

SECURE COMMUNICATION ON JAVA CARD

By

JAE HYUK JOO

Bachelor of Engineering

Tae-Jon University

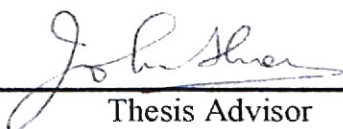
Tae-Jon, South Korea

1995

**Submitted to the Faculty of the
Graduate College of the
Oklahoma State University
in partial fulfillment of
the requirements for
the Degree of
MASTER OF SCIENCE
July, 2004**

SECURE COMMUNICATION ON JAVA CARD

Thesis Approved:



Thesis Advisor







Dean of the Graduate College

PREFACE

This research concentrates on secure communication between a host application and an applet on the Java card. We propose a one-time password based on one-way hash chains in this thesis. The passwords are generated by a one-way hash chain computation on the hash value of a Converted Applet (CAP) file. After each communication between a host application and an applet, the password is updated. More secure between a host application and an applet on the Java card is achieved successfully this way.

This thesis is organized into five chapters, references, and an appendix. Chapter I, Introduction, describes the background and current problems of Java Card technology. This chapter also states the objectives of this research. Chapter II, Literature Review, introduces the concepts of Smart Card and Java Card. Chapter III, Secure Communication, describes our approach to secure communications based on one-time passwords generated from a one-way hash chain. Chapter IV, Secure Communication Simulation, simulates the communication between a host application applet and a client applet. Various malicious attacks are also simulated. In Chapter V, we end with Conclusions, and suggest areas for future work. References for this thesis are listed next. The Appendix presents the source code designed and implemented for the simulation.

TABLE OF CONTENTS

Chapter	Page
I. INTRODUCTION	1
Background	1
Current Problems	2
Objectives	4
II. LITERATURE REVIEW.....	5
Smart Card	6
Basic Types Of Smart Card	6
Smart Card Merits And Weakness.....	7
Smart Card Hardware	7
Smart Card Communication	8
APDU Protocol	9
TPDU Protocol.....	11
ATR.....	11
Smart Card Operating Systems.....	11
Smart Card Systems.....	12
Open Platform And Open Card Framework	13
Java Card.....	14
Architecture.....	14
Language Subset	15
Java Card Virtual Machine	15
CAP File And Export File.....	16
Java Card Converter.....	16
Java Card Interpreter.....	17
Java Card Installer And Off-Card Installation Program	17
Java Card Runtime Environment	18
JCRE Lifetime	19
Java Card Runtime Features	20
Java Card Applets	20
Application Identifier (AID)	20
Objects	21
Atomicity	21
Exceptions.....	21
Applet Installation.....	22
Class Javacard.Framework.Applet.....	26

Chapter	Page
Java Card Security	27
Java Card Platform Security	28
Java Language Security	28
Additional Security Features of the Java Card Platform	28
Applet Firewall	29
Object Sharing Across Contexts	30
Java Card Platform Security Mechanisms	31
III. SECURE COMMUNICATION	33
One-way Function.....	33
One-way Hash Function	33
Message Digest (MD) 5	34
One-way Hash Chain	34
Applet Installation Process on the Java Card.....	35
Check In Compile Time.....	36
Class File Verification	37
Subset Checking.....	38
CAP File And Export File Verification	39
Check By Off-card Installation Program And Installer	41
Traditional Applet Communication And Risks	43
Applet Communication And Problems.....	43
Secure Applet Communication By One-time Password.....	46
Integrated System.....	53
Proposed one-time password merits	55
IV. SECURE COMMUNICATION SIMULATION BETWEEN HOST APPLICATION AND THE JAVA CARD.....	57
Prerequisites.....	57
Difference between the Java card and the simulation.....	58
Simulation.....	59

Chapter	Page
V. CONCLUSION.....	70
BIBLIOGRAPHY	72
APPENDIX.....	72
Ebank.java.....	74
Mask.java.....	78
HostEbank.java	82
Installer.java.....	101
HashMaker.java	118
InstallerInterface.java.....	135
Terminal.java	136

LIST OF TABLES

Table	Page
1. Command APDU structure	10
2. Response APDU structure	10
3. Supported and unsupported Java features.....	15
4. Exception classes in the java.lang package.....	22
5. Applet SELECT command	25
6. Methods in the class javacard.framework.Applet.....	27

LIST OF FIGURES

Figure	Page
1. Smart card communication model	9
2. Command and response APDU cases.....	11
3. ISO 7816-4 file system structure	12
4. Java Card Virtual Machine	15
5. Java card installer and off –card installation program	18
6. On-card system architecture	19
7. Structure of application identifiers (AID).....	20
8. Applet execution states and communication.....	24
9. The object system partitions on the Java Card platform.....	30
10. Shareable interface object mechanism.....	31
11. One-way hash chain generation process	35
12. Applet installation process.....	36
13. CAP file verification.....	41
14. After an applet installation.....	44
15. An applet selection from a host application.....	44
16. The PROCESS and DESELECT method	46
17. Initial state after applet installation.....	48
18. The proposed applet selection method.....	49
19. One-time password update.....	51

Figure	Page
20. i_{th} time selection an and update.....	53
21. Integrated System.....	54
22. The Real Java Card	58
23. Simulation Model.....	59
24. Masking hash installer	60
25. Ebank applet installation.....	61
26. Select and deposit with the first password.....	62
27. Reselect and deposit with the second password	64
28. A fake host application	65
29. Reselection after a fake host application tried	66
30. Original Ebank source code	67
31. Fake Ebank code.....	68
32. The result of fake Ebank applet	69

CHAPTER I

INTRODUCTION

Background

The explosion of the Internet and wireless digital communication has rapidly altered the way that people connect with each other. As the world has become more linked, the traditional business model that is face-to-face in-store transaction has been changed to the on-line transaction that uses a few mouse clicks not in a store but in our home or other places. To succeed in the electronic business market, the market must give the same confidence to people as the traditional face-to-face transaction market has given.

A smart card is similar to a credit card in an appearance. However, it has a microprocessor and memory chips in the plastic substrate of the card and has a computation capability. As result, smart cards offer great security and portability. Smart cards therefore are widely used in payment industries, banking industries, storage of identification and medical records, prepaid phone cards, retail royalty cards, and electronic purses, where data security and privacy are major concerns. The demand for smart cards has been growing every year. The total number of smart cards manufactured for use within the United States and Canada for 2000 is 28,430,000 [16]. The total number in 2001 is 41,320,000, a 45 percent growth rate [16]. Smart card shipments in the first half of 2001 totaled 14,800,000 and the second half of 2001 saw the growth of smart card shipments to 26,520,000, which is a 79 percent increase in smart card shipments from the first to second half of 2001[16]. Smart card growth from 1999 to 2000 was 37 percent [16].

However, smart cards still have some limitations although they are widely used. First one is the portability of applications. Second is the lack of flexibility to download applications into a card. This limitation is caused because smart card applications are burned in the chip. Therefore, once the card is made, the implanted application in the card cannot be updated or modified in any way. Finally, there is only a small universe of knowledgeable programmer who can develop the card applications. This is because the interior workings of the smart cards of each manufacturer are different even though the card size, shape, and communication protocol is normalized. Therefore, only a small group of highly skillful programmers can develop card applications.

A Java Card is simply defined as a smart card that is able to run several applications written in the java programming language within the limitations of memory size. The Java Card Virtual Machine offers a solution to the smart card's limitations with the following features. One is dynamic update. Card issuers might provide services by updating applets as they are developed. New applications can be added whenever needed. The only limitation is memory size. An existing applet can be deleted at any time. Another is security. The Java Programming Language is well known and proven as a secure language. The Java Card offers an applet firewall. This firewall provides a secure environment that allows several applets to reside on the same card without the leak of secure data. Java is also platform independent. Portability across different chip architectures, convenience of code reuse and all the benefits of object-programming languages are available to the Java Card Programmer.

Current Problems

Although Java Card technology offers a secure environment, there is still a hole in the security defense that malicious attackers can exploit to access secure data from a host application or the Java Card. Communication between a host application and the Java Card applet is one-way communication, known as a 'half-duplex'. The entire data exchanged between an applet and a host application takes place by using Application Protocol Data Units (APDUs). When the host application wants to communicate with a specific applet on the card, it sends an APDU that specifies the SELECT command, which is defined in the class `javacard.framework.Applet` and an application identifier (AID). When the Java Card Runtime Environment (JCRE) receives this command, it simply compares the command AID with the AID in internal table on the JCRE. If they match, the communication between a host application and an applet on the card is accepted and can proceed.

To select a specific applet as we have seen above, only the AID is needed in the host application, the JCRE, and the applet. If this AID is somehow revealed by the host application manager to malicious attackers or if this AID is obtained during the process of sending the SELECT method from a host application to the card by malevolent attackers, these attackers can pretend to be the host application to access secure information on the card.

In addition, Attackers can counterfeit the legal applet with this revealed or taken AID to harm the host application or by performing illegal functions. In Java Card technology, the Java Card Virtual Machine (VM) is divided in two parts: off-card VM and on-card VM. When an applet is installed on the card, the off-card VM verifies the

CAP (Converted Applet File) file, which the converter creates from class files as inputs. A CAP file is a binary representation of converted Java package. The CAP file includes class information, executable byte codes, linking information, verification information and so forth. After this process, the CAP file is installed to the card. However, this CAP file verification is only performed on the off-card part and not on the on-card portion. The lack of on-card verification makes it possible for a malicious applet to be installed onto a card via a legal applet installation process.

Objectives

The purpose of this research is to prevent malicious attackers from stealing secure information from the Java Card and a host application. Our objectives are the following:

- Our first objective is to verify a host application on the Java Card Runtime Environment (JCRE). As we have seen above, a malicious attacker can pretend to be a real host application with a stolen or revealed AID to select the Java Card applet for stealing secure information or performing illegal functions. This can happen since the JCRE uses only the AID to authorize a host application to access an applet. Therefore, the JCRE needs some verification functions for checking whether the host application is legal or not.
- Our second objective is to distinguish whether an applet on the card is legal or not. As we have seen above, currently this CAP file verification is performed only on the off-card VM. The JCRE therefore has to have some functions to verify the CAP file on the on-card VM. This makes an applet installation more secure.

CHAPTER II

LITERATURE REVIEW

There are three types of machine-readable cards. These are:

- embossing card
- magnetic stripe card, and
- smart card.

Embossing card is the oldest technique for marking ID cards in machine-readable form [2]. The embossed design on a card can be transferred to paper by printing using a simple and cheap device [2]. The type and positioning of the embossing are specified in ISO standard 7811 “Identification cards – Recording Technique” [2]. The simplicity of this technique has made worldwide proliferation possible, including developing countries [2]. Exploitation of this technique requires neither electric current nor connection to a telephone network [2].

Magnetic stripe card is read by pulling it across a reading head, either manually or automatically, whereby the data is read and stored electronically [2]. Processing this data no longer requires any paper [2]. This feature compensates one of the main drawbacks of the embossing card. The main drawback of the magnetic stripe technique is the considerable ease with which the stored data may be altered [2]. This type of card is often used in automatic machines where visual inspection is impossible, as in cash dispensers [2]. The potential criminal, having obtained valid card data, can use simple duplicates of cards at such unattended machines without having to forge the visual security features designed to prove the card’s authenticity [2].

The Smart card is the youngest and most sophisticated member of the identification card family in the ID-1 format [2]. It is characterized by an integrated circuit incorporated in the card, which contains elements used for data transmission, storage and processing [2]. We review a smart card in section 2.1, section 2.2 outlines a java card, and finally we review smart card security in section 2.3.

Smart Card

Smart cards are often called chip cards, or integrated circuit (IC) cards [1]. The integrated circuit incorporated in the credit card-sized plastic substrate contains elements used for data transmission, storage, and processing [1]. The initial smart card trials took place in France and Germany [1]. Due to vandalism and theft in the early 1980s, France's Public Telephone and Telegraph System began to move to a coinless public telephone system that used "smart" cards to hold a prepurchased value [3]. Microsoft employed smart card technology for the Microsoft Computer Dictionary. The smart card in computers and electronics is a circuit board with built-in logic or firmware that gives it some kind of independent decision-making ability [4]. In banking and finance, the smart card is like a credit card that contains an integrated circuit that gives it a limited amount of intelligence and memory [4].

Basic Types Of Smart Card

A smart card is divided into memory cards and microprocessor cards. Memory cards, and all smart cards for that matter, have some form of memory storage [3]. Memory cards are primarily designed for storing information or values and are commonly used for applications such as disposable prepaid telephone cards used in

public telephones [2]. Microprocessor cards are truly “smart” cards [2]. Henceforth in this thesis, a smart card refers to a microprocessor card.

Smart Card Merits And Weakness

A smart card can perform all functions as needed. A smart card can save transaction time since the smart card does not need to access remote databases. Smart cards are broadly used in areas such as payment and banking where data security and privacy are main concerns. However, in spite of these merits, there are limits to using a smart card widely. One is portability of applications. Another is flexibility to download applications on to a card. A small universe of knowledgeable programmers is another obstacle.

Smart Card Hardware

A smart card contains a central processing unit, RAM, ROM, mass storage (EEPROM). The central processing unit in most current smart card chips is an 8-bit microcontroller, usually based on the Motorola 6805 or Intel 8051 instruction set, and with clock speeds up to 5MHz[1]. High-end cards very often include a clock multiplier (by 2,4, or 8), which allows these cards to operate up to 40MHz (5MHz times 8) [1]. Newer smart card chips have a 16-bit or 32-bit microcontroller, and smart cards with reduced instruction set (RISC) architecture are also available [1].

RAM (random access memory) is used as temporary working space for storing and modifying data [1]. RAM is nonpersistent memory; that is, the information content is not preserved when power is removed from the memory cell. RAM can be accessed an unlimited number of times, and none of the restrictions found with EEPROM apply.

ROM (read-only memory) is used for storing the fixed program of the card [1]. As the name implies, this type of memory cannot be written [2]. No power is needed to hold data in this kind of memory. Because the data are stored in the chip by hard wiring, a smart card's ROM contains operating system routines as well as permanent data and user applications. These programs are built into the chip during production [2].

EEPROM (electrical erasable programmable read-only memory), like ROM, can preserve data content when power to the memory is turned off [1]. Functionally, an EEPROM corresponds to a PC hard disk, since data remain in memory in the absence of power and can be modified as necessary [2]. User applications can also be written into EEPROM after the card is made. The important electrical parameters of EEPROM are the number of write cycles over the lifetime of a card, data retention period, and access time. EEPROM in most smart cards can reliably accept at least 100,000 write cycles and can retain data for 10 years.

ROM is the least expensive of these three kinds of memory [1]. EEPROM is more expensive than ROM because an EEPROM cell takes up four times as much space as a ROM cell. RAM is very scarce in a smart card chip. A RAM cell tends to be approximately four times larger than an EEPROM cell.

The current generation of chip cards has an eight-bit processor, 16KB read-only memory, and 512 bytes of random-access memory [18]. This gives them the equivalent processing power of the original IBM-XT computer [18].

Smart Card Communication

A smart card is inserted in to a card acceptance device (CAD), which may be connected to another computer [1]. Card acceptance devices can be classified as two

types: readers and terminals [1]. A reader is connected to the serial, parallel, USB port of a computer, through which a smart card communicates. Terminals, on the other hand, are computers on their own. A terminal integrates a smart card reader as one of its components. The communication pathway between the card and the host is half-duplexed; that is, the data can either be sent from the host to the card or from the card to the host but not both at the same time.

Smart cards speak to other computers by using their own data packets – called APDUs (application protocol data units) [1]. APDU denotes internationally standardized data units in the application layer, which in the OSI model is layer 7 [2]. This is the layer, which in Smart Cards is located directly above the transmission protocols [2]. An APDU contains either a command or a response message. A host and a smart card mutually exchange command APDUs and response APDUs, as shown in Figure 1.

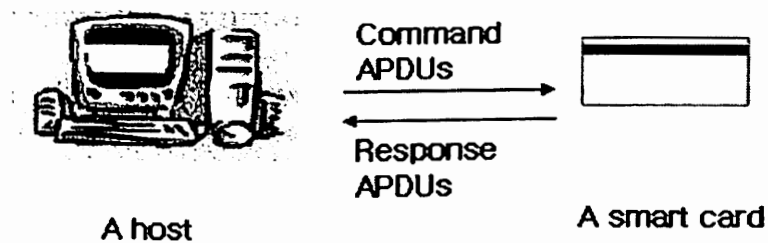


Figure 1. Smart card communication model [1]

APDU Protocol

The APDU protocol as specified in ISO 7816-4, is an application-level protocol between a smart card and a host application [1]. An APDU message consists of either a command APDU or a response APDU. A command APDU is used by a host and a response APDU is used by a smart card as shown in Figure 1. A command and response structure is illustrated in Table 1 and Table 2, respectively.

Mandatory header				Optional body		
CLA	INS	P1	P2	Lc	Data field	Le

Table 1. Command APDU structure [1]

Optional body	Mandatory Trailer	
Data field	SW1	SW2

Table 2. Response APDU structure [2]

The command APDU header consists of 4 bytes: CLA (class of instruction), INS (instruction code), and P1 and P2 (parameters 1 and 2) [1]. CLA field indicates a class of command APDUs. INS field mentions a command instruction. The two parameter bytes P1 and P2 are used to provide further qualifications to the instruction [1]. The section after the header in a command APDU is an optional body that varies in length [1]. The Lc field shows the length of a data field in bytes. The data field contains data that are sent to the card for executing the instruction specified in the APDU header [1]. The Le field indicates the length in bytes anticipated by the host in the card's reply. The response APDU is composed of an optional body (data) and a mandatory trailer field (sw1 and sw2). The body consists of the data field, whose length is determined by the Le field in the corresponding command APDU. The trailer consists of two fields SW1 and SW2, together called the status word, denoting the processing state in the card after executing the command APDU. APDUs communication between a host and a card can be divided into 4 cases as shown in Figure 2.

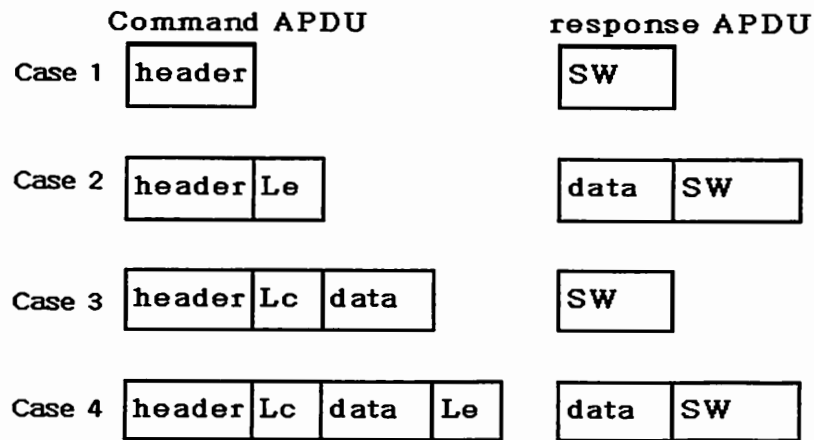


Figure 2. Command and response APDU cases [1]

TPDU Protocol

The data structures exchanged by a host and a card using the transport protocol are called transmission protocol data units, or TPDU's [1]. A total of 15 transmission protocols are provided, and their basic functions are defined in [2]. They are designated "T=" plus a serial number. Currently, two transmission protocols are used widely. One is the T=0 protocol which is byte-oriented and used for processing the smallest unit such as a single byte. The other is the T=1 protocol which is block-oriented. The unit of processing is a block-a sequence of bytes.

ATR

After booting the power supply, the clock, and the reset signal, the smart card sends out an answer to reset (ATR) at the I/O pin [2]. This conveys to the host the parameters required by the card for establishing a data communication pathway [1]. The ATR is up to 33 bytes. ATR is composed of transport protocol, data transmission rate, the chip serial number, mask version number, and other information that is needed by a host.

Smart Card Operating Systems

Smart card operating systems have little resemblance to desktop operating

systems, such as UNIX, Microsoft Windows, or even DOS [1]. Rather, the smart card operating system supports a collection of instructions on which user applications can be built [1]. ISO 7816-4 standardizes a wide range of instructions in the format of APDUs. ISO 7816-4 APDUs are largely file system-oriented commands, such as file selection and file access commands. Smart cards defined in ISO 7816-4 can have a hierarchical file system structure, as shown in Figure 3.

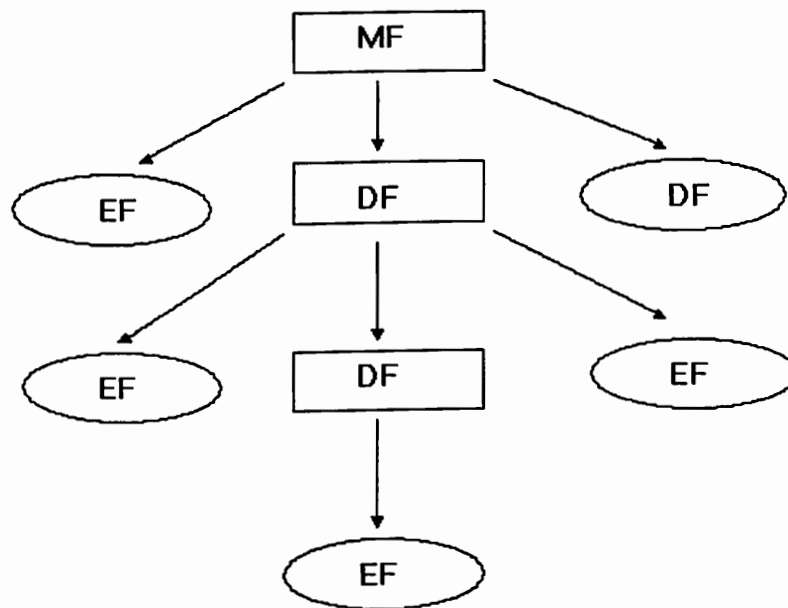


Figure 3. ISO 7816-4 file system structure [1]

There are three types of files. One is the master file (MF). The second one is the dedicated file (DF). The last one is the elementary file (EF). Each file is specified by either a 2-byte identifier or a symbolic name up to 16 bytes [1]. MF is the root, DF is a directory file, and EF is a data file. It is similar to the UNIX file system. The MF can contain DFs and EFs. A DF can contain other DFs and EFs. An EF can contain other EFs.

Smart Card Systems

Smart card systems are distributed systems that consist of two parts: the host

system residing in the computer connected to the reader or in the terminal and the card system inside a smart card [1]. Most smart card software, including system software and user application software, runs on the host side. The system software recognizes a specific smart card and handles communication between the user application and the card. The system software manages a card issuance and operations. The system software supports security and key management as well. User applications implement functions that work with a specific card or an application on the card.

Card software runs on the smart card itself. It contains system and user application software. The system software typically includes the operating system and utilities that control memory management, handle I/O communication with the host, ensure data integrity and security, support the ISO file system, and provide system utilities to the card application [1]. Card applications include data and maintain functions that work on data.

Open Platform And Open Card Framework

The Open platform (OP) defines an integrated environment for the development and operation of multiple-application smart card systems [1]. The OP is made up a card specification and a terminal specification. The cross-industry, nonproductive-specific requirements to apply an open platform and the off-card communication between the terminal and the on-card application management are prescribed in the card specification. The terminal specification defines the part of the application architecture within the terminal.

The OCF is a standard framework announced by an Industry consortium that provides for inter-operable smart cards solutions across many hardware and software

platforms [7]. OCF is the host-side application framework providing a standard interface for interacting with card readers and applications in the card [1]. The OCF is a functionally divisible structure model that divides functions among card terminal vendors, card operating system providers, and card issuers. This makes a card more independent.

Java Card

A Java Card is a smart card that is written in the java programming language. Java Card technology defines a secure, portable, and multi-applicable smart card platform that incorporates many of the advantages of the Java language. The Java Card extends the limits of the smart card by providing dynamic updates and more security.

Architecture

The Java Card 2.0 specification contains detailed information for building the java card virtual machine and application programming interface (API) in smart cards [5]. The minimum system requirements are 16 kilobytes of read-only memory (ROM), 8 kilobytes of EEPROM, and 256 bytes of random access memory (RAM). The java card virtual machine is divided into two parts: one part that works off-card and the other part that works on-card.

Java card technology essentially defines a platform on which applications written in the java programming language can run in smart cards and other memory-constrained devices [1]. This platform is distributed into both the smart card and desktop environment because of the split virtual machine features. It consists of three parts. The first one is the java card 2.1 virtual machine (JCVM), the second one is the java card 2.1 runtime environment (JCRE), and the third one is the java card 2.1 application programming interface (API).

Language Subset

As seen in Table 3, the java card terminal supports only a subset of the java language because of a java card's memory limitation.

Supported Java Features	Unsupported Java Features
<ul style="list-style-type: none"> -Small primitive data types: byte, boolean, short -One-dimensional arrays -Java packages, classes, interfaces, and exceptions -Java object-oriented features: inheritance, virtual methods, overloading and dynamic object creation, access scope, and binding rules -The int key word and 32-bit integer data type support are optional 	<ul style="list-style-type: none"> -Large primitive data types: long, double, float -Characters and strings -Multidimensional arrays -Dynamic class loading -Security manager -Garbage collection and finalization -Threads -Object serialization -Object cloning

Table 3. Supported and unsupported Java features [1]

Java Card Virtual Machine

JCVM is composed of two separate pieces, as depicted in Figure 4. The on-card piece includes the java card byte-code interpreter. The off-card piece includes the converter that runs on a pc or workstation.

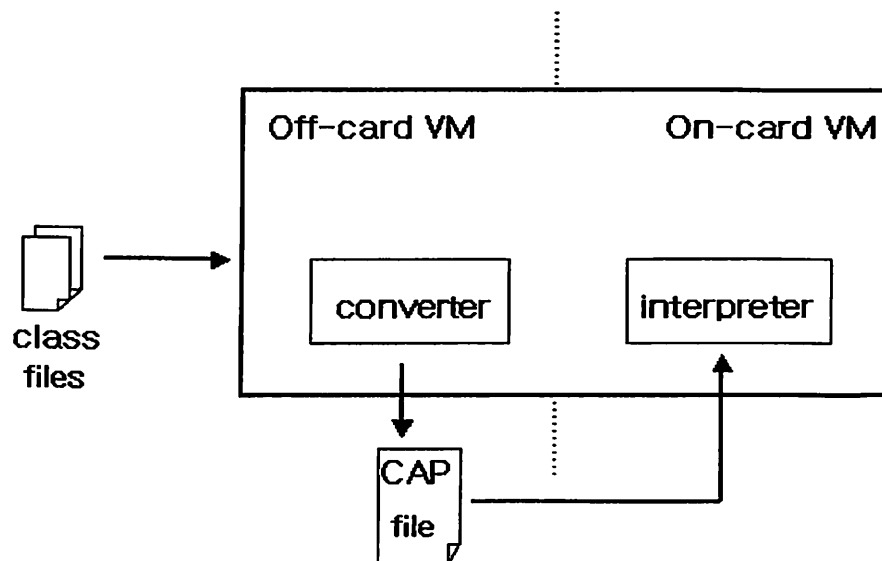


Figure 4. Java Card Virtual Machine

CAP File And Export File

A CAP (Converted Applet) file is a container file in JAR (Java™ Archive) file format. The JAR file format enables us to bundle multiple files into a single archive file. The JAR file format is used as the container format for CAP files [1]. A CAP file is a JAR file that contains a set of components, such as class information, executable byte-codes, linking information, verification information, and so forth. The CAP file also includes the manifest file. This manifest file offers additional human-readable information concerning the contents of the CAP file and the package that it stands for. A software form loading on the java card virtual machine is a CAP file format. For example, CAP files enable dynamic loading of applet classes after the card has been made.

Export files are not loaded onto smart cards and thus are not directly used by the interpreter. Exported files are produced and consumed by the converter for verification and linking purposes [1]. An export file contains public API information for an entire package of classes. It defines the access scope and name of a class and the access scope and signatures of the methods and fields of the class.

Java Card Converter

The conversion unit of the java card converter is a package. Class files are produced by java from source code. Then, the converter preprocesses all the class files that make up a java package and converts the package to a CAP file [1]. Besides producing a CAP file, the converter generates an export file for the converted package. During the conversion, the converter verifies that the load images of the java classes are well formed, performs static variable initialization, checks the Java Card language

subset's violations, resolves symbolic references to classes, methods, and fields into a more compact form, optimizes bytecode, allocates storage, and creates virtual machine data structures to represent classes.

Java Card Interpreter

The java card interpreter provides runtime support of the java language model and thus allows hardware independence of applet code [1]. The interpreter executes bytecode instructions, carries out applet, manages memory allocation and object creation, and plays an important role in guaranteeing runtime security.

Java Card Installer And Off-Card Installation Program

The off-card installation program is implemented to the pc or workstation side and the on-card installer is implemented to the smart card side as shown in Figure 5. The off-card installation program sends out the executable binary in a CAP file to the on-card installer via a card acceptance device (CAD). The on-card installer writes it to the smart card memory, connects it to the other classes already placed on the card, and creates any data structures used by the java card runtime environment.

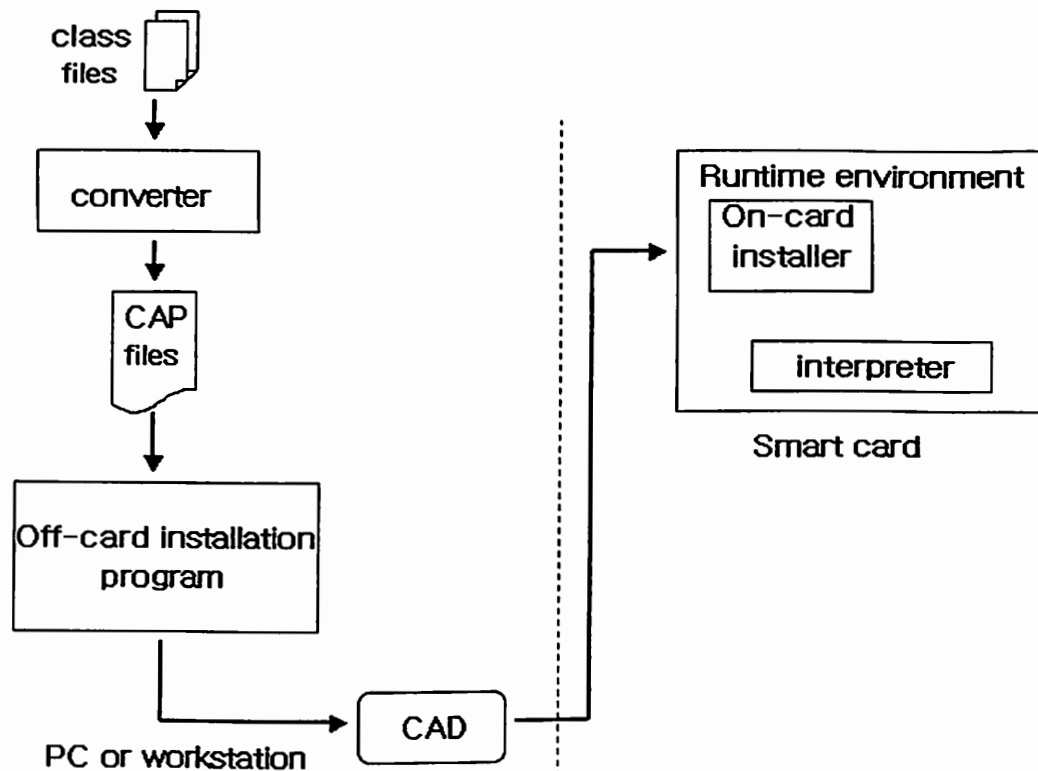


Figure 5. Java Card installer and off-card installation program [1]

Java Card Runtime Environment

The JCRE is responsible for card resource management, network communications, applet execution, and on-card system and applet security [1]. As illustrated in Figure 6, the JCRE is composed of APIs, industry-specific extensions, installer, System classes, java card virtual machine (JCVM) and native methods. Applets are separated from the JCRE.

The system classes are in charge of managing transactions, managing communication between the host applications and java card applets, and controlling applet creation, selection, and deselection [1]. To complete tasks, the system classes typically invoke native methods. A specific industry or business can supply add-on libraries to provide additional services or to refine the security and system model. The installer enables the secure downloading of software and applets onto the card after the

card is made and issued to the card holder [1].

Java card applets are applications on the java card platform. Applets can be downloaded and added to a java smart card after being manufactured and it is managed by the JCRE.

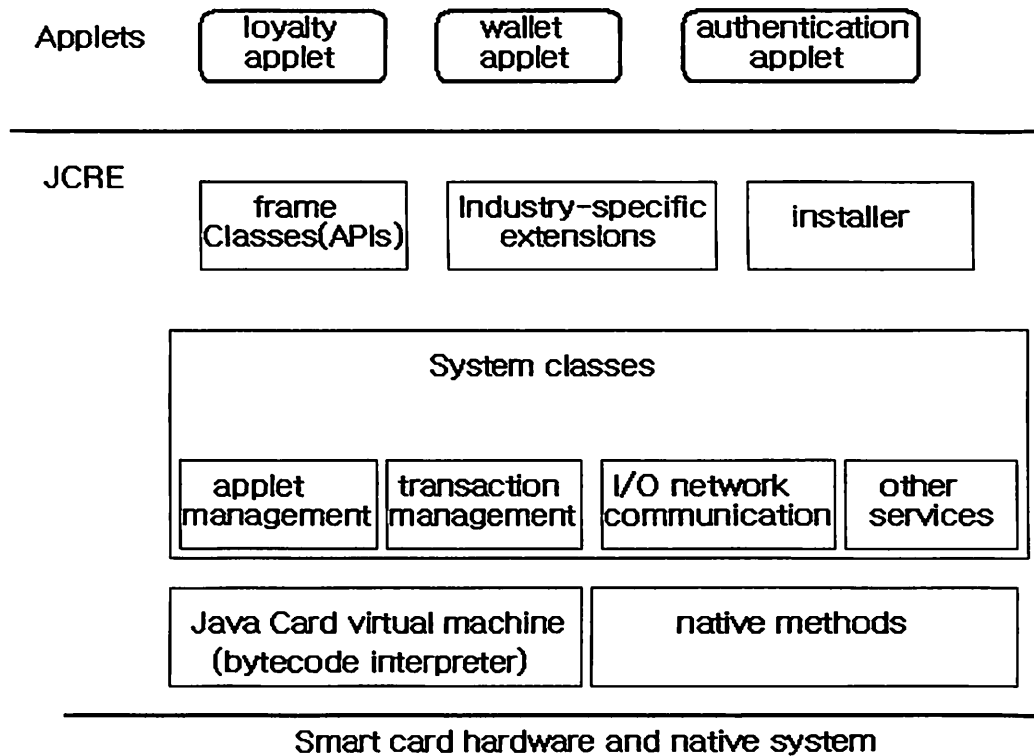


Figure 6. On-card system architecture

JCRE Lifetime

The JCVM's lifetime coincides with that of the card itself: It begins at some time after the card is manufactured and tested, and before it is issued to the cardholder, and it ends when the card is discarded or destroyed [6]. The JCRE initialization is performed at card initialization time and it is executed only once during the card lifetime. During this process, the JCRE initializes the virtual machine and creates objects for providing the JCRE services and managing applets [1]. As applets are installed, the JCRE creates applet instances, and applets create objects to store data.

Java Card Runtime Features

Java card assists three runtime features aside from java language runtime model: persistent and transient objects, atomic operations and transactions, and applet firewall and the sharing mechanisms.

Java Card Applets

An applet is an application that runs on java smart card. Java Card applets do not have to be burned into the ROM during the manufacturing time. These can be downloaded later after the card has been manufactured. An applet class must extend from the `javacard.framework.Applet` class [1]. The Applet class is the super class and defines the methods and variables of all applets. The JCRE supports a multi-application environment so multiple applets can exist together on a single card.

Application Identifier (AID)

Each applet instance and package in the java platform has a unique application identifier. ISO 7816 specifies AIDs to be used for unique identification of card applications and certain kinds of files in card file systems [1].

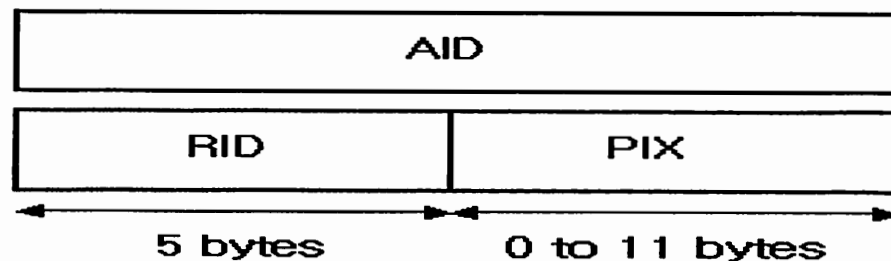


Figure 7. Structure of application identifiers (AID)

The AID is composed of two elements: RID and PIX as shown in Figure 7. The first element is the register identifier (RID) of 5 bytes length. It is allocated by a national

or international registration office, and contains a country code, an application category and a number identifying the application provider [2]. The second element is a variable-length value: PIX (proprietary identifier extension, 0~11 bytes). Thus an AID can range from 5 to 16 bytes in total length [1].

Objects

Runnable applets on the card are objects that are instances of classes or array types. A persistent object that is made by the new operator holds states and values across a CAD (Card Acceptance Device) session. Any update to a single field in a persistent object has an atomic property. A transient object created by invoking the Java Card APIs does not hold states and values across a CAD session. Any update to a single field in a transient object does not have an atomic property.

Atomicity

Atomicity means that any update to a single field in a persistent object or to a class field is guaranteed to either terminate successfully or else be restored its original value if error occurs during the update time. Atomicity on the java card supports any update to a single field in a persistent object or a single class field and a transactional model, in which an applet can group a set of updates into a transaction.

Exceptions

An exception is an event that disrupts the normal flow of instructions during the execution of a program [1]. When an applet detects or throws programmatically internal runtime problem, the JCRE and the JCVM throw exceptions. Exception classes in the java.lang package are listed in Table 4.

Throwable	Exception	RuntimeException
ArithmeticException	ArrayStoreException	ArrayIndexOutOfBoundsException
ClassCastException	NullPointerException	IndexOutOfBoundsException
SecurityException	NegativeArraySizeException	

Table 4. Exception classes in the java.lang package

The class Throwable is a common ancestor for all exception classes. It extends to the class Exception. All java Card checked exception classes extend from the class CardException that derives from the class Exception and all java Card unchecked exception classes extend from the class cardRuntimeException that comes from the class Exception. The class CardException is the root class for all checked exceptions that indicate a programming error in an applet. The class cardRuntimeException is the root class for all unchecked exceptions, often called runtime exceptions, that show unexpected runtime problems, programming errors, or erroneous APDU processing states in the java card platform. The java card exception classes give a short type numerical reason code that is used to describe optional details related to the throwing of the exception.

Applet Installation

A running applet in the JCRE is an instance of the applet class. A multiapplication environment is supported by the JCVM and each applet instance has a unique AID. Applet installation refers to the process of loading applet classes in a CAP file, combining them with the execution state of the JCRE, and creating an applet instance to bring the applet into a selectable and execution state [1].

To load an applet, the off-card installer takes the CAP file as an input. Then it transforms the input into a sequence of APDU commands that bring the CAP file content.

By exchanging the APDU commands with the off-card installation program, the on-card installer writes the CAP file content into the card's persistent memory (EEPROM) and links the classes in the CAP file with other classes that reside on the card [1]. The installer also creates and initializes any data that are used internally by the JCRE to support the applet [1]. At the last step of the installation, the installer invokes the install method.

The JCRE calls the install method as the last step during applet installation to create an applet instance—a runnable applet. The arguments to the install method carry the applet installation parameters [1]. The install method calls the applet's constructor. This constructor creates an applet instance by using the new operator. In the constructor, an applet typically creates objects that the applet needs during its lifetime and it initializes objects and the applet's internal variables. An applet also calls the register method.

The register method has two functions. One is that it stores a reference to the applet instance with JCRE. Second - it assigns an AID to the applet instance. As we have seen above, each applet instance on the card has a unique AID to identify. The CAP file that defines the applet classes contains a default AID [1]. However, an applet may choose to have an AID different from the default one. When an applet's instance is created and registered with JCRE, an applet's life starts.

During applet installation, the installation parameters are sent to the card along with the CAP files that define an applet [1]. Then, the JCRE supplies the installation parameters to the applet through the arguments to the install method. The applet designers or the card issuers define the content and format of the installation parameters. Usually, they include applet configuration parameters and initialization values.

Configuration parameters can be used to specify the size of an internal file, an array, AID (-not the default one in the CAP file) and so on [1]. For example, applet initialization values can specify the initial balance such as the card holder's ID and the account number in an electronic wallet. The installation process is transactional. This means that when errors such as programmatic failure, running out of memory, or card tear occur, the installer discards the CAP file and any applets it had created during the installation. It then recovers the space and the previous state of the JCRE.

When an applet is installed for the first time successfully, it enters an inactive state. This applet becomes active when a host application selects it. Through exchanging APDUs the communication between an applet and a host application is performed. This is illustrated in Figure 8.

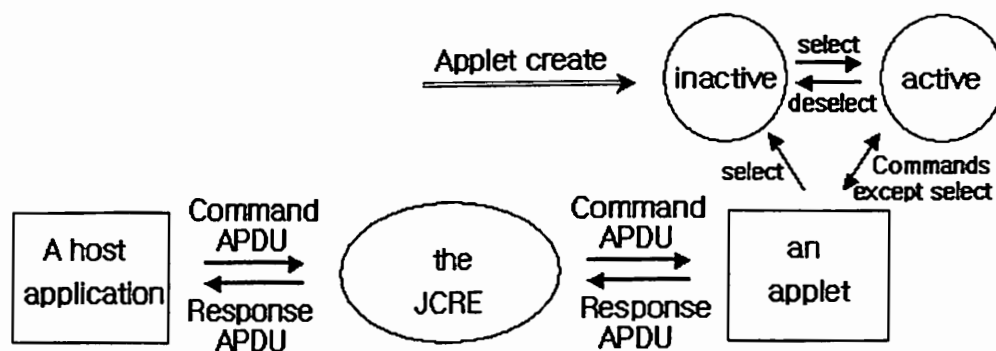


Figure 8. Applet execution states and communication

When the host application wants to select an applet to run, it sends an APDU that specifies the SELECT command and the AID of the requested applet [1]. The Select APDU command is the only APDU command that is uniformed on the Java Card platform. It ensures interoperable applet selection on various Java Card platform implementations. This standardized select command is depicted in Table 5. The data part of the SELECT APDU contains an applet AID. This AID length is between 5 and 16

bytes. The entire data field of the APDU must match the AID of the applet to be selected successfully.

CLA	INS	P1	P2	Lc	Data Field
0x0	0xA4	0x4	0x0	Length of AID	AID bytes

Table 5. Applet SELECT command

When the JCRE gets an APDU, it decodes its header (CLA, INS, P1, and P2) to determine whether this command is a select command or not. If this command is a select command, the JCRE determines whether the AID in the APDU data matches with that of an applet on the card or not. That is, the JCRE searches its internal table for an applet whose AID matches the one specified in the command [1]. If the JCRE can't find it, the JCRE returns the status word 0x6999 to the host application. This indicates that applet selection is failed. If it is found, the JCRE informs the applet of its selection by invoking its select method. In the SELECT method, the applet can check whether its conditions for selection have been met, and if so, it can set internal variables and states necessary to handle subsequent APDUs [1]. The applet returns true from the call to the select method if it is ready to accept incoming APDUs via its process method. All subsequent APDUs (including the SELECT APDU) are forwarded to the current applet until a new applet is selected.

Before a new applet is selected, the JCRE deactivates the current applet by calling its current applet [1]. This deselect method makes the applet carry out any cleanup operations to prepare to go "off stage" and makes another applet ready to execute. If the APDU is not for applet selection, the JCRE delivers it to the current applet for processing.

When the JCRE receives an APDU command, it calls the current applet's process method. An applet is expected to execute a function requested in the APDU in the process method.

Class Javacard.Framework.Applet

The JCRE calls the methods to install, register, select, process, and deselect.

Public static void	install (byte[] bArray, short bOffset, byte bLength) The JCRE calls this static method to create an instance of the Applet subclass.
Protected final void	register() This method is used by the applet to register this applet instance with the JCRE and to assign the default AID in the CAP file to the applet instance.
Protected final void	register (byte[] bArray, short bOffset, byte bLength) This method is used by the applet to register this applet instance with the JCRE and to assign to the applet instance the AID specified in the array bArray.
Public boolean	select () The JCRE calls this method to inform the applet that it has been selected.
Public abstract void	process (APDU apdu) This JCRE calls this method to instruct the applet to process an incoming APDU command.
Public void	deselect () The JCRE calls this method to inform the currently selected

	applet that another (or the same) applet will be selected.
--	--

Table 6. Methods in the class javacard.framework.Applet [1]

The JCRE calls install method to construct an instance of an applet. One of the two register methods registers the applet instance with the JCRE. When the JCRE accepts the select method, the JCRE checks first whether the applet is selected or not. If the applet is selected previously, the JCRE deselects the current applet, and then it selects the new applet by the select method. After successful selection, each APDU (including the SELECT APDU) is delivered to the active applet via a call to its process method [1]. The process method handles APDU commands and thus offers functions of the applet.

Java Card Security

Security refers to protection against unwanted disclosure, modification, or destruction of data in a system and also to the safeguarding of systems themselves [8]. It also refers to the technologies used to make a service resistant to unauthorized access to the data that it holds or for which it is responsible [4]. In order to exchange data securely, these distributed applications require access to a variety of security services, which include data confidentiality, data integrity, authentication, and non-repudiation [8]. Confidentiality means keeping confidential the content of users' data, even the users' identities – in fact anything which is not to be made generally known [9]. Data integrity pertains to protection of information from modification by unauthorized users [8]. Authentication is a mechanism or process that associates a particular person's identity with a statement, action, or event to verify that internal user identification is correctly associated with its owner [10].

Java Card Platform Security

The security features of the Java Card platform are a combination of the basics of Java language security and additional security protections defined by the Java Card platform [1].

Java Language Security

The Java Card platform supports a subset of the Java programming language and virtual machine specifications appropriate for smart card applications [1].

- The Java language is strongly typed. No illegal data conversions can be done.
- The Java Language enforces boundary checks on array access.
- Variables must be initialized before they are used.
- The level of access to all classes, methods, and fields is strictly controlled.

Additional Security Features of the Java Card Platform

Card issuers desire a secure computing platform to meet the special requirements of the smart card system [1].

- Transient and persistent object models – Objects are stored by default in persistent memory. For security reason, the Java Card platform allows temporary data to be stored in transient objects in RAM.
- Atomicity and transactions – Three security features are defined. First, a single update to a field of a persistent object or a class will be atomic. Second, the method `arrayCopy` in the class `javacard.framework.Util` guarantees atomicity for block updates of multiple data elements in an array. Third, the Java Card platform supports a transaction model in which an applet can atomically update several different fields in different persistent objects.

- Applet firewall - This is defined below.
- Object sharing - First, the JCRE is a privileged user that has full access to applets and to objects created by applets. Second, an applet gains access to JCRE services and resources through JCRE entry point objects. Third, applets in different contexts can share objects that are instances of a class implementing a shareable interface. This is detailed below. Finally, applets and the JCRE can share data through global arrays.
- Native methods in applets – Native methods are not executed by the Java Card virtual machine.

Applet Firewall

The applet firewall is a mechanism to protect sensitive data of single applets and to support cooperative applications. This isolates an applet. With applet isolation, the applet firewall provides protection against the most frequently anticipated security concerns: developer mistakes and design oversights that might allow sensitive data to be leaked to another applet [1].

The applet firewall partitions the Java Card object system into separate protected object spaces called contexts [1]. The boundary between one and another context is the firewall. When an instance of an applet is created the JCRE allocates it to a context that usually is a group context. When one applet accesses an object of different context, this is not permitted because of the firewall. Accessing an object in the same group context is allowed. Notice that there is only one active context within the virtual machines at any time: either the JCRE context or an applet's group context. There is also a firewall

between the applet space and the JCRE. Figure 9 describes the object system partitions on the java card platform.

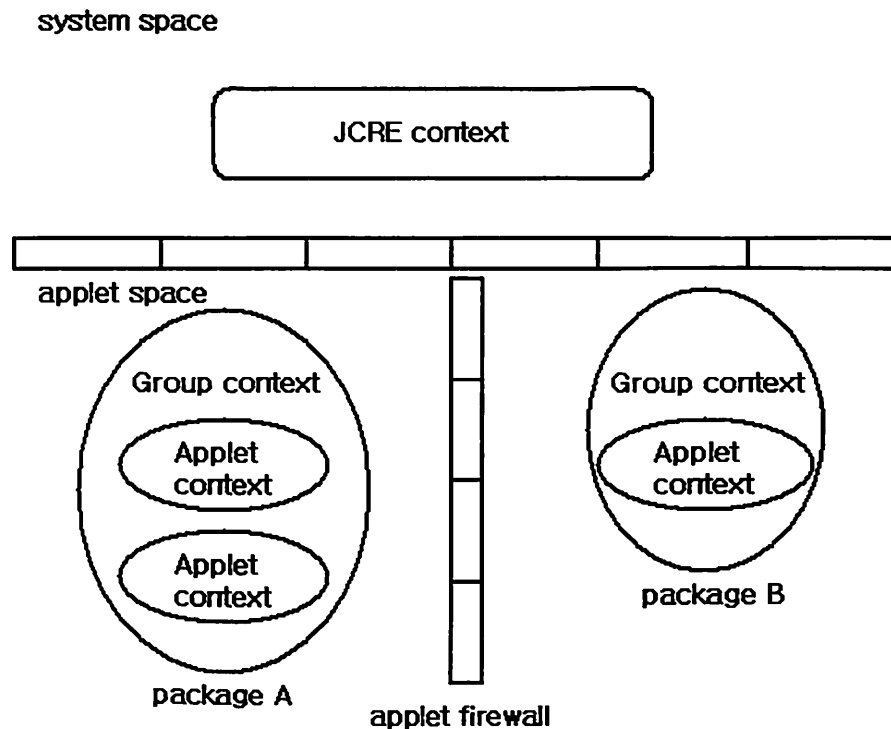


Figure 9. The object system partitions on the Java Card platform [1]

Object Sharing Across Contexts

The applet cannot reach beyond its context to access objects owned by the JCRE or by another applet in a different context [1]. But in situations where applets need to execute cooperatively, java card technology provides well-defined and secure sharing mechanisms that are accomplished by the following means: JCRE privileges, JCRE entry point objects, global arrays and shareable interfaces.

Since the JCRE is the system context that has particular privileges, it can call a method on any object or access an instance field of any object on the card. Such system privileges enable the JCRE to control system resources and manage applets [1]. And JCRE entry point objects are normal objects owned by the JCRE context, but they have

been flagged as containing entry point methods that are the gateways through which applets ask privileged JCRE services. Global arrays are handled by a particular kind of JCRE entry point object. The applet firewall enables public fields (array components and array length) of such arrays to be accessed from any context.

A shareable interface is simply an interface that extends, either directly or indirectly, the tagging interface `javacard.framework.Sharable` [1]. An object of a class that implements a shareable interface is called a shareable interface object (SIO). An SIO is a normal object to the owning context but is an instance of the shareable interface type to any other context. Methods only defined by SIO can access any other context. Shareable interface object mechanism is illustrated in Figure 10.

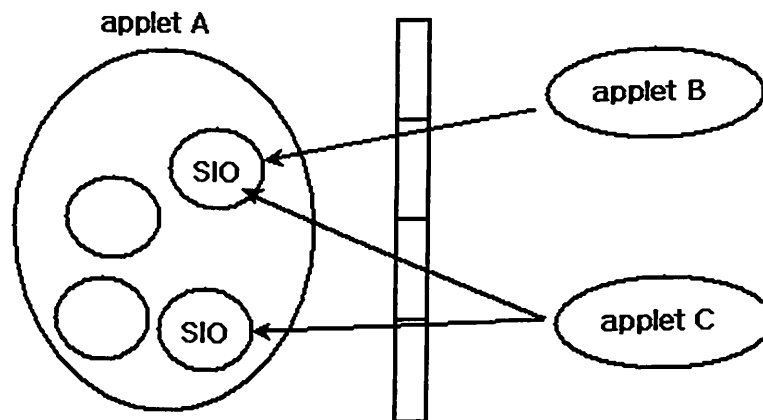


Figure 10. Shareable interface object mechanism

Java Card Platform Security Mechanisms

The Java Card security features are enforced through a number of mechanisms that are addressed at every level from the applet development process and the installation procedure to the runtime enforcement [1].

Section 3.1 describes each level security mechanism - compile-time checking, class file verification, subset checking, and CAP file and export file verification.

- Runtime Security Enforcement [1] – It covers both ensuring Java language type safety and enforcing applet isolation through the applet firewall. The CAP file contains sufficient type information to enable thorough type checking at runtime. To enforce the applet firewall, when an object is accessed, the Java Card interpreter performs checks to determine whether the access can be granted [1].

CHAPTER III

SECURE COMMUNICATION

Although Java Card technology offers a safe environment, Java Card technology does not standardize applets installation policy [1]. It is therefore possible for a malicious attacker to execute an applet's function to gain illegal access by pretending to be a legal host application. Moreover, attackers can legally install a fake applet that has unauthorized or malicious data or code on the card. In this thesis, we adapt one-time password via one-way hash chain to the Java Card to augment Java Card security.

One-way Function

A one-way function is a mathematical function that is significantly easier to compute in one direction (the forward direction) than in the opposite direction (the inverse direction) [11]. For example, it might be possible to work out the function in the forward way in seconds but to compute its opposite it could take months or years, if at all. We define the one-way function as $Y = F(X)$. If the X value and the function F are revealed, anyone can calculate the Y value. However, even if the Y value and the function F are revealed no one can calculate the X value.

One-way Hash Function

In our work we use one-way hash functions and chains to enhance security. A one-way Hash function can serve as a cryptographic checksum. A one-way hash function is a mathematical function that takes a message string of any length and returns a smaller fixed-length string (hash value) [12]. This has many names such as message digest, checksum, contraction function, data integrity check, message authentication code, message integrity check, and data authentication code. A hash function H accepts a

variable-size message M as input and outputs a fixed-size representation $H(M)$ of M [13]. M will be much larger than $H(M)$. $H(M)$ can be 64 or 128 bits but M may be a megabyte or more. To serve the authentication process properly a hash function F must have the following properties [13]:

- F can be applied to an argument of any size.
- F produces a fixed-size output. $F(x)$ is relatively easy to compute for any given x .
- For any given y it is computationally infeasible to find x with $F(x) = y$. This property indicates this function is a one-way function.
- For any fixed x it is computationally infeasible to find x' from x with $F(x)' = F(x)$. This ensures that an alternative message hashing to the same value as a given message can't be found. This prevents forgery and also allows F to function as a cryptographic checksum for integrity.

Message Digest (MD) 5

MD5 is a one-way hash function designed by Rivest after some cryptanalytic attacks were discovered against Rivest's previous MD4 algorithm [8]. MD5 handles arbitrary lengths of blocks as input and computes a message digest of 128 bits.

One-way Hash Chain

One-way chains are a widely-used cryptographic primitive. One of the first uses of one-way chains was for one-time passwords by Lamport [14]. Haller later used the same approach for the S/KEY one-time password system [15]. This chain is used in many other applications. This chain can be generated by repeatedly applying a one-way hash function on a random number. First we randomly pick the last element s to generate a chain's length n . We apply this to the hash function (H) to S_n as a seed. S_i is a hash value

from S_{i+1} by the hash function. S_{i+1} is a hash value from S_{i+2} by the hash function. i shows how many times the hash function is used. For example, n is 4 and i is 2, it shows hash function is used 3 times (repeat times = $n-i+1$). The verification of this one-way hash chain is in reverse order of its generation. For example a sender sends S_0 . The receiver stores S_0 . Next the sender sends S_1 . When a receiver accepts this, it can verify S_0 by computing $S_0 = H(S_1)$ as the receiver has S_1 . However the sender cannot generate S_2 from the S_1 it received. This verification can be performed until it meets S_n . The one-way hash chain's generation and verification order is depicted in Figure 11.

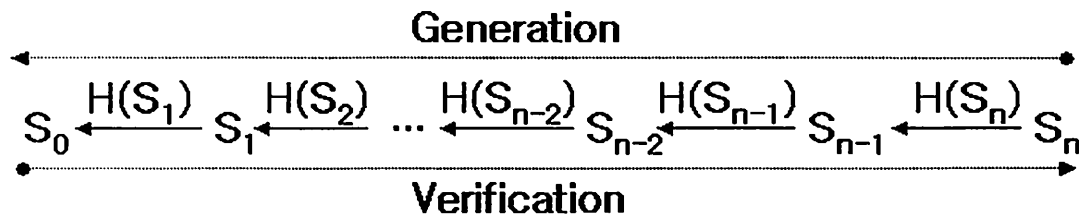


Figure 11. One-way hash chain generation process

By this one-time password, communication between a host application and the Java Card will be made secure. We describe the applet installation process in section 3.1. In section 3.2, we show existing communication mechanisms between a host application and an applet on the Java Card and we describe current security problems. Finally, our proposed one-time password is specified in section 3.3.

Applet Installation Process on the Java Card

Applets to run on the Java Card can be either burned into the ROM during manufacture time or can be downloaded whenever it is needed later after the card has been made. In this thesis we investigate security problems caused by installing applets after manufacture time by downloading them, that is, the applet installation process is performed from host side to the Java Card side via a card acceptance device (CAD). In

this section, we go through the applet installation process with security verification. This is illustrated in Figure 12.

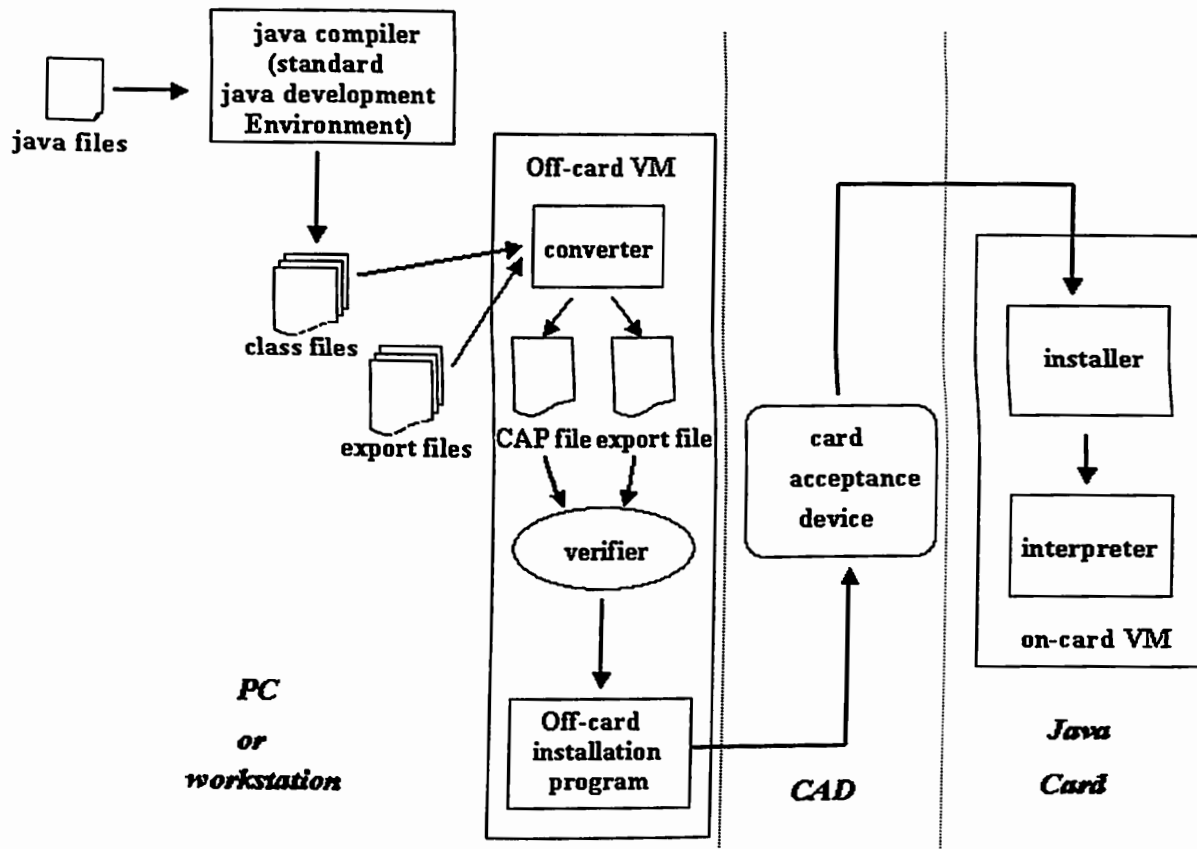


Figure 12. Applet installation process

An applet is composed of one or more Java files. These source codes are compiled by using any standard Java development environment like Symantec's Café or Sun's JDK. After compilation, binary class files are produced.

Check In Compile Time

The Java compiler performs extensive and stringent compile-time checking so that the compiler detects as many errors as possible. The Java language is strongly typed. Unlike C or C++, the type system does not have loopholes. Several examples are described below.

- Objects can't be cast to a subclass without an explicit runtime check.

- All references to methods and variables are checked to make sure that the objects are of the appropriate type.
- Integers can't be converted into objects and objects cannot be converted into integers.
- The Java compiler checks that access controls like referencing a private method or variable from another class are not violated.
- It also guarantees that a program does not access the value of a local variable that is not initialized.

Class File Verification

Although a trustworthy compiler can ensure that Java source code does not violate safety rules, class files could come from a network that is untrustworthy [1]. A verifier that resides in Java virtual machine checks all class files loaded in before they are executed. The class file verifier goes through class files in several passes to make sure that they have the correct format. It also ensures that byte codes of class files adhere to a set of structural constraints. In particular, class files are checked to ensure the following.

- There are no violations of memory management.
- There are no stack underflows or overflows. Restrictions of access are enforced.
- Methods are called with appropriate arguments of the proper type.
- Fields are altered with the correct type's values.
- Objects are accessed as what they are. For example, an APDU object is always used not as anything else but as an APDU object.
- Binary compatibility rules are put in force.
- No pointers are fabricated.

- Illegal data conversions are not allowed.

Unlike, the Java virtual machine, the Java Card virtual machine has a split architecture that consists of two pieces: the converter running off card on a PC or a workstation and the interpreter running inside a card as shown in Figure 12. The converter is the front end of the virtual machine and takes class files and export files as input. During this conversion, the class files of an applet are subject to the same level of rigorous verification as they would be by the Java virtual machine with its class file verifier at class-loading time. The converter can integrate the verifier component of the Java virtual machine, or it can plant its own class file verifier.

Subset Checking

The converter moreover should check class files further to make sure that only features in that subset are used because Java Card technology defines a subset of the Java language. This step is called subset checking [1]. During this subset checking, the converter checks that the applet does not violate the following rules.

- Unsupported data types such as the variables of type char, long, double, and float must not be used. Recall that in chapter two, int data type can be used only if the Java Card interpreter supports them.
- Unsupported java language features are used.
- Certain Java operations are used within limited ranges. These operations on the Java Card platform are smaller than those of the Java platform.
- No potential overflow or underflow that might cause arithmetic results to be computed in a different way than they would be on the Java Platform can occur.

CAP File And Export File Verification

The classes of an applet make up one or more packages. The converter takes all classes in a package as inputs and converts them into a CAP file. The CAP file is then loaded onto a Java smart card. Then, the interpreter on the on-card virtual machine executes it. Besides creating a CAP file, the converter generates an export file representing the public APIs of the package being converted. This export file is not directly used by the interpreter but used later to convert another package that imports classes from the package represented in the export file. That is, the export file's information is used for linking and external reference checking. The CAP file in the Java Card platform is an interoperable binary format for loading a java package onto a Java smart card.

In practice, there is no guarantee that a CAP file generated from verified class files by a trustworthy converter will immediately be loaded onto a Java smart card in a secure environment [1]. Thus, the CAP file's correctness and integrity cannot be taken for granted. The CAP file verifier ensures that the CAP file plays by the rules. Due to the limited memory space and computing power of a smart card, the CAP file verifier runs off card as shown in Figure 12. The verifier performs static checks on a CAP file before it is loaded onto a Java smart card. It ensures that a CAP file has the correct format and that the byte codes in it adhere to a set of structural constraints. Because of the analogous role of class files and CAP files, the CAP file verifier has a function that is similar to the class file verifier's function. In particular, it checks the CAP file to ensure the following.

- There are no violations of memory management.
- There are no stack underflows or overflows. Restrictions of access are enforced.

- Methods are called with appropriate arguments of the proper type.
- Fields are altered with the correct type's values.
- Objects are accessed as what they are. For example, an APDU object is always used not as anything else but as an APDU object.
- Binary compatibility rules are put in force.
- No pointers are fabricated. Illegal data conversions are not allowed.

Note that these checks are same to those of the class file verifier. In addition, the CAP file verifier enforces rules that are special to the CAP file structure and the Java Card environment [1].

- The package and each applet defined in the package should have a valid AID (Application Identifier). The range of AID lengths should be between 5 and 16 bytes. The package AID and the applet AIDs must share the same RID (Register Identifier) number that is the first 5 bytes in the AIDs.
- An applet must define an install method with the correct signature in order that instances of the applet can be appropriately created on the card.
- The order of class and interface definitions in a CAP file must follow the rules that interfaces appear ahead of classes and super-classes appear ahead of subclasses so that the CAP file-loading and linking process can be handled in sequence on the card.
- If the int type is used in the CAP file, the int flag should be set. This check allows a Java Card implementation that does not support the int type to reject the CAP file during loading by simply checking the int flag.

During verification in Figure 13, the CAP file verifier ensures that a CAP file is internally consistent, is consistent with the export files it imports, and is consistent with the export file that represents its API [1]. The verifier also examines whether the Java Card version rule, including those forced for binary compatibility that is defined in the Java Card 2.1 Virtual Machine Specification, have been followed.

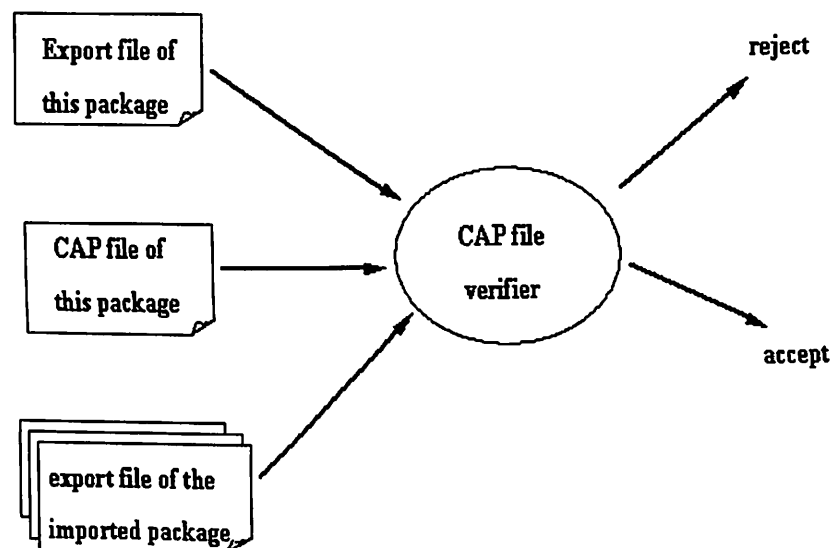


Figure 13. CAP file verification

Export file verification checks an export file both internally and against its corresponding CAP file to ensure that it is well formatted and satisfies the constraints required by the Java Card virtual machine specification [1].

Check By Off-card Installation Program And Installer

CAP file installation is accomplished through the collaboration of the off-card installation and the on-card installer. They load a CAP file. Then, if the CAP file defines any applets, they create one or more applet instances.

There are two levels of installation security. The first level is that the standard security protections enforced by the installer and the JCRE. The Second level is the security policies stated by the card issuers. These protect against the following [1]:

- Installation data corruption and tampering
- Inappropriateness between the CAP file and on-card resources
- Illegal access from outside the CAP file
- Insufficient resources and other errors during installation and initialization
- Inconsistent state due to card tearing or power loss during process of installation

The correctness and integrity of a CAP file are verified off-card. The Java Card installer does not perform most of the traditional Java verifications at class-loading time. Before any data is written on the card, the installer checks to see whether the card can support the CAP file, such as whether the card's available memory resources are sufficient for the CAP file or whether the int flag in the CAP file is set if the CAP file contains any int usage.

When the CAP file is read in, it can be linked either on the fly or after the entire CAP file is loaded [1]. The linking process includes resolving both internal and external references. The installer makes sure that internal references are in fact local to the package's memory and external references are linked to accessible locations of other packages and the JCRE. It also ensures that the CAP file is binary compatible with the existing software on the card.

Unlike the Java platform, the loading unit on the Java Card platform is a CAP file as a package. Since loading new classes incrementally is not supported, the installer ensures that the CAP file references only packages that are already on the card. If the CAP file defines any applets, the installer can create applets' instances by calling their install methods that are defined in Methods of the class `jvacard.framework.Applet`. The install method is typically called by the JCRE at the last step during applet installation to

create an applet instance. When an applet instance is created, a context is assigned to the applet instance. Multiple instances of the same applet and instances of multiple applets defined in the same package share one context. Recall from chapter 2 that the install method is combined with the register method. This register method stores a reference to the applet instance with the JCRE and assigns an AID to the applet instance.

The installation process is transactional. If an error, card tearing, or power loss occurs during installation, the installer discards the CAP file and any applets it had created during installation and recovers the space and the previous state of the JCRE [1].

Traditional Applet Communication And Risks

So far we have seen an applet installation process on the Java Card.

Communication between a host application and the Java Card applet is one-way communication, known as a 'half-duplex' as we have seen in section 2.2. The entire data exchanged between an applet and a host application takes place using Application Protocol Data Units (APDUs). The following shows traditional communication between a host application and the Java Card applet and current problems.

Applet Communication And Problems

This communication is processed by 4 steps: firstly applet installation, secondly the SELECT method to activate an applet, the PROCESS method for applet execution and finally the DESELECT method to deactivate the applet. After the applet is created on the Java Card, the applet enters an inactive state as we have seen in section 2.2. This is illustrated in Figure 14.

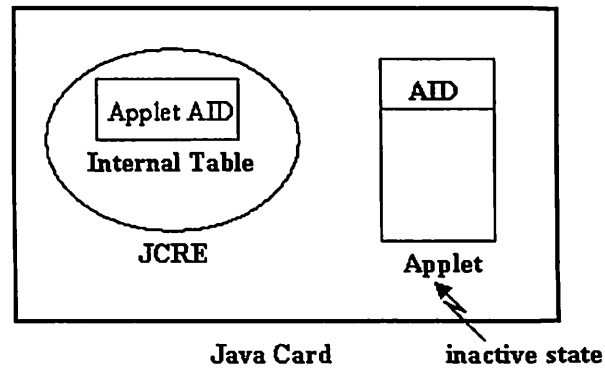


Figure 14. After an applet installation

This applet state enters an active state when a SELECT method with its AID is received by a host application. When the JCRE accepts the SELECT method with AID, it looks up its internal table for an applet whose AID matches the one specified in the command. If the JCRE can find it, this applet is verified. This is showed in Figure 15.

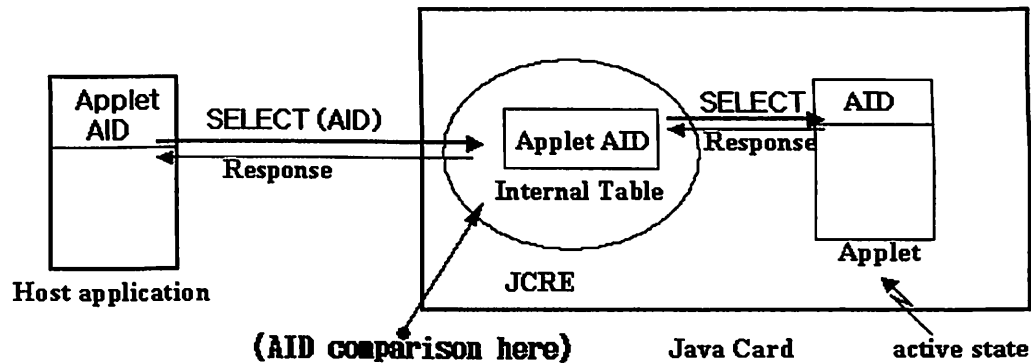


Figure 15. An applet selection from a host application

There is one major security problem is here. Malicious attackers can steal the AID of an applet when a host application sends AID with the SELECT method to the Java Card or the AID may be revealed by a host application manager to malevolent attackers. These attackers then make a forged application with the stolen AID and send the SELECT method with this AID to the Java Card. They can now access the applet on the card and access secure information on the card as well as perform illegal functions using

the applet. When the JCRE accepts this AID with the SELECT method, the JCRE cannot reject this request from the counterfeit application because the JCRE only compares the AID arriving with the SELECT method to the AID stored in the internal table in the JCRE.

A second problem arises when those malicious attackers can install a trojan or malicious applet on the Java Card with a revealed or stolen AID. After the spurious applet is installed, attackers can execute attack functions for their benefits by cheating a host application. This can happen because a host application only sends the AID to select the Java Card Applet. If the Java Card has a bogus applet with a revealed or stolen AID, the host application does not have a way to figure out that the called applet is a bogus applet. These two problems can be solved by a one-time password. This one-time password is explained in the next section.

After the applet is selected, the PROCESS method performs the communication between a host application and an applet. After using the Java Card, a host application sends the DESELECT method to the Java Card. After the JCRE accepts this method, it makes a currently selected applet inactive state. These steps are depicted in Figure 16.

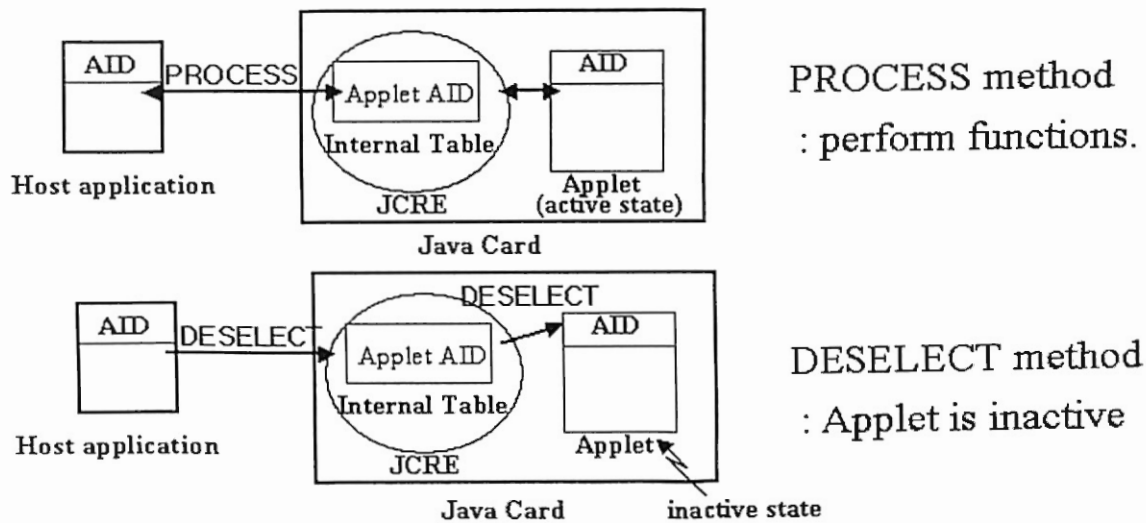


Figure 16. The PROCESS and DESELECT method

Secure Applet Communication By One-time Password

Recall from section 3.2 that current problems of communication between a host application and an applet are simply defined as verification of a host application and an applet. To verify a host application and an applet, this research adopts a one-time password that is implemented by a one-way hash chain to the Java Card. First, we show how the applet installation process on the card adopts a one-way hash chain. Secondly, we show how this one-way hash chain works on communication between a host application and an applet.

As we have seen in section 3.1, CAP file verification is not verified on the Java Card. This means that malicious attackers can install illegal applets by legal installation. Jang [17] verifies CAP files by using a one-way hash function on the on-card installer. In Jang's work when a client applet is installed, the JCRE stores a hash value from its own CAP file via a one-way hash function. The JCRE generates this hash value using the on-card hash value function unit. A host applet has this hash value before it is installed from a client applet provider. Whenever a client and server applet communicates with each

other, the JCRE compares the hash value on the JCRE and the hash value from a server applet. As the hash value is computed on-card, it is never transmitted to the card. An attacker cannot therefore obtain the hash value by intercepting communications between the card and the host application. However, his work focused on communication not between a host application and an applet on the card but between two applets on the card. A fundamental assumption in this work is that the JCRE is fully protected and attackers cannot gain access to the code or data stored in the JCRE. We make the same assumption in our work. We adapt this one-way hash function installer here. However, even though we install applets through this installer still there is a hole for attacks. Because this hash value is static, there is still a possibility that this hash value will be revealed. In this research, we adapt this hash value to verify the CAP file. In other words, this thesis focuses on a changeable hash value to protect a host application and every applet on the Java Card. The one time password that is introduced below is used to verify it. A 'one time password' is used once only and then discarded and becomes invalid.

This one-way hash function computes a hash value (H.V.) from a CAP file. By using this, the problem of the CAP file verification on the card is solved. This one-way hash function is also used for one-way hash chain to generate a one-time password.

When an applet is installed on the card, the JCRE stores its AID, the hash value (H.V.) computed from its CAP file, and the password. This password is the last one generated from the hash value in the one-way hash chain. As we saw in figure 11, the hash value is to be S_n and the password is to be S_0 . Figure 17 illustrates this.

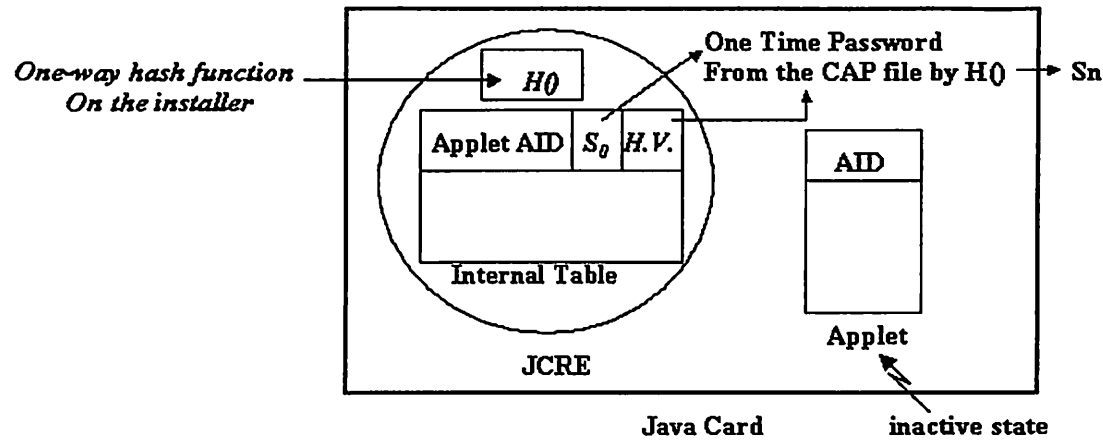


Figure 17. Initial state after applet installation

As we saw in chapter 2, S_0 is computed from a seed by a one-way hash function using n times repeatedly. The card provider decides this n value – that is how many times this one-way hash function will be performed. The seed S_n is generated from the applet CAP file’s entire contents. The hash value and S_0 are therefore not transmitted by the host to the card as they are generated on-card. This increases the security of the system. Figure 18 shows that the first communication between a host application and the Java Card applet by using the SELECT method as we have seen in section 3.2. In the host application side, the host application manager uses the S_{n-1} as an original seed at the host application. We assume that the host manager does not know the value of S_n . If the host manager knows S_n and he reveals S_n , the security is broken. As S_n is equal to the hash value HV (fig. 17), not only can the intruder compute S_{n-1} to S_0 , but he can also submit a correct HV value. On the other hand, if the host manager does reveal S_{n-1} , the system is still secure as the attacker still does not know the value HV. At the initial state, computations range from S_{n-1} to S_1 , resulting in $n-2$ hash computations. This serves as an index which reduces by one each time a communication with the card takes place $n-2$ value to make the first one-time password (S_1). The host application manager first obtains

the one-way hash function and the S_{n-1} value from the card provider (the host manager is never given S_n). A host application sends the SELECT method with AID and the S_1 to the Java Card. When the JCRE on the card receives this, the JCRE finds the matched AID on the internal table. If it is found it computes S_0 from S_1 , which is sent from the host application, and then it compares this computed S_0 with the S_0 stored on the table in the JCRE. If they match, the JCRE permits the host application's access to the applet. If they don't match, it rejects the host application's access and a fail message is sent as a response to the host application. This is illustrated in Figure 18.

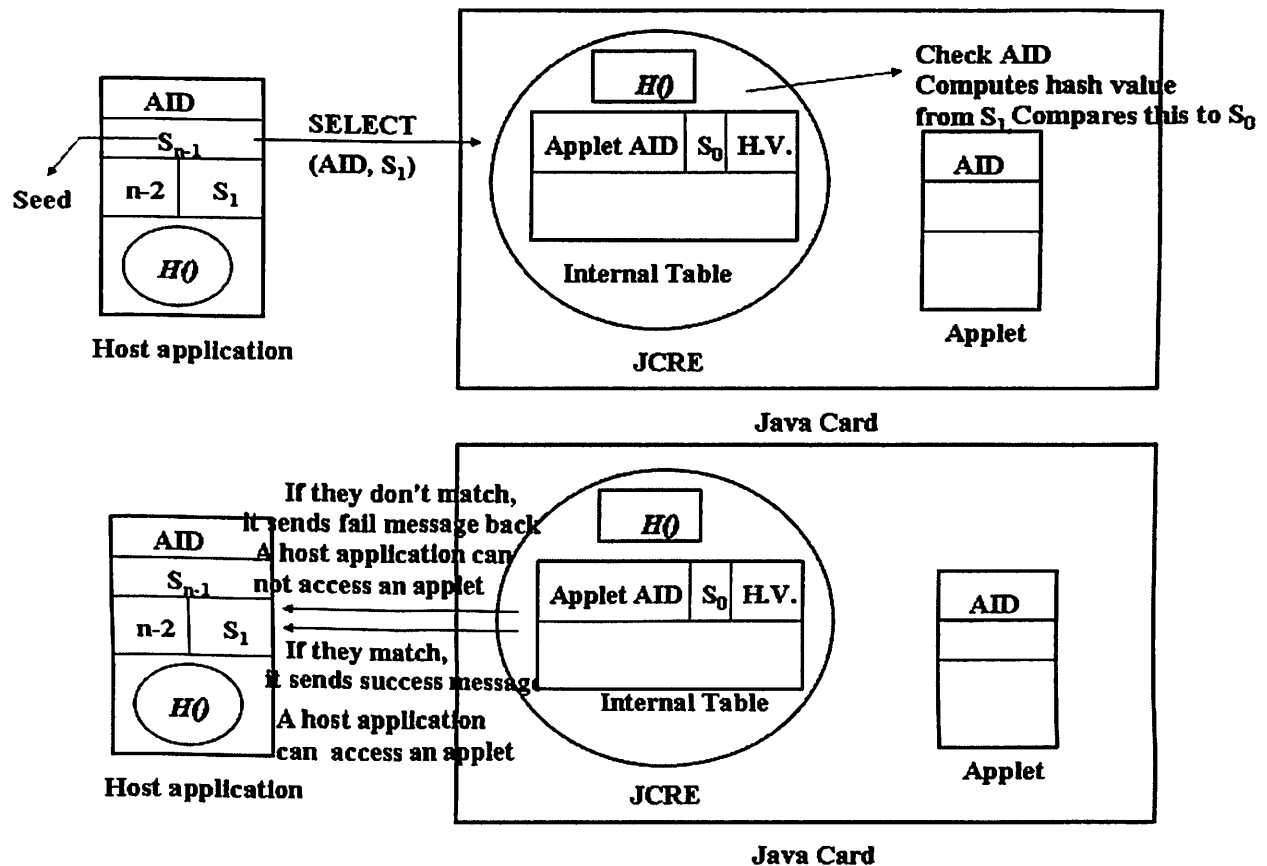


Figure 18. The proposed applet selection method

The communication between a host application and the Java Card applet is performed by the PROCESS method as described in section 3.2.

One-time passwords have a problem of synchronization when updating a password. In other words, the password submitted by the host application should be the one that is expected by the Java Card JCRE. For example, if the last password submitted by the host application is S_1 , the JCRE will be expecting S_2 next time. A host application may instead submit S_4 (instead of S_2). If there is no synchronization, the S_4 will be accepted by the JCRE as S_0 can be derived from S_4 . Clearly, if S_4 is submitted instead of S_2 , the JCRE should at least raise a alarm, even if it does not reject it outright. In the proposed scheme, this update is made easy. Before starting the next communication by the PROCESS method, the JCRE updates its table by changing the password (S_0) to the password (S_1) that the JCRE received from the host application in the previous communication (and the previous the SELECT method was a success). At the host end, on receiving a success message of an applet selection from an applet on the card, the host application updates its index from $n-2$ to $n-3$ and its one-time password from S_1 to S_2 by performing the hash function from the seed S_{n-1} with index $n-3$. This process is illustrated in Figure 19.

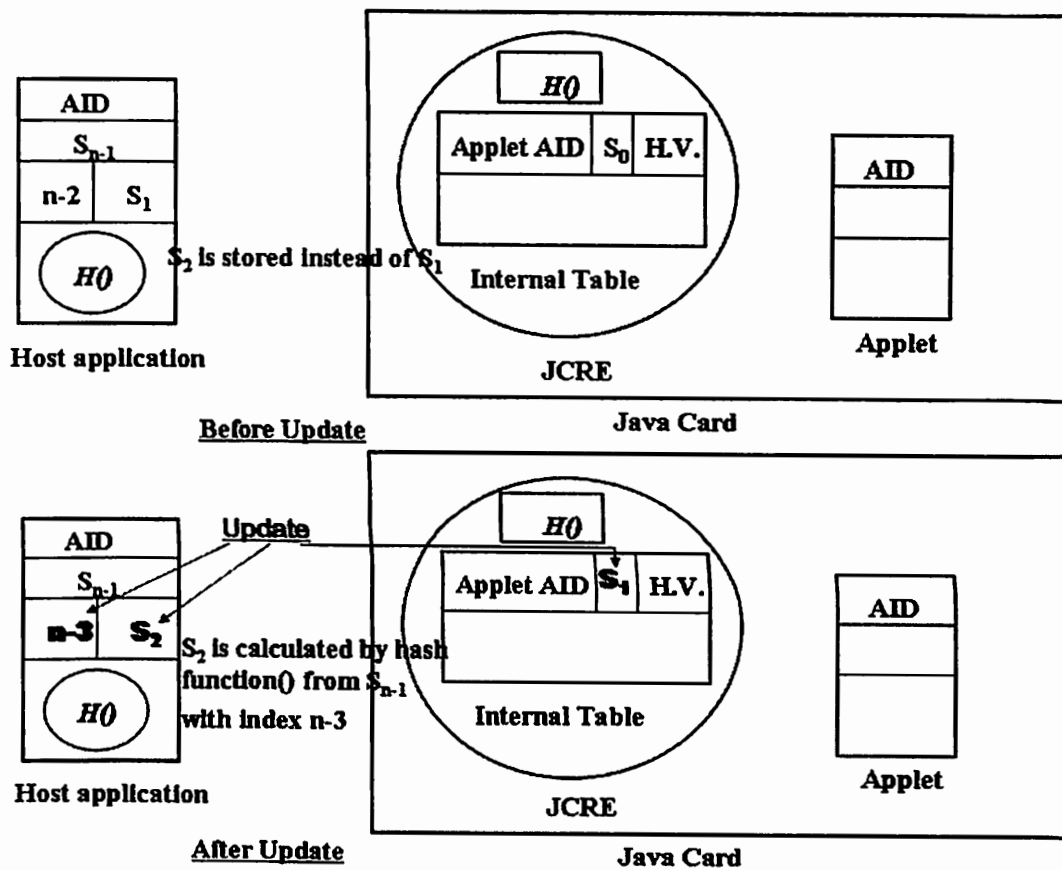


Figure 19. One-time password update

To generalize, assume in the most recent successful PROCESS (or SELECT) method the one-time password S_i was submitted by the host application. The JCRE updates its table to S_i . The host application updates its index entry to $n-2-i$ and its one-time password (the next one to be sent to the JCRE) to S_{i+1} . The last password is generated when $n-2-i$ equals zero. The index therefore serves to synchronize the password generation and defines the terminating condition when no more passwords will be generated. The next time (or the $i+1$ th time) the host application communicates with the applet it sends S_{i+1} (or S_2 if it is the second communication). Using this selection process, we can solve the security drawbacks of the Java Card. In case the AID is revealed or stolen by a malicious attacker, the attacker can make a forged application

with the stolen AID, send the SELECT method with this AID to the Java Card to access secure information from the card or perform malicious functions. However, our proposed one-time password can prevent this attack by computing S_0 from the transmitted S_1 value, and comparing with the stored S_0 value in the JCRE as shown above in Figure 18.

Therefore by comparing a one-time password, a forged applet problem with stolen AID can be solved. The malicious attacker, even if he has stolen S_0 cannot break the security of the system because he cannot generate the password S_1 (or S_i). If the attacker has stolen S_1 (or S_i) he cannot access the applet because S_1 (or S_i) is no longer valid and he cannot generate S_2 from S_1 (or S_{i+1} from S_i). Therefore an attacker obtaining AID or S_i illegally will not be able to break system security. The only remaining option for an attacker is to generate the hash value (HV) , that is, generate S_n . The S_n is generated from the CAP file of an applet. To generate S_n , an illegal applet source file should be the exactly same as an original applet source file. In other words, it is almost impossible for an attacker to generate S_n , unless the attacker is able to attack the host or the JCRE itself and obtain S_n . Therefore even if an attacker obtains AID or S_i illegally, it is virtually impossible for an attacker to generate S_n . Figure 20 illustrates i th time selection and update between a host application and an applet.

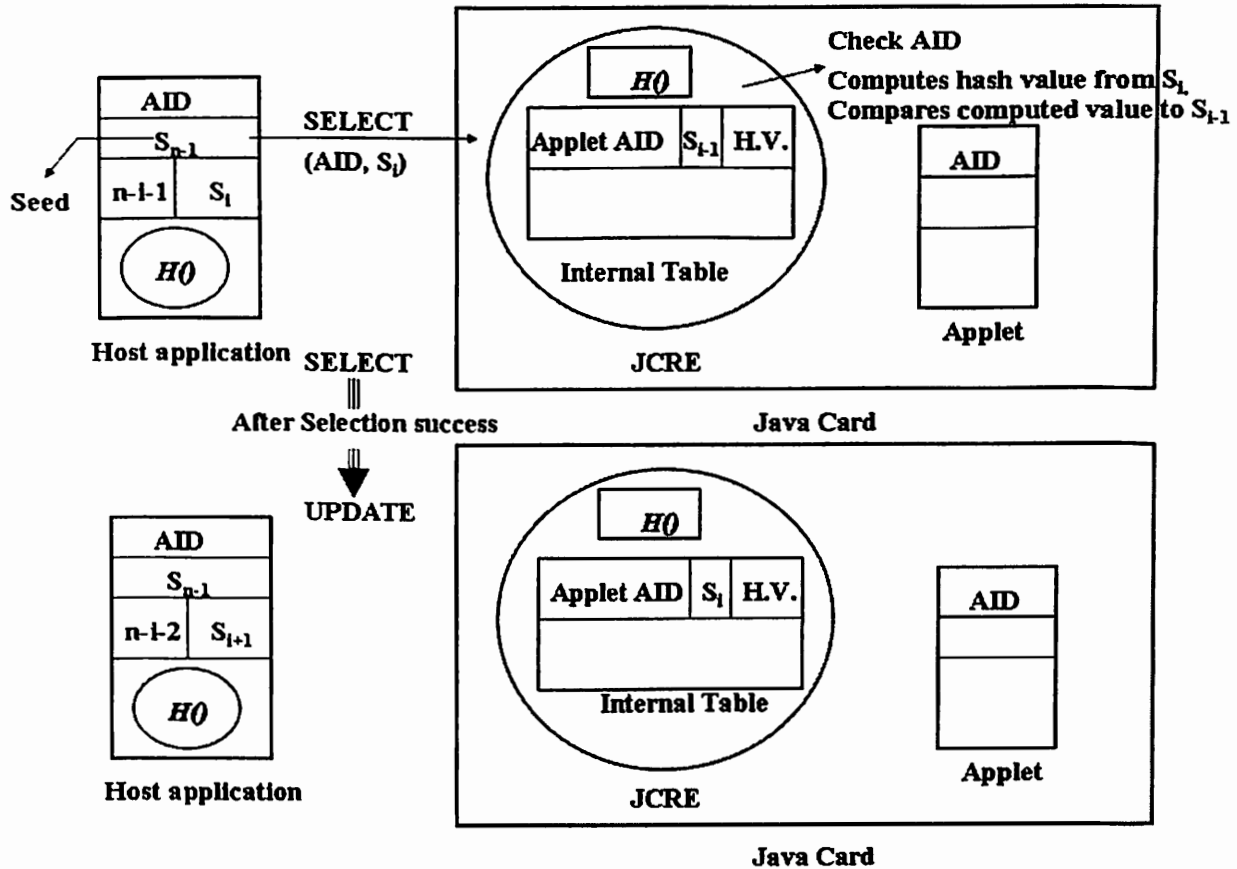


Figure 20. i -th time selection and update

Integrated System

We integrate this proposed method to the work reported in [17] – ‘Secure object sharing on Java Card’ – where secure communications between two on-card applets - a client and server applet is described. This server and client applet can communicate each other via Sharable Interface Object (SIO). In the integrated system, a host application, a client and a server applet after installation are illustrated in Figure 21. As we have seen above, every applet and host application communication is secured by our proposed method. Two applets (as a server and a client) on the card communicating via the SIO are secured by the method in [17].

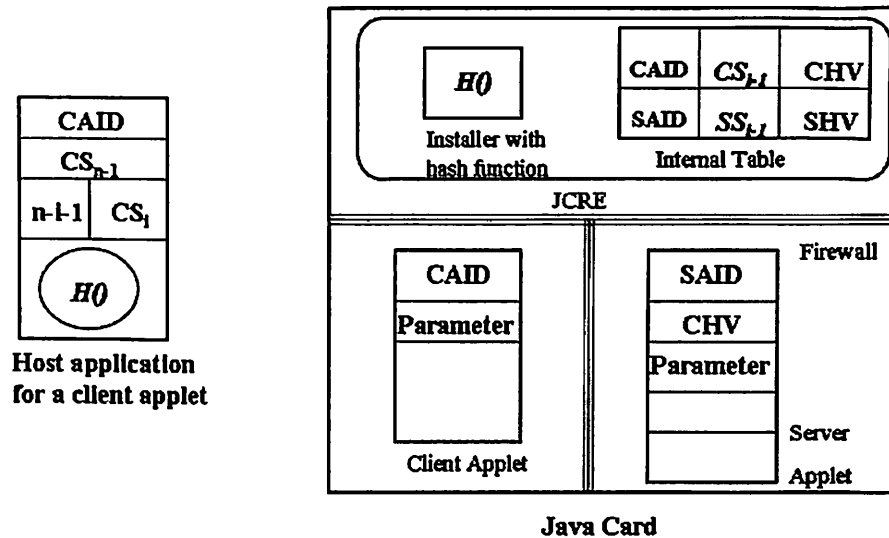


Figure 21. Integrated System

Figure 21 shows the integrated secure Java card architecture. CAID stands for the client applet AID and SAID for the server applet AID. CS_{n-1} is the seed provided at the host application by the card provider as described above. CS_i is the one-time password on the host application. CS_{i-1} specifies the client applet one-time password and SS_{i-1} indicates the server applet one-time password. Therefore n-i-1 is the index at the host application. CHV is the hash value of the client applet from the client applet CAP file. SHV is the hash value of the server applet from the server applet CAP file. Parameter is a security token, which carries a secret share by the server and the client. This parameter is decided by the applet provider. The parameter is used for secure communication between a client and a server applet on real card. However, even if the parameter is used – The JCRE compares a server applet’s parameter and client applet’s parameter, there can be a problem with security. The CHV is additionally used for communication between the client and server applet on the Java Card for more security. When the client applet communicates with the server applet, the JCRE compares not only the client applet parameter with the server applet parameter but also the CHV on the JCRE and SHV on

the server applet. The CHV and SHV on the JCRE are generated by the JCRE at time the applets are installed on the card. This communication is described detailed in [17]. The host application for the client applet and that for the server applet can be the same or they may be different. If they are different, there is another host application with a server applet password and a hash function for the server applet. If they are the same, one host application has a client applet password, a server applet password and a hash function.

Proposed one-time password merits

The proposed One-time password implemented by infinite one-way hash chains offers the following benefits:

- The proposed one-time password prevents an unauthorized host application from assuming a valid host application.
- It prevents an unauthorized applet from accessing an authorized host application.
- There can be other methods to verify a host application and the Java Card applet such as password exchange or key algorithms of encryption and decryption between a host application and the Java Card. However, these methods still have threats such as network eavesdropping and password cracking when a host application sends passwords or encrypted messages to the Java Card. The proposed method protects the Java Card from these kinds of attacks since the one-time password is not reusable.
- In this proposed one-time password it is easy to synchronize on a host application and the Java Card. Therefore, this feature makes simple management of the one-time password.

- On the memory resource side of the Java Card, only a one-way hash function is implemented on the card. This makes the Java Card memory resource usage minimal.

CHAPTER IV
SECURE COMMUNICATION SIMULATION
BETWEEN HOST APPLICATION AND THE JAVA CARD

To simulate our secure communication between a host application and the Java Card, we developed an installer with a hash function, an applet as a client, and a host application as a server. We also installed some tools for the simulation. In this chapter, we present our simulation environment in section 4.1. Section 4.2 shows the simulation proceeds for to a real card without our proposed security enhancements. Section 4.3 shows the simulation for our proposed method.

Prerequisites

For the simulation, we installed the Java Development Kit (JDK) version 1.3. JDK is a development environment to build applications and applets on the Java platform. We installed the Open Card Framework (OCF) version 1.2. It includes exportable source and executable code of the OpenCard Framework owned by the OpenCard Framework association. We also installed tools in the Java Card 2.2 development kit to install an applet. This kit is a collection of tools for implementing and developing applets based on the Java Card 2.2 framework. This facilitates java card implementations and development of applets based on the Java Card terminal 2.2 API (Application Programming Interface). After installing these, we copied the files base-core.jar and base-opt.jar from the OCF into the Java Card 2.2.

Difference between the Java card and the simulation

There are differences between the Java Card and simulation. The reason is that our proposed method actually requires modifications to the JCRE. Our proposed installer and comparison function should be implanted in the JCRE. However, we cannot modify the JCRE in a real card and our simulation is therefore not a direct reflection of a real Java Card.

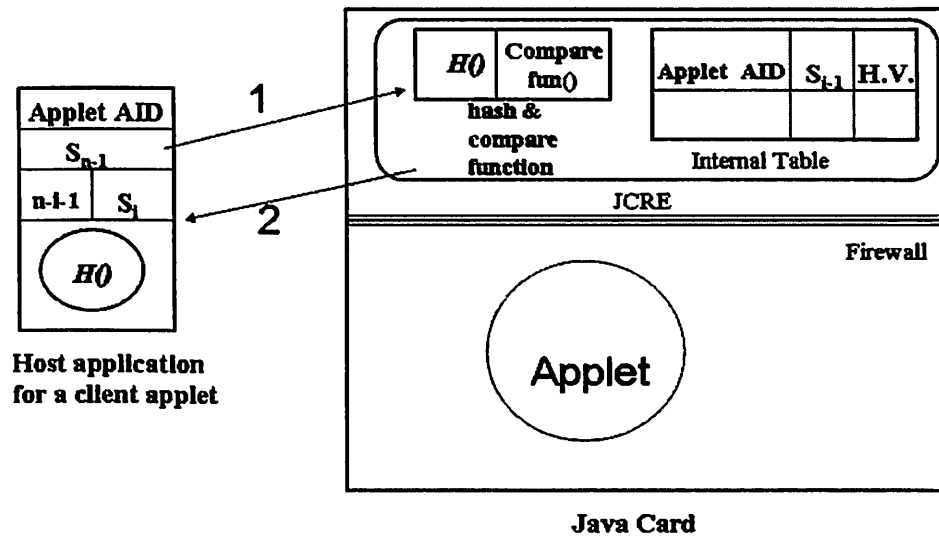


Figure 22. The real Java Card

Figure 22 shows the selection process of a real java card that implements our proposed one-time password. When a host application selects applet (1), the JCRE checks the applet AID and password. If they match, the JCRE sends ok message to the host application (2). Otherwise, the JCRE sends a fail message to the host application (2).

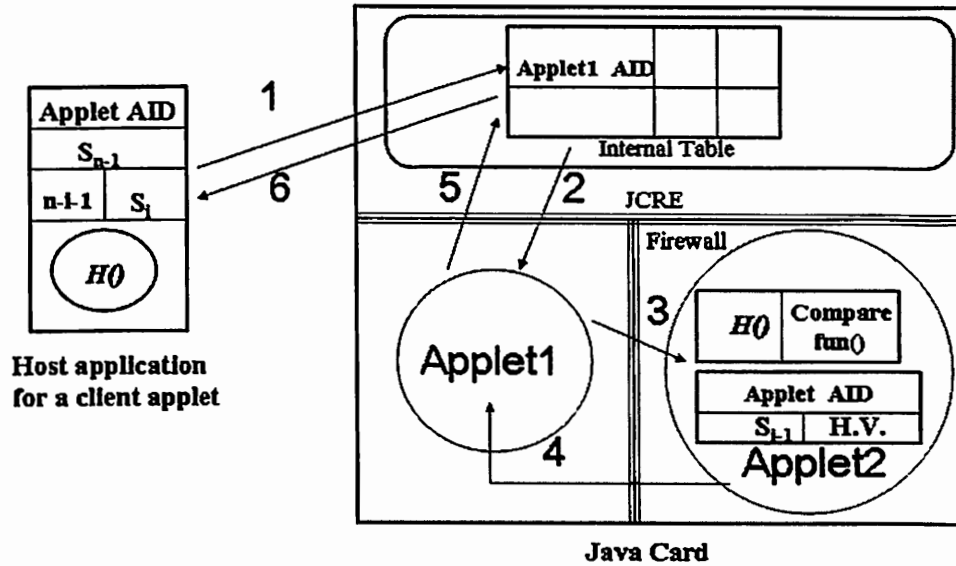


Figure 23. Simulation Model

As we see in Figure 23, simulation is different. First, a host application sends the select method to the JCRE (1). After checking an applet1's AID, JCRE sends the select method to applet1 (2). Applet1 sends the select method to an applet2 (3). Applet2 performs the hash function and compares passwords. If they match, it sends ok message to the applet 1 (4). Otherwise, it sends a fail message to applet1 (4). Applet1 forwards the result message to the JCRE (5). The JCRE sends the result message to the host application (6).

Simulation

An applet was implemented and installed on the C-language Java Card Runtime Environment (C-JCRE). We call this applet a hash installer. As this is a simulation environment instead of a real Java Card environment, we developed applets for installation. The hash installer has a hash function that generates a hash value when an applet is installed from the CAP file. So, the hash installer and the Java Card installer that exists in the C-JCRE perform as the traditional on-card installer.

To make a card, we use a command: `cref -o card` on the command prompt. The switch `o` means that the card will be saved. After making and saving the card, we can call and use it anytime by a command: `cref -i card -o card`. The switch `i` means we are calling an existing card. After making a card, we can use these commands to use it. Figure 23 shows that a hash installer is installed to the Java Card.

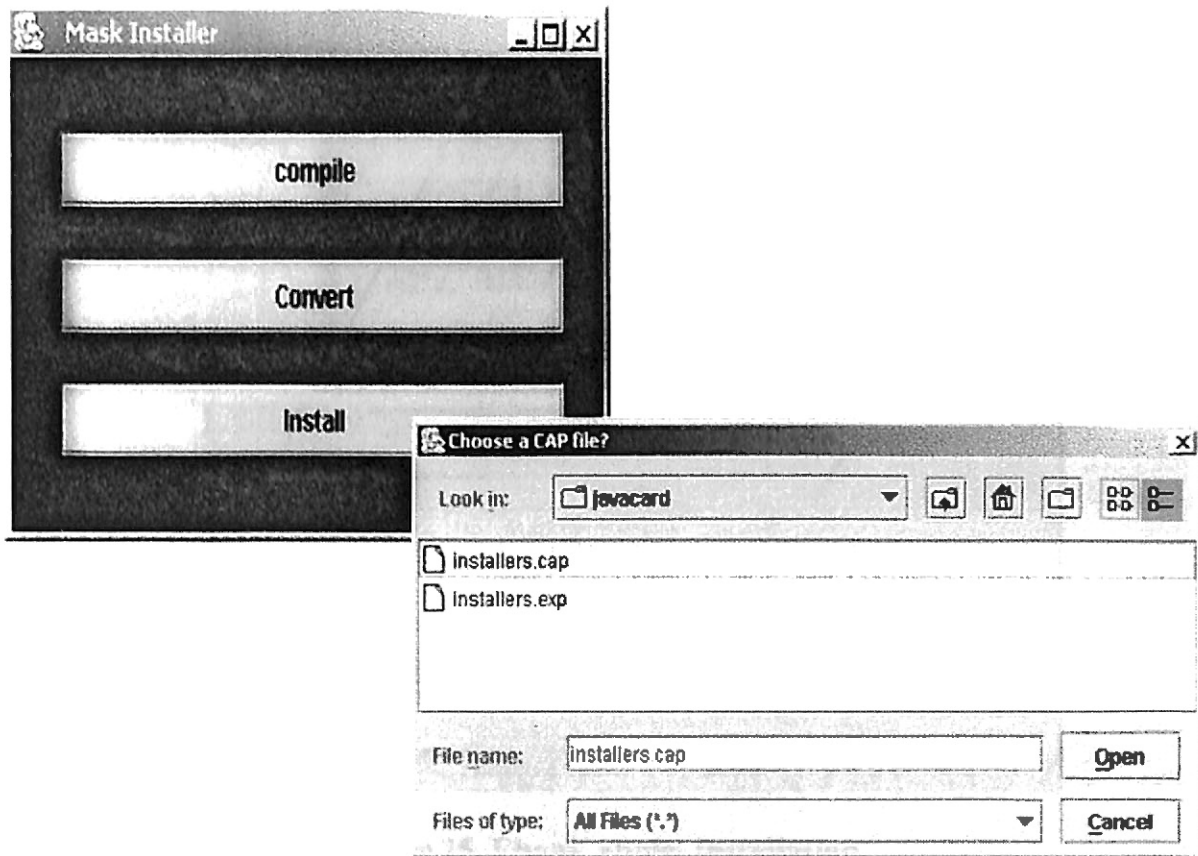


Figure 24. Masking hash installer

As we have seen in section 2.2, the off-card installation program and the installer on the C-JCRE perform an applet installation. In the simulation environment, the hash installer is also used. When the off-card installation program sends the CAP file to the installer, the installer sends this CAP file with an AID to the hash installer. Then, the hash installer generates a hash value from the CAP file. This hash installer installs the

password S_0 generated from the hash value (S_n). This password is therefore provided by the card provider. The card provider decides how many times the hash function can be used once the applet is installed to the card. This makes us achieve one of our objectives – CAP file verification on the on-card part.

Figure 25 illustrates that the Ebank applet is installed to the Java Card at terminal. The Ebank CAP file is installed to the card.

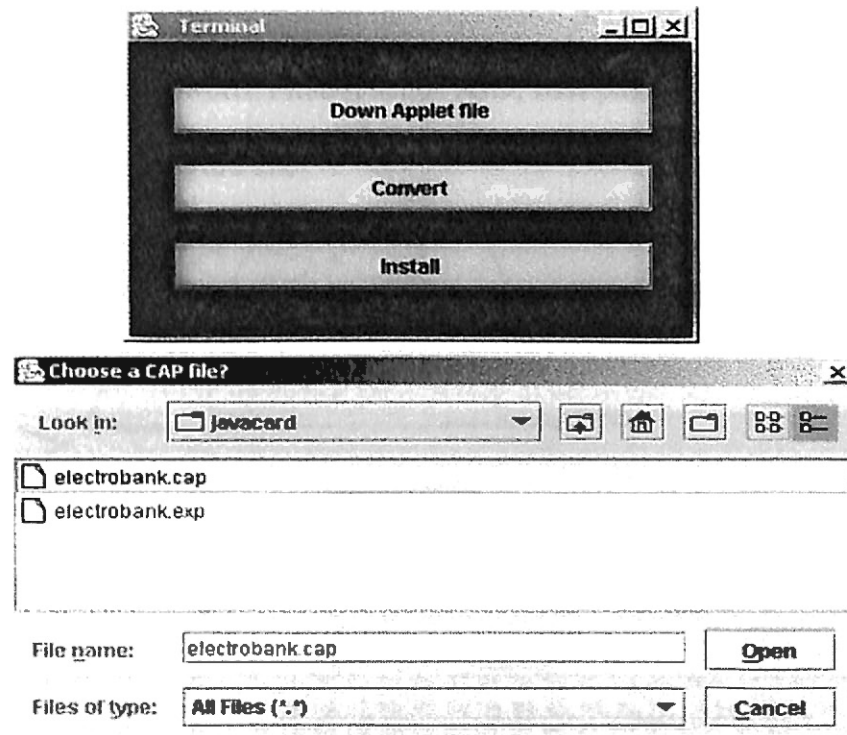


Figure 25. Ebank applet installation

After this Ebank applet is installed, the hash installer has this Ebank AID (0x00 0x00 0x00 0x0C 0x03 0x01 0x0C 0x06 0x01), Hash value (0x6e 0xf3 0x2a 0xa2 0x8a 0xe6 0x84 0xeb 0x30 0x68 0x0f 0xf6 0x7c 0xc7 0x1a 0xce), and password (0x6d 0x9f 0x9a 0x8e 0xd3 0x08 0xfd 0x2a 0x65 0xa6 0x1f 0x48 0x24 0x8a 0xf9 0xd6). This password is made from the hash value by using the hash function n times. And a host application has Ebank AID, $n-2$ value, and a hash value (0x21 0xcd 0xf4 0x44 0xdd 0x5a

0xcd 0x05 0x2a 0x00 0xd0 0x6b 0x6b 0xcd 0x54 0x57). After obtaining these values, the host application makes the first password from the n-2 index and the hash value. The password is 0x34 0x68 0x23 0xbb 0xa8 0x9e 0xbb 0x8d 0xe9 0x56 0xdc 0x13 0xc3 0x5e 0x35 0x8c.

The host application sends AID and the first password to the card (0x34 0x68 0x23 0xbb 0xa8 0x9e 0xbb 0x8d 0xe9 0x56 0xdc 0x13 0xc3 0x5e 0x35 0x8c). As we saw in section 3.3, the hash installer takes this password, performs the hash function, and compares with its own password. First, it sends AID, first password and deposits 0x10 (\$16). We see the result in figure 26.

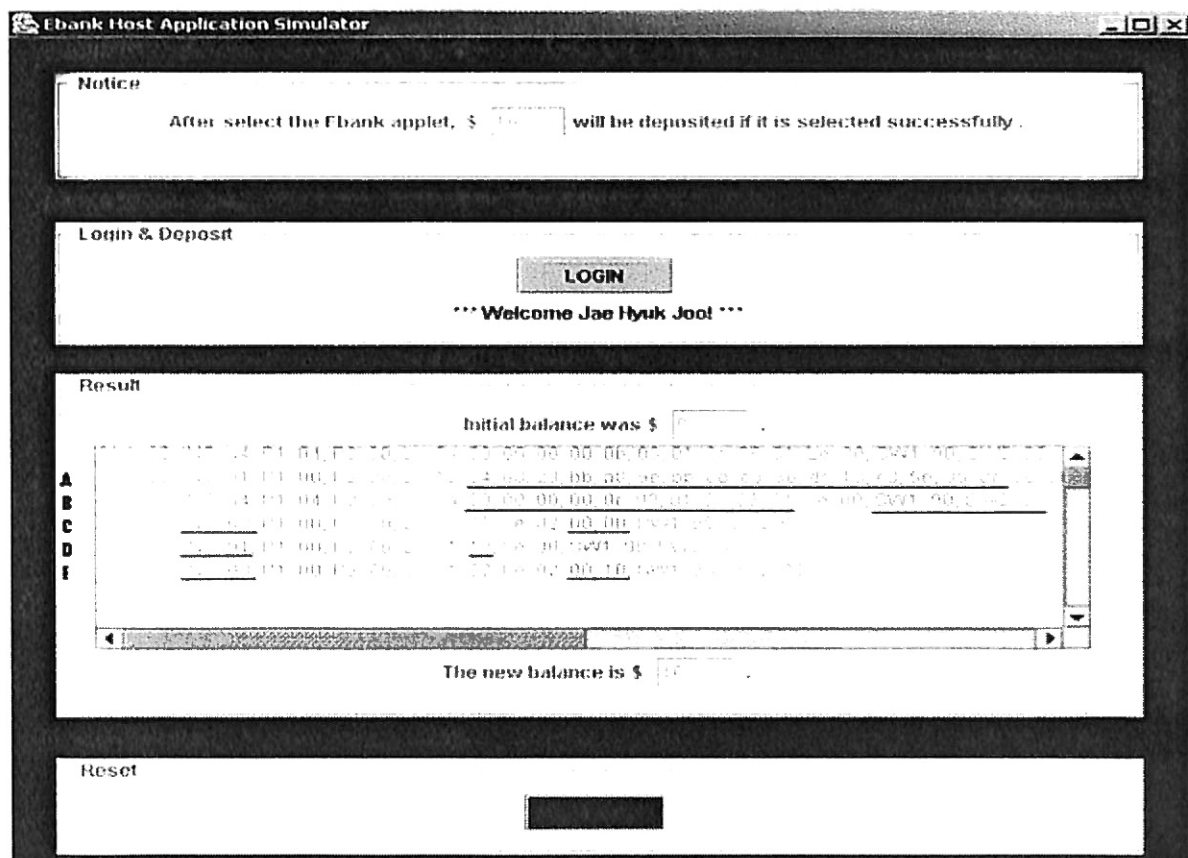


Figure 26. Select and deposit with the first password

As we see from the result window in figure 26, line A line shows that the first password (34 68 23 bb a8 9e bb 8d e9 56 dc 13 c3 5e 35 8c) is sent. Line B shows selection of the Ebank applet with AID and the result is indicated by SW1:90, SW2:00. The switch 9000 means that the selection of the Ebank applet is a success. If this selection failed, the switch would be 6999. The C line's INS:03 is the balance method and the balance is 0 as shown by the underlined 00, 00. The D line's INS:01 is the deposit method with \$16 (underlined 0x10). E line shows \$16 is deposited. After this selection and deposit, the password of the host application is updated to 0x2c 0x0e 0x91 0x24 0xe8 0xeb 0x9b 0x99 0x1a 0x12 0x11 0xc2 0x42 0x8b 0x3d 0xe3 as the second password. The password of the hash installer is updated to 0x34 0x68 0x23 0xbb 0xa8 0x9e 0xbb 0x8d 0xe9 0x56 0xdc 0x13 0xc3 0x5e 0x35 0x8c.

As the host application sends the second password to the Ebank applet, we can verify whether our proposed method works or not. At this time, \$32 will be deposited. After depositing this, the balance should be \$48.

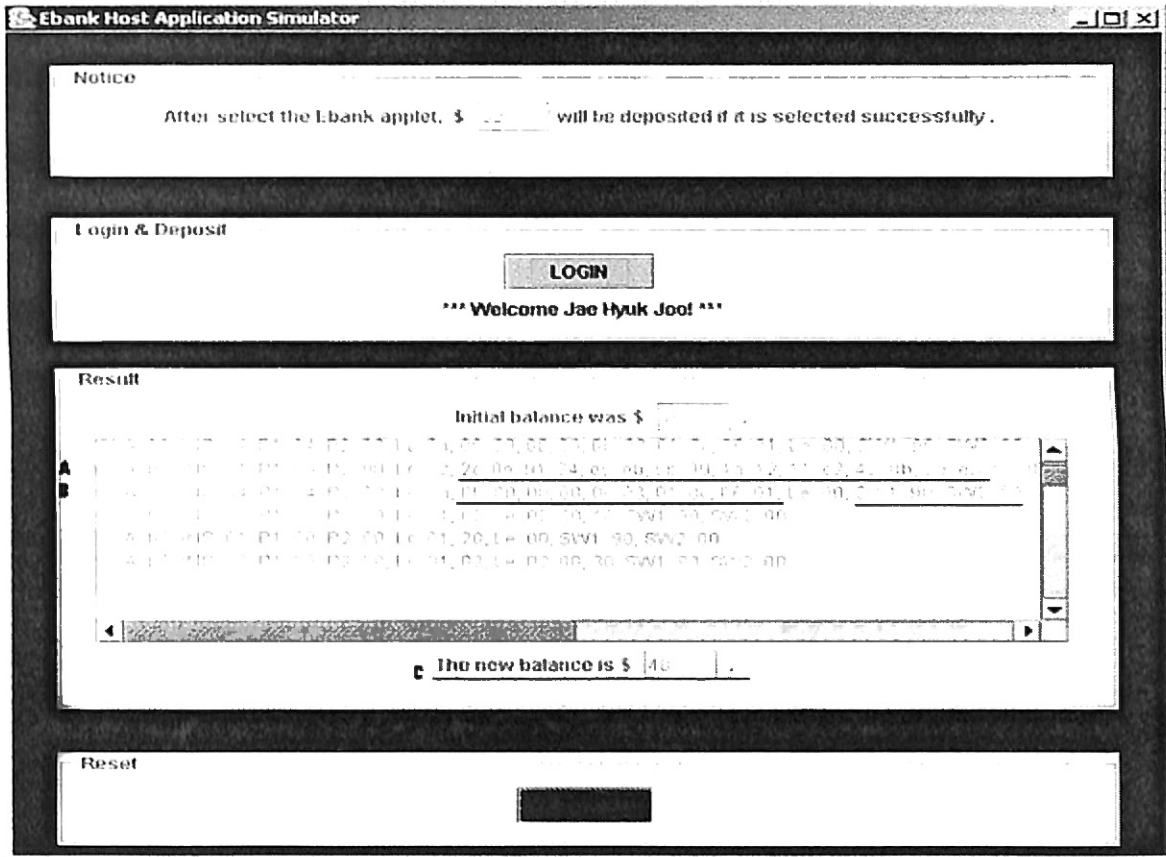


Figure 27. Reselect and deposit with the second password

Line A shows the second password. Line B tells us the selection is successful (9000). Line C shows the resulting balance to be \$48. This shows that our proposed one-time password works – that is, the password is updated each time after the applet is selected with the password. The next password of host application will be 0x56 0xb3 0x97 0x0d 0x13 0x2d 0x10 0xec 0x9c 0x3e 0x06 0x2f 0x2a 0x9b 0x91 0x3a. The next password of Ebank applet will be 0x2c 0x0e 0x91 0x24 0xe8 0xeb 0x9b 0x99 0x1a 0x12 0x11 0xc2 0x42 0x8b 0x3d 0xe3.

It is almost impossible for malicious attackers to obtain or guess the password because they have to know the hash function (), the n-2 index or the current value, and the original password. This original password comes from the Ebank CAP file. So, any

malicious input should exactly match the Ebank applet source code to make this password. This is practically impossible. The only way that attackers can obtain a password is to capture and replay the password at communication time between a host application and an applet. In figure 26 above, the host application sent the password (0x2c 0x0e 0x91 0x24 0xe8 0xeb 0x9b 0x99 0x1a 0x12 0x11 0xc2 0x42 0x8b 0x3d 0xe3). We assume that an attacker illegally catches that password and submits it to the Java Card. The result is shown in figure 28.

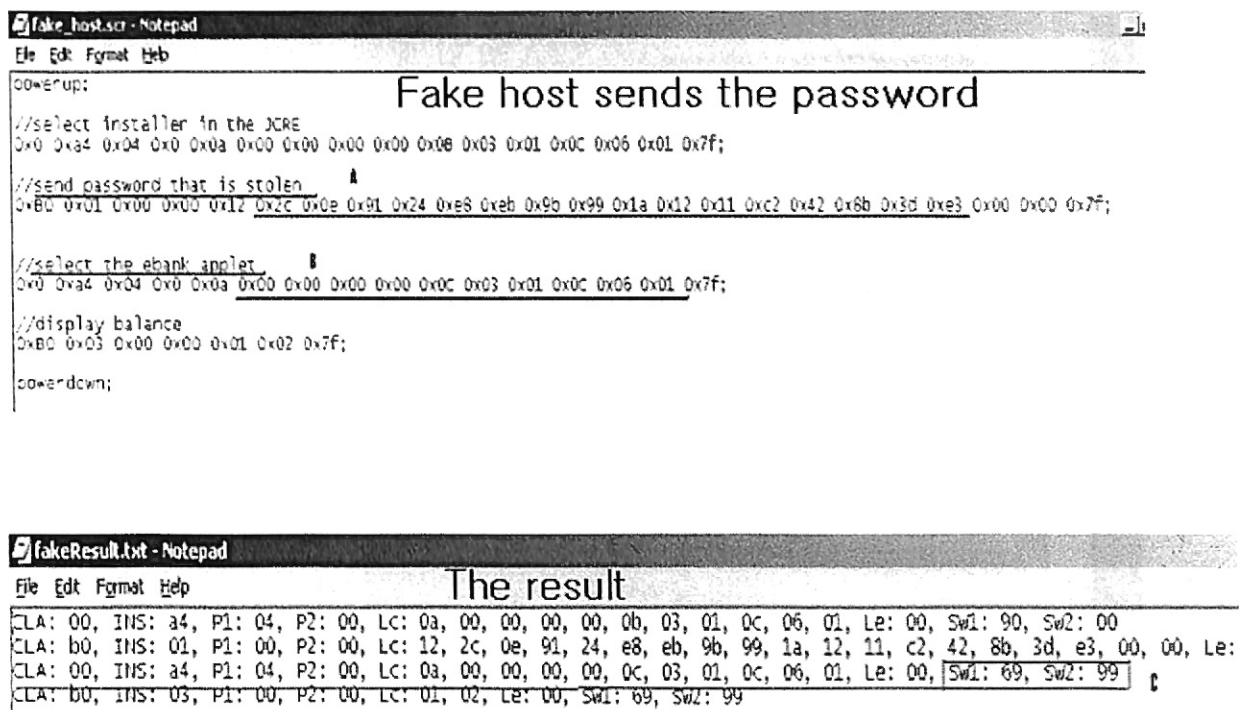


Figure 28. A fake host application

In figure 28, the fake host application sent a real password; this is captured at line A. The fake host application tried to select the Ebank applet at line B. The result is shown at line C. The switch 6999 means the selection has failed. Our proposed method can therefore detect a fake host application which is attempting to get the information illegally from the card. This is one of our objectives. Even though the real password has

applet. This added code does not affect the function in any way. This is shown in figures 30 and 31. After installing this fake applet, the real host application sends a real password to this applet. We observe whether this fake applet is able to communicate and get information from the real host application. This is illustrated in figure 30.

```
Ebank.java - Notepad
File Edit Format Help
package electrobank;

import javacard.framework.*;
import installers.InstallInterface;

public class Ebank extends Applet {

    //codes of CLA byte in the command APDUS
    private final static byte Ebank_CLA = (byte)0x80;

    //codes of INS byte in the command APDUS
    private final static byte DEPOSIT = (byte)0x01;
    private final static byte DEBIT = (byte)0x02;
    private final static byte BALANCE = (byte)0x03;

    //Applet-specific status words;
    private final static short SW_NEGATIVE_BALANCE = 0x6A01;
    private final static short SW_INVALID_DEBIT_AMOUNT=0x6A02;
    private final static short SW_NO_SERVER = 0x6A03;
    private final static short SW_SIO_GETTING_FAIL = 0x6A04;

    private short balance;

    //AID fo this applet instance
    private final byte[] own_aid = {0x00, 0x00, 0x00, 0x00, 0x0C, 0x03, 0x01, 0x0C, 0x06, 0x01};
    private final byte[] installer_aid = {0x00, 0x00, 0x00, 0x00, 0x0B, 0x03, 0x01, 0x0C, 0x06, 0x01};

    //parameters for sharing between Ebank and Installer
    private final byte sharepara = (byte) 0xcc;

    //start Ebank constructor
    private Ebank() {

        //initial check
        balance = (short) 0x00;

        //register this applet instance to the JCRE
        register(own_aid, (short)0, (byte)(own_aid.length));

    }
}
//end Ebank constructor
```

Figure 30. Original Ebank source code

```
Ebank.java - Notepad
File Edit Format Help
package electrobank;
import javacard.framework.*;
import installers.installinterface;
public class Ebank extends Applet {
//codes of CLA byte in the command APDUS
private final static byte Ebank_CLA = (byte)0xB0;
byte i;
//codes of INS byte in the command APDUS
private final static byte DEPOSIT = (byte)0x01;
private final static byte DEBIT = (byte)0x02;
private final static byte BALANCE = (byte)0x03;
//Applet-specific status words:
private final static short SW_NEGATIVE_BALANCE = 0x6401;
private final static short SW_INVALID_DEBIT_AMOUNT=0x6402;
private final static short SW_NO_SERVER = 0x6403;
private final static short SW_SIO_GETTING_FAIL = 0x6404;
private short balance;
//AID fo this applet instance
private final byte[] own_aid = {0x00, 0x00, 0x00, 0x00, 0x0C, 0x03, 0x01, 0x0C, 0x06, 0x01};
private final byte[] installer_aid = {0x00, 0x00, 0x00, 0x00, 0x0B, 0x03, 0x01, 0x0C, 0x06, 0x01};
//parameters for sharing between Ebank and Installer
private final byte sharepara = (byte) 0xCC;
//start Ebank constructor
private Ebank() {
//initial check
balance = (short) 0x00;
//register this applet instance to the JCRE
register(own_aid, (short)0, (byte)(own_aid.length));
}
}

```

Everything is same except this is added

Figure 31. Fake Ebank code

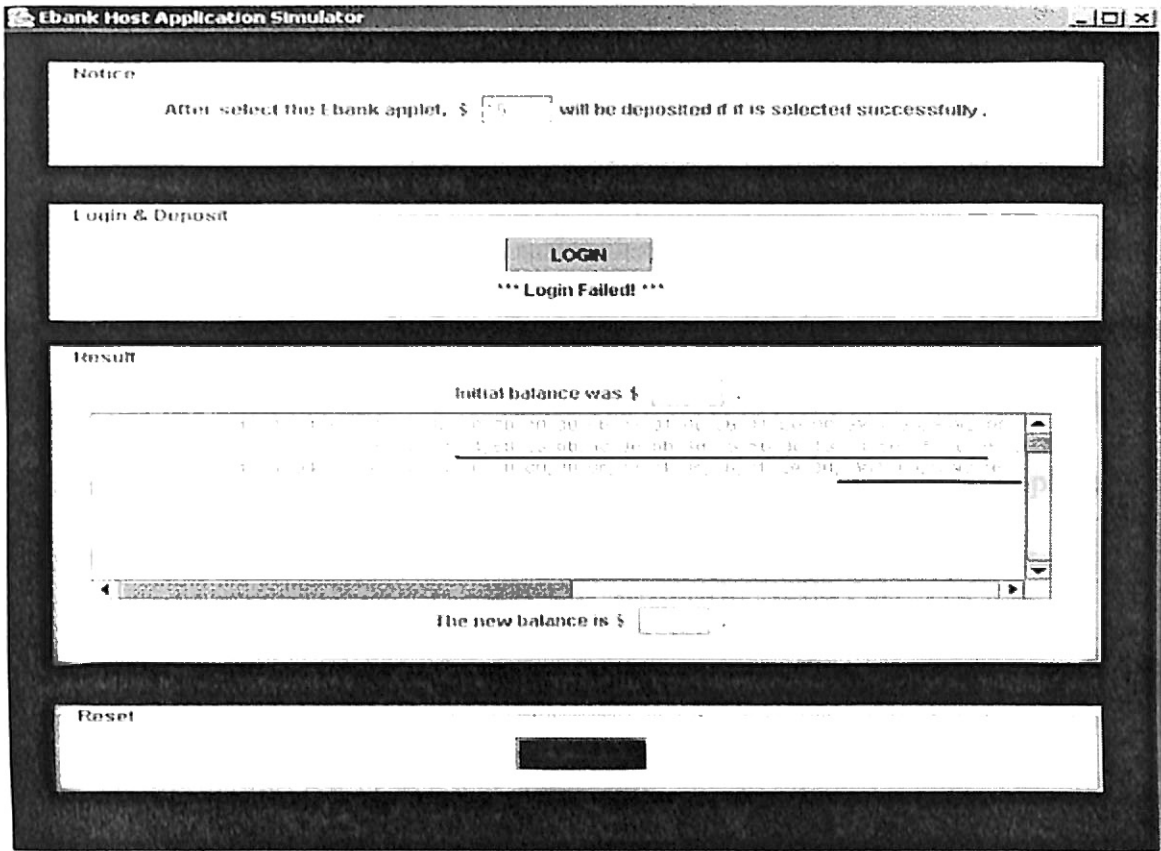


Figure 32. The result of fake Ebank applet

As we see in figure 32, the selection fails (6999). The real host application has the message 6999 and so it knows the applet is not a real applet but a fake applet.

CHAPTER V

CONCLUSION

Modern Java Card technology provides a secure environment. However, there is still a hole in the system that allows malevolent attackers to illegally obtain secure data from a host application or the Java Card. This is possible because a CAP file is verified only at the off-card portion. Hence malicious attackers can access and install fake applets on the card. Communication between a host application and an applet starts with the SELECT method specifying an applet AID. In the current Java card environment attackers can create a forged host application.

We have proposed a one-time password via a one-way hash chain in this thesis for secure communication between a host application and an on-card applet. The one-time password is generated by hashing an applet's CAP file. The JCRE has an applet's AID and a hash value from the CAP file. The password is updated every time after communication between a host application and an applet.

Even if malicious attackers make a fake host application and a counterfeit applet, they cannot access secure information. In the case of a fake host application, they have to know a hash function, the initial index of $n-2$ computations, and input value that the applet provider provided. This is almost impossible. The only way they can capture and replay a password is when a real application communicates with an applet on the card. However, since the password is changed after each communication with our proposed method, it is impossible to illegally access information on a card. In the case of a fake applet, attackers have to create an applet that is identical (not just similar) to the real applet installed on the card. An applet source code must therefore be exactly identical.

This is also practically impossible. We have simulated our scheme with the Java Card 2.2 Development Kit. It contains over 3,000 lines of code.

One of the disadvantages with the proposed scheme is the limitation imposed on the number of communications. In other words, the password is not infinite as only $n - 1$ secure communications are possible. Future work would involve developing schemes to remove the restriction on the number of secure communications between the host application and the JCRE. In this work we have also assumed that the JCRE is secure. Developing defenses against attacks on the JCRE have also to be devised. The first step would be to identify the vulnerabilities of the JCRE.

BIBLIOGRAPHY

1. Chen, Zhiqun. Java Card Technology for Smart Cards: architecture and programmer's guide. California: Sun Microsystems, Inc., 2000.
2. Rankl, Wolfgang and Effing, Wolfgang. Smart Card Handbook. England: John Wiley & Sons Ltd., 1997.
3. Dreifus, Henry and J. Thomas Monk. Smart Cards: A guide to building and managing smart card applications. United States of America: John Wiley & Sons, Inc., 1997.
4. Microsoft Computer Dictionary, Fifth Edition. Microsoft Corporation., 2002.
5. "Java Card Special Interest Group." Online. Internet. Jan. 2000. Available: http://www.javacard.org/others/what_is_java_card.htm#Applications.
6. "An Introduction to Java Card Technology – part 1." Online. Internet. May. 2003. Available: [An Introduction to Java Card Technology - Part 1](#).
7. "What is OpenCard and OpenCard Framework?" Online. Internet. May. 2003. Available: <http://www.opencard.org/overview.shtml>.
8. Ahuja, Vijay. Network & Internet Security. Michigan: AP Professional, 1996.
9. Purser, Michael. SECURE DATA NETWORKING. Norwood, MA: Artech House, Inc., 1993.
10. Richard E. Smith. Authentication From Passwords to Public Keys. Boston: Addison-Wesley., 2002.
11. "What is a one-way function?" Online. Internet. May. 2003. Available: <http://www.rsasecurity.com/rsalabs/faq/2-3-2.html>.
12. "One-way hash functions." Online. Internet. May. 2003. Available: <http://www.cs.bris.ac.uk/~cooper/HOT/guide.html>.
13. "Hash Functions and Message Digests." Online. Internet. May. 2003. Available: http://www.cs.nps.navy.mil/curricula/tracks/security/notes/chap05_19.html - HEADING18.
14. L. Lamport and P. Melliar-Smith. Password authentication with insecure communication. Communications of the ACM, 24(11): 770-772, November 1981.

15. . N. Haller. The S/Key one-time password system. In Proceedings of the Symposium on Network and Distributed Systems Security, pages 151-157. Internet Society, February 1994.
16. "Press Releases." Online. Internet. May. 2003. Available:
http://www.smartcardalliance.org/about_alliance/press_020702shipmentsurvey.cfm.
17. Jang, SyengHo. "SECURE OBJECT SHARING ON JAVA CARD." Oklahoma State university : Master thesis. May. 2003.
18. "Smart Card Overview." Online. Internet. May. 2003. Available:
<http://java.sun.com/products/javacard/smartcards.html>.

APPENDIX

Ebank.java

```
/*
*****
This applet works as a client.
When the JCRE decides that a host application is real, this applet accepts a host application's
request and perform it.
*****
*/

package electrobank;

import javacard.framework.*;
import installers.InstallInterface;

public class Ebank extends Applet {

    //codes of CLA byte in the command APDUs
    private final static byte Ebank_CLA = (byte)0xB0;

    //codes of INS byte in the command APDUs
    private final static byte DEPOSIT = (byte)0x01;
    private final static byte DEBIT = (byte)0x02;
    private final static byte BALANCE = (byte)0x03;

    //Applet-specific status words;
    private final static short SW_NEGATIVE_BALANCE = 0x6A01;
    private final static short SW_INVALID_DEBIT_AMOUNT=0x6A02;
    private final static short SW_NO_SERVER = 0x6A03;
    private final static short SW_SIO_GETTING_FAIL = 0x6A04;

    private short balance;

    //AID fo this applet instance
    private final byte[] own_aid = {0x00, 0x00, 0x00, 0x00, 0x0C, 0x03, 0x01, 0x0C, 0x06, 0x01};
    private final byte[] installer_aid = {0x00, 0x00, 0x00, 0x00, 0x0B, 0x03, 0x01, 0x0C, 0x06, 0x01};

    //parameters for sharing between Ebank and Installer
    private final byte sharepara = (byte) 0xCC;

    //start Ebank constructor
    private Ebank() {

        //initial check
        balance = (short) 0x00;

        //register this applet instance to the JCRE
        register(own_aid, (short)0, (byte)(own_aid.length));

    } //end Ebank constructor

    //start install method
    public static void install(byte[] bArray, short bOffset, byte bLength) {
```

```

//create an Ebank applet instance
new Ebank();

} //end install method

//initialize the applet when it is selected
public boolean select() {

    //obtain the server(installer) AID object
    AID sAid = JCSYSTEM.lookupAID(installer_aid, (short)0, (byte)installer_aid.length));

    //If there is no that server, then error
    if(sAid==null)
        ISOException.throwIt(SW_NO_SERVER);

    //request the sio from the server
    InstallInterface sio = (InstallInterface)(JCSYSTEM.getAppletShareableInterfaceObject(
        sAid, sharepara));

    //if the server does not have sharing interface, then error
    if(sio==null)
        ISOException.throwIt(SW_SIO_GETTING_FAIL);

    //call the installer applet to check password
    //if they are matched, return true and Ebank is selected
    if(sio.check()!=true)
        return false;
    //if they are not matched, this applet is not selected
    else
        return true;

} //end select method

//processmethod
public void process(APDU apdu) {

    //the APDU buffer
    byte[] buffer = apdu.getBuffer();

    //return if the APDU is the applet SELECT command
    if(selectingApplet())
        return;

    //verify the CLA byte
    if(buffer[ISO7816.OFFSET_CLA]!=Ebank_CLA)
        ISOException.throwIt(ISO7816.SW_CLA_NOT_SUPPORTED);

    //check the INS byte to decide which service method to call
    switch(buffer[ISO7816.OFFSET_INS]) {
        case DEPOSIT: deposit(apdu);
            return;
        case DEBIT: debit(apdu);
            return;
        case BALANCE: balance(apdu);
            return;
    }
}

```

```

        default: ISOException.throwIt(ISO7816.SW_INS_NOT_SUPPORTED);
    } //end switch method

} //end process method

/start deposit method
private void deposit(APDU apdu) {

    byte[] buffer = apdu.getBuffer();

    //get the number of bytes in the command field APDU's data field
    byte numBytes = buffer[ISO7816.OFFSET_LC];

    //read the data into the apdu buffer
    byte readByte = (byte)(apdu.setIncomingAndReceive());

    //if the data bytes read does not match the number in the LC byte
    //then error
    if((numBytes != 1) || (readByte != 1))
        ISOException.throwIt(ISO7816.SW_WRONG_LENGTH);

    //get the deposit amount
    byte depositAmount = buffer[ISO7816.OFFSET_CDATA];

    //deposit the amount
    alance = (short)(balance + depositAmount);

    return;
} //end deposit method

//start debit method
private void debit (APDU apdu) {

    byte[] buffer = apdu.getBuffer();

    //get the number of bytes in the command field APDU's data field
    byte numBytes = buffer[ISO7816.OFFSET_LC];

    //read the data into the apdu buffer
    byte readByte = (byte)(apdu.setIncomingAndReceive());

    //if the data bytes read does not match the number in the LC byte
    //then error
    if((numBytes != 1) || (readByte != 1))
        ISOException.throwIt(ISO7816.SW_WRONG_LENGTH);

    //get the deposit amount
    byte debitAmount = buffer[ISO7816.OFFSET_CDATA];

    //if debit amount is negative, then error
    if(debitAmount < 0)
        ISOException.throwIt(SW_INVALID_DEBIT_AMOUNT);

    //if new balance is negative then error

```

```

        if((short)(balance-debitAmount)<(short)0)
            ISOException.throwIt(SW_NEGATIVE_BALANCE);

        //new balance
        balance = (short)(balance-debitAmount);

    }//end debit method

    //start balance method
    private void balance (APDU apdu) {

        byte[] buffer = apdu.getBuffer();

        //notify the JCRE that the applet has data to return
        short le = apdu.setOutgoing();

        //set actual data byte's numbers
        apdu.setOutgoingLength((byte)2);

        //write the balance in to the APDU buffer
        Util.setShort(buffer, (short)0, balance);

        //send balance
        apdu.sendBytes((short)0, (short)2);

    }//end balance method

} //end Ebank class

```

```

//to install installer applet to the card
installIns = new JButton("Install");
installIns.setBackground(Color.yellow);
installIns.setBounds(25, 130, 275, 30);

//to compile
compileIns.addActionListener(new ActionListener(){
    public void actionPerformed (ActionEvent e){

        try{

            String order = "c:/defense/batch/compileInstaller.bat";
            Process child = Runtime.getRuntime().exec(order);

        }catch (Exception ee){}
    }
});

//to convert
converterIns.addActionListener(new ActionListener(){
    public void actionPerformed (ActionEvent e){

        try{

            String order = "c:/defense/batch/convertInstaller.bat";
            String order2 = "c:/defense/batch/moveInstallExp.bat";
            Process child = Runtime.getRuntime().exec(order);
            child = Runtime.getRuntime().exec(order2);
        }catch (Exception ee){}
    }
});

//to mask
installIns.addActionListener(new ActionListener() {
    public void actionPerformed (ActionEvent e) {

        fileChooser = new JFileChooser("c:\\defense\\class\\installers\\javacard");
        fileChooser.setDialogTitle("Choose a CAP file?");
        int forError = fileChooser.showOpenDialog(Mask.this);

        //for checking CAP selection error
        if(forError == JFileChooser.APPROVE_OPTION)
            capFile = fileChooser.getSelectedFile();
        else{
            JOptionPane.showMessageDialog(Mask.this,
                "Need a CAP file", "ERROR",JOptionPane.ERROR_MESSAGE);
            capFile = null;
            return;
        }
    }
});

//to make batch file and script file for sending them to JCRE
try{
    //make script batch file
    batchFile = new File("script.bat");
    //make script file
    scriptFile = new File("apdu.scr");
}

```

```

BufferedWriter text = new BufferedWriter(new FileWriter(batchFile));
text.write("@echo off\n");
text.write("scriptgen -o " + scriptFile.getAbsolutePath() + " "
          + capFile.getAbsolutePath() + "\n");
text.close();
String path = batchFile.getAbsolutePath();
path = path.replace('\\', '/');
Process child = Runtime.getRuntime().exec(path);
child.waitFor();
} catch (Exception e1){
} finally {
    batchFile.delete();
    batchFile = null;
}

//to send power.scr file to the installer
outputFile = new File("power.scr");
String readData = new String();

try {
    BufferedReader reader = new BufferedReader(new FileReader(scriptFile));
    BufferedWriter writer = new BufferedWriter(new FileWriter(outputFile));

    //turn on the card
    writer.write("powerup;\n");

    //to wake up the JCRE installer
    writer.write("//select the installer \n");

    //apdu command
    writer.write("0x00 0xA4 0x04 0x00 0x09 0xA0 0x00 0x00 0x00
                0x62 0x03 0x01 0x08 0x01 0x7F;\n\n");

    //read whole data from script file that is made by scriptgen command
    while((readData = reader.readLine()) != null){
        writer.write(readData + "\n");
    }

    //create installer
    writer.write("\n//Install this applet\n");
    writer.write("0x80 0xb8 0x00 0x00 0x0c 0x0a 0x00 0x00 0x00 0x00
                0x0B 0x03 0x01 0x0C 0x06 0x01 0x00 0x7f;\n\n");
    //turn card off
    writer.write("powerdown;");

    //done read data
    reader.close();
    writer.close();

} catch (FileNotFoundException ee){
} catch (IOException eee){
} finally {
    scriptFile.delete();
    scriptFile = null;
}

```



```

//create installer applet
try{
    //make ebank install batch file
    batchFile = new File("install.bat");
    BufferedWriter toWrite = new BufferedWriter(new FileWriter(batchFile));
    toWrite.write("@echo off\n");

    //to send installer and have an answer from the JCRE installer applet
    toWrite.write("apdutool -o answer " + outputFile.getAbsolutePath() + "\n");
    toWrite.close();

    String path = batchFile.getAbsolutePath();
    path = path.replace("\\", '/');
    Process child = Runtime.getRuntime().exec(path);
    child.waitFor();
} catch (Exception e1){
} finally{
    outputFile.delete();
    outputFile = null;
    batchFile.delete();
    batchFile = null;
}
});

//add these to simulation window
getContentPane().add(compileIns);
getContentPane().add(converterIns);
getContentPane().add(installIns);

} //end constructor

//start main method
public static void main (String[] args){

    //call the constructor
    JFrame mainWindow = new Mask();

    //to set simulation window boundary
    mainWindow.setBounds(0, 0, 330, 218);
    mainWindow.setVisible(true);

} //end main method

} //end class Mask

```

HostEbank.java

```

/*****
HostEbank can work as a host application.
It sends a one-time password to select a client (Ebank) applet.
A hash function to make a password is included.
Actual original name is the MD5 message-Digest Algorithm from RFC 1321
It is implemented as C language on webpage (http://www.faqs.org/rfcs/rfc1321.html).
*****/
import java.io.*;
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;

class HostEbank extends JFrame {
    //to make visible window
    //define panel, text, label, button

    //make first panel
    private JPanel first_panel;
    private JLabel message1;
    private JTextField amount;
    private JLabel message2;
    //make second panel
    private JPanel second_panel;
    private JButton login;
    private JLabel response;
    //make third panel
    private JPanel third_panel;
    private JLabel initial;
    private JTextField initial_bal;
    private JLabel initial2;
    private JTextArea area;
    private JScrollPane scroll;
    private JLabel after;
    private JTextField after_bal;
    private JLabel after2;
    //make fourth panel
    private JPanel fourth_panel;
    private JButton reset;

    private int money;
    private int pre_index;
    private int index;
    private String message_digest;

    private File scr_file;
    private File batch;
    private File result;

    private byte[] password = new byte [16];
    private byte[] hashval = new byte [16];
    private byte[] tempMessage = new byte [64];
    private MD5 md5 = new MD5();
    private int[][] index_table = new int[4][2];

```

```

private int index_table_index = 0;

HostEbank () {
    //title
    setTitle("Ebank Host Application Simulator");
    setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    //color
    getContentPane().setLayout(null);
    getContentPane().setBackground(Color.blue);
    //size & color
    first_panel = new JPanel();
    first_panel.setBounds(25, 25, 630, 80);
    first_panel.setBackground(Color.white);
    first_panel.setBorder(BorderFactory.createTitledBorder
        (BorderFactory.createLineBorder(Color.orange), " Notice "));
    //message
    message1 = new JLabel("After select the Ebank applet, $ ");
    amount = new JTextField(4); //amount will be saved in text field 4
    message2 = new JLabel("will be deposited if it is selected successfully .");
    //add message1, amount, message2 to the first panel
    first_panel.add(message1);
    first_panel.add(amount);
    first_panel.add(message2);

    //second panel
    second_panel = new JPanel();
    second_panel.setBounds(25, 135, 630, 90);
    second_panel.setBackground(Color.white);
    second_panel.setBorder(BorderFactory.createTitledBorder
        (BorderFactory.createLineBorder(Color.orange), " Login & Deposit "));
    //make label
    JLabel ghost = new JLabel(" ");
    login = new JButton(" LOGIN ");
    login.setBackground(Color.orange);
    JLabel ghost2 = new JLabel(" ");
    response = new JLabel(" ", SwingConstants.CENTER);
    response.setForeground(Color.blue);
    //add them to second panel
    second_panel.add(ghost);
    second_panel.add(login);
    second_panel.add(ghost2);
    second_panel.add(response);

    //make third panel
    third_panel = new JPanel();
    third_panel.setBounds(25, 245, 630, 250);
    third_panel.setBackground(Color.white);
    third_panel.setBorder(BorderFactory.createTitledBorder
        (BorderFactory.createLineBorder(Color.orange), " Result "));
    //make label and text
    initial = new JLabel(" Initial balance was $ ");
    initial_bal = new JTextField(4);
    initial_bal.setEnabled(false);
    initial2 = new JLabel(" .");
    JLabel ghost3 = new JLabel("t");
    area = new JTextArea(250,100);

```

```

area.setBackground(Color.white);
area.setForeground(Color.black);
area.setEnabled(false);
scroll = new JScrollPane(area);
scroll.setPreferredSize(new Dimension(580, 150));
JLabel ghost4 = new JLabel("\t");
after = new JLabel("The new balance is $ ");
after_bal = new JTextField(4);
after_bal.setEnabled(false);
after2 = new JLabel(" .");
//add them to third pannel
third_panel.add(initial);
third_panel.add(initial_bal);
third_panel.add(initial2);
third_panel.add(ghost3);
third_panel.add(scroll);
third_panel.add(ghost4);
third_panel.add(after);
third_panel.add(after_bal);
third_panel.add(after2);

//start fourth panel
fourth_panel = new JPanel();
fourth_panel.setBounds(25, 525, 630, 70);
fourth_panel.setBackground(Color.white);
fourth_panel.setBorder(BorderFactory.createTitledBorder
    (BorderFactory.createLineBorder(Color.orange), " Reset "));
//initialize reset button
reset = new JButton(" RESET ");
reset.setBackground(Color.blue);
//add it to fourth panel
fourth_panel.add(reset);

//this index used for times to make hash value
pre_index = 4;
index = 6;

//initial hash value
hashval[0] = (byte) 0x21;
hashval[1] = (byte) 0xcd;
hashval[2] = (byte) 0xf4;
hashval[3] = (byte) 0x44;
hashval[4] = (byte) 0xdd;
hashval[5] = (byte) 0x5a;
hashval[6] = (byte) 0xcd;
hashval[7] = (byte) 0x05;
hashval[8] = (byte) 0x2a;
hashval[9] = (byte) 0x00;
hashval[10] = (byte) 0xd0;
hashval[11] = (byte) 0x6b;
hashval[12] = (byte) 0x6b;
hashval[13] = (byte) 0xcd;
hashval[14] = (byte) 0x54;
hashval[15] = (byte) 0x57;

//index table

```

```

index_table[0][0] = 4;
index_table[0][1] = 6;
index_table[1][0] = 3;
index_table[1][1] = 0;
index_table[2][0] = 2;
index_table[2][1] = 0;
index_table[3][0] = 1;
index_table[3][1] = 0;

// login button -> power up -> select instller -> send password and index -> select ebank
// -> check up -> hash -> reponse update label
// -> current balance -> deposit (amount -> money) -> current balance -> power down.
login.addActionListener(new ActionListener() {
    public void actionPerformed (ActionEvent e) {

        try {
            //set as a false
            amount.setEnabled(false);

            // stroe the value of deposit
            money = Integer.parseInt(amount.getText());

            String money16 = Integer.toString(money, 16);
            if (money16.length() == 1)
                money16 = "0" + money16;

            //to make a password from hash value
            //from the hash value buffer to temp message buffer
            for(int j=0; j<16; j++)
                tempMessage[j] = hashval[j];

            //to make initial password
            for(short k=0; k<index_table[index_table_index][0]; k++){

                //md5 initialize
                md5.md5Init();
                //md5 update
                md5.md5Update(tempMessage, (short)16);
                //md5 generate and save it
                md5.md5Gen(password);

                for(int l=0; l<16; l++)
                    tempMessage[l] = password[l];
            }

            String index16 = Integer.toString(index_table[index_table_index][1], 16);
            if (index16.length() == 1)
                index16 = "0" + index16;
            //make defined apdu file
            scr_file = new File("apdu.scr");
            BufferedWriter out = new BufferedWriter(new FileWriter(scr_file));

            //write apdu command file
            //turn the card on
            out.write("powerup;\n\n");

```

```

out.write("// select installer applet\n\n");
out.write("0x0 0xa4 0x04 0x0 0x0a 0x00 0x00 0x00 0x00 0x0B 0x03 0x01
          0x0C 0x06 0x01 0x7f;\n\n");

message_digest = new String();
String temp = new String();

//append 0x0000000F -> show that one command is finished
for (int i = 0; i < 16; i++) {

    temp = Integer.toString(password[i]&0x000000FF, 16);

    if (temp.length() == 1)
        temp = "0" + temp;

    temp = "0x" + temp;
    message_digest = message_digest + temp + " ";
}

out.write("// store initial password to the installer\n\n");
out.write("0xB0 0x01 0x00 0x00 0x12 " + message_digest + "0x" + index16 +
          " 0x00 0x7f;\n\n");

out.write("// Select Ebank applet\n");
out.write("0x0 0xa4 0x04 0x0 0x0a 0x00 0x00 0x00 0x00 0x0C 0x03 0x01
          0x0C 0x06 0x01 0x7f;\n\n");

out.write("// display balance\n");
out.write("0xB0 0x03 0x00 0x00 0x01 0x02 0x7f;\n\n");

out.write("// deposit\n");
out.write("0xB0 0x01 0x00 0x00 0x01 0x" + money16 + " 0x7F;\n\n");

out.write("// display balance\n");
out.write("0xB0 0x03 0x00 0x00 0x01 0x02 0x7f;\n\n");

out.write("powerdown;");
out.close();

} catch (Exception ee) {
}

try {
//make them into batch file to execute
batch = new File("execute.bat");
result = new File("result");
BufferedWriter out = new BufferedWriter(new FileWriter(batch));
out.write("@echo off\n");
//to send commands to the card
out.write("apdutool -o " + result.getAbsolutePath() + " "
          + scr_file.getAbsolutePath() + "\n");
out.close();
// replace \\ to / to make a path
String batch_path = batch.getAbsolutePath();
batch_path = batch_path.replace("\\", '/');
Process child = Runtime.getRuntime().exec(batch_path);

```

```

    child.waitFor();

    } catch (Exception e2) {
    }

String text;
int access_line = 0;
boolean access_result = false;
String access_text = new String();
String first_bal = new String();
String end_bal = new String();

try {
    //show result
    BufferedReader reader = new BufferedReader(new FileReader(result));
    //read line by line
    while ((text = reader.readLine()) != null) {

        ++ access_line;

        if (access_line == 3) {
            access_text = text;
            access_text = access_text.substring(95,97);
            //90 means success
            if (Integer.parseInt(access_text) == 90)
                access_result = true;
        }

        if (access_line == 4) {

            if (access_result == true) {
                first_bal = text;
                first_bal = first_bal.substring(54,56) +
                first_bal.substring(58,60);
                //make balance
                first_bal =String.valueOf(Integer.parseInt(first_bal, 16));
            }

            else {
                //set up when it is failed
                response.setForeground(Color.red);
                response.setText("*** Login Failed! ***");
                area.append("\n");
                result.delete();
                result = null;
                return;
            }
        }
        //compute banlance
        if (access_line == 6) {
            end_bal = text;
            end_bal = end_bal.substring(54,56) +end_bal.substring(58,60);
            end_bal = String.valueOf(Integer.parseInt(end_bal, 16));
        }
    }
}

```

```

        area.append(text+ "\n");

    } // end of while

    reader.close();
    //set up response
    response.setForeground(Color.blue);
    response.setText(" *** Welcome Jae Hyuk Joo! *** ");
    area.append("\n");

    initial_bal.setText(first_bal);
    after_bal.setText(end_bal);

    index_table_index++;

    if (index_table_index == 4)
        index_table_index = 0;

} catch (Exception e3) { //error
} finally { //delete them all after done
    scr_file.delete();
    scr_file = null;
    batch.delete();
    batch = null;
    result.delete();
    result = null;
}

}
});

reset.addActionListener(new ActionListener() {
    public void actionPerformed (ActionEvent e) {
        //make them null
        amount.setText(null);
        amount.setEnabled(true);
        response.setText(null);
        initial_bal.setText(null);
        area.append("\n");
        after_bal.setText(null);

    }
});

getContentPane().add(first_panel);
getContentPane().add(second_panel);
getContentPane().add(third_panel);
getContentPane().add(fourth_panel);

} // end of constructor

//main method
public static void main (String[] args) {

```



```

        //set up boundary
        JFrame frame = new HostEbank();
        frame.setBounds(0, 0, 695, 670);
        frame.setVisible(true);

    }//end main method

}//end HostEbank

//start md5 class
class MD5 {

    /* The source of this MD5 algorithm is from RFC 1321 from MIT laboratory for
    Computer Science and RSA data security.inc.
    The original source is untimely distributed.
    This MD5 is changed and adapted to the JAVA CARD    */

    /* Constants for MD5Transform routine */
    private static final byte S11 = 7; private static final byte S12 = 12;
    private static final byte S13 = 17; private static final byte S14 = 22;
    private static final byte S21 = 5; private static final byte S22 = 9;
    private static final byte S23 = 14; private static final byte S24 = 20;
    private static final byte S31 = 4; private static final byte S32 = 11;
    private static final byte S33 = 16; private static final byte S34 = 23;
    private static final byte S41 = 6; private static final byte S42 = 10;
    private static final byte S43 = 15; private static final byte S44 = 21;

    //to append padding bits - 128bits (64 bytes)
    private static final byte[] padding = {
        (byte)0x80, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
        0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
        0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
        0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
        0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
        0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00
    };

    //for four-word buffer(A,B,C,D)
    //These register buffers are used for computing the message digest
    private byte[] wordA = new byte[4];
    private byte[] wordB = new byte[4];
    private byte[] wordC = new byte[4];
    private byte[] wordD = new byte[4];

    //temp four-word buffer for increment each of the four registers by the value
    private byte[] wordAA = new byte[4];
    private byte[] wordBB = new byte[4];
    private byte[] wordCC = new byte[4];
    private byte[] wordDD = new byte[4];

    //for before padding initialize it to false

```

```

private boolean beforePadding = false;

//a 64-element table
//This is constructed from the sine function
private byte[] sixtyFourETable = new byte[64];

//for total message length
private byte[] totalML = new byte[8];

//for partial message
private byte[] partialM = new byte[4];
private byte[] partialM2 = new byte[4];

//start md5Init method for MD5 initialization
//Begins an MD5 operation
void md5Init () {

    //wordA = 0x67452301
    wordA[0] = (byte)0x67; wordA[1] = (byte)0x45;
    wordA[2] = (byte)0x23; wordA[3] = (byte)0x01;
    //wordB = 0xefcdab89
    wordB[0] = (byte)0xEF; wordB[1] = (byte)0xCD;
    wordB[2] = (byte)0xAB; wordB[3] = (byte)0x89;
    //wordC = 0x98badcfe
    wordC[0] = (byte)0x98; wordC[1] = (byte)0xBA;
    wordC[2] = (byte)0xDC; wordC[3] = (byte)0xFE;
    //wordD = 0x10325476
    wordD[0] = (byte)0x10; wordD[1] = (byte)0x32;
    wordD[2] = (byte)0x54; wordD[3] = (byte)0x76;

    //initialize total message length buffer
    for (short i = 0; i<8; i++)
        totalML[i] = 0x00;

    //initialize beforePadding as a false
    beforePadding = false;

} //end md5Init method

//hash value update
void md5Update(byte[] buffer, short length) {

    //by input length - initial case is 0
    short processByCase = 0;

    switch(length) {

        //In case each data block bits are 512 bits
        case 64: processByCase = 1;
        break;

        //after processing all data blocks
        case 0:
        if (beforePadding == false){
            processByCase = -1;
            break;
        }
    }
}

```

```

    }
    else
        return;

    //for the last data block
    default:    processByCase = 3;
    beforePadding = true;
}

//to add message length
appendLength(totalML, length);

//copy from Array wordA to paste Array wodrdAA
arrayCopy(wordA, (short)0, wordAA, (short)0, (short)4);

//copy from Array wordB to paste Array wodrdBB
arrayCopy(wordB, (short)0, wordBB, (short)0, (short)4);

//copy from Array wordC to paste Array wodrdCC
arrayCopy(wordC, (short)0, wordCC, (short)0, (short)4);

//copy from Array wordD to paste Array wodrdDD
arrayCopy(wordD, (short)0, wordDD, (short)0, (short)4);

do{

    //making 64-element table T[1...64]
    switch (processByCase) {

        //padding 448 bits (100...0) and save it with 64bits (total message)
        case -1:
            arrayCopy(padding, (short)0, sixtyFourETable, (short)0, (short)56);
            arrayCopy(totalML, (short)0, sixtyFourETable, (short)56, (short)8);
            processByCase = 0;
            break;

        //save 512 bit message
        case 1:
            arrayCopy(buffer, (short)0, sixtyFourETable, (short)0, (short)64);
            processByCase = 0;
            break;

        //padding 448 bits (000...0) and save it with 64bits (total message)
        case 2:
            arrayCopy(padding, (short)8, sixtyFourETable, (short)0, (short)56);
            arrayCopy(totalML, (short)0, sixtyFourETable, (short)56, (short)8);
            processByCase = 0;
            break;

        //to make the last input(8bits~512bits) 512bits and make it 128 bits
        case 3:
            if(length < 56) {
                short required_pad = (short)((short)56 - length);
                arrayCopy(buffer, (short)0, sixtyFourETable, (short)0, length);
                arrayCopy(padding, (short)0, sixtyFourETable, (short)length, required_pad);
            }
    }
}

```

```

        arrayCopy(totalML, (short)0, sixtyFourETable, (short)56, (short)8);
        processByCase = 0;
    }
    else {
        short required_pad = (short)((short)64 - length);
        arrayCopy(buffer, (short)0, sixtyFourETable, (short)0, length);
        arrayCopy(padding, (short)0, sixtyFourETable, (short)length, required_pad);
        processByCase = 2;
    }
    break;
}

```

/* call methods - Round 1,2,3,4.
each method is called by total 16*/

```

//round 1
roundOne (wordA, wordB, wordC, wordD, sixtyFourETable, (short)0, S11, (byte)0xD7, (byte)0x6A,
          (byte)0xA4, (byte)0x78);
roundOne (wordD, wordA, wordB, wordC, sixtyFourETable, (short)4, S12, (byte)0xE8, (byte)0xC7,
          (byte)0xB7, (byte)0x56);
roundOne (wordC, wordD, wordA, wordB, sixtyFourETable, (short)8, S13, (byte)0x24, (byte)0x20,
          (byte)0x70, (byte)0xDB);
roundOne (wordB, wordC, wordD, wordA, sixtyFourETable, (short)12, S14, (byte)0xC1, (byte)0xBD,
          (byte)0xCE, (byte)0xEE);
roundOne (wordA, wordB, wordC, wordD, sixtyFourETable, (short)16, S11, (byte)0xF5, (byte)0x7C,
          (byte)0x0F, (byte)0xAF);
roundOne (wordD, wordA, wordB, wordC, sixtyFourETable, (short)20, S12, (byte)0x47, (byte)0x87,
          (byte)0xC6, (byte)0x2A);
roundOne (wordC, wordD, wordA, wordB, sixtyFourETable, (short)24, S13, (byte)0xA8, (byte)0x30,
          (byte)0x46, (byte)0x13);
roundOne (wordB, wordD, wordD, wordA, sixtyFourETable, (short)28, S14, (byte)0xFD, (byte)0x46,
          (byte)0x95, (byte)0x01);
roundOne (wordA, wordB, wordC, wordD, sixtyFourETable, (short)32, S11, (byte)0x69, (byte)0x80,
          (byte)0x98, (byte)0xD8);
roundOne (wordD, wordA, wordB, wordC, sixtyFourETable, (short)36, S12, (byte)0x8B, (byte)0x44,
          (byte)0xF7, (byte)0xAF);
roundOne (wordC, wordD, wordA, wordB, sixtyFourETable, (short)40, S13, (byte)0xFF, (byte)0xFF,
          (byte)0x5B, (byte)0xB1);
roundOne (wordB, wordC, wordD, wordA, sixtyFourETable, (short)44, S14, (byte)0x89, (byte)0x5C,
          (byte)0xD7, (byte)0xBE);
roundOne (wordA, wordB, wordC, wordD, sixtyFourETable, (short)48, S11, (byte)0x6B, (byte)0x90,
          (byte)0x11, (byte)0x22);
roundOne (wordD, wordA, wordB, wordC, sixtyFourETable, (short)52, S12, (byte)0xFD, (byte)0x98,
          (byte)0x71, (byte)0x93);
roundOne (wordC, wordD, wordA, wordB, sixtyFourETable, (short)56, S13, (byte)0xA6, (byte)0x79,
          (byte)0x43, (byte)0x8E);
roundOne (wordB, wordC, wordD, wordA, sixtyFourETable, (short)60, S14, (byte)0x49, (byte)0xB4,
          (byte)0x08, (byte)0x21);

```

```

//round 2
roundTwo (wordA, wordB, wordC, wordD, sixtyFourETable, (short)4, S21, (byte)0xF6, (byte)0x1E,
          (byte)0x25, (byte)0x62);
roundTwo (wordD, wordA, wordB, wordC, sixtyFourETable, (short)24, S22, (byte)0xC0, (byte)0x40,

```

```

        (byte)0xB3, (byte)0x40);
roundTwo (wordC, wordD, wordA, wordB, sixtyFourETable, (short)44, S23, (byte)0x26, (byte)0x5E,
        (byte)0x5A, (byte)0x51);
roundTwo (wordB, wordC, wordD, wordA, sixtyFourETable, (short)0, S24, (byte)0xE9, (byte)0xB6,
        (byte)0xC7, (byte)0xAA);
roundTwo (wordA, wordB, wordC, wordD, sixtyFourETable, (short)20, S21, (byte)0xD6, (byte)0x2F,
        (byte)0x10, (byte)0x5D);
roundTwo (wordD, wordA, wordB, wordC, sixtyFourETable, (short)40, S22, (byte)0x2, (byte)0x44,
        (byte)0x14, (byte)0x53);
roundTwo (wordC, wordD, wordA, wordB, sixtyFourETable, (short)60, S23, (byte)0xD8, (byte)0xA1,
        (byte)0xE6, (byte)0x81);
roundTwo (wordB, wordD, wordD, wordA, sixtyFourETable, (short)24, S24, (byte)0xE7, (byte)0xD3,
        (byte)0xFB, (byte)0xC8);
roundTwo (wordA, wordB, wordC, wordD, sixtyFourETable, (short)36, S21, (byte)0x21, (byte)0xE1,
        (byte)0xCD, (byte)0xE6);
roundTwo (wordD, wordA, wordB, wordC, sixtyFourETable, (short)56, S22, (byte)0xC3, (byte)0x37,
        (byte)0x07, (byte)0xD6);
roundTwo (wordC, wordD, wordA, wordB, sixtyFourETable, (short)12, S23, (byte)0xF4, (byte)0xD5,
        (byte)0x0D, (byte)0x87);
roundTwo (wordB, wordC, wordD, wordA, sixtyFourETable, (short)32, S24, (byte)0x45, (byte)0x5A,
        (byte)0x14, (byte)0xED);
roundTwo (wordA, wordB, wordC, wordD, sixtyFourETable, (short)52, S21, (byte)0xA9, (byte)0xE3,
        (byte)0xE9, (byte)0x05);
roundTwo (wordD, wordA, wordB, wordC, sixtyFourETable, (short)8, S22, (byte)0xFC, (byte)0xEF,
        (byte)0xA3, (byte)0xF8);
roundTwo (wordC, wordD, wordA, wordB, sixtyFourETable, (short)28, S23, (byte)0x67, (byte)0x6F,
        (byte)0x02, (byte)0xD9);
roundTwo (wordB, wordC, wordD, wordA, sixtyFourETable, (short)48, S24, (byte)0x8D, (byte)0x2A,
        (byte)0x4C, (byte)0x8A);

```

```
//round 3
```

```

roundThree(wordA, wordB, wordC, wordD, sixtyFourETable, (short)20, S31, (byte)0xFF,
        (byte)0xFA, (byte)0x39, (byte)0x42);
roundThree(wordD, wordA, wordB, wordC, sixtyFourETable, (short)32, S32, (byte)0x87,
        (byte)0x71, (byte)0xF6, (byte)0x81);
roundThree(wordC, wordD, wordA, wordB, sixtyFourETable, (short)44, S33, (byte)0x6D,
        (byte)0x9D, (byte)0x61, (byte)0x22);
roundThree(wordB, wordC, wordD, wordA, sixtyFourETable, (short)56, S34, (byte)0xFD,
        (byte)0xE5, (byte)0x38, (byte)0x0C);
roundThree(wordA, wordB, wordC, wordD, sixtyFourETable, (short)4, S31, (byte)0xA4,
        (byte)0xBE, (byte)0xEA, (byte)0x44);
roundThree(wordD, wordA, wordB, wordC, sixtyFourETable, (short)16, S32, (byte)0x4B,
        (byte)0xDE, (byte)0xCF, (byte)0xA9);
roundThree(wordC, wordD, wordA, wordB, sixtyFourETable, (short)28, S33, (byte)0xF6,
        (byte)0xBB, (byte)0x4B, (byte)0x80);
roundThree(wordB, wordD, wordD, wordA, sixtyFourETable, (short)40, S34, (byte)0xBE,
        (byte)0xBF, (byte)0xBC, (byte)0x70);
roundThree(wordA, wordB, wordC, wordD, sixtyFourETable, (short)52, S31, (byte)0x28,
        (byte)0x9B, (byte)0x7E, (byte)0xC6);
roundThree(wordD, wordA, wordB, wordC, sixtyFourETable, (short)0, S32, (byte)0xEA,
        (byte)0xA1, (byte)0x27, (byte)0xFA);
roundThree(wordC, wordD, wordA, wordB, sixtyFourETable, (short)12, S33, (byte)0xD4,
        (byte)0xEF, (byte)0x30, (byte)0x85);
roundThree(wordB, wordC, wordD, wordA, sixtyFourETable, (short)24, S34, (byte)0x4,
        (byte)0x88, (byte)0x1D, (byte)0x05);
roundThree(wordA, wordB, wordC, wordD, sixtyFourETable, (short)36, S31, (byte)0xD9,

```

```

        (byte)0xD4, (byte)0xD0, (byte)0x39);
roundThree(wordD, wordA, wordB, wordC, sixtyFourETable, (short)48, S32, (byte)0xE6,
        (byte)0xDB, (byte)0x99, (byte)0xE5);
roundThree(wordC, wordD, wordA, wordB, sixtyFourETable, (short)60, S33, (byte)0x1F,
        (byte)0xA2, (byte)0x7C, (byte)0xF8);
roundThree(wordB, wordC, wordD, wordA, sixtyFourETable, (short)8, S34, (byte)0xC4,
        (byte)0xAC, (byte)0x56, (byte)0x65);

//round 4
roundFour (wordA, wordB, wordC, wordD, sixtyFourETable, (short)0, S41, (byte)0xF4, (byte)0x29,
        (byte)0x22, (byte)0x44);
roundFour (wordD, wordA, wordB, wordC, sixtyFourETable, (short)28, S42, (byte)0x43, (byte)0x2A,
        (byte)0xFF, (byte)0x97);
roundFour (wordC, wordD, wordA, wordB, sixtyFourETable, (short)56, S43, (byte)0xAB, (byte)0x94,
        (byte)0x23, (byte)0xA7);
roundFour (wordB, wordC, wordD, wordA, sixtyFourETable, (short)20, S44, (byte)0xFC, (byte)0x93,
        (byte)0xA0, (byte)0x39);
roundFour (wordA, wordB, wordC, wordD, sixtyFourETable, (short)48, S41, (byte)0x65, (byte)0x5B,
        (byte)0x59, (byte)0xC3);
roundFour (wordD, wordA, wordB, wordC, sixtyFourETable, (short)12, S42, (byte)0x8F, (byte)0x0C,
        (byte)0xCC, (byte)0x92);
roundFour (wordC, wordD, wordA, wordB, sixtyFourETable, (short)40, S43, (byte)0xFF, (byte)0xEF,
        (byte)0xF4, (byte)0x7D);
roundFour (wordB, wordD, wordD, wordA, sixtyFourETable, (short)4, S44, (byte)0x85, (byte)0x84,
        (byte)0x5D, (byte)0xD1);
roundFour (wordA, wordB, wordC, wordD, sixtyFourETable, (short)32, S41, (byte)0x6F, (byte)0xA8,
        (byte)0x7E, (byte)0x4F);
roundFour (wordD, wordA, wordB, wordC, sixtyFourETable, (short)60, S42, (byte)0xFE, (byte)0x2C,
        (byte)0xE6, (byte)0xE0);
roundFour (wordC, wordD, wordA, wordB, sixtyFourETable, (short)24, S43, (byte)0xA3, (byte)0x01,
        (byte)0x43, (byte)0x14);
roundFour (wordB, wordC, wordD, wordA, sixtyFourETable, (short)52, S44, (byte)0x4E, (byte)0x08,
        (byte)0x11, (byte)0xA1);
roundFour (wordA, wordB, wordC, wordD, sixtyFourETable, (short)16, S41, (byte)0xF7, (byte)0x53,
        (byte)0x7E, (byte)0x82);
roundFour (wordD, wordA, wordB, wordC, sixtyFourETable, (short)44, S42, (byte)0xBD,
        (byte)0x3A, (byte)0xF2, (byte)0x35);
roundFour (wordC, wordD, wordA, wordB, sixtyFourETable, (short)8, S43, (byte)0x2A, (byte)0xD7,
        (byte)0xD2, (byte)0xBB);
roundFour (wordB, wordC, wordD, wordA, sixtyFourETable, (short)36, S44, (byte)0xEB, (byte)0x86,
        (byte)0xD3, (byte)0x91);

/* save A as AA, B as BB, C as CC, and D as DD */
byteAddFun(wordA, wordAA, (short)0, (short)3);
byteAddFun(wordB, wordBB, (short)0, (short)3);
byteAddFun(wordC, wordCC, (short)0, (short)3);
byteAddFun(wordD, wordDD, (short)0, (short)3);

} while (processByCase != 0);

} //end md5 update method

//start md5Gen method
void md5Gen (byte[] md) {

```

```

//generate output A
for (short i = 0, j = 3; j >= 0; ++i, --j)
    md[i] = wordA[j];

//generate output B
for (short i = 4, j = 3; j >= 0; ++i, --j)
    md[i] = wordB[j];

//generate output C
for (short i = 8, j = 3; j >= 0; ++i, --j)
    md[i] = wordC[j];

//generate output D
for (short i = 12, j = 3; j >= 0; ++i, --j)
    md[i] = wordD[j];

} //end md5Gen method

//for round 1: a = b + ((a + F(b,c,d) + X[k] + T[i]) <<< s)
private void roundOne (byte[] a, byte[] b, byte[] c, byte[] d, byte[] x,
    short point, byte s, byte t1, byte t2, byte t3, byte t4) {

    andFun(b, c, partialM);
    compFun(b, partialM2);
    andFun(partialM2, d, partialM2);
    orFun(partialM, partialM2, partialM2);
    arrayCopy(x, (short)point, partialM, (short)0, (short)4);
    byteAddFun(partialM2, partialM, (short)0, (short)3);
    partialM[0] = t1; partialM[1] = t2;
    partialM[2] = t3; partialM[3] = t4;
    byteAddFun (partialM2, partialM, (short)0, (short)3);
    byteAddFun (partialM2, a, (short)0, (short)3);
    leftRotation(partialM2, s, partialM2);
    byteAddFun(partialM2, b, (short)0, (short)3);
    arrayCopy(partialM2, (short)0, a, (short)0, (short)4);

} //end round 1

//for round 2: a = b + ((a + G(b,c,d) + X[k] + T[i]) <<< s)
private void roundTwo (byte[] a, byte[] b, byte[] c, byte[] d, byte[] x,
    short point, byte s, byte t1, byte t2, byte t3, byte t4) {

    andFun(b, d, partialM);
    compFun(d, partialM2);
    andFun(c, partialM2, partialM2);
    orFun(partialM, partialM2, partialM2);
    arrayCopy(x, (short)point, partialM, (short)0, (short)4);
    byteAddFun(partialM2, partialM, (short)0, (short)3);
    partialM[0] = t1; partialM[1] = t2;
    partialM[2] = t3; partialM[3] = t4;
    byteAddFun (partialM2, partialM, (short)0, (short)3);
    byteAddFun (partialM2, a, (short)0, (short)3);
    leftRotation(partialM2, s, partialM2);
    byteAddFun(partialM2, b, (short)0, (short)3);

```

```

        arrayCopy(partialM2, (short)0, a, (short)0, (short)4);
    } //end round 2

//round 3: a = b + ((a + H(b,c,d) + X[k] + T[i]) <<< s)
private void roundThree (byte[] a, byte[] b, byte[] c, byte[] d, byte[] x,
        short point, byte s, byte t1, byte t2, byte t3, byte t4) {

    xorFun(b, c, partialM);
    xorFun(partialM, d, partialM2);
    arrayCopy(x, (short)point, partialM, (short)0, (short)4);
    byteAddFun(partialM2, partialM, (short)0, (short)3);
    partialM[0] = t1;    partialM[1] = t2;
    partialM[2] = t3;    partialM[3] = t4;
    byteAddFun (partialM2, partialM, (short)0, (short)3);
    byteAddFun (partialM2, a, (short)0, (short)3);
    leftRotation(partialM2, s, partialM2);
    byteAddFun(partialM2, b, (short)0, (short)3);
    arrayCopy(partialM2, (short)0, a, (short)0, (short)4);

} //end round 3

//round 4: a = b + ((a + I(b,c,d) + X[k] + T[i]) <<< s)
private void roundFour (byte[] a, byte[] b, byte[] c, byte[] d, byte[] x,
        short point, byte s, byte t1, byte t2, byte t3, byte t4) {

    compFun(d, partialM);
    orFun(a, partialM, partialM);
    xorFun(c, partialM, partialM2);
    arrayCopy(x, (short)point, partialM, (short)0, (short)4);
    byteAddFun(partialM2, partialM, (short)0, (short)3);
    partialM[0] = t1; partialM[1] = t2;
    partialM[2] = t3;    partialM[3] = t4;
    byteAddFun (partialM2, partialM, (short)0, (short)3);
    byteAddFun (partialM2, a, (short)0, (short)3);
    leftRotation(partialM2, s, partialM2);
    byteAddFun(partialM2, b, (short)0, (short)3);
    arrayCopy(partialM2, (short)0, a, (short)0, (short)4);

} //end round 4

//start array copy method
private void arrayCopy (byte[] original, short oStart,
        byte[] destine, short dStart, short length) {

    short index = 0;

    for (index = length; index > 0; --index) {
        destine[dStart] = original[oStart];
        oStart++;
        dStart++;
    }
}

```



```

} //end array copy method

//start byte add function
private void byteAddFun (byte[] a, byte[] b, short point2, short length) {

    if (length == -1)
        return;

    short oneByte = (short)((short)(a[length]&0x00FF) +
        (short)(b[length]&0x00FF) + point2);

    a[length] = (byte)(oneByte & (short)0x00FF);

    if (overFlow(oneByte))
        byteAddFun(a, b, (short)1, --length);
    else
        byteAddFun(a, b, (short)0, --length);

    return;

} //end byte add function

private void appendLength (byte[] lengthTotal, short length) {

    byte[] partialLength = new byte[5];
    short point = -1;

    while (true) {
        if (length >= 127) {
            partialLength[++point] = (byte)0x7F;
            length = (short)(length - (short)127);
        }
        else {
            partialLength[++point] = (byte)(length%((short)127));
            break;
        }
    }

    for (short index = 7; point >= 0; --point, index = 7) {
        short oneByte = (short)((short)(lengthTotal[index]&0x00ff) +
            (short)(partialLength[point]&0x00ff));
        lengthTotal[index] = (byte)(oneByte & (short)0x00FF);
        if (overFlow(oneByte))
            lengthTotal = round(lengthTotal, --index, (short)1);
    }

} //end appendLength method

//start overflow function
private boolean overFlow (short number) {

    if ((short)(number&(short)0xFF00) >= (short)0x0100)
        return true;
}

```

```

else
    return false;

} //end overflow function

//start left rotation function
private void leftRotation (byte[] array, byte shifting, byte[] answer) {

    byte one = 0x00; byte two = 0x00;
    byte three = 0x00; byte four = 0x00;
    byte pointer = 0x00; byte shifter = 0x00;
    byte oppositeShift = 0x00;
    byte current = 0x00; short next = 0x00;

    switch (shifting % (byte)8) {
        case 0: pointer = (byte)0xFF;
            break;
        case 1: pointer = (byte)0x80;
            break;
        case 2: pointer = (byte)0xC0;
            break;
        case 3: pointer = (byte)0xE0;
            break;
        case 4: pointer = (byte)0xF0;
            break;
        case 5: pointer = (byte)0xF8;
            break;
        case 6: pointer = (byte)0xFC;
            break;
        case 7: pointer = (byte)0xFE;
            break;
    }

    shifter = (byte)(shifting % (byte)8);
    oppositeShift = (byte)((byte)0x08 - shifter);

    if (shifter == 0) {
        shifter = 8;
        oppositeShift = 0;
    }

    if (shifting <= 8) {

        one = array[0]; two = array[1];
        three = array[2]; four = array[3];

    }

    else if (shifting <= 16) {

        one = array[1]; two = array[2];
        three = array[3]; four = array[0];

    }

    else if (shifting <= 24) {

```

```

one = array[2]; two = array[3];
three = array[0]; four = array[1];

}

next = (short)((four & pointer) & 0x00FF);
answer[3] = (byte)(four << shifter);
current = (byte)(next >>> oppositeShift);
next = (short)((three & pointer) & 0x00FF);
answer[2] = (byte)((three << shifter) | current);
current = (byte)(next >>> oppositeShift);
next = (short)((two & pointer) & 0x00FF);
answer[1] = (byte)((two << shifter) | current);
current = (byte)(next >>> oppositeShift);
next = (short)((one & pointer) & 0x00FF);
answer[0] = (byte)((one << shifter) | current);
current = (byte)(next >>> oppositeShift);
answer[3] = (byte)(answer[3] | (current));

} //end left Rotation method

//start round function
private byte[] round(byte[] total, short index, short point2) {

    if (index < 0)
        return total;

    short oneByte = (short)((short)(total[index]&0x00ff) + point2);
    total[index] = (byte)(oneByte & (short)0x00FF);

    if (overflow(oneByte))
        total = round(total, --index, (short)1);

    return total;

} //end round function

//start bit complementation function
private void compFun (byte[] array, byte[] answer) {

    for (short i = 0; i < (short)4; i++)
        answer[i] = (byte)~(array[i]);

} //end bit complementation function

//start bit and function
private void andFun (byte[] array1, byte[] array2, byte[] answer) {

    for (short i = 0; i < (short)4; i++)
        answer[i] = (byte)(array1[i] & array2[i]);

} //end bit and function

```

```
//start bit or function
private void orFun (byte[] array1, byte[] array2, byte[] answer) {

    for (short i = 0; i < (short)4; i++)
        answer[i] = (byte)(array1[i] | array2[i]);

} //end bit or function

//start xor function
private void xorFun (byte[] array1, byte[] array2, byte[] answer) {

    for (short i = 0; i < (short)4; i++)
        answer[i] = (byte)(array1[i] ^ array2[i]);

} //end xor function

} //end md5 class
```

Installer.java

```
/*  
    This applet resides on the JCRE by masking  
    When an applet is installed, it makes original input by hash function from CAP file  
    Hash function is included here.  
*/  
  
package installers;  
  
import javacard.framework.*;  
  
public class Installer extends Applet  
    implements InstallInterface {  
  
    //CLA byte  
    private final static byte Installer_CLA = (byte)0xB0;  
  
    //INS byte to store temp password  
    private final static byte STORE = (byte)0x01;  
    //for initialize md5  
    private final static byte INITIMD5 = (byte)0x02;  
    //for make and update md5  
    private final static byte UPDATEMD5 = (byte)0x03;  
    //for save md5  
    private final static byte GENMD5 = (byte)0x04;  
    //for display md5 password  
    private final static byte DISPLAYPASS= (byte)0x05;  
    //for display md5 hashvalue  
    private final static byte DISPLAYHASH= (byte)0x06;  
  
    //this applet instance's AID  
    private final byte[] own_aid = {0x00, 0x00, 0x00, 0x00, 0x0B, 0x03, 0x01, 0x0C, 0x06, 0x01};  
    //client(Ebank)'s AID  
    private final byte[] cAid = {0x00, 0x00, 0x00, 0x00, 0x0C, 0x03, 0x01, 0x0C, 0x06, 0x01};  
  
    //status word  
    private final static short INVALID_MESSAGE_BLOCK_LENGTH = 0x6A01;  
  
    //parameters for sharing between Ebank and Installer  
    private final byte parashare = (byte) 0xCC;  
  
    //for instance of md5  
    private MD5 md5;  
  
    //AFTER GENERATING HASH BALUE FROM EBANK CAP FILE-  
    //instance variables declaration  
    //for static hash value from installer (128 bits)  
    private final byte[] hashValue;  
    //for changeable password (128 bits)  
    private final byte[] password;  
    //for the temp password from host application  
    private byte[] tempPass;
```

```

//to making hash value
private byte[] tempMessage;

//start Installer constructor
private Installer(){

    //md5 instance
    md5 = new MD5();

    //for hash value
    hashValue = new byte[16];
    //for password
    password = new byte[16];
    //make storage for the temp-pass
    tempPass = new byte[16];
    //for message block
    tempMessage = new byte[64];

    //register this applet instance to the JCRE
    register(own_aid, (short)0, (byte)(own_aid.length));

} //end Installer constructor

//start install method
public static void install(byte[] bArray, short bOffset, byte bLength) {

    //create this applet instance
    new Installer();

} //end install method

//start process method
public void process(APDU apdu) {

    byte[] buffer = apdu.getBuffer();

    //if this apdu is select method, then return
    if(selectingApplet())
        return;

    //check the CLA bytes
    if(buffer[ISO7816.OFFSET_CLA] != Installer_CLA)
        ISOException.throwIt(ISO7816.SW_CLA_NOT_SUPPORTED);

    //check the INS byte to decide which service method to call
    switch(buffer[ISO7816.OFFSET_INS]) {
        case STORE:    store(apdu);
            return;
        case INITMD5:  initmd5(apdu);
            return;
        case UPDATMD5: updatemd5(apdu);
            return;
        case GENMD5:   genmd5(apdu);
            return;
    }
}

```

```

        case DISPLAYPASS: displaypass(apdu);
            return;
        case DISPLAYHASH: displayhash(apdu);
            return;
        default: ISOException.throwIt(ISO7816.SW_INS_NOT_SUPPORTED);
    } //end switch method

} //end process method

//start store method to store temp password from host application
private void store(APDU apdu){

    byte[] buffer = apdu.getBuffer();

    //set to data_length value
    short data_length = (short)(buffer[ISO7816.OFFSET_LC]&0xFF);

    //set to data receive mode
    apdu.setIncomingAndReceive();

    //temp buffer
    byte[] temp;

    //set to temp buffer length
    temp = new byte[data_length];

    //copy buffer from buffer[] to temp[]
    Util.arrayCopy(buffer,(short)(ISO7816.OFFSET_CDATA&0x00FF),
        temp,(short)0, (short)buffer[ISO7816.OFFSET_LC]);

    //for times value to make first password
    short length = (short)16;
    short timesValue = (short) temp[length];

    //copy buffer from temp[] to tempPass[]
    Util.arrayCopy(temp,(short)0,tempPass,(short)0, (short)16);

    //to make first password from hash value
    if(timesValue != (short)0) {
        //to make a password from hash value
        //from the hash value buffer to temp message buffer
        Util.arrayCopy(hashValue,(short)0,tempMessage,(short)0, (short)16);

        //to make initial password
        for (short i=0; i<timesValue; i++){
            //md5 initialize
            md5.md5Init();
            //md5 update
            md5.md5Update(tempMessage, (short)16);

            //md5 generate and save it
            md5.md5Gen(password);

            Util.arrayCopy(password,(short)0,tempMessage,(short)0, (short)16);
        }
    }
}

```

```

    }

} //end store method

//start initialization method to initialize md5 instance
private void initmd5(APDU apdu){

    //initialize md5
    md5.md5Init();

} //end initialize method

//start updatemd5 method to update hash value
private void updatemd5(APDU apdu){

    byte[] buffer = apdu.getBuffer();

    //set mode to receiving data
    apdu.setIncomingAndReceive();

    //set to message block length value
    short length = (short)(buffer[ISO7816.OFFSET_LC]&0x00FF);

    //check whether the length is valid or not
    if(length<1 || length>64)
        ISOException.throwIt(INVALID_MESSAGE_BLOCK_LENGTH);

    //copy data from buffer's data field to tempMessage
    Util.arrayCopy(buffer,(short)(ISO7816.OFFSET_CDATA&0x00FF),
        tempMessage,(short)0, length);

    //call md5 update method to update hash value
    md5.md5Update(tempMessage, (short)length);

} //end update method

//start generate md5 method to store hash value
private void genmd5(APDU apdu){

    //call md5 update method to update hash value
    md5.md5Update(tempMessage, (short)0);

    //save md5 hash value into password
    md5.md5Gen(hashValue);
} //end generate method

//start display method to show hash value
private void displaypass(APDU apdu) {

    byte[] buffer = apdu.getBuffer();

    //notify the JCRE that the applet has data to return
    short le = apdu.setOutgoing();

```



```

//set actual data byte's numbers
apdu.setOutgoingLength((byte)16);

apdu.sendBytesLong(password,(short)0,(short)16);

} //end display method

//start display method to show hash value
private void displayhash(APDU apdu) {

    byte[] buffer = apdu.getBuffer();

    //notify the JCRE that the applet has data to return
    short le = apdu.setOutgoing();

    //set actual data byte's numbers
    apdu.setOutgoingLength((byte)16);

    apdu.sendBytesLong(hashValue,(short)0,(short)16);

} //end display method

//start shareable interface object method
public Shareable getShareableInterfaceObject(AID clientAid, byte parameter){

    //if the client AID does not match, then return null
    if(clientAid.equals(cAid, (short)0, (byte)(cAid.length))!= true)
        return null;

    //check the parameter
    if(parameter != parashare)
        return null;

    //if client AID and parameter matched, then return this SIO
    return this;

} //end shareable interface object method

//to check password
public boolean check(){

    //to compare between pass and updated temp pass(checkupPass)
    byte[] checkupPass = new byte[16];

    //copy tempPass to checkupPass
    Util.arrayCopy(tempPass,(short)0,checkupPass,(short)0, (short)16);

    //to generate next password from the checkupPass
    //and update it to temp pass2
    Util.arrayCopy(checkupPass,(short)0,tempMessage,(short)0, (short)16);

    //md5 initialize
    md5.md5Init();

```

```

//md5 update
md5.md5Update(tempMessage, (short)16);

//md5 generate and save it
md5.md5Gen(checkupPass);

//comapre between checkupPass and password
//In case that they are not matched, then return false
if(Util.arrayCompare(checkupPass,(short)0,password,(short)0,(short)16)!= 0x00){

    //tempPass is set up to 0x00
    for(short i = 0; i<16; i++)
        tempPass[i] = 0x00;

    return false;
}
//If they are matched, update password and return true
else{

    //update password
    Util.arrayCopy(tempPass,(short)0,password,(short)0, (short)16);

    //tempPass is set up to 0x00
    for(short i = 0; i<16; i++)
        tempPass[i] = 0x00;

    return true;
}
}
} //end Installer class

```

```

//start md5 class
class MD5 {

```

```

/* The source of this MD5 algorithm is from RFC 1321 from MIT laboratory for
Computer Science and RSA data security.inc.
The original source is untimly distributed.
This MD5 is changed and adapted to the JAVA CARD */

```

```

/* Constants for MD5Transform routine */

```

```

private static final byte S11 = 7; private static final byte S12 = 12;
private static final byte S13 = 17; private static final byte S14 = 22;
private static final byte S21 = 5; private static final byte S22 = 9;
private static final byte S23 = 14; private static final byte S24 = 20;
private static final byte S31 = 4; private static final byte S32 = 11;
private static final byte S33 = 16; private static final byte S34 = 23;
private static final byte S41 = 6; private static final byte S42 = 10;
private static final byte S43 = 15; private static final byte S44 = 21;

```

```

//to append padding bits - 128bits (64 bytes)

```

```

private static final byte[] padding = {
    (byte)0x80, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
    0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
    0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,

```

```
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00};
```

```
//for four-word buffer(A,B,C,D)
//These register buffers are used for computing the message digest
private byte[] wordA = new byte[4];
private byte[] wordB = new byte[4];
private byte[] wordC = new byte[4];
private byte[] wordD = new byte[4];

//temp four-word buffer for increment each of the four registers by the value
private byte[] wordAA = new byte[4];
private byte[] wordBB = new byte[4];
private byte[] wordCC = new byte[4];
private byte[] wordDD = new byte[4];

//for before padding initialize it to false
private boolean beforePadding = false;

//a 64-element table
//This is constructed from the sine function
private byte[] sixtyFourETable = new byte[64];

//for total message length
private byte[] totalML = new byte[8];

//for partial message
private byte[] partialM = new byte[4];
private byte[] partialM2 = new byte[4];

//start md5Init method for MD5 initialization
//Begins an MD5 operation
void md5Init () {

    //wordA = 0x67452301
    wordA[0] = (byte)0x67; wordA[1] = (byte)0x45;
    wordA[2] = (byte)0x23; wordA[3] = (byte)0x01;
    //wordB = 0xefcdab89
    wordB[0] = (byte)0xEF; wordB[1] = (byte)0xCD;
    wordB[2] = (byte)0xAB; wordB[3] = (byte)0x89;
    //wordC = 0x98badcfe
    wordC[0] = (byte)0x98; wordC[1] = (byte)0xBA;
    wordC[2] = (byte)0xDC; wordC[3] = (byte)0xFE;
    //wordD = 0x10325476
    wordD[0] = (byte)0x10; wordD[1] = (byte)0x32;
    wordD[2] = (byte)0x54; wordD[3] = (byte)0x76;

    //initialize total message length buffer
    for (short i = 0; i<8; i++)
```

```

        totalML[i] = 0x00;

//initialize beforePadding as a false
beforePadding = false;

} //end md5Init method

void md5Update(byte[] buffer, short length) {

//by input length - initial case is 0
short processByCase = 0;

switch(length) {

//In case each data block bits are 512 bits
case 64: processByCase = 1;
        break;

//after processing all data blocks
case 0:
    if (beforePadding == false){
        processByCase = -1;
        break;
    }
    else
        return;

//for the last data block
default:
    processByCase = 3;
    beforePadding = true;
}

//to add message length
appendLength(totalML, length);

//copy from Array wordA to paste Array wordAA
arrayCopy(wordA, (short)0, wordAA, (short)0, (short)4);

//copy from Array wordB to paste Array wordBB
arrayCopy(wordB, (short)0, wordBB, (short)0, (short)4);

//copy from Array wordC to paste Array wordCC
arrayCopy(wordC, (short)0, wordCC, (short)0, (short)4);

//copy from Array wordD to paste Array wordDD
arrayCopy(wordD, (short)0, wordDD, (short)0, (short)4);

do{

//making 64-element table T[1...64]
switch (processByCase) {

//padding 448 bits (100...0) and save it with 64bits (total message)
case -1:

```

```

arrayCopy(padding, (short)0, sixtyFourETable, (short)0, (short)56);
arrayCopy(totalML, (short)0, sixtyFourETable, (short)56, (short)8);
processByCase = 0;
break;

//save 512 bit message
case 1:
arrayCopy(buffer, (short)0, sixtyFourETable, (short)0, (short)64);
processByCase = 0;
break;

// padding 448 bits (000...0) and save it with 64bits (tottal message)
case 2:
arrayCopy(padding, (short)8, sixtyFourETable, (short)0, (short)56);
arrayCopy(totalML, (short)0, sixtyFourETable, (short)56, (short)8);
processByCase = 0;
break;

//to make the last input(8bits~512bits) 512bits and make it 128 bits
case 3:
if(length < 56) {
short required_pad = (short)((short)56 - length);
arrayCopy(buffer, (short)0, sixtyFourETable, (short)0, length);
arrayCopy(padding, (short)0, sixtyFourETable, (short)length, required_pad);
arrayCopy(totalML, (short)0, sixtyFourETable, (short)56, (short)8);
processByCase = 0;
}
else {
short required_pad = (short)((short)64 - length);
arrayCopy(buffer, (short)0, sixtyFourETable, (short)0, length);
arrayCopy(padding, (short)0, sixtyFourETable, (short)length, required_pad);
processByCase = 2;
}
break;
}

/* call methods - Round 1,2,3,4.
each method is called by total 16*/

//round 1
roundOne (wordA, wordB, wordC, wordD, sixtyFourETable, (short)0, S11, (byte)0xD7,
(byte)0x6A, (byte)0xA4, (byte)0x78);
roundOne (wordD, wordA, wordB, wordC, sixtyFourETable, (short)4, S12, (byte)0xE8,
(byte)0xC7, (byte)0xB7, (byte)0x56);
roundOne (wordC, wordD, wordA, wordB, sixtyFourETable, (short)8, S13, (byte)0x24,
(byte)0x20, (byte)0x70, (byte)0xDB);
roundOne (wordB, wordC, wordD, wordA, sixtyFourETable, (short)12, S14, (byte)0xC1,
(byte)0xBD, (byte)0xCE, (byte)0xEE);
roundOne (wordA, wordB, wordC, wordD, sixtyFourETable, (short)16, S11, (byte)0xF5,
(byte)0x7C, (byte)0x0F, (byte)0xAF);
roundOne (wordD, wordA, wordB, wordC, sixtyFourETable, (short)20, S12, (byte)0x47,
(byte)0x87, (byte)0xC6, (byte)0x2A);
roundOne (wordC, wordD, wordA, wordB, sixtyFourETable, (short)24, S13, (byte)0xA8,
(byte)0x30, (byte)0x46, (byte)0x13);
roundOne (wordB, wordD, wordD, wordA, sixtyFourETable, (short)28, S14, (byte)0xFD,

```

```

        (byte)0x46, (byte)0x95, (byte)0x01);
roundOne (wordA, wordB, wordC, wordD, sixtyFourETable, (short)32, S11, (byte)0x69,
        (byte)0x80, (byte)0x98, (byte)0xD8);
roundOne (wordD, wordA, wordB, wordC, sixtyFourETable, (short)36, S12, (byte)0x8B,
        (byte)0x44, (byte)0xF7, (byte)0xAF);
roundOne (wordC, wordD, wordA, wordB, sixtyFourETable, (short)40, S13, (byte)0xFF,
        (byte)0xFF, (byte)0x5B, (byte)0xB1);
roundOne (wordB, wordC, wordD, wordA, sixtyFourETable, (short)44, S14, (byte)0x89,
        (byte)0x5C, (byte)0xD7, (byte)0xBE);
roundOne (wordA, wordB, wordC, wordD, sixtyFourETable, (short)48, S11, (byte)0x6B,
        (byte)0x90, (byte)0x11, (byte)0x22);
roundOne (wordD, wordA, wordB, wordC, sixtyFourETable, (short)52, S12, (byte)0xFD,
        (byte)0x98, (byte)0x71, (byte)0x93);
roundOne (wordC, wordD, wordA, wordB, sixtyFourETable, (short)56, S13, (byte)0xA6,
        (byte)0x79, (byte)0x43, (byte)0x8E);
roundOne (wordB, wordC, wordD, wordA, sixtyFourETable, (short)60, S14, (byte)0x49,
        (byte)0xB4, (byte)0x08, (byte)0x21);

//round 2
roundTwo (wordA, wordB, wordC, wordD, sixtyFourETable, (short)4, S21, (byte)0xF6,
        (byte)0x1E, (byte)0x25, (byte)0x62);
roundTwo (wordD, wordA, wordB, wordC, sixtyFourETable, (short)24, S22, (byte)0xC0,
        (byte)0x40, (byte)0xB3, (byte)0x40);
roundTwo (wordC, wordD, wordA, wordB, sixtyFourETable, (short)44, S23, (byte)0x26,
        (byte)0x5E, (byte)0x5A, (byte)0x51);
roundTwo (wordB, wordC, wordD, wordA, sixtyFourETable, (short)0, S24, (byte)0xE9,
        (byte)0xB6, (byte)0xC7, (byte)0xAA);
roundTwo (wordA, wordB, wordC, wordD, sixtyFourETable, (short)20, S21, (byte)0xD6,
        (byte)0x2F, (byte)0x10, (byte)0x5D);
roundTwo (wordD, wordA, wordB, wordC, sixtyFourETable, (short)40, S22, (byte)0x2,
        (byte)0x44, (byte)0x14, (byte)0x53);
roundTwo (wordC, wordD, wordA, wordB, sixtyFourETable, (short)60, S23, (byte)0xD8,
        (byte)0xA1, (byte)0xE6, (byte)0x81);
roundTwo (wordB, wordD, wordD, wordA, sixtyFourETable, (short)24, S24, (byte)0xE7,
        (byte)0xD3, (byte)0xFB, (byte)0xC8);
roundTwo (wordA, wordB, wordC, wordD, sixtyFourETable, (short)36, S21, (byte)0x21,
        (byte)0xE1, (byte)0xCD, (byte)0xE6);
roundTwo (wordD, wordA, wordB, wordC, sixtyFourETable, (short)56, S22, (byte)0xC3,
        (byte)0x37, (byte)0x07, (byte)0xD6);
roundTwo (wordC, wordD, wordA, wordB, sixtyFourETable, (short)12, S23, (byte)0xF4,
        (byte)0xD5, (byte)0x0D, (byte)0x87);
roundTwo (wordB, wordC, wordD, wordA, sixtyFourETable, (short)32, S24, (byte)0x45,
        (byte)0x5A, (byte)0x14, (byte)0xED);
roundTwo (wordA, wordB, wordC, wordD, sixtyFourETable, (short)52, S21, (byte)0xA9,
        (byte)0xE3, (byte)0xE9, (byte)0x05);
roundTwo (wordD, wordA, wordB, wordC, sixtyFourETable, (short)8, S22, (byte)0xFC,
        (byte)0xEF, (byte)0xA3, (byte)0xF8);
roundTwo (wordC, wordD, wordA, wordB, sixtyFourETable, (short)28, S23, (byte)0x67,
        (byte)0x6F, (byte)0x02, (byte)0xD9);
roundTwo (wordB, wordC, wordD, wordA, sixtyFourETable, (short)48, S24, (byte)0x8D,
        (byte)0x2A, (byte)0x4C, (byte)0x8A);

//round 3
roundThree(wordA, wordB, wordC, wordD, sixtyFourETable, (short)20, S31, (byte)0xFF,
        (byte)0xFA, (byte)0x39, (byte)0x42);
roundThree(wordD, wordA, wordB, wordC, sixtyFourETable, (short)32, S32, (byte)0x87,

```

```

        (byte)0x71, (byte)0xF6, (byte)0x81);
roundThree(wordC, wordD, wordA, wordB, sixtyFourETable, (short)44, S33, (byte)0x6D,
        (byte)0x9D, (byte)0x61, (byte)0x22);
roundThree(wordB, wordC, wordD, wordA, sixtyFourETable, (short)56, S34, (byte)0xFD,
        (byte)0xE5, (byte)0x38, (byte)0x0C);
roundThree(wordA, wordB, wordC, wordD, sixtyFourETable, (short)4, S31, (byte)0xA4,
        (byte)0xBE, (byte)0xEA, (byte)0x44);
roundThree(wordD, wordA, wordB, wordC, sixtyFourETable, (short)16, S32, (byte)0x4B,
        (byte)0xDE, (byte)0xCF, (byte)0xA9);
roundThree(wordC, wordD, wordA, wordB, sixtyFourETable, (short)28, S33, (byte)0xF6,
        (byte)0xBB, (byte)0x4B, (byte)0x80);
roundThree(wordB, wordD, wordD, wordA, sixtyFourETable, (short)40, S34, (byte)0xBE,
        byte)0xBF, (byte)0xBC, (byte)0x70);
roundThree(wordA, wordB, wordC, wordD, sixtyFourETable, (short)52, S31, (byte)0x28,
        (byte)0x9B, (byte)0x7E, (byte)0xC6);
roundThree(wordD, wordA, wordB, wordC, sixtyFourETable, (short)0, S32, (byte)0xEA,
        (byte)0xA1, (byte)0x27, (byte)0xFA);
roundThree(wordC, wordD, wordA, wordB, sixtyFourETable, (short)12, S33, (byte)0xD4,
        (byte)0xEF, (byte)0x30, (byte)0x85);
roundThree(wordB, wordC, wordD, wordA, sixtyFourETable, (short)24, S34, (byte)0x4,
        (byte)0x88, (byte)0x1D, (byte)0x05);
roundThree(wordA, wordB, wordC, wordD, sixtyFourETable, (short)36, S31, (byte)0xD9,
        (byte)0xD4, (byte)0xD0, (byte)0x39);
roundThree(wordD, wordA, wordB, wordC, sixtyFourETable, (short)48, S32, (byte)0xE6,
        (byte)0xDB, (byte)0x99, (byte)0xE5);
roundThree(wordC, wordD, wordA, wordB, sixtyFourETable, (short)60, S33, (byte)0x1F,
        (byte)0xA2, (byte)0x7C, (byte)0xF8);
roundThree(wordB, wordC, wordD, wordA, sixtyFourETable, (short)8, S34, (byte)0xC4,
        (byte)0xAC, (byte)0x56, (byte)0x65);

```

```
//round 4
```

```

roundFour (wordA, wordB, wordC, wordD, sixtyFourETable, (short)0, S41, (byte)0xF4,
        (byte)0x29, (byte)0x22, (byte)0x44);
roundFour (wordD, wordA, wordB, wordC, sixtyFourETable, (short)28, S42, (byte)0x43,
        (byte)0x2A, (byte)0xFF, (byte)0x97);
roundFour (wordC, wordD, wordA, wordB, sixtyFourETable, (short)56, S43, (byte)0xAB,
        (byte)0x94, (byte)0x23, (byte)0xA7);
roundFour (wordB, wordC, wordD, wordA, sixtyFourETable, (short)20, S44, (byte)0xFC,
        (byte)0x93, (byte)0xA0, (byte)0x39);
roundFour (wordA, wordB, wordC, wordD, sixtyFourETable, (short)48, S41, (byte)0x65,
        (byte)0x5B, (byte)0x59, (byte)0xC3);
roundFour (wordD, wordA, wordB, wordC, sixtyFourETable, (short)12, S42, (byte)0x8F,
        (byte)0x0C, (byte)0xCC, (byte)0x92);
roundFour (wordC, wordD, wordA, wordB, sixtyFourETable, (short)40, S43, (byte)0xFF,
        (byte)0xEF, (byte)0xF4, (byte)0x7D);
roundFour (wordB, wordD, wordD, wordA, sixtyFourETable, (short)4, S44, (byte)0x85,
        (byte)0x84, (byte)0x5D, (byte)0xD1);
roundFour (wordA, wordB, wordC, wordD, sixtyFourETable, (short)32, S41, (byte)0x6F,
        (byte)0xA8, (byte)0x7E, (byte)0x4F);
roundFour (wordD, wordA, wordB, wordC, sixtyFourETable, (short)60, S42, (byte)0xFE,
        (byte)0x2C, (byte)0xE6, (byte)0xE0);
roundFour (wordC, wordD, wordA, wordB, sixtyFourETable, (short)24, S43, (byte)0xA3,
        (byte)0x01, (byte)0x43, (byte)0x14);
roundFour (wordB, wordC, wordD, wordA, sixtyFourETable, (short)52, S44, (byte)0x4E,
        (byte)0x08, (byte)0x11, (byte)0xA1);
roundFour (wordA, wordB, wordC, wordD, sixtyFourETable, (short)16, S41, (byte)0xF7,

```

```

        (byte)0x53, (byte)0x7E, (byte)0x82);
roundFour (wordD, wordA, wordB, wordC, sixtyFourETable, (short)44, S42, (byte)0xBD,
        (byte)0x3A, (byte)0xF2, (byte)0x35);
roundFour (wordC, wordD, wordA, wordB, sixtyFourETable, (short)8, S43, (byte)0x2A,
        (byte)0xD7, (byte)0xD2, (byte)0xBB);
roundFour (wordB, wordC, wordD, wordA, sixtyFourETable, (short)36, S44, (byte)0xEB,
        (byte)0x86, (byte)0xD3, (byte)0x91);

    /* save A as AA, B as BB, C as CC, and D as DD */
    byteAddFun(wordA, wordAA, (short)0, (short)3);
    byteAddFun(wordB, wordBB, (short)0, (short)3);
    byteAddFun(wordC, wordCC, (short)0, (short)3);
    byteAddFun(wordD, wordDD, (short)0, (short)3);

} while (processByCase != 0);

} //end md5 update method

//start md5Gen method
void md5Gen (byte[] md) {

    //generate output A
    for (short i = 0, j = 3; j >= 0; ++i, --j)
        md[i] = wordA[j];

    //generate output B
    for (short i = 4, j = 3; j >= 0; ++i, --j)
        md[i] = wordB[j];

    //generate output C
    for (short i = 8, j = 3; j >= 0; ++i, --j)
        md[i] = wordC[j];

    //generate output D
    for (short i = 12, j = 3; j >= 0; ++i, --j)
        md[i] = wordD[j];

} //end md5Gen method

//for round 1: a = b + ((a + F(b,c,d) + X[k] + T[i]) <<< s)
private void roundOne (byte[] a, byte[] b, byte[] c, byte[] d, byte[] x,
    hort point, byte s, byte t1, byte t2, byte t3, byte t4) {

    andFun(b, c, partialM);
    compFun(b, partialM2);
    andFun(partialM2, d, partialM2);
    orFun(partialM, partialM2, partialM2);
    arrayCopy(x, (short)point, partialM, (short)0, (short)4);
    byteAddFun(partialM2, partialM, (short)0, (short)3);
    partialM[0] = t1; partialM[1] = t2;
    partialM[2] = t3; partialM[3] = t4;
    byteAddFun (partialM2, partialM, (short)0, (short)3);
    byteAddFun (partialM2, a, (short)0, (short)3);

```



```

    leftRotation(partialM2, s, partialM2);
    byteAddFun(partialM2, b, (short)0, (short)3);
    arrayCopy(partialM2, (short)0, a, (short)0, (short)4);

} //end round 1

//for round 2: a = b + ((a + G(b,c,d) + X[k] + T[i]) <<< s)
private void roundTwo (byte[] a, byte[] b, byte[] c, byte[] d, byte[] x,
    short point, byte s, byte t1, byte t2, byte t3, byte t4) {

    andFun(b, d, partialM);
    compFun(d, partialM2);
    andFun(c, partialM2, partialM2);
    orFun(partialM, partialM2, partialM2);
    arrayCopy(x, (short)point, partialM, (short)0, (short)4);
    byteAddFun(partialM2, partialM, (short)0, (short)3);
    partialM[0] = t1; partialM[1] = t2;
    partialM[2] = t3; partialM[3] = t4;
    byteAddFun (partialM2, partialM, (short)0, (short)3);
    byteAddFun (partialM2, a, (short)0, (short)3);
    leftRotation(partialM2, s, partialM2);
    byteAddFun(partialM2, b, (short)0, (short)3);
    arrayCopy(partialM2, (short)0, a, (short)0, (short)4);

} //end round 2

//round 3: a = b + ((a + H(b,c,d) + X[k] + T[i]) <<< s)
private void roundThree (byte[] a, byte[] b, byte[] c, byte[] d, byte[] x,
    short point, byte s, byte t1, byte t2, byte t3, byte t4) {

    xorFun(b, c, partialM);
    xorFun(partialM, d, partialM2);
    arrayCopy(x, (short)point, partialM, (short)0, (short)4);
    byteAddFun(partialM2, partialM, (short)0, (short)3);
    partialM[0] = t1;
    partialM[1] = t2;
    partialM[2] = t3;
    partialM[3] = t4;
    byteAddFun (partialM2, partialM, (short)0, (short)3);
    byteAddFun (partialM2, a, (short)0, (short)3);
    leftRotation(partialM2, s, partialM2);
    byteAddFun(partialM2, b, (short)0, (short)3);
    arrayCopy(partialM2, (short)0, a, (short)0, (short)4);

} //end round 3

//round 4: a = b + ((a + I(b,c,d) + X[k] + T[i]) <<< s)
private void roundFour (byte[] a, byte[] b, byte[] c, byte[] d, byte[] x,
    short point, byte s, byte t1, byte t2, byte t3, byte t4) {

    compFun(d, partialM);
    orFun(a, partialM, partialM);
    xorFun(c, partialM, partialM2);

```

```

arrayCopy(x, (short)point, partialM, (short)0, (short)4);
byteAddFun(partialM2, partialM, (short)0, (short)3);
partialM[0] = t1;
partialM[1] = t2;
partialM[2] = t3;
partialM[3] = t4;
byteAddFun (partialM2, partialM, (short)0, (short)3);
byteAddFun (partialM2, a, (short)0, (short)3);
leftRotation(partialM2, s, partialM2);
byteAddFun(partialM2, b, (short)0, (short)3);
arrayCopy(partialM2, (short)0, a, (short)0, (short)4);

} //end round 4

//start array copy method
private void arrayCopy (byte[] original, short oStart,
byte[] destine, short dStart, short length) {

    short index = 0;

    for (index = length; index > 0; --index) {
        destine[dStart] = original[oStart];
        oStart++;
        dStart++;
    }

} //end array copy method

//start byte add function
private void byteAddFun (byte[] a, byte[] b, short point2, short length) {

    if (length == -1)
        return;

    short oneByte = (short)((short)(a[length]&0x00FF) +
        (short)(b[length]&0x00FF) + point2);

    a[length] = (byte)(oneByte & (short)0x00FF);

    if (overflow(oneByte))
        byteAddFun(a, b, (short)1, --length);
    else
        byteAddFun(a, b, (short)0, --length);

    return;

} //end byte add function

private void appendLength (byte[] lengthTotal, short length) {

    byte[] partialLength = new byte[5];
    short point = -1;

```

```

while (true) {
    if (length >= 127) {
        partialLength[++point] = (byte)0x7F;
        length = (short)(length - (short)127);
    }
    else {
        partialLength[++point] = (byte)(length%((short)127));
        break;
    }
}

for (short index = 7; point >= 0; --point, index = 7) {
    short oneByte = (short)((short)(lengthTotal[index]&0x00ff) +
        (short)(partialLength[point]&0x00ff));
    lengthTotal[index] = (byte)(oneByte & (short)0x00FF);
    if (overFlow(oneByte))
        lengthTotal = round(lengthTotal, --index, (short)1);
}

} //end appendLength method

//start overflow function
private boolean overFlow (short number) {

    if ((short)(number&(short)0xFF00) >= (short)0x0100)
        return true;
    else
        return false;
} //end overflow function

//start left rotation function
private void leftRotation (byte[] array, byte shifting, byte[] answer) {

    byte one = 0x00; byte two = 0x00;
    byte three = 0x00; byte four = 0x00;
    byte pointer = 0x00; byte shifter = 0x00;
    byte oppositeShift = 0x00;
    byte current = 0x00; short next = 0x00;

    switch (shifting % (byte)8) {
        case 0: pointer = (byte)0xFF;
            break;
        case 1: pointer = (byte)0x80;
            break;
        case 2: pointer = (byte)0xC0;
            break;
        case 3: pointer = (byte)0xE0;
            break;
        case 4: pointer = (byte)0xF0;
            break;
        case 5: pointer = (byte)0xF8;
            break;
        case 6: pointer = (byte)0xFC;
            break;
    }
}

```

```

        break;
    case 7: pointer = (byte)0xFE;
        break;
    }

    shifter = (byte)(shifting % (byte)8);
    oppositeShift = (byte)((byte)0x08 - shifter);

    if (shifter == 0) {
        shifter = 8;
        oppositeShift = 0;
    }

    if (shifting <= 8) {

        one = array[0]; two = array[1];
        three = array[2]; four = array[3];

    }

    else if (shifting <= 16) {

        one = array[1]; two = array[2];
        three = array[3]; four = array[0];

    }

    else if (shifting <= 24) {

        one = array[2]; two = array[3];
        three = array[0]; four = array[1];

    }

    next = (short)((four & pointer) & 0x00FF);
    answer[3] = (byte)(four << shifter);
    current = (byte)(next >>> oppositeShift);
    next = (short)((three & pointer) & 0x00FF);
    answer[2] = (byte)((three << shifter) | current);
    current = (byte)(next >>> oppositeShift);
    next = (short)((two & pointer) & 0x00FF);
    answer[1] = (byte)((two << shifter) | current);
    current = (byte)(next >>> oppositeShift);
    next = (short)((one & pointer) & 0x00FF);
    answer[0] = (byte)((one << shifter) | current);
    current = (byte)(next >>> oppositeShift);
    answer[3] = (byte)(answer[3] | (current));

} //end left Rotation method

//start round function
private byte[] round(byte[] total, short index, short point2) {

    if (index < 0)
        return total;

```

```

short oneByte = (short)((short)(total[index]&0x00ff) + point2);
total[index] = (byte)(oneByte & (short)0x00FF);

if (overFlow(oneByte))
total = round(total, --index, (short)1);

return total;
} //end round function

//start bit complementation function
private void compFun (byte[] array, byte[] answer) {

    for (short i = 0; i < (short)4; i++)
        answer[i] = (byte)~(array[i]);

} //end bit complementation function

//start bit and function
private void andFun (byte[] array1, byte[] array2, byte[] answer) {

    for (short i = 0; i < (short)4; i++)
        answer[i] = (byte)(array1[i] & array2[i]);

} //end bit and function

//start bit or function
private void orFun (byte[] array1, byte[] array2, byte[] answer) {

    for (short i = 0; i < (short)4; i++)
        answer[i] = (byte)(array1[i] | array2[i]);

} //end bit or function

//start xor function
private void xorFun (byte[] array1, byte[] array2, byte[] answer) {

    for (short i = 0; i < (short)4; i++)
        answer[i] = (byte)(array1[i] ^ array2[i]);

} //end xor function

} //end md5 class

```

HashMaker.java

```
/*
*****
We can check the password from the input by this applet.
This is made to help on simulation.
*****
*/

import java.io.*;
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;

//start main class hash maker
class HashMaker extends JFrame {

//to declare variables
//for simulation window, buttons, and text fields
private JButton selection;
private JButton valueMaker;
private JButton nextMaker;
private JLabel name;
private JLabel name2;
private JTextField boundary;
private JTextField boundary2;

//for cap file choose
private JFileChooser fileChooser;

//to read cap file
private File capFile;
private File dumpFile;
private File batch;

//to declare instance of md5 class
private MD5 md5;

//to generate hash value
private byte[] data = new byte[64];
private byte[] hv = new byte[16];

//start HashMaker constructor
HashMaker(){

//to set simulation window and color
getContentPane().setLayout(null);
getContentPane().setBackground(Color.blue);

//set window title
setTitle("Hash Value Maker");
setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

//for choosing cap file
selection = new JButton("What Cap File ?");
```

```

selection.setBackground(Color.yellow);
selection.setBounds(25,20,275,30);

//for button to make hash value
valueMaker = new JButton("Make Hash Value");
valueMaker.setBackground(Color.yellow);
valueMaker.setBounds(25, 65, 275, 30);

//to show generated hash value from cap file
name = new JLabel("Made Hash Value");
name.setBounds(25, 105, 240, 30);
boundary = new JTextField();
boundary.setEditable(false);
boundary.setBounds(25, 135, 275, 25);

//to make next hash value from the pre-made hash value
nextMaker = new JButton("Make next Hash Value");
nextMaker.setBackground(Color.yellow);
nextMaker.setBounds(25, 200, 275, 30);

////to show generated hash value from pre-made hash value
name2 = new JLabel("Made Hash Value");
name2.setBounds(25, 240, 240, 30);
boundary2 = new JTextField();
boundary2.setEditable(false);
boundary2.setBounds(25, 270, 275, 25);

//for cap file chooser
fileChooser = new JFileChooser();

//make instance of MD5 class
md5 = new MD5();

// select a CAP file
selection.addActionListener(new ActionListener() {
    public void actionPerformed (ActionEvent e) {

        //initialize text field
        boundary.setText(null);

        //to make directroy for choosing cap file
        fileChooser = new JFileChooser("c:\\defense\\terminal\\electrobank\\javacard");
        int returnVal = fileChooser.showOpenDialog(HashMaker.this);

        //choose cap file
        if(returnVal == JFileChooser.APPROVE_OPTION){
            capFile = fileChooser.getSelectedFile();
        }
        else{
            //if it is not a valid cap file, then shows error message
            JOptionPane.showMessageDialog (HashMaker.this,
                "It's not a valid cap file", "ERROR",JOptionPane.ERROR_MESSAGE);
            capFile = null;
            return;
        }
    }
}

```

```

//if it is not a valid cap file, then shows error message
String temp = capFile.getName();
if (temp.indexOf(".cap") == -1){
    JOptionPane.showMessageDialog(HashMaker.this,
        "It's not a valid cap file", "ERROR",JOptionPane.ERROR_MESSAGE);
    capFile = null;
}
});

/* This method is for making Hash Value from the valid CAP file
first, choose the valid cap file
second, read this file line by line until 64 bytes
Third, sends this reading bytes to the md5 update method
Fourth, repeat third step until meeting the last line
Fifth, after processed the last line
sends the vlaue that it's updated until now
to md5 hash value making method */
valueMaker.addActionListener(new ActionListener(){
public void actionPerformed (ActionEvent e){
    try{

        //First step
        batch = new File("dump.bat");
        BufferedWriter out = new BufferedWriter(new FileWriter(batch));
        out.write("capdump " + capFile + "\n");
        out.close();
        Process child = Runtime.getRuntime().exec("dump.bat");
        dumpFile = new File("capdump");
        BufferedReader in = new BufferedReader(new InputStreamReader(child.getInputStream()));
        BufferedWriter out2 = new BufferedWriter(new FileWriter(dumpFile));
        String input;
        int control = 0;
        while ((input = in.readLine()) != null){
            if(control < 2){
                control++;
                continue;
            }
            out2.write(input + "\n");
        }
        out2.close();
    }catch (Exception ee){
    }finally{
        if(dumpFile.length() == 0){
            batch.delete();
            batch = null;
            dumpFile.delete();
            umpFile = null;
            JOptionPane.showMessageDialog
            (HashMaker.this, "You have to select a valid CAP file",
            "ERROR",JOptionPane.ERROR_MESSAGE);
            capFile = null;
            return;
        }
    }

    //for read line byte length
    short length = 0;

```



```

//to show made hash value
String message_digest = new String();

//to initialize the md5 instance
md5.md5Init();

//second and third step
try{
    BufferedInputStream bufferin = new BufferedInputStream
        (new FileInputStream(dumpFile));
    while (true){
        length = (short)(bufferin.read(data));
        if (length == -1)
            break;
        md5.md5Update(data, length);
    }
    bufferin.close();
    //third step
    md5.md5Update(data, (short)0);
    //fourth step
    md5.md5Gen(hv);
    String temp;
    //to show a made hash value
    for (int i = 0; i < 16; i++){
        temp = Integer.toString(hv[i]&0x000000FF, 16);
        if(temp.length() == 1)
            temp = "0" + temp;
        if (i!=15)
            message_digest = message_digest + temp + ":";
        else
            message_digest = message_digest + temp;
    }
    boundary.setText(message_digest);
    batch.delete();
    batch = null;
    dumpFile.delete();
    dumpFile = null;
} catch (FileNotFoundException ee){
} catch (IOException eee){}
}});

//This method for making hash value from
//original hash value befor made from the cap file
nextMaker.addActionListener(new ActionListener(){
public void actionPerformed (ActionEvent e){
if(boundary.getText().length() == 0){
//make a hash value from the cap file first
JOptionPane.showMessageDialog(HashMaker.this,
    "Must get a hash value of a CAP file first!",
    "ERROR",JOptionPane.ERROR_MESSAGE);
return;
}
boundary2.setText(null);
short length = 0;
//to show the hash value
String message_digest = new String();

```

```

//for hash value generation
for(byte i = 0x00; i<0x10; i++)
    data[i] = hv[i];

//md5 initialize, update, and generation
md5.md5Init();
md5.md5Update(data, (short)16);
md5.md5Gen(hv);

//to show the generated hash value
String temp;
for (int i = 0; i<16; i++){
    temp = Integer.toString(hv[i]&0x000000FF, 16);
    if (temp.length() == 1)
        temp = "0" + temp;
    if (i != 15)
        message_digest = message_digest + temp + ":";
    else
        message_digest = message_digest + temp;
}
boundary2.setText(message_digest);
}});

//add these to simulation window
getContentPane().add(selection);
getContentPane().add(valueMaker);
getContentPane().add(nextMaker);
getContentPane().add(name);
getContentPane().add(name2);
getContentPane().add(boundary);
getContentPane().add(boundary2);
} //end constructor

//start main method
public static void main (String[] args){

    //call the constructor
    JFrame mainWindow = new HashMaker();

    //to set simulation window boundary
    mainWindow.setBounds(0, 0, 340, 350);
    mainWindow.setVisible(true);

} //end main method

} //end start main class hash maker

//start md5 class
class MD5 {

    /* The source of this MD5 algorithm is from RFC 1321 from MIT laboratory for
    Computer Science and RSA data security.inc.

```

The original source is untimely distributed.

This MD5 is changed and adapted to the JAVA CARD */

/* Constants for MD5Transform routine */

```
private static final byte S11 = 7; private static final byte S12 = 12;
private static final byte S13 = 17; private static final byte S14 = 22;
private static final byte S21 = 5; private static final byte S22 = 9;
private static final byte S23 = 14; private static final byte S24 = 20;
private static final byte S31 = 4; private static final byte S32 = 11;
private static final byte S33 = 16; private static final byte S34 = 23;
private static final byte S41 = 6; private static final byte S42 = 10;
private static final byte S43 = 15; private static final byte S44 = 21;
```

//to append padding bits - 128bits (64 bytes)

```
private static final byte[] padding = {
    (byte)0x80, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
    0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
    0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
    0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
    0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
    0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00};
```

//for four-word buffer(A,B,C,D)

//These register buffers are used for computing the message digest

```
private byte[] wordA = new byte[4];
private byte[] wordB = new byte[4];
private byte[] wordC = new byte[4];
private byte[] wordD = new byte[4];
```

//temp four-word buffer for increment each of the four registers by the value

```
private byte[] wordAA = new byte[4];
private byte[] wordBB = new byte[4];
private byte[] wordCC = new byte[4];
private byte[] wordDD = new byte[4];
```

//for before padding initialize it to false

```
private boolean beforePadding = false;
```

//a 64-element table

//This is constructed from the sine function

```
private byte[] sixtyFourETable = new byte[64];
```

//for total message length

```
private byte[] totalML = new byte[8];
```

//for partial message

```
private byte[] partialM = new byte[4];
private byte[] partialM2 = new byte[4];
```

//start md5Init method for MD5 initialization

```

//Begins an MD5 operation
void md5Init () {

    //wordA = 0x67452301
    wordA[0] = (byte)0x67; wordA[1] = (byte)0x45;
    wordA[2] = (byte)0x23; wordA[3] = (byte)0x01;
    //wordB = 0xefcdab89
    wordB[0] = (byte)0xEF; wordB[1] = (byte)0xCD;
    wordB[2] = (byte)0xAB; wordB[3] = (byte)0x89;
    //wordC = 0x98badcfe
    wordC[0] = (byte)0x98; wordC[1] = (byte)0xBA;
    wordC[2] = (byte)0xDC; wordC[3] = (byte)0xFE;
    //wordD = 0x10325476
    wordD[0] = (byte)0x10; wordD[1] = (byte)0x32;
    wordD[2] = (byte)0x54; wordD[3] = (byte)0x76;

    //initialize total message length buffer
    for (short i = 0; i<8; i++)
        totalML[i] = 0x00;

    //initialize beforePadding as a false
    beforePadding = false;

} //end md5Init method

void md5Update(byte[] buffer, short length) {

    //by input length - initial case is 0
    short processByCase = 0;

    switch(length) {

        //In case each data block bits are 512 bits
        case 64: processByCase = 1;
                break;

        //after processing all data blocks
        case 0:
            if (beforePadding == false){
                processByCase = -1;
                break;
            }
            else
                return;

        //for the last data block
        default:
            processByCase = 3;
            beforePadding = true;
    }

    //to add message length
    appendLength(totalML, length);

    //copy from Array wordA to paste Array wordAA

```

```

arrayCopy(wordA, (short)0, wordAA, (short)0, (short)4);

//copy from Array wordB to paste Array wordBB
arrayCopy(wordB, (short)0, wordBB, (short)0, (short)4);

//copy from Array wordC to paste Array wordCC
arrayCopy(wordC, (short)0, wordCC, (short)0, (short)4);

//copy from Array wordD to paste Array wordDD
arrayCopy(wordD, (short)0, wordDD, (short)0, (short)4);

do{

    //making 64-element table T[1...64]
    switch (processByCase) {

        //padding 448 bits (100...0) and save it with 64bits (total message)
        case -1:
            arrayCopy(padding, (short)0, sixtyFourETable, (short)0, (short)56);
            arrayCopy(totalML, (short)0, sixtyFourETable, (short)56, (short)8);
            processByCase = 0;
            break;

        //save 512 bit message
        case 1:
            arrayCopy(buffer, (short)0, sixtyFourETable, (short)0, (short)64);
            processByCase = 0;
            break;

        // padding 448 bits (000...0) and save it with 64bits (total message)
        case 2:
            arrayCopy(padding, (short)8, sixtyFourETable, (short)0, (short)56);
            arrayCopy(totalML, (short)0, sixtyFourETable, (short)56, (short)8);
            processByCase = 0;
            break;

        //to make the last input(8bits~512bits) 512bits and make it 128 bits
        case 3:
            if(length < 56) {
                short required_pad = (short)((short)56 - length);
                arrayCopy(buffer, (short)0, sixtyFourETable, (short)0, length);
                arrayCopy(padding, (short)0, sixtyFourETable, (short)length, required_pad);
                arrayCopy(totalML, (short)0, sixtyFourETable, (short)56, (short)8);
                processByCase = 0;
            }
            else {
                short required_pad = (short)((short)64 - length);
                arrayCopy(buffer, (short)0, sixtyFourETable, (short)0, length);
                arrayCopy(padding, (short)0, sixtyFourETable, (short)length, required_pad);
                processByCase = 2;
            }
            break;
    }
}

/* call methods - Round 1,2,3,4.

```

each method is called by total 16*/

```
//round 1
roundOne (wordA, wordB, wordC, wordD, sixtyFourETable, (short)0, S11, (byte)0xD7,
          (byte)0x6A, (byte)0xA4, (byte)0x78);
roundOne (wordD, wordA, wordB, wordC, sixtyFourETable, (short)4, S12, (byte)0xE8,
          (byte)0xC7, (byte)0xB7, (byte)0x56);
roundOne (wordC, wordD, wordA, wordB, sixtyFourETable, (short)8, S13, (byte)0x24,
          (byte)0x20, (byte)0x70, (byte)0xDB);
roundOne (wordB, wordC, wordD, wordA, sixtyFourETable, (short)12, S14, (byte)0xC1,
          (byte)0xBD, (byte)0xCE, (byte)0xEE);
roundOne (wordA, wordB, wordC, wordD, sixtyFourETable, (short)16, S11, (byte)0xF5,
          (byte)0x7C, (byte)0x0F, (byte)0xAF);
roundOne (wordD, wordA, wordB, wordC, sixtyFourETable, (short)20, S12, (byte)0x47,
          (byte)0x87, (byte)0xC6, (byte)0x2A);
roundOne (wordC, wordD, wordA, wordB, sixtyFourETable, (short)24, S13, (byte)0xA8,
          (byte)0x30, (byte)0x46, (byte)0x13);
roundOne (wordB, wordD, wordD, wordA, sixtyFourETable, (short)28, S14, (byte)0xFD,
          (byte)0x46, (byte)0x95, (byte)0x01);
roundOne (wordA, wordB, wordC, wordD, sixtyFourETable, (short)32, S11, (byte)0x69,
          (byte)0x80, (byte)0x98, (byte)0xD8);
roundOne (wordD, wordA, wordB, wordC, sixtyFourETable, (short)36, S12, (byte)0x8B,
          (byte)0x44, (byte)0xF7, (byte)0xAF);
roundOne (wordC, wordD, wordA, wordB, sixtyFourETable, (short)40, S13, (byte)0xFF,
          (byte)0xFF, (byte)0x5B, (byte)0xB1);
roundOne (wordB, wordC, wordD, wordA, sixtyFourETable, (short)44, S14, (byte)0x89,
          (byte)0x5C, (byte)0xD7, (byte)0xBE);
roundOne (wordA, wordB, wordC, wordD, sixtyFourETable, (short)48, S11, (byte)0x6B,
          (byte)0x90, (byte)0x11, (byte)0x22);
roundOne (wordD, wordA, wordB, wordC, sixtyFourETable, (short)52, S12, (byte)0xFD,
          (byte)0x98, (byte)0x71, (byte)0x93);
roundOne (wordC, wordD, wordA, wordB, sixtyFourETable, (short)56, S13, (byte)0xA6,
          (byte)0x79, (byte)0x43, (byte)0x8E);
roundOne (wordB, wordC, wordD, wordA, sixtyFourETable, (short)60, S14, (byte)0x49,
          (byte)0xB4, (byte)0x08, (byte)0x21);

//round 2
roundTwo (wordA, wordB, wordC, wordD, sixtyFourETable, (short)4, S21, (byte)0xF6,
          (byte)0x1E, (byte)0x25, (byte)0x62);
roundTwo (wordD, wordA, wordB, wordC, sixtyFourETable, (short)24, S22, (byte)0xC0,
          (byte)0x40, (byte)0xB3, (byte)0x40);
roundTwo (wordC, wordD, wordA, wordB, sixtyFourETable, (short)44, S23, (byte)0x26,
          (byte)0x5E, (byte)0x5A, (byte)0x51);
roundTwo (wordB, wordC, wordD, wordA, sixtyFourETable, (short)0, S24, (byte)0xE9,
          (byte)0xB6, (byte)0xC7, (byte)0xAA);
roundTwo (wordA, wordB, wordC, wordD, sixtyFourETable, (short)20, S21, (byte)0xD6,
          (byte)0x2F, (byte)0x10, (byte)0x5D);
roundTwo (wordD, wordA, wordB, wordC, sixtyFourETable, (short)40, S22, (byte)0x2,
          (byte)0x44, (byte)0x14, (byte)0x53);
roundTwo (wordC, wordD, wordA, wordB, sixtyFourETable, (short)60, S23, (byte)0xD8,
          (byte)0xA1, (byte)0xE6, (byte)0x81);
roundTwo (wordB, wordD, wordD, wordA, sixtyFourETable, (short)24, S24, (byte)0xE7,
          (byte)0xD3, (byte)0xFB, (byte)0xC8);
roundTwo (wordA, wordB, wordC, wordD, sixtyFourETable, (short)36, S21, (byte)0x21,
          (byte)0xE1, (byte)0xCD, (byte)0xE6);
roundTwo (wordD, wordA, wordB, wordC, sixtyFourETable, (short)56, S22, (byte)0xC3,
```

```

        (byte)0x37, (byte)0x07, (byte)0xD6);
roundTwo (wordC, wordD, wordA, wordB, sixtyFourETable, (short)12, S23, (byte)0xF4,
        (byte)0xD5, (byte)0x0D, (byte)0x87);
roundTwo (wordB, wordC, wordD, wordA, sixtyFourETable, (short)32, S24, (byte)0x45,
        (byte)0x5A, (byte)0x14, (byte)0xED);
roundTwo (wordA, wordB, wordC, wordD, sixtyFourETable, (short)52, S21, (byte)0xA9,
        (byte)0xE3, (byte)0xE9, (byte)0x05);
roundTwo (wordD, wordA, wordB, wordC, sixtyFourETable, (short)8, S22, (byte)0xFC,
        (byte)0xEF, (byte)0xA3, (byte)0xF8);
roundTwo (wordC, wordD, wordA, wordB, sixtyFourETable, (short)28, S23, (byte)0x67,
        (byte)0x6F, (byte)0x02, (byte)0xD9);
roundTwo (wordB, wordC, wordD, wordA, sixtyFourETable, (short)48, S24, (byte)0x8D,
        (byte)0x2A, (byte)0x4C, (byte)0x8A);

```

```
//round 3
```

```

roundThree(wordA, wordB, wordC, wordD, sixtyFourETable, (short)20, S31, (byte)0xFF,
        (byte)0xFA, (byte)0x39, (byte)0x42);
roundThree(wordD, wordA, wordB, wordC, sixtyFourETable, (short)32, S32, (byte)0x87,
        (byte)0x71, (byte)0xF6, (byte)0x81);
roundThree(wordC, wordD, wordA, wordB, sixtyFourETable, (short)44, S33, (byte)0x6D,
        (byte)0x9D, (byte)0x61, (byte)0x22);
roundThree(wordB, wordC, wordD, wordA, sixtyFourETable, (short)56, S34, (byte)0xFD,
        (byte)0xE5, (byte)0x38, (byte)0x0C);
roundThree(wordA, wordB, wordC, wordD, sixtyFourETable, (short)4, S31, (byte)0xA4,
        (byte)0xBE, (byte)0xEA, (byte)0x44);
roundThree(wordD, wordA, wordB, wordC, sixtyFourETable, (short)16, S32, (byte)0x4B,
        (byte)0xDE, (byte)0xCF, (byte)0xA9);
roundThree(wordC, wordD, wordA, wordB, sixtyFourETable, (short)28, S33, (byte)0xF6,
        (byte)0xBB, (byte)0x4B, (byte)0x80);
roundThree(wordB, wordD, wordD, wordA, sixtyFourETable, (short)40, S34, (byte)0xBE,
        byte)0xBF, (byte)0xBC, (byte)0x70);
roundThree(wordA, wordB, wordC, wordD, sixtyFourETable, (short)52, S31, (byte)0x28,
        (byte)0x9B, (byte)0x7E, (byte)0xC6);
roundThree(wordD, wordA, wordB, wordC, sixtyFourETable, (short)0, S32, (byte)0xEA,
        (byte)0xA1, (byte)0x27, (byte)0xFA);
roundThree(wordC, wordD, wordA, wordB, sixtyFourETable, (short)12, S33, (byte)0xD4,
        (byte)0xEF, (byte)0x30, (byte)0x85);
roundThree(wordB, wordC, wordD, wordA, sixtyFourETable, (short)24, S34, (byte)0x4,
        (byte)0x88, (byte)0x1D, (byte)0x05);
roundThree(wordA, wordB, wordC, wordD, sixtyFourETable, (short)36, S31, (byte)0xD9,
        (byte)0xD4, (byte)0xD0, (byte)0x39);
roundThree(wordD, wordA, wordB, wordC, sixtyFourETable, (short)48, S32, (byte)0xE6,
        (byte)0xDB, (byte)0x99, (byte)0xE5);
roundThree(wordC, wordD, wordA, wordB, sixtyFourETable, (short)60, S33, (byte)0x1F,
        (byte)0xA2, (byte)0x7C, (byte)0xF8);
roundThree(wordB, wordC, wordD, wordA, sixtyFourETable, (short)8, S34, (byte)0xC4,
        (byte)0xAC, (byte)0x56, (byte)0x65);

```

```
//round 4
```

```

roundFour (wordA, wordB, wordC, wordD, sixtyFourETable, (short)0, S41, (byte)0xF4,
        (byte)0x29, (byte)0x22, (byte)0x44);
roundFour (wordD, wordA, wordB, wordC, sixtyFourETable, (short)28, S42, (byte)0x43,
        (byte)0x2A, (byte)0xFF, (byte)0x97);
roundFour (wordC, wordD, wordA, wordB, sixtyFourETable, (short)56, S43, (byte)0xAB,
        (byte)0x94, (byte)0x23, (byte)0xA7);
roundFour (wordB, wordC, wordD, wordA, sixtyFourETable, (short)20, S44, (byte)0xFC,

```

```

        (byte)0x93, (byte)0xA0, (byte)0x39);
roundFour (wordA, wordB, wordC, wordD, sixtyFourETable, (short)48, S41, (byte)0x65,
        (byte)0x5B, (byte)0x59, (byte)0xC3);
roundFour (wordD, wordA, wordB, wordC, sixtyFourETable, (short)12, S42, (byte)0x8F,
        (byte)0x0C, (byte)0xCC, (byte)0x92);
roundFour (wordC, wordD, wordA, wordB, sixtyFourETable, (short)40, S43, (byte)0xFF,
        (byte)0xEF, (byte)0xF4, (byte)0x7D);
roundFour (wordB, wordD, wordD, wordA, sixtyFourETable, (short)4, S44, (byte)0x85,
        (byte)0x84, (byte)0x5D, (byte)0xD1);
roundFour (wordA, wordB, wordC, wordD, sixtyFourETable, (short)32, S41, (byte)0x6F,
        (byte)0xA8, (byte)0x7E, (byte)0x4F);
roundFour (wordD, wordA, wordB, wordC, sixtyFourETable, (short)60, S42, (byte)0xFE,
        (byte)0x2C, (byte)0xE6, (byte)0xE0);
roundFour (wordC, wordD, wordA, wordB, sixtyFourETable, (short)24, S43, (byte)0xA3,
        (byte)0x01, (byte)0x43, (byte)0x14);
roundFour (wordB, wordC, wordD, wordA, sixtyFourETable, (short)52, S44, (byte)0x4E,
        (byte)0x08, (byte)0x11, (byte)0xA1);
roundFour (wordA, wordB, wordC, wordD, sixtyFourETable, (short)16, S41, (byte)0xF7,
        (byte)0x53, (byte)0x7E, (byte)0x82);
roundFour (wordD, wordA, wordB, wordC, sixtyFourETable, (short)44, S42, (byte)0xBD,
        (byte)0x3A, (byte)0xF2, (byte)0x35);
roundFour (wordC, wordD, wordA, wordB, sixtyFourETable, (short)8, S43, (byte)0x2A,
        (byte)0xD7, (byte)0xD2, (byte)0xBB);
roundFour (wordB, wordC, wordD, wordA, sixtyFourETable, (short)36, S44, (byte)0xEB,
        (byte)0x86, (byte)0xD3, (byte)0x91);

    /* save A as AA, B as BB, C as CC, and D as DD */
    byteAddFun(wordA, wordAA, (short)0, (short)3);
    byteAddFun(wordB, wordBB, (short)0, (short)3);
    byteAddFun(wordC, wordCC, (short)0, (short)3);
    byteAddFun(wordD, wordDD, (short)0, (short)3);

} while (processByCase != 0);

} //end md5 update method

//start md5Gen method
void md5Gen (byte[] md) {

    //generate output A
    for (short i = 0, j = 3; j >= 0; ++i, --j)
        md[i] = wordA[j];

    //generate output B
    for (short i = 4, j = 3; j >= 0; ++i, --j)
        md[i] = wordB[j];

    //generate output C
    for (short i = 8, j = 3; j >= 0; ++i, --j)
        md[i] = wordC[j];

    //generate output D
    for (short i = 12, j = 3; j >= 0; ++i, --j)
        md[i] = wordD[j];

```



```

} //end md5Gen method

//for round 1: a = b + ((a + F(b,c,d) + X[k] + T[i]) <<< s)
private void roundOne (byte[] a, byte[] b, byte[] c, byte[] d, byte[] x,
short point, byte s, byte t1, byte t2, byte t3, byte t4) {

    andFun(b, c, partialM);
    compFun(b, partialM2);
    andFun(partialM2, d, partialM2);
    orFun(partialM, partialM2, partialM2);
    arrayCopy(x, (short)point, partialM, (short)0, (short)4);
    byteAddFun(partialM2, partialM, (short)0, (short)3);
    partialM[0] = t1; partialM[1] = t2;
    partialM[2] = t3; partialM[3] = t4;
    byteAddFun (partialM2, partialM, (short)0, (short)3);
    byteAddFun (partialM2, a, (short)0, (short)3);
    leftRotation(partialM2, s, partialM2);
    byteAddFun(partialM2, b, (short)0, (short)3);
    arrayCopy(partialM2, (short)0, a, (short)0, (short)4);

} //end round 1

//for round 2: a = b + ((a + G(b,c,d) + X[k] + T[i]) <<< s)
private void roundTwo (byte[] a, byte[] b, byte[] c, byte[] d, byte[] x,
short point, byte s, byte t1, byte t2, byte t3, byte t4) {

    andFun(b, d, partialM);
    compFun(d, partialM2);
    andFun(c, partialM2, partialM2);
    orFun(partialM, partialM2, partialM2);
    arrayCopy(x, (short)point, partialM, (short)0, (short)4);
    byteAddFun(partialM2, partialM, (short)0, (short)3);
    partialM[0] = t1; partialM[1] = t2;
    partialM[2] = t3; partialM[3] = t4;
    byteAddFun (partialM2, partialM, (short)0, (short)3);
    byteAddFun (partialM2, a, (short)0, (short)3);
    leftRotation(partialM2, s, partialM2);
    byteAddFun(partialM2, b, (short)0, (short)3);
    arrayCopy(partialM2, (short)0, a, (short)0, (short)4);

} //end round 2

//round 3: a = b + ((a + H(b,c,d) + X[k] + T[i]) <<< s)
private void roundThree (byte[] a, byte[] b, byte[] c, byte[] d, byte[] x,
short point, byte s, byte t1, byte t2, byte t3, byte t4) {

    xorFun(b, c, partialM);
    xorFun(partialM, d, partialM2);
    arrayCopy(x, (short)point, partialM, (short)0, (short)4);
    byteAddFun(partialM2, partialM, (short)0, (short)3);
    partialM[0] = t1;
    partialM[1] = t2;
    partialM[2] = t3;

```

```

    partialM[3] = t4;
    byteAddFun (partialM2, partialM, (short)0, (short)3);
    byteAddFun (partialM2, a, (short)0, (short)3);
    leftRotation(partialM2, s, partialM2);
    byteAddFun(partialM2, b, (short)0, (short)3);
    arrayCopy(partialM2, (short)0, a, (short)0, (short)4);

} //end round 3

//round 4: a = b + ((a + l(b,c,d) + X[k] + T[i]) <<< s)
private void roundFour (byte[] a, byte[] b, byte[] c, byte[] d, byte[] x,
    short point, byte s, byte t1, byte t2, byte t3, byte t4) {

    compFun(d, partialM);
    orFun(a, partialM, partialM);
    xorFun(c, partialM, partialM2);
    arrayCopy(x, (short)point, partialM, (short)0, (short)4);
    byteAddFun(partialM2, partialM, (short)0, (short)3);
    partialM[0] = t1;
    partialM[1] = t2;
    partialM[2] = t3;
    partialM[3] = t4;
    byteAddFun (partialM2, partialM, (short)0, (short)3);
    byteAddFun (partialM2, a, (short)0, (short)3);
    leftRotation(partialM2, s, partialM2);
    byteAddFun(partialM2, b, (short)0, (short)3);
    arrayCopy(partialM2, (short)0, a, (short)0, (short)4);

} //end round 4

//start array copy method
private void arrayCopy (byte[] original, short oStart,
    byte[] destine, short dStart, short length) {

    short index = 0;

    for (index = length; index > 0; --index) {
        destine[dStart] = original[oStart];
        oStart++;
        dStart++;
    }

} //end array copy method

//start byte add function
private void byteAddFun (byte[] a, byte[] b, short point2, short length) {

    if (length == -1)
        return;

    short oneByte = (short)((short)(a[length]&0x00FF) +
        (short)(b[length]&0x00FF) + point2);

```

```

a[length] = (byte)(oneByte & (short)0x00FF);

if (overFlow(oneByte))
    byteAddFun(a, b, (short)1, --length);
else
    byteAddFun(a, b, (short)0, --length);

return;

} //end byte add function

private void appendLength (byte[] lengthTotal, short length) {

    byte[] partialLength = new byte[5];
    short point = -1;

    while (true) {
        if (length >= 127) {
            partialLength[++point] = (byte)0x7F;
            length = (short)(length - (short)127);
        }
        else {
            partialLength[++point] = (byte)(length%((short)127));
            break;
        }
    }

    for (short index = 7; point >= 0; --point, index = 7) {
        short oneByte = (short)((short)(lengthTotal[index]&0x00ff) +
            (short)(partialLength[point]&0x00ff));
        lengthTotal[index] = (byte)(oneByte & (short)0x00FF);
        if (overFlow(oneByte))
            lengthTotal = round(lengthTotal, --index, (short)1);
    }

} //end appendLength method

//start overflow function
private boolean overFlow (short number) {

    if ((short)(number&(short)0xFF00) >= (short)0x0100)
        return true;
    else
        return false;

} //end overflow function

//start left rotation function
private void leftRotation (byte[] array, byte shifting, byte[] answer) {

    byte one = 0x00; byte two = 0x00;
    byte three = 0x00; byte four = 0x00;
    byte pointer = 0x00; byte shifter = 0x00;

```

```

byte oppositeShift = 0x00;
byte current = 0x00; short next = 0x00;

switch (shifting % (byte)8) {
    case 0: pointer = (byte)0xFF;
            break;
    case 1: pointer = (byte)0x80;
            break;
    case 2: pointer = (byte)0xC0;
            break;
    case 3: pointer = (byte)0xE0;
            break;
    case 4: pointer = (byte)0xF0;
            break;
    case 5: pointer = (byte)0xF8;
            break;
    case 6: pointer = (byte)0xFC;
            break;
    case 7: pointer = (byte)0xFE;
            break;
}

shifter = (byte)(shifting % (byte)8);
oppositeShift = (byte)((byte)0x08 - shifter);

if (shifter == 0) {
    shifter = 8;
    oppositeShift = 0;
}

if (shifting <= 8) {

    one = array[0]; two = array[1];
    three = array[2]; four = array[3];

}

else if (shifting <= 16) {

    one = array[1]; two = array[2];
    three = array[3]; four = array[0];

}

else if (shifting <= 24) {

    one = array[2]; two = array[3];
    three = array[0]; four = array[1];

}

next = (short)((four & pointer) & 0x00FF);
answer[3] = (byte)(four << shifter);
current = (byte)(next >>> oppositeShift);
next = (short)((three & pointer) & 0x00FF);
answer[2] = (byte)((three << shifter) | current);

```

```

current = (byte)(next >>> oppositeShift);
next = (short)((two & pointer) & 0x00FF);
answer[1] = (byte)((two << shifter) | current);
current = (byte)(next >>> oppositeShift);
next = (short)((one & pointer) & 0x00FF);
answer[0] = (byte)((one << shifter) | current);
current = (byte)(next >>> oppositeShift);
answer[3] = (byte)(answer[3] | (current));

} //end left Rotation method

//start round function
private byte[] round(byte[] total, short index, short point2) {

    if (index < 0)
        return total;

    short oneByte = (short)((short)(total[index]&0x00ff) + point2);
    total[index] = (byte)(oneByte & (short)0x00FF);

    if (overFlow(oneByte))
        total = round(total, --index, (short)1);

    return total;

} //end round function

//start bit complementation function
private void compFun (byte[] array, byte[] answer) {

    for (short i = 0; i < (short)4; i++)
        answer[i] = (byte)~(array[i]);

} //end bit complementation function

//start bit and function
private void andFun (byte[] array1, byte[] array2, byte[] answer) {

    for (short i = 0; i < (short)4; i++)
        answer[i] = (byte)(array1[i] & array2[i]);

} //end bit and function

//start bit or function
private void orFun (byte[] array1, byte[] array2, byte[] answer) {

    for (short i = 0; i < (short)4; i++)
        answer[i] = (byte)(array1[i] | array2[i]);

} //end bit or function

```

```
//start xor function
private void xorFun (byte[] array1, byte[] array2, byte[] answer) {

    for (short i = 0; i < (short)4; i++)
        answer[i] = (byte)(array1[i] ^ array2[i]);

} //end xor function

} //end md5 class
```

InstallerInterface.java

```
/*
*****
This applet make a way to connect between the installer and Ebank using SIO.
*****
*/

package installers;

import javacard.framework.Shareable;

public interface InstallInterface extends Shareable{

    //to check password
    public boolean check();
}
```

Terminal.java

```
/*
*****
It is used for install applets.
The terminal download applet's class and convert it to its CAP file.
And then send it to the installer on the JCRE.
*****
*/

import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
import java.io.*;

//start Terminal applet
class Terminal extends JFrame {

    //to declare variables
    //for simulation window, buttons, and text fields
    //for Ebankk applet
    private JButton ebankApp;
    private JButton converterEbank;
    private JButton installEbank;

    //for cap file choosing
    private JFileChooser fileChooser;
    //to read cap file
    private File capFile;
    //to make dump file
    private File dumpFile;
    //to make batch file
    private File batchFile;
    //to make script file
    private File scriptFile;
    //for making output file to show result
    private File outputFile;

    //start Terminal constructor
    Terminal(){

        //to set simulation window and color
        getContentPane().setLayout(null);
        getContentPane().setBackground(Color.blue);
        //set main window title
        setTitle(" Terminal");
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

        /*for simulation window */
        //for choosing Installer applet file
        ebankApp = new JButton("Down Applet file");
        ebankApp.setBackground(Color.yellow);
        ebankApp.setBounds(25,30,275,30);
    }
}
```



```

//to convert Installer
converterEbank = new JButton("Convert");
converterEbank.setBackground(Color.yellow);
converterEbank.setBounds(25, 80, 275, 30);

//to install installer applet to the card
installEbank = new JButton("Install");
installEbank.setBackground(Color.yellow);
installEbank.setBounds(25, 130, 275, 30);

//to download Ebank applet
ebankApp.addActionListener(new ActionListener(){
    public void actionPerformed (ActionEvent e){

        try{

            String order = "c:/defense/batch/downEbankClass.bat";
            Process child = Runtime.getRuntime().exec(order);

        } catch (Exception ee){}
    }});

//to convert Ebank applet
converterEbank.addActionListener(new ActionListener(){
    public void actionPerformed (ActionEvent e){

        try{

            String order = "c:/defense/batch/convertEbank.bat";
            Process child = Runtime.getRuntime().exec(order);

        } catch (Exception ee){}
    }});

//to installEbank applet
installEbank.addActionListener(new ActionListener(){
    public void actionPerformed (ActionEvent e){

        fileChooser = new JFileChooser("c:\\defense\\terminal");
        fileChooser.setDialogTitle("Choose a CAP file?");
        int forError = fileChooser.showOpenDialog(Terminal.this);

        //for checking CAP selection error
        if(forError == JFileChooser.APPROVE_OPTION)
            capFile = fileChooser.getSelectedFile();
        else{
            JOptionPane.showMessageDialog(Terminal.this,
                "Need a CAP file", "ERROR",JOptionPane.ERROR_MESSAGE);
            capFile = null;
            return;
        }
    }
}

```

```

//to make batch file and script file for sending them to installer
try {
    //make script batch file
    batchFile = new File("script.bat");
    //make script file
    scriptFile = new File("apdu.scr");
    BufferedWriter text = new BufferedWriter(new FileWriter(batchFile));
    text.write("@echo off\n");
    text.write("scriptgen -o " + scriptFile.getAbsolutePath() + " "
        + capFile.getAbsolutePath() + "\n");
    text.close();
    String path = batchFile.getAbsolutePath();
    path = path.replace("\\", '/');
    Process child = Runtime.getRuntime().exec(path);
    child.waitFor();
} catch (Exception e) {
} finally {
    batchFile.delete();
    batchFile = null;
}

//to make dump file to see a CAP file
String makeText = new String();
dumpFile = new File("dump");

try {
    batchFile = new File("makedump.bat");
    BufferedWriter toWrite = new BufferedWriter(new FileWriter(batchFile));
    toWrite.write("@echo off\n");
    toWrite.write("capdump " + capFile.getAbsolutePath() + "\n");
    toWrite.close();

    String path = batchFile.getAbsolutePath();
    path = path.replace("\\", '/');
    Process child = Runtime.getRuntime().exec(path);

    BufferedReader readData = new BufferedReader(new
        InputStreamReader(child.getInputStream()));
    BufferedWriter towrite = new BufferedWriter(new FileWriter(dumpFile));

    while ((makeText = readData.readLine()) != null)
        towrite.write(makeText + "\n");
    towrite.close();
} catch (Exception ee) {
} finally {
    batchFile.delete();
    batchFile = null;
    capFile = null;
}

//to send power.scr file to the installer
outputFile = new File("power.scr");
byte[] sendBuffer = new byte[64];
String readData = new String();
int length = 0;
byte length2 = 0;

```

```

String byte2 = new String();

try{
    BufferedReader reader = new BufferedReader(new FileReader(scriptFile));
    BufferedWriter writer = new BufferedWriter(new FileWriter(outputFile));

    //turn on the card
    writer.write("powerup;\n");

    //to wake up the JCRE installer
    writer.write("//select the installer \n");

    //apdu command
    writer.write("0x00 0xA4 0x04 0x00 0x09 0xA0 0x00 0x00 0x00
                0x62 0x03 0x01 0x08 0x01 0x7F;\n\n");

    //read whole data from script file that is made by scriptgen command
    while((readData = reader.readLine()) != null){
        writer.write(readData + "\n");
    }

    // create an instance of this applet
    writer.write("\n//Install this applet\n");
    writer.write("0x80 0xb8 0x00 0x00 0x0c 0x0a 0x00 0x00 0x00 0x00
                0x0C 0x03 0x01 0x0C 0x06 0x01 0x00 0x7f;\n\n");

    //done read data
    reader.close();

    //to send script file to the installer to make hash value
    BufferedInputStream reader2 = new BufferedInputStream(
        new FileInputStream(dumpFile));

    //select installer applet on the JCRE
    writer.write("//Select installer\n");
    writer.write("0x0 0xa4 0x04 0x00 0x0a 0x00 0x00 0x00 0x00 0x0B
                0x03 0x01 0x0C 0x06 0x01 0x7f;\n\n");

    //for md5 init apducommand
    writer.write("//cap file start\n");
    writer.write("0xB0 0x02 0x00 0x00 0x00 0x7F;\n\n");

    //send cap file to the installer
    while(true){
        //data read length
        length = (reader2.read(sendBuffer));

        //if no data
        if (length == -1)
            break;

        length2 = (byte)(length & 0x000000FF);
        byte2 = (Integer.toString(length2 & 0x000000FF, 16)).toUpperCase();
        if (byte2.length() == 1)
            byte2 = "0" + byte2;
    }
}

```

```

//after read one line, start md5 update apdu command
writer.write("0xB0 0x03 0x00 0x00 " + "0x" + byte2 + " ");

//start for loop
for(int i = 0; i < length; i++){
    byte2 = (Integer.toString(sendBuffer[i] & 0x000000FF, 16)).toUpperCase();
    if (byte2.length() == 1)
        byte2 = "0" + byte2;
    writer.write("0x" + byte2 + " ");
} //end for loop

    writer.write("0x7F:\n");
} //end while loop

//to generate hash value
writer.write("\n");
writer.write("//CAP file end\n");

//md5 generate command
writer.write("0xB0 0x04 0x00 0x00 0x00 0x7F;\n\n");

//turn off the card
writer.write("powerdown;");

//stop read data and write data
reader2.close();
writer.close();
} catch (FileNotFoundException ee) {
} catch (IOException eee) {
} finally {
dumpFile.delete();
dumpFile = null;
scriptFile.delete();
scriptFile = null;
}

try {
//make ebank install batch file
batchFile = new File("install.bat");
BufferedWriter toWrite = new BufferedWriter(new FileWriter(batchFile));
toWrite.write("@echo off\n");

//to send installer and have an answer from the JCRE installer applet
toWrite.write("apdutool -o answer " + outputFile.getAbsolutePath() + "\n");
toWrite.close();

String path = batchFile.getAbsolutePath();
path = path.replace("\\", '/');
Process child = Runtime.getRuntime().exec(path);
child.waitFor();
} catch (Exception e1) {
} finally {
    outputFile.delete();
    outputFile = null;
    batchFile.delete();
    batchFile = null;
}
}

```

```
    }}
  }):

  //add these to simulation window
  getContentPane().add(ebankApp);
  getContentPane().add(converterEbank);
  getContentPane().add(installEbank);

} //end constructor

//start main method
public static void main (String[] args){

  //call the constructor
  JFrame mainWindow = new Terminal();

  //to set simulation window boundary
  mainWindow.setBounds(0, 0, 330, 218);
  mainWindow.setVisible(true);

} //end main method

} //end class Terminal
```

VITA



JAE HYUK JOO

Candidate for the Degree of

Master of Science

Thesis: SECURE COMMUNICATION ON JAVA CARD

Major Field: Computer Science

Biographical:

Personal: The son of Kap Eung Joo and Jong Sun Kim
The brother of Eun Hee, Eun Ha, and Jae Sung

Education: Received Bachelor of Engineering degree in Environmental
Engineering from Tae Jon University, Tae Jon, Korea in February 1995.
Completed the requirements for the Master of Science degree with a major
in Computer Science at Oklahoma State University in July, 2004.