

**SECURE OBJECT SHARING ON JAVA CARD**

By

**SYENG HO JANG**

Bachelor of Science

Oklahoma State University

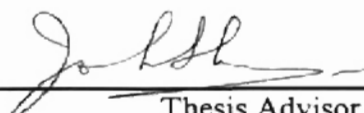
Stillwater, Oklahoma

2000

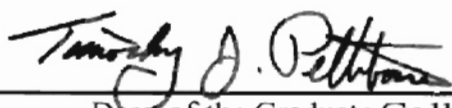
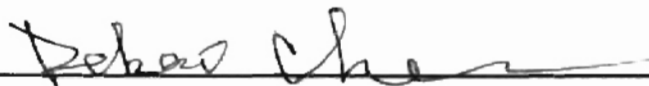
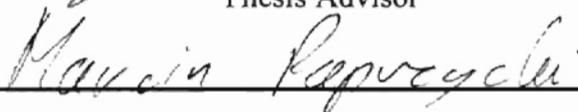
Submitted to the Faculty of the  
Graduate College of the  
Oklahoma State University  
in partial fulfillment of  
the requirements for  
the Degree of  
**MASTER OF SCIENCE**  
May, 2003

## SECURE OBJECT SHARING ON JAVA CARD

Thesis Approved:



Thesis Advisor



Dean of the Graduate College

## PREFACE

This research concerns enhancement in on-card verification of downloaded applets in Java Card technology. In this thesis, we propose the on-card installer with a one-way hash function to support on-card verification of download applets. The hash value generated from the on-card installer is used to verify download applets when they try to gain a SIO from a server applet.

This thesis is organized into five chapters and an appendix. Chapter 1, Introduction, depicts the background of Java Card technology, the current problem in the Java Card platform, and the objectives of this research. Chapter 2, Literature Review, introduces fundamental concepts and background knowledge on Smart Card, Java Card, and message digest algorithms. Chapter 3, Secure Object Sharing, presents solution to meet the objectives. Chapter 4, Secure Object Sharing Simulation, simulates the object sharing process between a server applet and a client applet. Chapter 5, Conclusion, draws a conclusion of enhancement in on-card verification of downloaded applets. Appendix presents source codes that are used for the simulation.

## TABLE OF CONTENTS

Chapter	Page
1. INTRODUCTION .....	1
1.1 Background .....	1
1.2 Current Problems .....	2
1.3 Objectives .....	3
2. LITERATURE REVIEW .....	5
2.1 Smart Card .....	5
Smart Card Classification .....	5
Smart Card Hardware .....	7
Smart Card Operating System .....	8
Card Acceptance Device .....	9
Smart Card Communication .....	9
Smart Card Standards and Specification .....	11
2.2 Java Card .....	12
Applet .....	13
Java Card Language Subset .....	14
Java Card Virtual Machine .....	14
CAP File Format .....	16
Java Card Framework .....	16
Java Card Runtime Environment .....	16
Applet Execution .....	18
Applet Object .....	20
Java Card Transaction .....	21
Applet Firewall .....	22
Context Switch .....	23
OpenCard FrameWork .....	25
2.3 Message Digest .....	25
One-Way Hash Function .....	26
Message Digest Algorithms .....	27
Secure Hash Algorithm .....	28



Chapter	Page
3. SECURE OBJECT SHARING.....	29
3.1 Applet Installation.....	29
Converter.....	30
Verifier.....	31
Off-Card Installation Program and Installer.....	31
Illegal Installation.....	32
3.2 Object Sharing.....	33
Shareable Interface Mechanism.....	34
Client Applet Verification.....	37
3.3 Enhanced Applet Verification.....	38
Installer with a One-Way Hash Function.....	39
Secure Object Sharing Process.....	42
4. SECURE OBJECT SHARING SIMULATION.....	45
4.1 Java Card 2.2 Development Kit.....	47
4.2 Object Sharing Process Simulation with the Existing Approach.....	47
Shared Information.....	47
Illegal SIO Access.....	48
4.3 Message Digest.....	50
Proposed On-Card Installer.....	50
Hash Value of the Client Applet.....	51
Applet Installation.....	52
Enhanced Client Applet Verification.....	53
Updated a Hash Value of the Client Applet.....	57
5. CONCLUSION.....	59
BIBLIOGRAPHY.....	61
APPENDIX.....	63
AirMile.java.....	63
AriMileInterface.java.....	67
EPurse.java.....	67
HostAirMile.java.....	70
HostEPurse.java.....	74
HashInstaller.java.....	78
HashInstallerInterface.java.....	97

Chapter	Page
HVGenerator.java .....	97
Terminal.java .....	112

## LIST OF TABLES

Table	Page
1. Smart Card Comparisons .....	6
2. The Methods in the Class javacard.framework.Applet.....	19

## LIST OF FIGURES

Figure	Page
1. Command APDU Structure .....	11
2. Response APDU Structure.....	11
3. Structure of Application Identifier .....	14
4. Java Card Virtual Machine .....	15
5. Java Card Runtime Environment .....	18
6. Applet Execution States.....	20
7. Applet Communication.....	20
8. Applet Firewall .....	23
9. One-Way Hash Function.....	26
10. Applet Installation Process .....	30
11. Object Sharing Process .....	35
12. Installer with a One-Way Hash Function.....	41
13. Secure Object Sharing Process .....	42
14. The Usage of Tools in the Java Card 2.2 Development Kit .....	46
15. Host Application for the AirMile with Existing Approach .....	49
16. Host Application for the EPurse with Existing Approach.....	49
17. Hash Value Generator with the Hash Value of EPurse Applet .....	51
18. Terminal Application.....	53
19. Secure Object Sharing Simulation.....	54

Figure	Page
20. Host Application for the AirMile with Our Approach.....	55
21. Host Application for the EPurse with Our Approach.....	56
22. Update the Hash Value of the EPurse Applet in the AirMile Applet.....	58

# **1. INTRODUCTION**

## **1.1 Background**

The rapid growth of the Internet has changed the mode of business dealings from the traditional face to face in-store transactions to worldwide on-line transactions directed with a few mouse clicks in the home or office. For the electronic business market to reach its full potential, technology must offer the same level of trust as face to face business relations and must be able to handle business transactions in an efficient and user-friendly manner.

Smart cards have the potential to provide great security and portability due to the added intelligence of a microprocessor on the card, their size and computing ability. Smart cards therefore afford a way to enable secure transactions and a broad range of electronic business. These days, smart cards are mainly used for prepaid phone cards, electronic purses, retail royalty cards, and storage of identification and medical records, among other applications. The demand for smart cards is growing at a rate of 40 percent per year [1]. Currently, over 3 billion cards are in circulation worldwide with over 15% of the total in use in the United States and Canada [2].

However, smart cards have some limitations such as a small universe of knowledgeable programmers and limited flexibility to download applications into cards [3]. These factors prevent a broader deployment of smart card applications. The inner workings of smart cards differ widely from one manufacturer to another even though the cards are standardized in size, shape, and communication protocol. As a result, developing smart card applications have been limited to a small group of highly skilled and specialized programmers. Also, in the traditional approach, smart card applications

are burned in the chip, so after the card is issued, the embedded application cannot evolve.

A Java card is simply defined as a smart card that is capable of running applications called applets written in the Java programming language, and it offers a way to overcome the limitations of existing smart cards. While the Java Card platform preserves many of the benefits of the Java programming language such as productivity, security, robustness and portability across different chip architectures, it also provides several unique benefits. In the Java Card platform, new applets can be installed securely at any time, limited only by the memory size after a card has been issued, so card issuers can have the ability to dynamically respond to their customer's changing needs. Also, the Java Card platform provides a secure environment using an applet firewall that enables multiple applications supplied by different service providers to coexist securely on a single card.

### **1.2 Current Problems**

Even though Java Card technology provides a secure environment to enable multiple applets supplied by different vendors to coexist via the applet firewall mechanism and cooperate securely by way of the object shareable interface mechanism on a single card, enhancement in on-card verification of downloaded applets is still under consideration.

In the Java Card platform, the Java Card installer, an on-card component to install an applet on Java Cards, does not verify a converted applet (CAP) file that consists of applet classes and is the loadable and installable unit on the Java Card platform [3]. This means the correctness and integrity of a CAP file are verified off-card, and the installer

on the Java Card platform does not perform most of the traditional Java verifications at class-loading time. For this reason, it is possible for a malicious applet to be installed onto a card via illegal applet installation process.

In Java Card technology, cooperation between applets provided by different vendors is achieved through a shareable interface object (SIO) that is an object instance of a class implementing one or more shareable interfaces of an applet. To cooperate with other applets provided by different vendors, the behavior of a SIO should be available for other application providers who are asked not to reveal the behavior of the SIO. However, there is no guarantee that the application providers will not share it with unauthorized means. Therefore, a malicious applet that can get a SIO from other applets can be developed. Once the malicious applet is installed onto a card by way of an illegal applet installation process, the malicious applet can access sensitive data and service of other applets supplied by different vendors by invoking one of other applets' SIO.

### **1.3 Objectives**

The purpose of this research is to enhance Java Card security, and our objectives are following:

The first objective is to improve on-card verification mechanism of downloaded applets. Beyond minimum-security protections enforced by the on-card installer and the Java Card Runtime Environment in the Java Card platform, Java Card technology does not standardize applets installation policy. For this reason, a malicious applet can be installed onto a card via illegal applet installation process, and determination of validation of applets relies on cryptographic exchange algorithms that may exist between



applets. However, the most frequently anticipated security concerns have been developer mistakes and design oversights.

The second objective is to improve a client applet verification mechanism before a server applet agrees to share its SIO with a client applet. In the Java Card platform, a server applet determines validation of a client applet only by the client's AID and secret parameter before the server applet agrees to share its SIO. Therefore, there is no way for a server applet to reject a malicious client applet that impersonates with a valid AID and secret parameter before it agrees to share its SIO with the malicious applet.

## **2. LITERATURE REVIEW**

In the Java Card platform, card applications called applets can be added at any time onto the cards after the card has been issued while applications in many embedded systems need to be burned into the ROM during manufacture. Also, multiple applets from different service providers can coexist and cooperate securely on a single Java Card. This thesis considers on-card verification of downloaded applets, and in our novel approach, the on-card installer with a one-way hash function supports on-card verification of downloaded applets that cooperates with other applets from different service providers on a single card. Here, the one-way hash function is used in the on-card installer to support verification of downloaded applets that cooperates with other applets residing on the same card because developing a malicious applet that can request service to other applets and also has the same hash value is almost impossible. In this chapter, we first review smart cards and Java cards in sections 2.1 and 2.2 respectively. Then, we review message digest algorithms in section 2.3.

### **2.1 Smart Card**

A smart card is similar in appearance to a credit card, but has electronic circuits embedded in silicon in the plastic substrate of the card. Unlike a magnetic stripe card, a smart card can carry all necessary functions and information on the card, so it does not require access to remote database at the time of the transaction. Smart cards have enjoyed wide acceptance in many application areas due to its added intelligence of a microprocessor, size and computing ability.

#### **Smart Card Classification**

There are two major categories of smart cards; embedded with either only a

memory chip with non-programmable logic, which is called a memory card, or a microprocessor and memory chip which is called a microprocessor card [3]. The comparison of two categories of smart cards with magnetic stripe cards is illustrated in Table 1.

Type of Cards	Max. Data Capacity	Processing Power	Cost of Card	Cost of Reader and Connection
Magnetic Strip Cards	140 bytes	None	\$0.20 - \$0.75	\$750
Memory Cards	1 Kbytes	None	\$1 - \$2.50	\$500
Microprocessor Cards	8 Kbytes	8-bit CPU, moving to 16-bit and 32-bit	\$7 - \$15	\$500

Table1. Smart Card Comparisons [2]

The typical memory cards hold up to 1- 4 KB of data, but do not have processor on the card with which to manipulate that data [2]. Memory cards are popular as high-security alternatives to magnetic strip cards, and they are used primarily as prepaid cards for public phones or similar applications that are sold against prepayment [3]. Since a memory card has no processor, a simple circuit capable of executing a few preprogrammed instructions performs its data processing [3]. For example, prepaid phone cards contains the hard wired logic that treats the chip memory as a counter by allowing one bits to be set to zero bits but not the reverse to prevent the value from being increased [4]. However, such a circuit has limited functions and cannot be reprogrammed, so memory cards cannot be reused after the value in the card is spent.

Microprocessor cards, as the name implies, contain a processor. They have the power to process data as well as to store and secure information. Also, they can integrate several different applications. Therefore, microprocessor cards offer more functional capability and increased security than memory cards. Typical microprocessor cards have

an 8-bit processor, 16KB read-only memory, and 512 bytes of random-access memory [2]. They are widely used where data security and privacy are major concern such as payment and banking industries, payment of parking and tolls, storage of identification and medical records, and access to satellite television, among other applications.

The term smart card generally refers to both memory cards and microprocessor cards [3]. However, some publications refer to call only microprocessor cards as smart card because memory cards are not really smart due to lack of intelligence to process data [3]. In this thesis, the term smart card refers to microprocessor cards.

### **Smart Card Hardware**

A typical smart card contains a Central Processing Unit (CPU) and three different types of memory - Read Only Memory (ROM), Electrical Erasable Programmable Read Only Memory (EEPROM) and Random Access Memory (RAM).

The most popular CPU in smart cards is an 8-bit micro controller, usually using the Motorola 6805 or Intel 8051 instruction set with clock speeds up to 5 MHz [3]. A 16-bit or 32-bit micro controller with Reduced Instruction Set (RISC) architecture is becoming more common [3]. Also, some smart cards have a built-in coprocessor for use in security applications that require computationally expensive cryptographic operations on voluminous data.

Read Only Memory (ROM), as the name implies, cannot be written to after the card is manufactured. Also, no voltage is needed to hold data in memory. Therefore, ROM in smart cards is used for containing operation systems' routines, permanent data as well as various testing and diagnostic functions [5]. Currently, chips with more than 32KB of ROM are available for smart cards [4].

Electrical Erasable Programmable Read Only Memory (EEPROM) functionally corresponds to a PC hard disk because it not only can preserve data content when power to the memory is turned off, but also can be modified during normal use of the card [5]. Therefore, EEPROM is used for data storage in smart cards. However, writing to EEPROM is 1,000 times slower than writing to random access memory (RAM) even though reading from EEPROM is as fast as reading from RAM [3]. Currently, chips are available with more than 8KB of EEPROM for smart cards [4].

Random Access Memory (RAM) needs a power supply for its operation, and once it is switched off, the data contents in RAM are evaporated. RAM is used in smart card as temporary working space for storing and modifying data. These days, chips are available with more than 64KB of RAM for smart cards [4].

Currently, other memory technologies such as Flash EEPROM are gaining popularity for smart cards [3]. Also, some experts prefer to use Recoverable Persistent Memory (RPM) to hide differences between volatile and non-volatile memories in application development [6].

### **Smart Card Operating System**

The International Organization for Standardization (ISO) standardizes a wide range of instructions for smart cards, and smart card operating systems support some or all of these instructions as well as the manufacturer's additions to do secure program execution and protect access to data [3]. Also, most of smart card operating systems support a hierarchical file structure with file directories to support a smart card-based service such as file selection and file selection command [3]. In this case, a card application is a data file that stores application-specific information, and smart card

operating system is the one that implements the semantics and instructions to access the card application data file [3].

However, the hierarchical approach for smart card operating system has limited services available on the smart card such as processing data with card application-specific security requirements, so newer operating systems that support multiple applications simultaneously are becoming more popular [8].

### **Card Acceptance Device**

Card Acceptance Device (CAD), into which a smart card is inserted, can be classified as two types: readers and terminals [3]. A reader, that has a slot into which a card is placed, is connected to the parallel, serial or USB port of a computer, and it can write to the card as well as read data from it [7]. In general, a reader does not have the power to process data while some readers have the power to detect errors and correct functions when transmitted data are not compliant with the underlying transport protocol [3]. A terminal that integrates a smart card reader as one of its components is a computer on its own. A terminal has the power to process data exchanged between itself and the smart card, and Bank ATMs and devices used in gas station for payment or stores for payments and credit card transactions are the most commonly seen terminals [3].

In this thesis, the application that interfaces with the smart card through a card acceptance device is referred to as host application whether it reside in the computer connected to the reader or in the terminal.

### **Smart Card Communication**

The communication between the smart card and the host application is one-way communication, known as a 'half-duplex' pathway. That is, the data can either be sent

from the host application or the smart card in turn but not both at the same time, so the other party must always be in reception mode. The entire data exchanged between smart card and host application takes place using Application Protocol Data Units (APDUs) that is the layer that is located directly above the transmission protocols in smart card, and the APDU is carried by the transmission protocol without modification or interpretation [5].

For the data exchange between smart card and host application, the master-slave model is employed, and a smart card always plays the slave role [3]. When a smart card is inserted in a CAD, the smart card is waiting for a command APDU from the host application. Then the smart card executes the instruction specified in the command APDU, and replies to the host application with a respond APDU.

Command APDU that is sent by a host application to a smart card consists of a header and an optional body [3]. The header consists of 4 bytes: Class of Instruction (CLA), Instruction Code (INS), and Parameter 1 and 2 (P1 and P2). The CLA byte is used to identify a category of the command and respond APDUs, and the INS byte specifies the instruction of the command. The two parameters bytes are used to describe more closely the instruction selected by the INS byte. The optional body that consists of data field, Lc, and Le varies in length. The data field contains data that are sent to the smart card for executing the instruction of the command and the length of the data field is specified in the Lc byte. The last byte in the command APDU is the Le that specifies the length of the data field expected by the host application in the smart card's response. The structure of command APDU is illustrated in Figure 1.

Header				Body		
CLA	INS	P1	P2	Lc	Data field	Le

Figure 1. Command APDU Structure [3]

Response APDU that sent by a smart card in reply to a command APDU of a host application consists of an optional body and a trailer [3]. The optional body consists of the data field that its length was determined in the previous command APDU's Le byte.

The trailer consists of two bytes, Status Word 1 (SW1) and 2 (SW2) that contain the processing state in the card after executing the command APDU. The structure of response APDU is illustrated in Figure 2.

Body (Optional)	Trailer	
Data field	SW1	SW2

Figure 2. Response APDU Structure [3]

### Smart Card Standards and Specification

Because the proliferation of smart cards depends seriously on the existence of national and international standards and generally recognized specifications, for the past 15 years, a number of smart card standards and specifications have been defined to ensure that smart card issuers, card acceptance device vendors, and application developers can work together [3].

The ISO 7816 that was published by The International Organization for Standardization (ISO) in 1987 is the first set of standard defining the characteristics of integrated circuit cards that uses electrical contacts [4]. The ISO 7816 consists of distinct sections, and each section covers various aspects of smart cards; physical characteristics, dimension and location of contacts, electronic signal and transmission protocols, inter-industry commands for interchange, numbering system and registration procedure for



application identifiers, and inter-industry data elements [4].

Europay International, MasterCard International and Visa International (EMV) have made efforts to create common application standard where the ISO 7816 does not address smart card applications [4]. The EMV specification is based on the ISO 7816 series of standards with additional proprietary features to meet the specific needs of the financial industry such as the cooperative development of financial payments standards [3].

The Global System for Mobile Communications (GSM) defined by the European Telecommunications Standards Institute (ETSI) is a specification that covers smart cards for use in an international terrestrial mobile telephone system [3]. The GSM specifies the digital authorization and authentication procedures and programs that are stored in a smart card or a smaller form called a Subscriber Identity Module (SIM) [1].

## **2.2 Java Card**

Although smart cards have been widely used in many application areas due to their underlying advantage of providing powerful and secure computing capabilities come in simple and portable forms, their limitations such as chip architecture dependent of application development, no standardized high-level application interfaces, small universe of knowledgeable programmers and lack of interoperability have prevented a broader deployment of smart cards.

In 1996, Schlumberger proposed the implementation of a virtual machine for a subset of the Java programming language on a standard 8-bit smart card microprocessor. Base on this proposal, in 1997 SUN launched the Java Card 2.0 API specification, which is compatible with the smart card international standard ISO 7816 [4]. The latest version

of this specification is 2.2 and is now the standard specification for Java Card.

While Java Card technology preserves many of the benefits of the Java programming language such as productivity, security, robustness and portability across different chip architectures, it also has several unique benefits over existing smart cards. New applications can be installed securely at any time limited only by the memory size after a card has been issued. It provides card issuers with the ability to dynamically respond to their customer's changing needs. Also, Java Card technology provides a secure environment to enable multiple applications from different service providers to coexist and cooperate securely on a single card using the applet firewall mechanism and the shareable interface object mechanism.

### **Applet**

An applet is a Java Card application that runs within the Java Card Runtime Environment (JCRE) on card. Although it is different from Java applet that is intended to run within a browser environment, the reason the name applet was chosen for Java Card applications is that Java Card applets can be loaded dynamically onto the Java Card in a distributed network such as the Internet within a secure environment over a Card Accepting Device (CAD) [3]. It can be downloaded anytime onto the cards after the card has been issued while applications in many embedded systems need to be burned into the ROM during manufacture [3]. This makes the Java Card a very powerful platform for developing new applications. Also, the Java Card applet can be burned into ROM together with the JCRE and other system components during the process of card manufacturing as other applications in typical embedded systems [3].

Each applet instance is uniquely identified by the Application Identifier (AID).

The AID controlled by the International Organization for Standardization (ISO) can be 5 to 16 bytes long, and is constructed of two data elements: Resource Identifier (RID) that is a 5 bytes value and Proprietary Identifier Extension (PIX) that can be from 0 to 11 bytes in length. The structure of the AID is shown in Figure 3 [5].

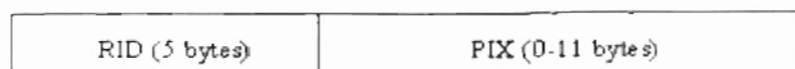


Figure 3. Structure of Application Identifier [5]

ISO controls the assignment of RIDs to the application providers, so each application provider has a unique RID. Then, each application provider can add its own PIX after RID in AID. In the Java Card platform, each applet instance is identified and selected by its unique AID that is constructed by concatenating the company's RID and the PIX for that applet [3].

### **Java Card Language Subset**

Because of limited memory resource and computing power, the Java Card platform supports only a subset of the language features defined in the Java Language Specification. For example, dynamic class loading, security manager, garbage collection and finalization, thread and cloning are unsupported Java language features on the Java Card platform [4]. Also, Java Card platform does not support arrays of more than one dimension, and types long, float, double, char and string [3]. Notice that the int keyword and its 32-bit integer data type are optionally supported, and a Java Card Virtual Machine (JCVM) that does not support int data type rejects programs using that type [3].

### **Java Card Virtual Machine**

Java Card Virtual Machine (JCVM) is implemented as two separate pieces as illustrated in Figure 4.

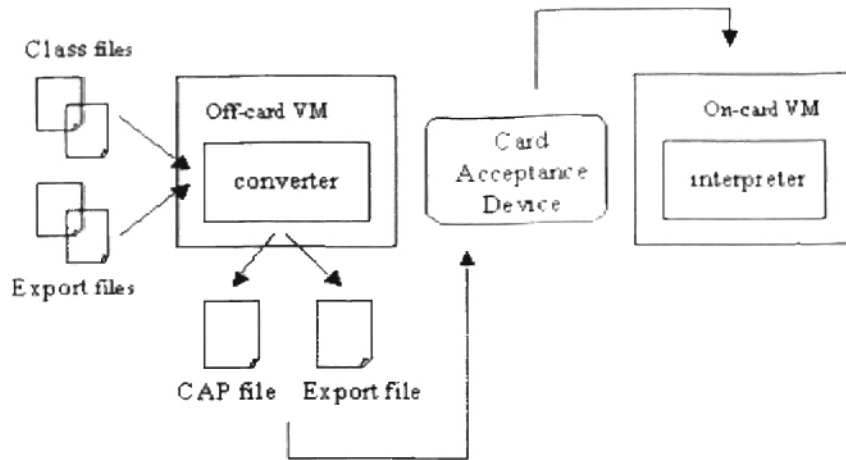


Figure 4. Java Card Virtual Machine [3]

In the off-card portion of the JCVM that runs on a PC or workstation, the converter converts a package composed of all class files that are generated by compilation of the source code to a converted applet (CAP) file that contains an executable binary representation of the classes in the package [3]. A CAP file is the loadable and installable unit on the Java Card platform, and it consists of classes that make up a Java package [3]. That is, the CAP file format is the form in which a card application is loaded onto Java Cards [3]. Also, the converter takes as input one or more export files and produce another export file. An export file that can be thought of as the header files in the C programming language contains name and link information for the contents of other packages that are imported by the classes being converted [9].

The on-card portion of the JCVM includes interpreter that provides runtime support of the Java language model and thus allows hardware independence of applet code [3]. The interpreter executes bytecode instructions and ultimately executes applets [3].

In a PC or workstation, the Java Virtual Machine (JVM) runs as an operating system process, and when the operating system process is terminated, the JVM and

objects that were created in RAM are automatically destroyed [9]. Unlike the JVM, the JCVM on a card appears to run forever even when no power is supplied [9]. When power is removed, the JCVM on the card only stops temporarily, and when the next time the card is energized, the JCVM starts up again and execute from the beginning of the main loop instead from the exact point where it lost power [3].

### **CAP File Format**

A CAP file generated by the converter consists of a set of components, which describe a Java package, and utilizes the JAR file format. Each component in a CAP file describes a set of element in the Java package defined [8]. In addition to the components, the CAP file also contains a manifest file. The manifest file provides additional human-readable information, such as the creation time of the CAP file, the version of the converter, and the provider of the converter, and it can be used to facilitate the distribution of the CAP file.

### **Java Card Framework**

The Java Card Framework is compatible with formal international standards, especially ISO 7816 standard, and defines a set of Application Programming Interface (API) classes for developing applets that provide system services to those applications. It provides applet developers with a relatively easy and straightforward programming interface by hiding the details of the smart card infrastructure [4]. Also, it allows applets written for one Java Card enabled platform to run on any other Java Card enabled platform.

### **Java Card Runtime Environment**

The Java Card Runtime Environment (JCRE) serves as the operating system of

the Java Card [3]. It is responsible for card resource management, network communication, applet execution, and on-card system and applet security [3]. The JCRE that sits on top of a specific integrated circuit and native operation system implementation contains on-card portion of the JCVVM, JCRE system classes, Java Card framework (and industry specific extensions), and installer

The system classes are responsible for managing transactions, communication between the host applications and applets, and instance creation, selection and deselection of applets [3]. The industry specific extensions supplied by a specific industry or business are add-on libraries to provide additional services or to refine the security and system model for industry or business [3].

The installer downloads applets onto the card after the card is made and issued. It cooperates with an off-card installation program that resides on PC or workstation. The off-card installation program transmits a CAP file that contains an executable binary representation of the classes to the installer running on the card via a card acceptance device, then the installer writes the binary into the card memory, links it with the other classes that have already been placed on the card, and creates an instance of the applet [3]. More detail about the installer and off-card installation program will be given in the next chapter with applet installation process. The JCRE is illustrated in Figure 5.

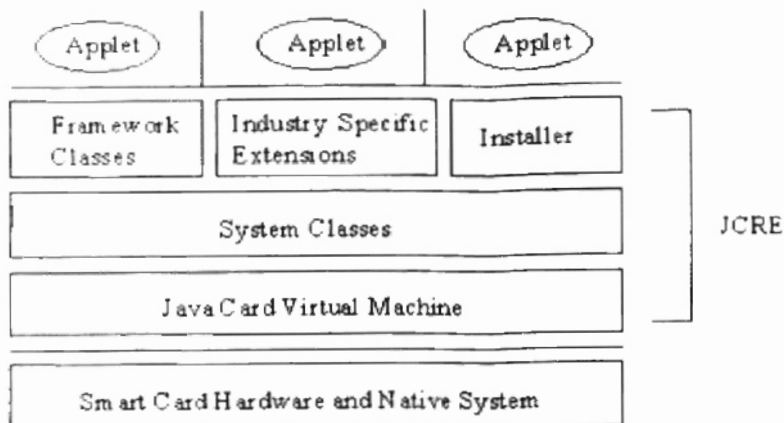


Figure 5. Java Card Runtime Environment [3]

The JCRC is initialized at card initialization time, and during this process the JCRC creates its objects for providing the JCRC service and managing applets and store them in EEPROM that can hold the information even when power is removed from the card [3]. When power is removed, the JCVM is only suspended, and the state of the JCRC and applets on the card are preserved in EEPROM [3]. The next time the card is inserted, the JCVM restarts the JCRC by loading data from EEPROM [3].

### Applet Execution

Every applet is implemented by creating a subclass of the class `javacard.framework.Applet`. The base Applet class is the super class for all applets residing on a Java Card, and it defines `install`, `register`, `select`, `process`, and `deselect` methods. However, the base Applet class provides only the default behavior for these methods, so an applet needs to override some or all of these methods to implement its function. Table 2 lists the methods invoked by the JCRC during applet creation and execution.

Public static void	install (byte[] bArray, short bOffset, byte bLength)  The JCRE calls this method to create an instance of the Applet subclass. It is similar to the main method in a Java application, and the arguments to the install method carry the applet installation parameters.
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 using its AID specified in the array bArray.
Public Boolean	select ( )  The JCRE calls this method to inform the applet that it has been selected. An applet remains in a suspended state until this method is explicitly called by JCRE.
Public abstract void	process (APDU apdu)  The JCRE calls this method to instruct the applet to process an incoming APDU command. On receiving an APDU command, this method decodes the APDU header and calls a service method to execute the requested function. An applet must directly or indirectly override this method.
Public void	deselect ( )  The JCRE calls this method to inform the currently selected applet that another applet will be selected. The deselected applet will be remained in a suspended state until the next time it is selected.

Table 2. The Methods in the Class javacard.framework.Applet [3]

After an applet is correctly loaded into the card and linked with the Java Card Framework and other libraries on the card, an applet's life starts when the JCRE calls the applet's install method to create an instance of the applet in EEPROM. The JCRE then calls the applet's register method to register this applet instance with the JCRE using the applet's AID. On successful return from the install and register method, the applet is ready to be selected and to process the upcoming APDU commands.

The JCRE is a single thread environment - means that only one applet is running at a time [3]. An applet on the card remains in a suspended state until it is explicitly



selected. Figure 6 illustrates execution states of an applet.

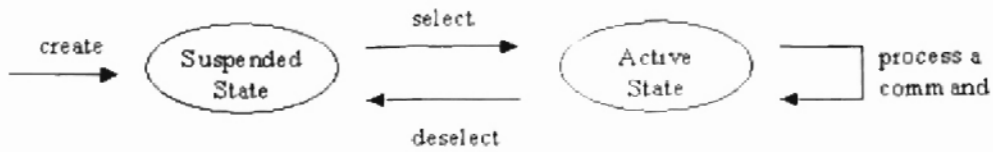


Figure 6. Applet Execution States [3]

A host application sends a select APDU command to the JCRE. The JCRE suspends the currently selected applet and invokes the applet’s deselect method to perform any necessary cleanup. Then, the JCRE calls the select method of the newly selected applet whose AID is specified in the select APDU command as the currently selected applet. Once the typical applet is selected, it waits for an application running on the host side to send a command APDU. The applet then executes the command APDU and returns a response APDU to the host via the JCRE. All subsequent APDUs are forwarded to the current applet until a new applet is selected or the card is removed from the card acceptance device, and once the applet is deselected, the applet remains in a suspended state until the next time it is selected. Figure 7 illustrates applet communication.

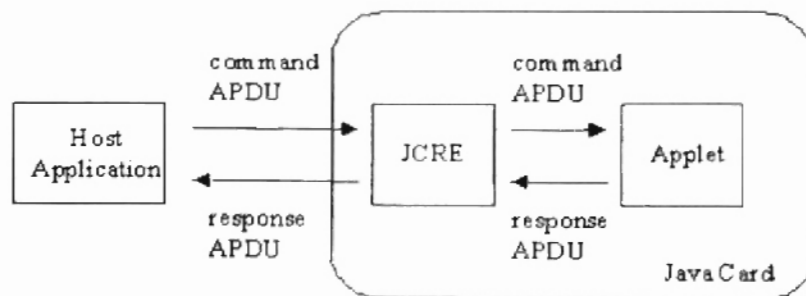


Figure 7. Applet Communication [3]

### Applet Object

In the Java platform, objects are created in RAM, and they are automatically

destroyed when the Java Virtual machine exits. In Java Card technology, however, most JCRE and applet objects are created in EEPROM using the new operator, so their information can be preserved even when power is removed. Those objects are called the persistent objects, and their reference and data are preserved across CAD sessions [3].

For security and performance reasons, applets sometimes require objects that contain temporary data that need not be persistent across CAD sessions. Those objects are called transient objects, and applets can create the transient objects in RAM by invoking the Java Card API [3]. The data of the transient object is cleared and reset to their default value when certain events such as power loss, card reset or applet deselection are occurred. The reference of the transient objects, however, is saved in a persistent file, so the next time the card is energized, the applet uses the same object reference to access the transient object even though the object data from the previous session are lost [3].

### **Java Card Transaction**

Java Card technology guarantees atomic transactions, so the updated field either gets the new value in case of normal execution or is restored to the previous value in case of accidental events such as power loss in the middle of a transaction or program errors that might cause data corruption [9]. An applet, sometimes, needs to update several different fields atomically, and the Java Card platform provides a transactional model with commit and rollback capability to guarantee that complex operations can be accomplished atomically [9]. Either all updates are completed correctly and consistently, or their partial results are not put into effect and restored to their previous values. Therefore, the atomicity of all the updates is ensured.

## Applet Firewall

Multiple applets from different vendors can coexist in a single Java Card, so each applet should be protected from others because an applet might contain highly security information such as electronic money and private cryptographic key. In the Java Card platform, the applet firewall provides a secure execution environment between different applets in the same card [8].

The applet firewall confines an applet to its own designated area called contexts, and the firewall is the boundary between one context and another. When an applet instance is created, the JCRE assigns it a context. Recall that on the Java Card platform, the loading and installable unit is a CAP file, and a CAP file consists of classes that make up a single Java package. At this point, the context is essentially a group context, so all applet instances of a single Java package share the same group context. There is no firewall between two applets instance in a same group context, so object access between applets in the same group context is allowed unless members and methods of the objects are declared as private. However, the applet cannot reach beyond its context to access the contents or behaviors of objects in a different context. In addition, the JCRE maintains its own JCRE context, and accessing from the JCRE context to any applet's context in the same card is allowed while accessing from an applet's context to the JCRE context is denied by the firewall. Figure 8 illustrates the applet firewall.

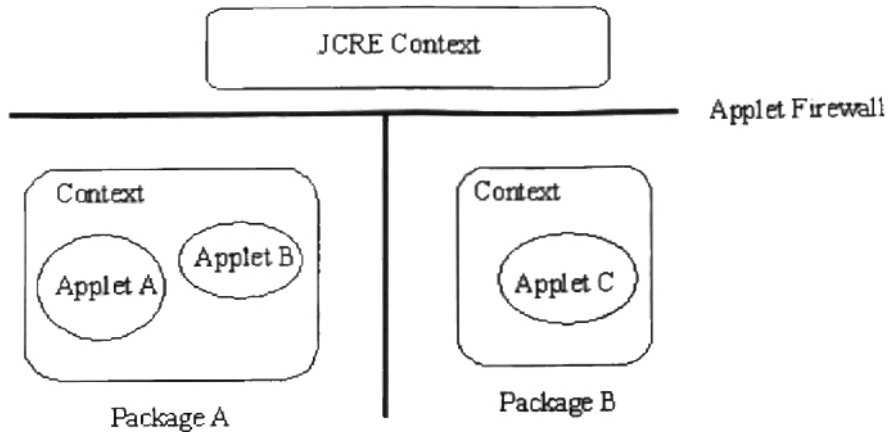


Figure 8. Applet Firewall [8]

At any point in time, there is only one active context within the execution of the JVM [8]. This can be either an applet's group context or the JCRE context. When a new object is created, its ownership is assigned to the current active context, and the object can be accessed from within that context, that is, by all applet instances in its owning context. If the JCRE context is the currently active context, the object is owned by the JCRE.

All bytecodes that access an object are checked at runtime [8]. A private instance method, for example, cannot be invoked from outside the object. Also, when an object is accessed, the object's owning context is compared to the currently active context in order to determine if the access is allowed. If the contexts do not match, the access is denied and a `SecurityException` is thrown.

### Context Switch

As mentioned at the previous section, there is only one active context at any time within the JVM, and all object accesses are denied by the applet firewall if the owning context of the object being accessed is not the same as the currently active context. However, for some situations where applets need to execute cooperatively, context

switches are occurred within the JCVM. Context switches only occur during invocation of and return from certain methods of an object owned by a different context, and the result of a context switch is a new currently active context [8].

During a context switch, the current context and object owner information is pushed on an internal JCVM stack, and a new context becomes the currently active context. The invoked method is now executing in this new context. When the method exits from a normal return or an exception, the JCVM performs a restoring context switch. The original context is popped from the stack and is restored as the currently active context.

Recall that the JCRE context to any applet's context in the same card is allowed. For example, when the JCRE receives an APDU command from a host application, it invokes the currently selected applet's select, deselect, or process method. When the JCRE invokes an applet's those methods, the JCRE context is switched to the applet's context [3]. Also, on return from the applet's method, the JCRE context is restored [3].

Recall that applets are not allowed to access the JCRE context. However, a secure computer system must have a way for users who don't have privilege to request system service that are performed by privileged system routine, and this requirement is accomplished by using JCRE entry point objects in the Java Card platform [8]. The JCRE entry point objects that are objects owned by the JCRE context have been flagged as containing entry point methods. The entry point designation allows the public methods of such objects to be invoked from any context through the firewall, and when such invocation occurs, a context switch to the JCRE context is performed. Notice that only the public methods of the JCRE entry point objects are accessible, and the fields of

these objects are still protected by the firewall [3]. The JCRE-owned AID instance that the JCRE creates to encapsulate an applet's AID when the applet instance is created is an example of the JCRE entry point object.

Java Card technology enables multiple applications from different service providers to cooperate securely on a single card via the sharable interface mechanism. During the process of cooperation between applets that reside on different contexts, context switches are also occurred. The sharable interface mechanism will be discussed in the next chapter.

### **OpenCard Framework**

OpenCard Framework (OCF) developed by the OpenCard consortium is the host application framework providing common interface for both the card readers and the applications in the card [10]. OCF implemented in the Java programming language is designed to reduce dependence on each of terminal vendors, card operating system providers, card issuers and application developers.

### **2.3 Message Digest**

When security information is transmitted over the network, it must be protected from unauthorized disclosure – that is called as data confidentiality, and it can be accomplished by encryption scheme. Encryption is the process of converting some information from an easily understandable formant into an unintelligible form in such a way that the original data can be obtained only by using the decryption process [11]. Triple-DES (Triple Data Encryption Standard) and public key scheme are the most popular encryption algorithms. The main difference between two algorithms is that same keys are used for both encryption and decryption in Triple-DES while the public key

scheme uses two different keys, called as a public key that is used for encryption and a private key that is used for decryption.

A Message digest algorithm has much in common with techniques used in encryption, but to a different end; verification that data have not been modified since the signature was published – that is called as data integrity that pertains to protection of information from alteration by unauthorized means.

### One-Way Hash Function

A one-way hash function takes an arbitrary length plaintext as input and outputs a relatively small fixed-length string called hash value. Assume  $F$  as a one-way hash function and  $M$  as a data message to be protected. Then, we can obtain a hash value  $H$  of the data message by applying the function on the data message. That is  $H = F(M)$ .

Figure 9 illustrates one-way hash function.

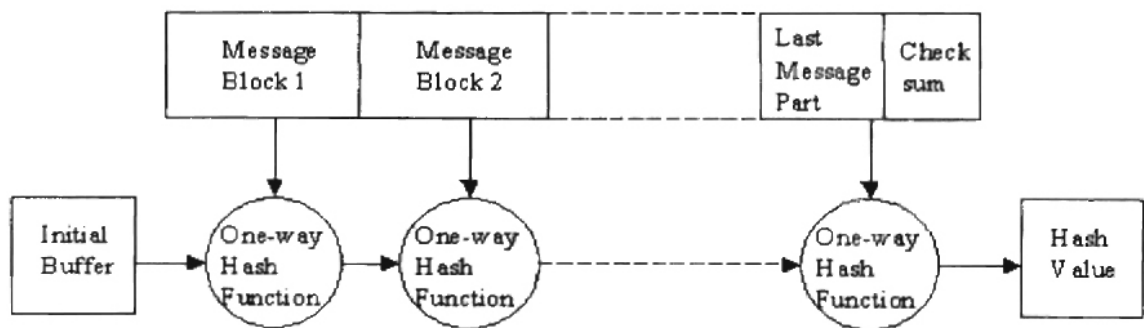


Figure 9. One-Way Hash Function [13]

Each unique message fed to a one-way hash function produces a unique hash value of the entire message, and it is virtually impossible to generate the original message from the hash value - that is the reason why it is called a one-way hash function [12]. The purpose of a one-way hash function is to provide the means for the receiver to detect whether the message has been modified by unauthorized users [11].

There are some basic requirements for one-way hash functions [13]. First, the one-way hash function can be applied to an arbitrary length of data message. Second, the hash value from the one-way hash function has a fixed size. Next, it is impossible to find the data message from the hash value. Finally, it is very hard to find a data block called collision that has the same hash value with the data message.

### **Message Digest Algorithms**

MD2, MD4 and MD5 are message digest algorithms developed by Ronald L. Rivest [14]. All message digest algorithms take a data message of arbitrary length as input and produce a 128-bit message digest of the data message, but MD2 was designed for 8-bit machines while MD4 and MD5 were optimized for 32-bit machines [14].

MD2 that was developed in 1989 is designed for use with Privacy Enhanced Mail (PEM). The data message is first padded so its length in bytes is divisible by 16, then a 16-bit checksum is appended. The message digest is computed on the resulting message. However, Rogier and Chauvaud found that a collision for MD2 could be constructed if the calculation of the checksum is omitted [14].

MD4 that was developed in 1990 also requires padding, but to a multiple of 512 bits. The padding always includes a 64-bit value that indicates the length of the unpadded message, and this adds a significant measure of security over MD2 because if it is difficult to produce two messages that have the same 128-bit message digest, it is all the more difficult to do it with two messages that have the same length [15]. However, Dobbertin showed how a collision for MD4 could be found in under a minute on a typical PC, so MD4 now might be considered broken [14].

MD5 that is an extension of the MD4 was developed in 1991 for use with digital



signature applications where a large message has to be compressed securely before being signed with the private key [16]. The algorithm consists of four distinct rounds while MD4 consists of three distinct rounds, so it is more secure than MD4 while it is slightly slower. Van Oorschot and Wiener estimate a collision search machine designed specifically for MD5 with brute-force search could find a collision for MD5 in 24 days on average [14].

### **Secure Hash Algorithm**

Secure Hash Algorithm (SHA) developed by the U.S. National Institute of Standards and Technology (NIST) in 1994. SHA produces a 160-bit hash value, and its design is very similar to MD4. The algorithm is slightly slower than MD5, but the larger hash value increases its protection ability [17].

### **3. SECURE OBJECT SHARING**

Beyond minimum-security protections enforced by the on-card installer and the Java Card Runtime Environment in the Java Card platform, Java Card technology does not standardize applets installation policy [3]. For this reason, enhancement in on-card verification of downloaded applets is still under consideration. In this thesis, the on-card installer with a one-way hash function is proposed to support on-card verification of downloaded applets that cooperates with other applets from different service providers on a single card. We first go through the applet installation process and address the possibility of illegal installation of a malicious applet in section 3.1. In section 3.2, we show how multiple applets from different providers can cooperate on a single card and point out the need for improvement in on-card verification of applets when they cooperate. Then, our proposed installer and secure object sharing are detailed in section 3.3.

#### **3.1 Applet Installation**

In Java Card technology, applets can be either downloaded anytime onto the cards after the card has been issued or burned into ROM together with the JCRE during the process of card manufacturing. In this thesis, however, applet installation process refers to the process of loading applet classes via a Card Acceptance Device (CAD) onto a Java Card and creating an applet instances to bring the applet into a selectable and executable state.

Applet installation process consists of conversion, verification and a CAP file installation on the card. During conversion process, the converter takes a package as input, and outputs a CAP file that contains an executable binary representation of the

classes in the package. After conversion, the verifier performs static checks on the CAP file before it is loaded onto a Java Card. For a CAP file installation process, Java Card installer cooperates with an off-card installation program, and together, they load a CAP file via a CAD onto a card and create one or more applets instances in the card's persistent memory such as EEPROM. Applet installation process is illustrated in Figure 10.

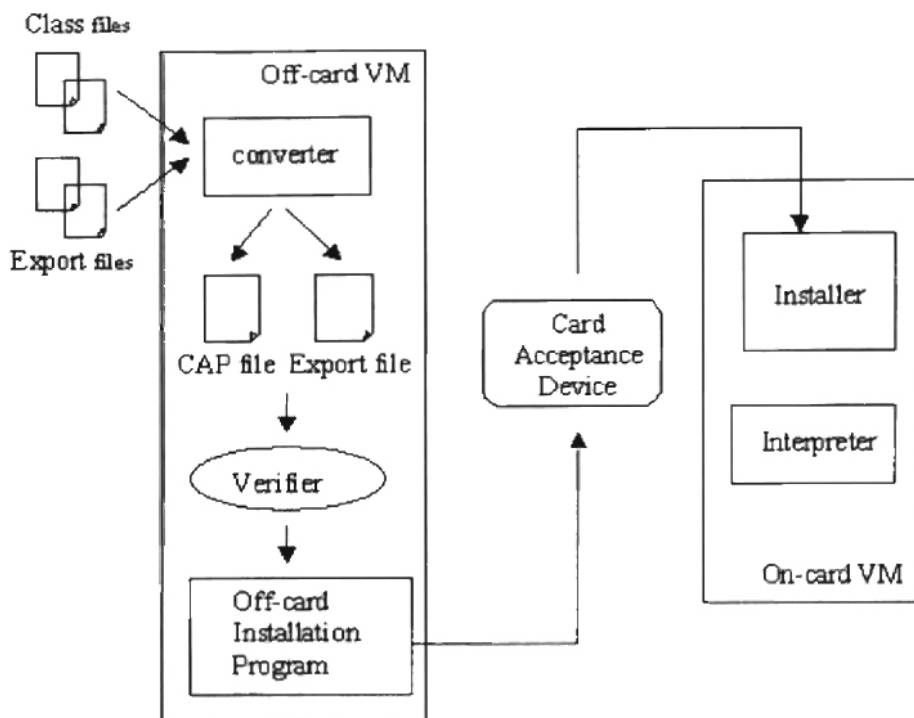


Figure 10. Applet Installation Process

The applet installation process is transactional, so if an error or power loss occurs during applet installation, the installer discards the CAP file and any applet instances created during installation, and recovers the space and the previous state of the JCRE [3].

### Converter

Although a trustworthy compiler can ensure that Java source code does not violate safety rules, class files could come from a network that is unsecured [3]. The converter is

the front end of the off-card virtual machine, and it takes as input all of Java class files for a package and converts them to a CAP file that is the loading and installable unit on the Java Card platform and consists of classes that make up a Java package. It also takes one or more export files that can be thought of as the header files in the C programming language as input, and generates an export file for the converted package. During conversion, the converter performs tasks that a Java virtual machine in a desktop environment would perform at class-loading time such as checking for Java Card language subset violations, performing static variables initialization, optimizing bytecode, allocating storage and creating virtual machine data structures to represent classes [3].

### **Verifier**

The correctness and integrity of a CAP file cannot be taken for granted even though a CAP file generated by a trustworthy converter will immediately be loaded onto Java Card in a secure environment [3]. The verifier that runs off card due to the limited memory space and computing power of a smart card verifies a CAP file before it is loaded onto a Java Card. That is, the verifier provides a means to assert that a CAP file conforms to the Java Card specification, providing additional assurance that the executable code in a CAP files will not compromise the integrity of JCVM [18].

### **Off-Card Installation Program and Installer**

Applet installation on Java Card is completed through the cooperation of an off-card installation program and the on-card installer. The off-card installer program transmits the executable binary in the CAP file to the installer running on the card with a sequence of APDU commands by way of card acceptance device (CAD). When the CAP

file is read in, the installer first checks to see whether the card can support the CAP file by checking such as availability of the card's memory resource [3]. Then, the installer writes the CAP file content into the card's persistent memory such as EEPROM and links it with other classes that already reside on the card.

In the last step during installation, the installer creates instances of applets in EEPROM by calling the applets' install methods, and when instances of the applets are created, a context is assigned to the instances of the applets. Recall that a context is essentially a group context, so instances of multiple applets defined in the same package share one context. Also, the installer registers the applets' AIDs with the JCRE by calling their register methods. Notice that each applet within a Java Card is associated with unique AID, so if the same AID has been previously successfully registered on the JCRE, an error occurs and the installer discards the CAP file and destroys any applet instances created during installation [19].

On successful return from the install and register methods, applet installation process is completed, and the applets are now ready to be selected and to process the upcoming APDU commands.

### **Illegal Installation**

Applet installation security consists of two levels: standard security protections enforced by the converter, verifier and the JCRE, and security policies dictated by the issuers [3]. This means that beyond the minimum-security protections such as Java Card language subset violation enforced by the converter and insufficient card's memory and illegal AID usage enforced by the JCRE, Java Card technology does not standardize the installation policy, so the issuers have the flexibility to define security policies.

The simplest scheme of protection is to authenticate an off-card installation program by using PIN – thus providing a measure of trust to the CAP file provider and the content of the CAP file [3]. A more powerful scheme can use digital signature and data encryption [3]. For example, any Java Card file such as class file, export file or CAP file can be encrypted and digitally signed to ensure integrity and to provide the identify of its provider during transportation between development and on-card installation. To build such strengthened security scheme, a card issuer needs to define security polices using key management and cryptographic mechanisms.

However, the Java Card installer does not verify a CAP file [3]. This means the correctness and integrity of a CAP file are verified off-card, and the installer on the Java Card platform does not perform most of the traditional Java verifications at class-loading time. Therefore, it is possible a malicious applet to be installed onto a card via illegal applet installation process due to lack of a CAP file verification power in the Java Card installer. For this reason, enhancement in on-card verification of downloaded applets is still under consideration. Our proposed on-card installer with a one-way hash function support on-card verification of downloaded applets that cooperates with other applets supplied by different service providers on a single card. More detail about the proposed installer will be given in section 3.3.

### **3.2 Object Sharing**

In Java Card technology, multiple applets from different vendors can coexist in a single card, and the applet firewall enables that each applet that may store highly sensitive information can be protected from other applets. An applet is prevented from reaching the contents of objects owned by other applets residing on a different context by

the firewall.

In real card world, however, applets from different service providers often need to execute cooperatively. For example, consider cooperation between an airline loyalty applet and an electronic purse applet - when buying a flight ticket with the electronic purse, add miles to the airline loyalty program. In this case, shared information from the airline loyalty applet and the electronic purse may be received by the airline loyalty applet and the electronic purse applet. To support cooperation between applications from different application providers on a single card, Java Card technology provides well-defined and secure shareable interface mechanism.

### **Shareable Interface Mechanism**

Shareable interface mechanism is a feature in the Java Card API to enable object sharing between applets [9]. An object instance of a class implementing one or more shareable interfaces is called a Shareable Interface Object (SIO), and these interface methods can be invoked from one context even if the SIO is owned by an applet in another context.

To cooperate with other applets provided by different vendors, the behavior of a SIO of an applet, such as name of methods defined in the SIO, needed parameters type and expected results, should be available for other application providers. However, the class type of SIO is not exposed to other application providers even though the behavior of an SIO is available for other application providers. During sharing process, only methods defined in a server SIO are presented to client applets in another context, and all other members and methods of the server SIO are protected by the firewall. Therefore, with shareable interface mechanism, an applet can execute cooperatively with other

applets from different application providers without worrying about exposition of their sensitive data.

The object sharing process can be described as a client-server relationship as below [3]. Let's consider cooperation between an airline loyalty applet (server applet) and an electronic purse applet (client applet) supplied by different service providers. The wallet applet stores electronic cash, and the money can be spent to purchase goods. Similar to the wallet applet, the airline loyalty applet stores values – the miles the card holder has traveled. Under a co-marketing deal between two service providers, for every dollar spent using the wallet applet, one air mile is credited to the airline loyalty applet. Suppose that the electronic purse applet now requests air miles to the airline loyalty applet after its debit transaction for purchasing goods. Following are the steps during cooperation between two applets residing on different contexts, and Figure 11 illustrates the object sharing process.

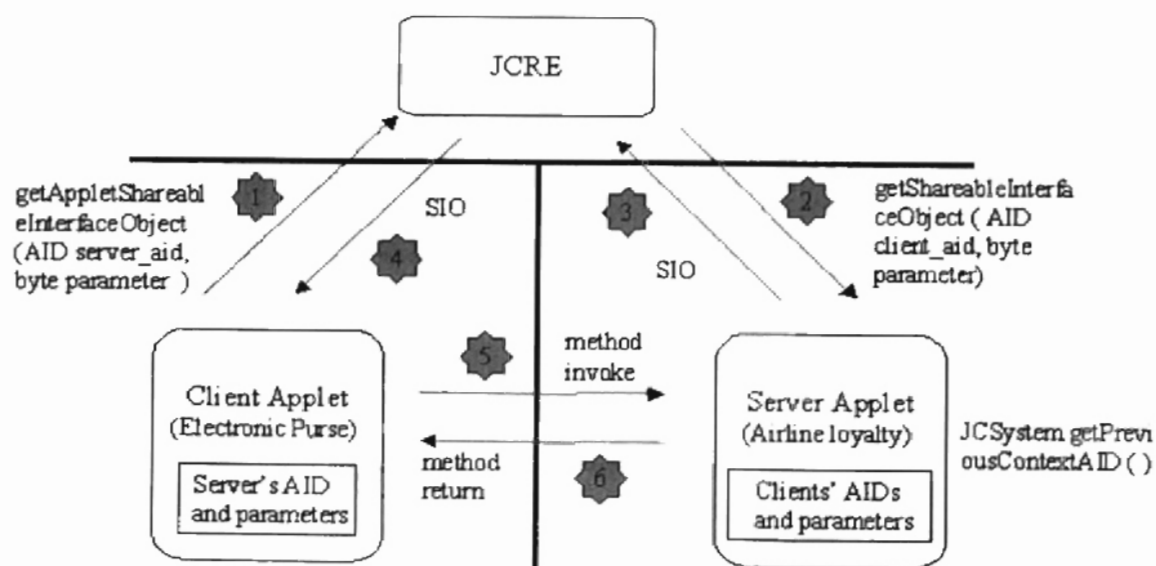


Figure 11. Object Sharing Process

In step 1, to get the airline loyalty applet's SIO, the electronic purse applet that



resides on a different context calls the method `JCSystem.getAppletShareableInterfaceObject` with the airline loyalty applet's AID and a parameter. In here, the sever applet's AID and the parameter are programmed into the client applet code before the client applet is installed on the card. The parameter can be used later as a security token, which carries a secret shared by the server and the client. Also, recall from section 2.2 that there is only one active context at any time within the JCVm - either an applet's group context or the JCRE context, and context switches occur during invocation of and return from certain methods of an object owned by a different context. Therefore, when the electronic purse applet invokes the method `JCSystem.getAppletShareableInterfaceObject` that belongs to the JCRE, a context switch occurs from the context on which the electronic purse applet resides to the JCRE context.

In step 2, the JCRE identifies the electronic purse applet and the airline loyalty applet with their AIDs that were registered on the JCRE when the applets were installed onto the card. After the JCRE ensures that the airline loyalty applet is already residing on the card, the JCRE invokes the airline loyalty applet's `getShareableInterfaceObject` method. In the server's `getShareableInterfaceObject` method, the JCRE replaces the first argument with the electronic purse applet's AID and passes along the same parameter supplied by the electronic purse applet. Now, the airline loyalty applet's group context is a currently active context.

In step 3, with the electronic purse applet's AID and the parameter, the airline loyalty applet determines validation of the client applet. In here, the client applet's AID and the parameter are programmed into the server applet code before the server applet is installed on the card. If the airline loyalty applet agrees to share its SIO with the

electronic purse applet, then the airline loyalty applet returns its SIO to the JCRE, and a context switch occurs.

In step 4, the JCRE returns the server's SIO to the electronic purse applet, and at the end of the step 4, the electronic purse applet's group context is restored as a current active context.

In step 5, once the electronic purse applet gets the SIO from the airline loyalty applet, the client applet requests air miles to the server applet by invoking a service method of the airline loyalty SIO. During the method invocation, a context switch is occurs, and the airline loyalty applet becomes the currently active context.

In step 6, When the service method is invoked from the electronic purse applet, the airline loyalty applet should verify the client again because the electronic purse applet that originally requests the SIO could break the contract and share the SIO with a third party without getting the proper permission [3]. To find out the AID of the actual caller, the airline loyalty applet invokes the `JCSystem.getPreviousContextAID` method. After performing the service for the electronic purse applet, the electronic purse applet's group context is restored as a current active context.

Notice here that, for a higher degree security, the server applet developer must define some cryptographic exchange algorithms to verify its clients when the client applet requests service to the applet by invoking one of shareable interface methods [3].

### **Client Applet Verification**

Recall that, to cooperate with other applets from different vendors, the behavior of a shareable interface object (SIO) and the secret parameter of an applet should be available for other application providers. The application providers are now asked not to

share the information with unauthorized means. However, there is no guarantee that the application providers will not make them available worldwide. Once they are revealed, a malicious applet if installed with an illegal installation process can get a SIO from a server applet. At this point, there is no way for a server applet to reject the malicious client applet that impersonates with a valid AID and parameter. The server applet will agree to share its SIO with the malicious because the server applet determines validation of the client applet only by the client's AID and parameter.

In addition, as previously mentioned, after a server applet agrees to share its SIO with a client applet, there may exist some cryptographic exchange algorithms between the server applet and the client applet to determine validation of the client applet. However, the most frequently anticipated security concerns have been developer mistakes and design oversights. Therefore, enhancement in on-card verification of a client applet before a server applet agrees to share its SIO is needed. Our proposed on-card installer with a one-way hash function helps a server applet to determine more surely whether the client applet is a valid or not before the server applet agrees to share its SIO with the client applet. More detail about the proposed installer will be given in the next section.

### **3.3 Enhanced Applet Verification**

Recall from the section 3.1 that, in Java Card technology, the correctness and integrity of a CAP file are verified off-card and the on-card installer does not verify a CAP file at class loading time. Therefore, installation of a malicious applet onto a card may be possible by way of an illegal applet installation process.

Recall from section 3.2 that, to cooperate with other applets provided by different

service providers, the behavior of a shareable interface object (SIO) of an applet should be available for other application providers who are asked not to reveal the shared information such as the behavior of SIO, a secret parameter and existing cryptographic exchange algorithms between a server applet and a client applet. Therefore, it is possible that the information may be shared with unauthorized means. Once the malicious applet is installed onto a card via an illegal applet installation process, it can get a SIO from a server applet.

Moreover, if a cryptographic exchange algorithm that may exist between a server applet and a client applet can be revealed or broken, then the malicious applet can be allowed to access sensitive data and service of the server applet by invoking one of server SIO. Therefore, enhancement in on-card verification of downloaded applets that cooperates with other applets from different vendors is required urgently.

Our approach using the on-card installer with a one-way hash function enhances on-card verification of downloaded applets that cooperates with other applets supplied by different vendors on a single card. In our novel approach, a malicious applet if installed with an illegal installation process cannot access the SIO of a server applet. Any client whose hash value is not stored in the server code will not be granted access.

### **Installer with a One-Way Hash Function**

In this thesis, the installer with a one-way hash function is proposed to support on-card verification of downloaded applets that cooperates with other applets supplied by different application providers on a single card. While the installer in the Java Card platform registers only an applet's AID on the JCRE, the proposed installer registers not only an applet's AID, but also an applet's hash value that is computed by the proposed

installer during the applet installation process.

Recall that, during the applet installation process, when a CAP file is read in, the installer writes the CAP file content into EEPROM, and in the last step of the installation, the installer invokes JCRE entry point methods to registers an applet's AID with the JCRE. However, with the proposed installer, while a CAP file is read in, the proposed installer computes a hash value of the CAP file contents before it write them into EEPROM. Also, in the last step of the installation, the proposed installer registers with the JCRE not only an applet's AID but also the CAP file's hash value computed by the proposed installer during the CAP file is read in.

Recall from section 2.2 that a CAP file contains not only a set of components that describe a Java package but also a manifest file that provides additional human-readable information such as the creation time of the CAP file and the version of the converter. For this reason, each CAP file in itself does not have an identical hash value even though each CAP files is created from the same Java package. However, a hash value of a set of components in each CAP file is identical if each CAP file is created from the same Java package. Therefore, when a CAP file is loaded onto a card, our proposed installer generates a hash value of a set of components in the CAP file. Hence, in this thesis, a hash value of the CAP file means a hash value of a set of components in the CAP file.

Once an AID of an applet and a hash value of a CAP file are registered in the JCRE object, applets provided by different vendors can now share their data and service in a more secure manner. With our proposed installer, even though it is still possible a malicious applet to be installed onto a card via illegal applet installation process, it is extremely hard to get a service from a server applet because developing a malicious

applet that can request service to other applets and also has the same hash value is almost impossible.

Notice here that it is safe to assume that the installer is trustworthy as the installer is programmed by the card manufacturer. The installer is burned into a card with on-card portion of the JCVM, JCRE system classes and Java Card framework. To the JCRE, the installer appears to be an applet, so at card initialization time, the installer registers its AID with the JCRE. Later, when the installer registers an applet's AID with the JCRE, the installer needs to invoke JCRE entry point methods, and the JCRE identifies the installer applet by its AID. Therefore, installing a malicious installer with the valid AID onto a card is impossible because the same AID cannot be registered on the JCRE.

For the one-way hash function used in the proposed installer, MD5 or other hash algorithms, which are secure and optimize card resource usage can be chosen. Figure 12 illustrates the applet installation process with the proposed installer.

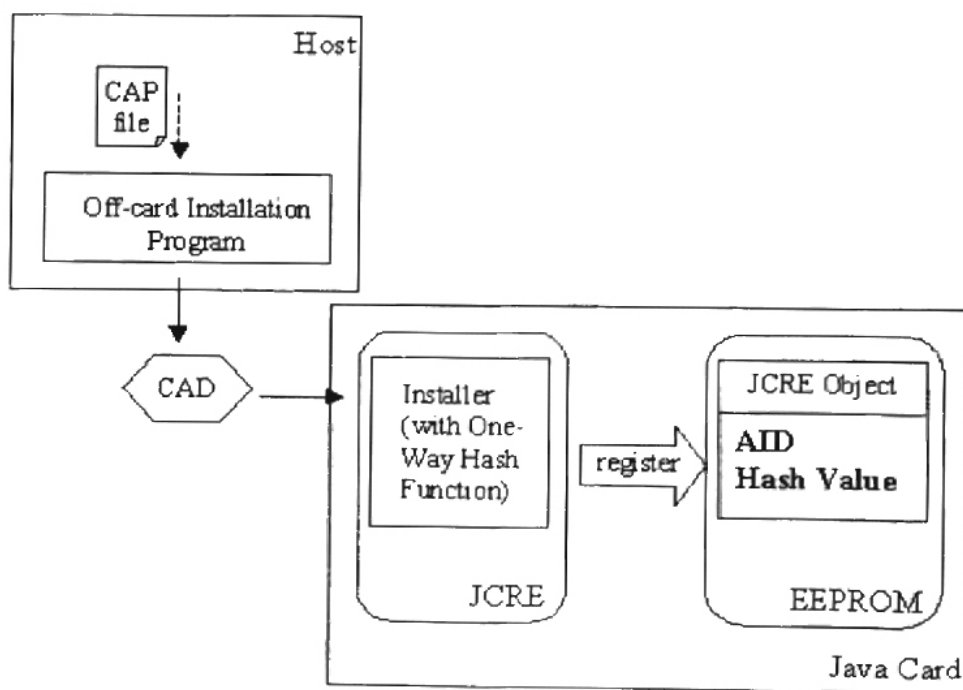


Figure 12. Installer with a One-Way Hash Function

Notice here that our approach assumes that servers are provided the hash values of the client who are likely to request their services. When server applets are being developed by the application developers, the hash values of client applets that may use their services are programmed into the server code before the server applet is installed on the card. Any client whose hash value is not stored in the server code will not be granted access.

### Secure Object Sharing Process

Because the proposed installer registers not only an applet's AID, but also an applet's hash value, the object sharing process of two applets that reside on different contexts is now different as below, and Figure 13 illustrates our secure object sharing process.

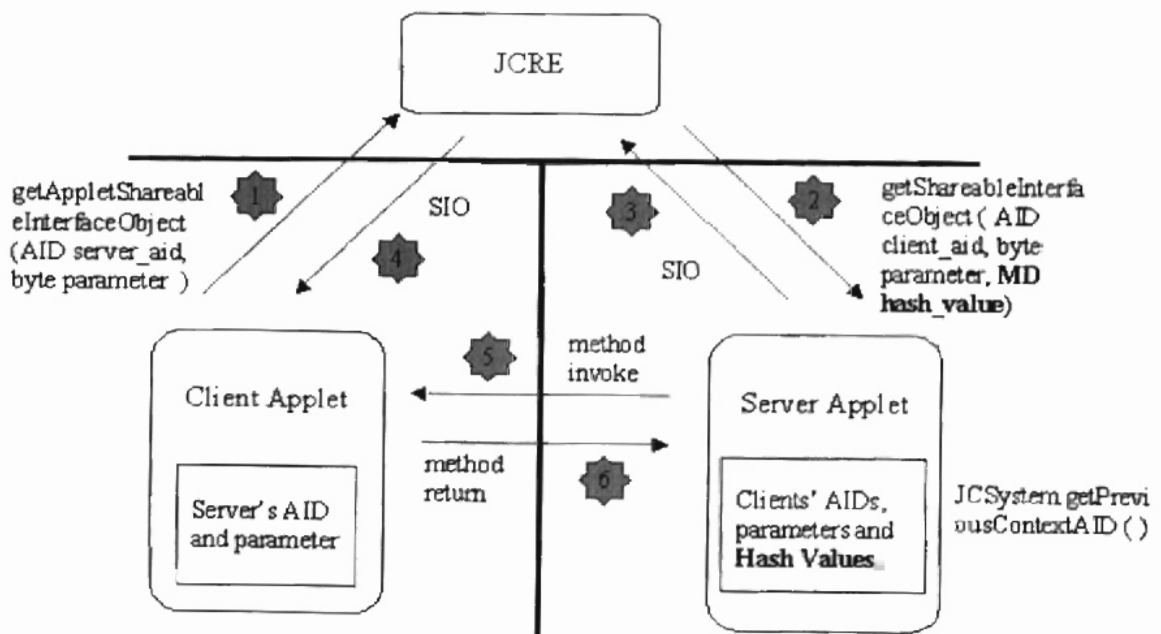


Figure 13. Secure Object Sharing Process

The step 1 is identical with the step 1 in the existing object sharing process illustrated in section 3.2.

In step 2, the JCRE invokes the server applet's `getShareableInterfaceObject` with the client applet's AID, the parameter, and a hash value that is computed by the installer. Our proposed installer registers with the JCRE not only an applet's AID but also the CAP file's hash value computed by the proposed installer when the CAP file is read in.

In step 3, the server applet determines validation of the client applet with the client applet's AID, the parameter, and the hash value. At this point, the server applet now can determine validation of the client applet in more secure manner because it is near impossible for the malicious to have the same hash value as the CAP file containing the valid client applet's classes has. This is also true even if the hash value of the CAP file were revealed. If the server applet agrees to share its SIO with the client applet, then the server applet returns its SIO to the JCRE. In our approach, the hash values of client applets are programmed into the server applet's code before the server applet is installed on the card. Therefore, any client whose hash value is not stored in the server code will not be granted access.

The steps 4, 5 and 6 are identical with the steps 4, 5 and 6 in the existing object sharing process illustrated in section 3.2.

Recall that the hash values of client applets should be saved in a server applet before the server applet is installed on the card. If the application code were burned into the card at manufacture time such a scheme would be a major drawback. However, in the Java Card platform, both client applets and server applets can be downloaded at any time. In the case of a new valid client whose hash value has not been programmed into the server, the up-to-date version of the server can be downloaded very easily. As the Java



Card platform has ability to dynamically respond to a card issuer's changing needs, such problems could be solved very easily.

Therefore, our approach that provides enhanced on-card verification of downloaded applets is practical, and with our approach, an applet can verify other applets supplied by different vendors in a more secure manner along with cryptographic exchange algorithms that may exist between applets.

## **4. SECURE OBJECT SHARING SIMULATION**

To demonstrate the enhanced on-card verification of downloaded applets by using the proposed installer, we simulate the object sharing process between a server applet and a client applet. We simulate the object sharing process not only with our proposed approach that uses hashing for the verification of client applets but also with the existing approach that does not use Hashing. Our objective is to demonstrate that our approach does indeed enhance security.

In this chapter, we first introduce the Java Card 2.2 development kit in section 4.1. Next, in section 4.2, the object sharing process simulation with the existing approach shows that a malicious applet if installed with an illegal installation process can get a SIO from a server applet. Then, in section 4.3, we simulate the object sharing process with our approach that use hashing to demonstrate enhanced on-card verification of a client applet.

### **4.1 Java Card 2.2 Development Kit**

The Java Card 2.2 development kit provided by SUN Microsystems is a collection of tools for designing Java Card technology-based implementations and for developing applets based on the Java Card 2.2 framework [19]. Some tools, which are used for our simulation, in the Java Card 2.2 development kit are described below.

Recall from section 3.1 that the applet installation process consists of conversion, verification and a CAP file installation on the card. For the conversion process, the converter tool is used, and it converts class files that make up a Java package to a CAP file. Also, the converter tool takes as input one or more export files and produce another

export file. The verifycap tool is used for the verification process, and it confirms whether a CAP file is internally consistent. For the CAP file installation process, the scriptgen tool and the apdutool tool are used. The scriptgen tool converts a CAP file into a script file that contains a sequence of APDUs in ASCII format. Then, the apdutool tool reads a script file containing APDUs and sends them to a Java Card. Therefore, both tools work together as off-card installers in Java Card technology. Figure 14 illustrates the usage of tools that is in the Java Card 2.2 development kit.

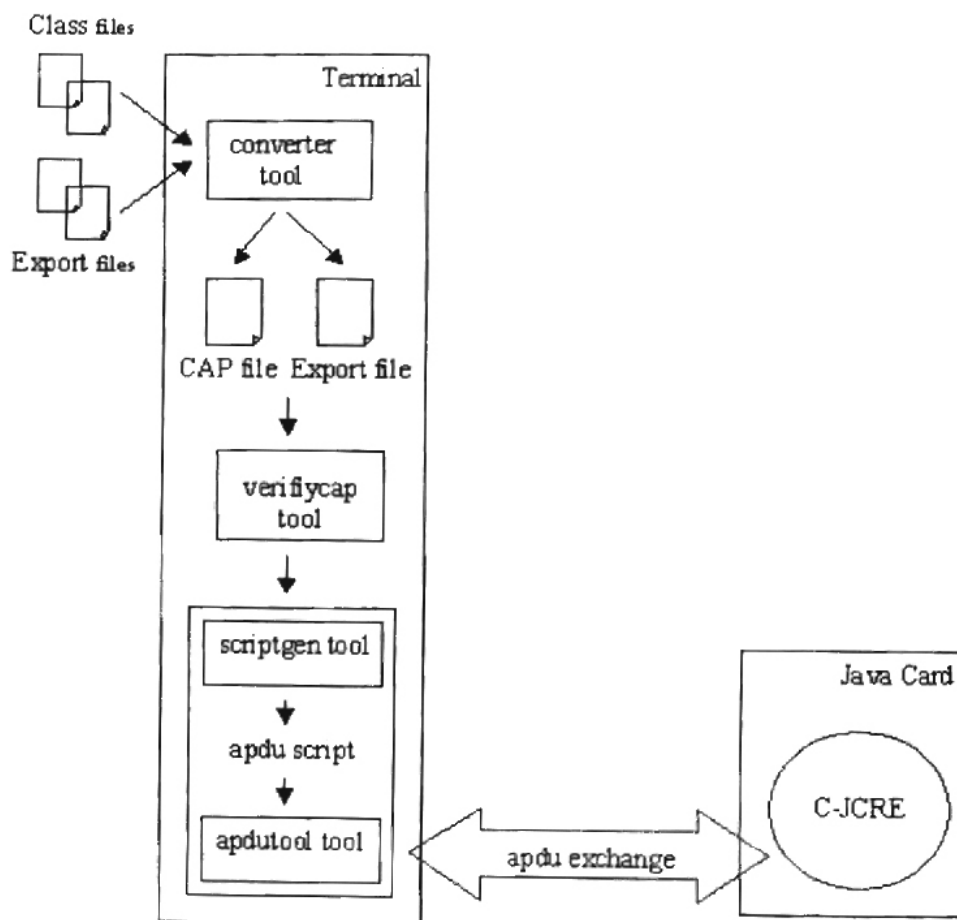


Figure 14. The Usage of Tools in the Java Card 2.2 Development Kit

The C-language Java Card Runtime Environment (C-JCRE) in the Java Card 2.2 Development Kit is a simulator that presents a real Java Card in a card acceptance device

(CAD). It has ability to simulate persistent memory such as EEPROM, and to save and restore the contents of EEPROM to and from the disk file. Also, the on-card installer, interpreter, and Java Card framework are already masked into the C-JCRE, so applets developed by a user can be installed and executed in the C-JCRE. The C-JCRE is supplied as pre-built executable, cref.exe for Windows [19].

#### **4.2 Object Sharing Process Simulation with the Existing Approach**

For the object sharing process simulation with the existing approach, let's consider cooperation between an AirMile applet (server applet) and an EPurse applet (client applet) supplied by different service providers as described in section 3.2. The AirMile applet stores values – the miles the card holder has traveled. Similar to the AirMile applet, the EPurse applet stores electronic cash, and the money can be spent to purchase goods. Assume that under a co-marketing deal with between two service providers, for every dollar spent using the EPurse applet, one air mile is credited to the AirMile applet.

##### **Shared Information**

Recall from section 3.2 that to cooperate with other applets from different vendors, certain information, such as the behavior of a shareable interface object (SIO) and the secrete parameter, should be available for other application providers. The shared information between two applets is below. (Notice that 0x stands for hex.)

- ? The AID of the AirMile applet: 0x0 0x0 0x0 0x0 0xB 0x0 0x0 0x1
- ? The AID of the EPurse applet: 0x0 0x0 0x0 0x0 0xC 0x0 0x0 0x1
- ? The secrete parameter: 0xAF
- ? The behavior of a SIO: void bonusMile (short mileage)

- AirMile applet grants mileages on request from the EPurse applet.
- ? The airmileloyalty package that contains classes of AirMile and AirMileInterface.
- ? The export file of the airmileloyalty package.

The source codes of the AirMile applet and EPurse applet are in Appendix.

### **Illegal SIO Access**

The EPurse application provider is asked not to share the information with unauthorized means. However, there is no guarantee that the provider will not share the information with unauthorized means. Once the shared information is revealed, a malicious applet (client applet) that can get a SIO from the AirMile applet (server applet) can be developed.

For simulating a malicious applet, we add a class variable, which does not affect the compilation and functionality of the client applet, into the source code of the EPurse applet as below.

```
public class EPurse extends Applet {  
    ...  
    private byte for_malicious = (byte)0x0;  
    ...  
}
```

Recall from section 3.1 that the correctness and integrity of a CAP file are verified off-card, and the installer on the Java Card platform does not perform most of the traditional Java verifications at class-loading time. Therefore, it is possible a malicious applet to be installed onto a card via illegal applet installation process due to lack of a CAP file verification power in the Java Card installer.

The malicious applet if installed with an illegal installation process can get a SIO from the AirMile applet as shown in Figure 15 and 16. Notice that every value appeared

in the result box in the host applications is hex format. The source codes of two host applications are in Appendix.

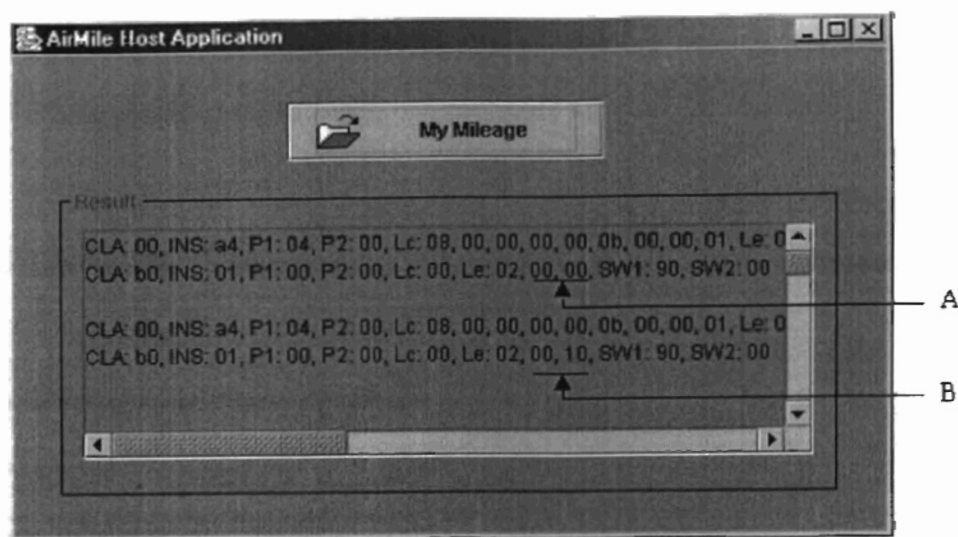


Figure 15. Host Application for the AirMile with Existing Approach

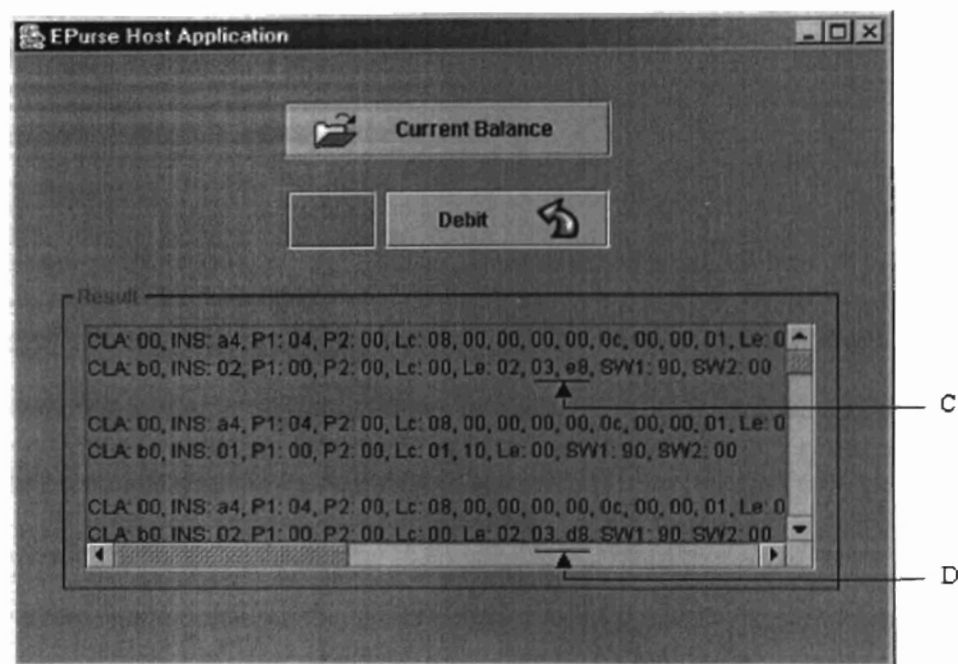


Figure 16. Host Application for the EPurse with Existing Approach

The arrow A in Figure 15 indicates that the initial mileage of the AirMile applet set as zero. Also, the arrow C in Figure 16 shows that the initial balance of the EPurse

applet set as one thousand.

After a debit transaction with sixteen dollars, the arrow D in Figure 16 indicates that the balance in the EPurse applet has been changed. Also, the arrow B in Figure 15 shows that the debit transaction causes to add sixteen mileages in the AirMile applet. This means the malicious client applet have gotten a SIO from the server, and invoke the bonusMile method in the SIO successfully.

As shown in the above simulation, when the malicious applet asks to share a SIO of the AirMile applet, there is no way for the AirMile applet to reject the malicious client applet that impersonates with a valid AID and parameter. Because the AirMile applet determines validation of the client applet only by the client's AID and parameter, the server applet agrees to share its SIO with the malicious applet.

#### **4.3 Object Sharing Process Simulation with Our Approach**

For simulating the object sharing process with our approach, we will consider the same situation environment, which is specified between the AirMile applet and the EPurse applet.

##### **Proposed On-Card Installer**

To simulate the object sharing process with our approach that use hashing for the verification of client applets, an applet called HashInstaller should be installed onto a Java Card before other applets are installed. The HashInstaller generates a hash value of a CAP file when the CAP file is read in and registers the hash value in its instance. Therefore, the HashInstaller applet and the existing Java Card installer residing on the Java Card platform together perform as the proposed on-card installer. The source code of the HashInstaller applet is in Appendix A.

The MD5 algorithm is implemented in the HashInstaller applet to generate a hash value of a CAP file. The reason for choosing MD5 algorithm is that it is not only one of the most robust one-way hash functions, but also as it does not require any large substitution tables, it can optimize card resource usage [16].

### Hash Value of the Client Applet

Recall from section 3.3 that, in our secure object sharing scheme, a hash value of a client applet needs to be programmed into a server applet's code before the server applet is installed. For this, a card issuer - or the AirMile applet provider if it issues the card - provides a hash value generator to the EPurse applet provider. Notice here that a hash value generated from the given hash value generator should be the same hash value that the proposed on-card installer generates when the CAP file is read in.

After the client applet provider developed the EPurse applet along with the shared information, it generates a hash value of the CAP file that contains the EPurse applet classes by using the given hash value generator. Figure 17 is a screen shot of the hash value generator that generated the hash value of the EPurse applet.

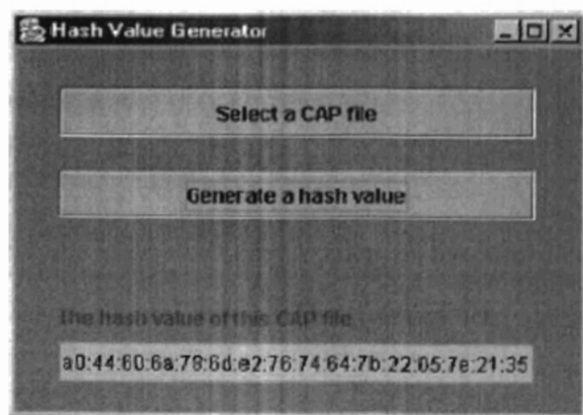


Figure 17. Hash Value Generator with the Hash Value of EPurse Applet

Now, the hash value of the CAP file that contains the EPurse applet classes need



to be programmed into the AirMile applet's code as below before the server applet is installed.

```
public class AirMile extends Applet implements AirMileInterface {
    ...
    private byte[] md_epurse = {
        (byte)0xA0, (byte)0x44, (byte)0x60, (byte)0x6A, (byte)0x78,
        (byte)0x6D, (byte)0xE2, (byte)0x76, (byte)0x74, (byte)0x64,
        (byte)0x7B, (byte)0x22, (byte)0x05, (byte)0x7E, (byte)0x21
        (byte)0x35 };
    ...
}
```

The source codes of the hash value generator is in Appendix.

### **Applet Installation**

Recall from section 3.1 that applet installation on Java Card is completed through the cooperation of an off-card installation program and the on-card installer. The off-card installer program transmits a CAP file to the installer running on the card via a card acceptance device (CAD). To support our simulation, however, we provide our own off-card installation program that transmits a CAP file not only to the Java Card installer but also to the HashInstaller installed onto a card before other applets are installed.

Notice that, in real implementation of our proposal scheme, an off-card installation program will transmit a CAP file only once to the proposed on-card installer, and the proposed on-card installer computes a hash value of the CAP file contents before writing it into EEPROM while a CAP file is read in.

Figure 18 is a screen shot of the application that is for simulating the applet installation process in a terminal. This application contains our off-card installation program, and the source code of the application is in Appendix.

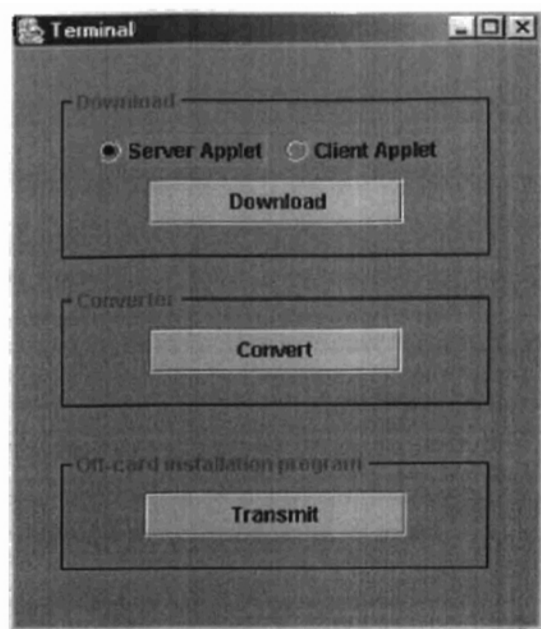


Figure 18. Terminal Application

When a user clicks the Transmit button, our off-card installation program transmits a CAP file to the Java Card installer, and it also transmits the CAP file to the HashInstaller to generate a hash value of the CAP file. Therefore, in the last step of the applet installation process, the Java Card installer registers an applet's AID with the JCRE, and the HashInstaller registers an applet's AID and a hash value of the CAP file in its own instance.

### **Enhanced Client Applet Verification**

Figure 19 illustrates the steps in the simulation during the object sharing process between the AirMile applet and the EPurse applet. Here we use hashing to demonstrate enhanced on-card verification of a client applet.

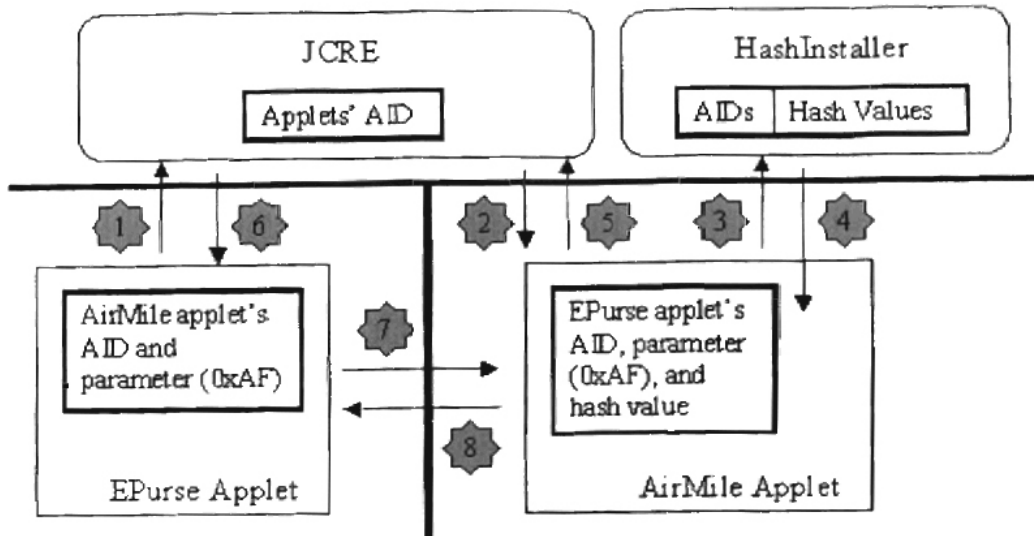


Figure 19. Secure Object Sharing Simulation

The steps 1 and 2 are identical with the steps 1 and 2 in the existing object sharing process illustrated in section 3.2.

In step 3, The AirMile applet verifies the client applet with the electronic purse applet's AID and the secret parameter that were programmed into the AirMile applet code before the server applet was installed. If the client applet is valid, then the AirMile applet passes the EPurse applet's AID and hash value that was also programmed into the server applet code to the HashInstaller.

In step 4, The HashInstaller compares the hash value that is from the AirMile applet with the hash value computed when the CAP file was read in. If they are identical, the HashInstaller returns true to the AirMile applet. Otherwise, it returns false to the AirMile applet. Once the AirMile gets a response from the HashInstaller, the sever applet now can determine the validation of the client applet.

The steps from 5 to 8 are identical with the steps from 3 to 6 in the existing object sharing process illustrated in section 3.2.

To compare our proposed approach with the existing approach, we used the same

malicious applet used in the previous section. As shown in the previous section with the existing approach, the malicious applet if installed with an illegal installation process can get a SIO from the AirMile applet. However, with our approach that uses hashing for the verification of client applets, the malicious applet cannot get a SIO from the AirMile applet even though it is still possible for a malicious applet to be installed onto a card via an illegal applet installation process as shown in Figures 20 and 21.

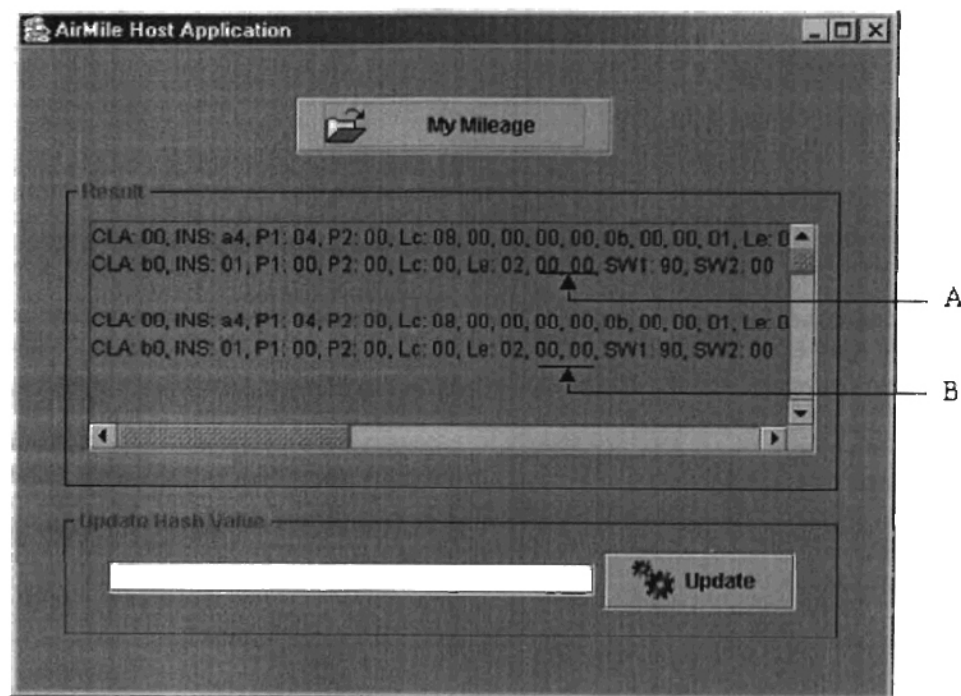


Figure 20. Host Application for the AirMile with Our Approach

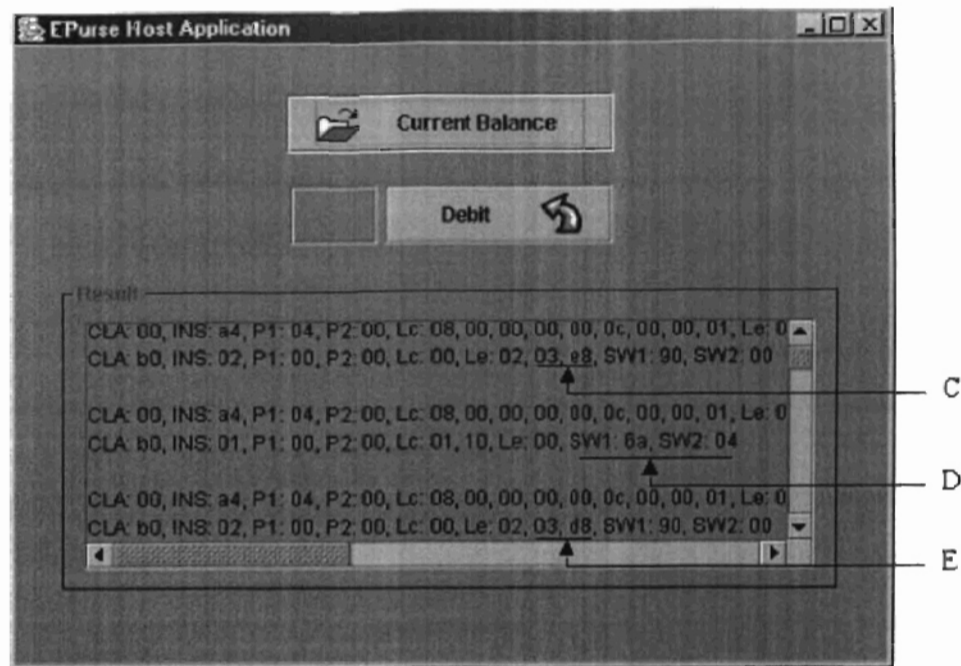


Figure 21. Host Application for the EPurse with Our Approach

In Figure 20 and 21, the arrow A indicates that the initial mileage of the AirMile applet set as zero, and the arrow C shows that the initial balance of the EPurse applet set to one thousand.

The arrow D in Figure 21 indicates that during a debit transaction with sixteen dollars, granting a SIO from the server applet has failed. Notice that the response APDU can be either an Acknowledgement (called an ACK) or Negative Acknowledgement (called a NAK). The value for an ACK frame SW1SW2 is 9000, and the value for a NAK frame SW1SW2 is 6XXX.

After the debit transaction with sixteen dollars, the arrow E in Figure 21 indicates that the balance in the EPurse applet has been changed. However, the arrow B in Figure 20 shows that the mileage in the AirMile applet has not been changed. This means the malicious client applet although installed with an illegal installation process applet failed to get a SIO from the server.

As shown in the above simulation, with our proposed approach that uses hashing for the verification of client applets, even though it is still possible for a malicious applet to be installed onto a card via an illegal applet installation process, it is extremely hard to get a service from a server applet because it is almost impossible for the malicious applet to have the same hash value as the value of the valid client's applet classes contained in the CAP file.

### **Update a Hash Value of the Client Applet**

Recall that applets can be installed and deleted at any time in the Java Card platform. Let's assume that the EPurse applet provider found some bugs later in the EPurse applet, and the provider wants to distribute a new version of the EPurse applet to its costumers. So, when card holders (the customers) insert a card into a CAD next time, the provider deletes the existing EPurse applet from the card and installs the new version of the EPurse applet onto the card. With our proposed on-card installer, when the new version of the EPurse applet is installed, the new hash value of the EPurse applet is registered with the JCRE.

Recall that, in our approach, the hash values of client applets should be saved in a server applet for verification of client applets. If the application code were burned into the chip at manufacture time as in the traditional smart card approach, this would be a major drawback. However, because the Java Card platform has the ability to dynamically respond to a card issuer's changing needs, such problems could be solved very easily. Figure 22 shows that the hash value of the client applet in the AirMile applet can be updated any time by the AirMile applet provider. The only constraint is that the new client's hash value has been stored in the server's code previously.

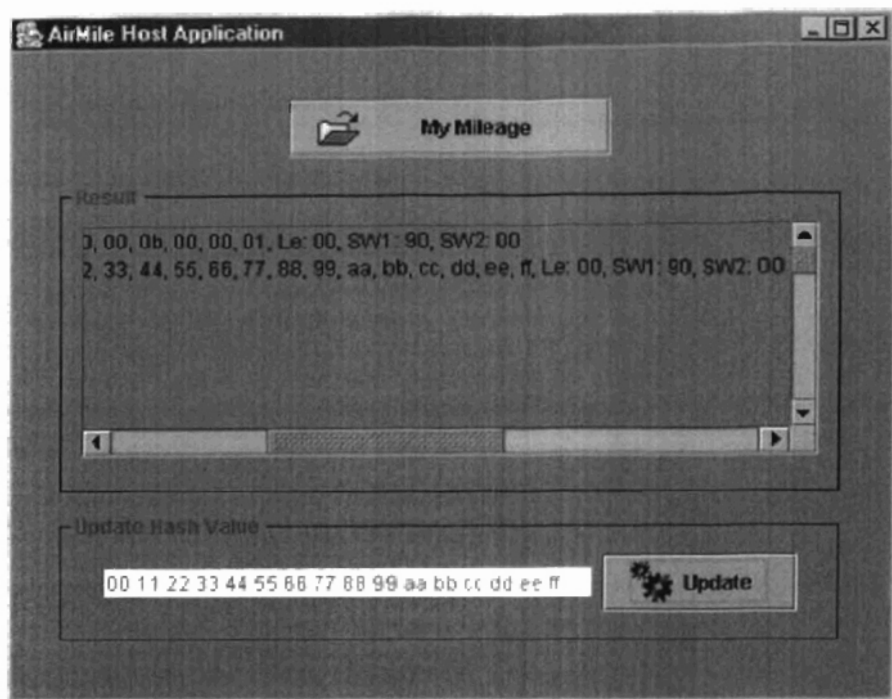


Figure 22. Update the Hash Value of the EPurse Applet in the AirMile Applet

The simulation tools were made available on the web. The purpose was to allow outsiders to attack our proposed security mechanism. At the time of writing, no-one had succeeded in breaking our system.

## **5. CONCLUSION**

In Java Card technology, the correctness and integrity of a CAP file are verified off card, and the on-card installer does not verify a CAP file at class loading time. Therefore, installation of a malicious applet onto a card may be possible by way of an illegal applet installation process. Also, to cooperate with other applets provided by different service providers, certain information should be available for other application providers who should not to reveal this information. However, it is possible that the information may be shared with unauthorized means. Therefore, a malicious applet if installed with an illegal installation process can get a SIO from a server applet because there is no way the server applet can reject the malicious client applet that impersonates with a valid AID and parameter.

We proposed in this thesis an on-card installer with a one-way hash function to support on-card verification of download applets by using a hash value of a CAP file. While the installer in the Java Card platform registers only an applet's AID on the JCRE, our proposed installer registers not only an applet's AID but also an applet's hash value that is computed by the proposed installer during the applet installation process. Later, the hash value is used to verify the client applet when it tries to gain a SIO from a server applet.

With our approach, even though it is still possible for a malicious applet to be installed onto a card via an illegal applet installation process, it is extremely hard to get a SIO from a server applet because it is almost impossible for the malicious applet to have the same hash value as the value of the valid client's applet classes contained in the CAP file. We have validated our proposal by simulating a Java Card environment



development using the Java Card 2.2 Development Kit. This simulation tool contains over 4000 lines of code.

## **BIBLIOGRAPHY**

1. Dreifus, Henry and Monk, Thomas. Smart Card: a guide to building and managing smart card applications. New York: John Wiley & Sons, Inc., 1997.
2. "Smart Card Overview." Online. Internet. Aug. 2002. Available: <http://java.sun.com/products/javacard/samrtcards.html>.
3. Chen, Zhiqun. Java Card Technology for Smart Cards: architecture and programmer's guide. California: Sun Microsystems, Inc., 2000.
4. "Smart Card Overview." Online. Internet. Aug. 2002. Available: <http://www.javacard.org>.
5. Rankl, Wolfgang and Effing, Wolfgang. Smart Card Handbook. England: John Wiley & Sons Ltd., 1997.
6. Donsez, D. Grimaud, G. and Lecomte, S. "Recoverable Persistent Memory for SmartCard." Proceedings of the Third International Conference, CARDIS'98 (1998): 134-140.
7. Hendry, Mike. Smart card security and applications. Boston: Artech House, 1997.
8. "Java Card 2.2 Java Card Virtual Machine Specification." Online. Internet. Aug. 2002. Available: <http://java.sun.com/products/javacard>.
9. "Java Card 2.2 Java Card Runtime Environment Specification." Online. Internet. Aug. 2002. Available: <http://java.sun.com/products/javacard>.
10. "What is OpenCard and the OpenCard Framework?" Online. Internet. Sep. 2002. Available: <http://www.opencard.org>.
11. Ahuja, Vijay. Network & Internet Security. Michigan: AP Professional, 1996.
12. Hughes, Larry. Internet Security Techniques. Indiana: New Riders, 1995.
13. "What is a hash function?" Online. Internet. Nov. 2002. Available: <http://www.rsasecurity.com/rsalabs/faq/2-1-6.html>.
14. "What are MD2, MD4, and MD5?" Online. Internet. Nov. 2002. Available: <http://www.rsasecurity.com/rsalabs/faq/3-6-6.html>.
15. Hughes, Larry. Actually Useful Internet Security Techniques. Indiana: New Riders, 1995.

16. "The MD5 Message-Digest Algorithm." MIT Laboratory. Apr. 1992.
17. "What are SHA and SHA-1?" Online. Internet. Nov. 2002. Available: <http://www.rsasecurity.com/rsalabs/faq/2-1-6.html>.
18. "Java Card Platform Security." Online. Internet. Oct. 2002. Available: <http://java.sun.com/products/javacard>.
19. "Java Card 2.2 Application Programming Interface." Online. Internet. Aug. 2002. Available: <http://java.sun.com/products/javacard>.
20. "Java Card 2.2 Development Kit User Guide." Online. Internet. Aug. 2000. Available: <http://java.sun.com/products/javacard>.

## APPENDIX

### AirMile.java

```
/*
*****
The class AirMile contains the miles the card holder has traveled. It implements the
verifyHashValue method in the HashInstallerInterface interface. When a client applet requests
a SIO, it asks a validation of the client applet to the HashInstaller. If the client applet is valid,
it shares its SIO with the client applet.
*****
*/
package airmileloyalty;

import javacard.framework.*;
import installersupport.HashInstallerInterface;

public class AirMile extends Applet implements AirMileInterface {

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

    // codes of INS byte in the command APDUs
    private final static byte DISPLAY = (byte)0x01;

    // used with the HashInstaller
    private final static byte UPDATEMD = (byte)0x02;
    private final static byte ADD = (byte)0x03;

    // applet-specific status words
    // used with the HashInstaller
    private final static short SW_UNAUTHORIZED_CLIENT = 0x6A01;
    private final static short SW_FAILED_TO_GET_SERVER_SIO = 0x6A02;
    private final static short SW_FAILED_TO_VERIFY = 0x6A03;
    private final static short SW_INVALID_MD_LENGTH = 0x6A04;

    // AID of this applet instance
    private final byte[] own_aid = {0x00, 0x00, 0x00, 0x00, 0x0B, 0x00, 0x00, 0x01};
    // AID of the server applet (HashInstaller) instance
    // used with the HashInstaller
    private final byte[] hash_installer_aid = {0x00, 0x00, 0x00, 0x00, 0x0A, 0x00, 0x00, 0x01};
    // AID of the client applet (EPurse) instance
    private final byte[] epurse_aid = {0x00, 0x00, 0x00, 0x00, 0x0C, 0x00, 0x00, 0x01};

    // parameter between AirMile and HashInstaller
    private final byte password_hash = (byte)0x0A;
    // parameter between EPurse and AirMile
    private final byte password_epurse = (byte)0xAF;

    // the hash value of the client applet (EPurse)
    // used with the HashInstaller
    private byte[] md_epurse;

    private short mileage;

    private AirMile () {
```

```

md_cpurse = new byte[16];

// hashvalue of the CAP file contains EPurse applet
// used with the HashInstaller
md_epurse[0] = (byte)0xA0;
md_epurse[1] = (byte)0x44;
md_epurse[2] = (byte)0x60;
md_epurse[3] = (byte)0x6A;
md_epurse[4] = (byte)0x78;
md_epurse[5] = (byte)0x6D;
md_epurse[6] = (byte)0xE2;
md_epurse[7] = (byte)0x76;
md_epurse[8] = (byte)0x74;
md_epurse[9] = (byte)0x64;
md_epurse[10] = (byte)0x7B;
md_epurse[11] = (byte)0x22;
md_epurse[12] = (byte)0x05;
md_epurse[13] = (byte)0x7E;
md_epurse[14] = (byte)0x21;
md_epurse[15] = (byte)0x35;

mileage = (short)0;

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

} // end of constructor

public static void install (byte[] bArray, short bOffset, byte blength) {

    // create a AirMile applet instance
    new AirMile();

} // end of install method

public void process (APDU apdu) {

    byte[] buffer = apdu.getBuffer();

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

    // verify the CLA bytes
    if (buffer[ISO7816.OFFSET_CLA] != AirMile_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 DISPLAY: display(apdu);
            return;
        // used with the HashInstaller
        case UPDATEMD: updateMD(apdu);
            return;
        case ADD: addMile (apdu);
            return;
    }
}

```

```

    default: ISOException.throwIt(ISO7816.SW_INS_NOT_SUPPORTED);
;

} // end of process method

// used with the HashInstaller
public Shareable getShareableInterfaceObject (AID client_aid, byte parameter) {

    if (client_aid.equals(epurse_aid, (short)0, (byte)(epurse_aid.length)) == false)
        return null;

    if (parameter != password_epurse)
        return null;

    // check the hash value of the client applet
    // obtain the HashInstaller AID object
    AID server_aid = JCSYSTEM.lookupAID(hash_installer_aid, (short)0, (byte)(hash_installer_aid.length));

    // request the sio from the server
    HashInstallerInterface sio
        = (HashInstallerInterface)(JCSYSTEM.getAppletShareableInterfaceObject
            (server_aid, password_hash));

    byte aid1 = 0, aid2 = 0, aid3 = 0, aid4 = 0, aid5 = 0, aid6 = 0, aid7 = 0, aid8 = 0,
        aid9 = 0, aid10 = 0, aid11 = 0, aid12 = 0, aid13 = 0, aid14 = 0, aid15 = 0, aid16 = 0;
    byte aid_length = (byte)epurse_aid.length;
    if (aid_length >= 1) aid1 = epurse_aid[0];
    if (aid_length >= 2) aid2 = epurse_aid[1];
    if (aid_length >= 3) aid3 = epurse_aid[2];
    if (aid_length >= 4) aid4 = epurse_aid[3];
    if (aid_length >= 5) aid5 = epurse_aid[4];
    if (aid_length >= 6) aid6 = epurse_aid[5];
    if (aid_length >= 7) aid7 = epurse_aid[6];
    if (aid_length >= 8) aid8 = epurse_aid[7];
    if (aid_length >= 9) aid9 = epurse_aid[8];
    if (aid_length >= 10) aid10 = epurse_aid[9];
    if (aid_length >= 11) aid11 = epurse_aid[10];
    if (aid_length >= 12) aid12 = epurse_aid[11];
    if (aid_length >= 13) aid13 = epurse_aid[12];
    if (aid_length >= 14) aid14 = epurse_aid[13];
    if (aid_length >= 15) aid15 = epurse_aid[14];
    if (aid_length >= 16) aid16 = epurse_aid[15];

    byte md1 = 0, md2 = 0, md3 = 0, md4 = 0, md5 = 0, md6 = 0, md7 = 0, md8 = 0,
        md9 = 0, md10 = 0, md11 = 0, md12 = 0, md13 = 0, md14 = 0, md15 = 0, md16 = 0;
    md1 = md_epurse[0];
    md2 = md_epurse[1];
    md3 = md_epurse[2];
    md4 = md_epurse[3];
    md5 = md_epurse[4];
    md6 = md_epurse[5];
    md7 = md_epurse[6];
    md8 = md_epurse[7];
    md9 = md_epurse[8];
    md10 = md_epurse[9];
    md11 = md_epurse[10];

```

```

md12 = md_epurse[11];
md13 = md_epurse[12];
md14 = md_epurse[13];
md15 = md_epurse[14];
md16 = md_epurse[15];

// verify the client applet with its hash value
boolean result = sio.verifyHashValue
    (aid_length, aid1, aid2, aid3, aid4, aid5, aid6, aid7, aid8, aid9, aid10,
    aid11, aid12, aid13, aid14, aid15, aid16, md1, md2, md3, md4, md5, md6,
    md7, md8, md9, md10, md11, md12, md13, md14, md15, md16);

if (result == false)
    ISOException.throwIt(SW_FAILED_TO_VERIFY);

return (this);
} // end of method getShareableInterfaceObject

public void bonusMile (short bonus) {

    // get the caller's AID
    AID client_aid = JCSysSystem.getPreviousContextAID();

    // check if the actual caller is the EPurse applet
    if (client_aid.equals(epurse_aid, (short)0, (byte)(epurse_aid.length)) == false)
        ISOException.throwIt(SW_UNAUTHORIZED_CLIENT);

    mileage = (short)(mileage + bonus);
} // end of method bonusMile

private void display (APDU apdu) {

    byte[] buffer = apdu.getBuffer();

    // inform the JCRE that the applet has data to return
    apdu.setOutgoing();

    // set the actual number of the outgoing data bytes
    apdu.setOutgoingLength((byte)2);

    // write the balance into the APDU buffer at the offset 0
    Util.setShort(buffer, (short)0, mileage);

    // send the 2-byte balance at the offset 0 in the apdu buffer
    apdu.sendBytes((short)0, (short)2);
} // end of method balance

// used with the HashInstaller
private void updateMD (APDU apdu) {

    byte[] apdu_buffer = apdu.getBuffer();

    // set the JCRE into the data receiving mode

```

```

apdu.setIncomingAndReceive();

// the length of a message block is from 1 to 64 bytes
short md_length = (short)(apdu_buffer[ISO7816.OFFSET_LC] & 0x00FF);

if (md_length != 16)
    ISOException.throwIt(SW_INVALID_MD_LENGTH);

// copy a message that is contained from index 5
Util.arrayCopy(apdu_buffer, (short)(ISO7816.OFFSET_CDATA & 0x00FF),
               md_epurse, (short)0, md_length);

} // end of method update

private void addMile (APDU apdu) {

    byte[] apdu_buffer = apdu.getBuffer();

    // set the JCRE into the data_receiving mode
    apdu.setIncomingAndReceive();

    short mile = (short)(apdu_buffer[ISO7816.OFFSET_CDATA] & 0x00FF);

    mileage = (short)(mileage + mile);

} // end of method addMile

} // end of class AirMile

```

## AirMileInterface.java

```

/*****
The interface AirMileInterface contains the method bonusMile that is an abstract method
used for grant a mileage.
*****/

```

```

package airmileloyalty;

import javacard.framework.Shareable;

public interface AirMileInterface extends Shareable {

    public void bonusMile (short mileage);

}

```

## EPurse.java

```

/*****
The class EPurse contains electronic cash used for buy goods. It has an ability to get a SIO
and invoke the bonusMile method from the AirMile applet.
*****/

```

```

package electronicpurse;

```



```

import javacard.framework.*;
import airmileloyalty.AirMileInterface;

public class EPurse extends Applet {

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

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

    // Applet-specific status words
    private final static short SW_INVALID_AMOUNT = 0x6A01;
    private final static short SW_NEGATIVE_BALANCE = 0x6A02;
    private final static short SW_SERVER_NOT_EXIST = 0x6A03;
    private final static short SW_FAILED_TO_GET_SERVER_SIO = 0x6A04;

    // AID of this applet instance
    private final byte[] own_aid = {0x00, 0x00, 0x00, 0x00, 0x0C, 0x00, 0x00, 0x01};
    // AID of the server applet(AirMile applet) instance
    private final byte[] airmile_aid = {0x00, 0x00, 0x00, 0x00, 0x0B, 0x00, 0x00, 0x01};

    // parameter between EPurse and AirMile
    private final byte password = (byte)0xAF;
    // maximum transaction amount
    private final byte max_payment = (byte)100;

    private short money;

    private EPurse () {

        money = (short)1000;

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

    } // end of constructor

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

        // create a EPurse applet instance
        new EPurse ();

    } // end of install method

    public void process (APDU apdu) {

        byte[] buffer = apdu.getBuffer();

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

        // verify the CLA bytes
        if (buffer[ISO7816.OFFSET_CLA] != EPurse_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 PAY: payment(apdu);
            return;
    case BALANCE: balance(apdu);
            return;
    default: ISOException.throwIt(ISO7816.SW_INS_NOT_SUPPORTED);
}

} // end of process method

private void payment (APDU apdu) {

    byte[] buffer = apdu.getBuffer();

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

if (numBytes != 1)
    ISOException.throwIt(ISO7816.SW_WRONG_LENGTH);

// receive data
apdu.setIncomingAndReceive();

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

if ((amount > max_payment) || (amount < 0))
    ISOException.throwIt(SW_INVALID_AMOUNT);

if ((short)(money - amount) < (short)0)
    ISOException.throwIt(SW_NEGATIVE_BALANCE);

// new balance
money = (short)(money - amount);

// obtain the server AID object
AID server_aid = JCSYSTEM.lookupAID(airmile_aid, (short)0, (byte)(airmile_aid.length));

if (server_aid == null)
    ISOException.throwIt(SW_SERVER_NOT_EXIST);

// request the sio from the server
AirMileInterface sio
    = (AirMileInterface)(JCSYSTEM.getAppletShareableInterfaceObject(server_aid, password));

if (sio == null)
    ISOException.throwIt(SW_FAILED_TO_GET_SERVER_SIO);

// ask the server to bonus miles
sio.bonusMile((short)(amount&0x00FF));

} // end of method payment

```

```

private void balance (APDU apdu) {

    byte[] buffer = apdu.getBuffer();

    // inform the JCRE that the applet has data to return
    apdu.setOutgoing();

    // set the actual number of the outgoing data bytes
    apdu.setOutgoingLength((byte)2);

    // write the balance into the APDU buffer at the offset 0
    Util.setShort(buffer, (short)0, money);

    // send the 2-byte balance at the offset 0 in the apdu buffer
    apdu.sendBytes((short)0, (short)2);

} // end of method balance

} // end of class EPurse

```

## HostAirMile.java

```

/*****
The class HostAirMile is a host application for the AirMile applet resided on the terminals.
It has an ability to extract a current mileage from the AirMile applet.
*****/

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

class HostAirMile extends JFrame {

    private JPanel first_panel;
    private JButton display_mile;

    private JPanel second_panel;
    private JScrollPane scroll;
    private JTextArea area;

    // only used with the HashInstaller applet
    private JPanel third_panel;
    private JButton update_md;
    private JTextField md;

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

    HostAirMile () {

        setTitle("AirMile Host Application");
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

```

```

getContentPane().setLayout(null);
getContentPane().setBackground(Color.green);

first_panel = new JPanel();
first_panel.setBounds(25, 25, 450, 55);
first_panel.setBackground(Color.green);

display_mile = new JButton("    My Mileage    ",
                           new ImageIcon("c:/demo/sourcecode/gif/display.gif"));
display_mile.setBackground(Color.orange);
display_mile.setSize(100, 40);
first_panel.add(display_mile);

second_panel = new JPanel();
second_panel.setBounds(25, 80, 450, 190);
second_panel.setBackground(Color.green);
second_panel.setBorder(BorderFactory.createTitledBorder
                      (BorderFactory.createLineBorder(Color.blue), " Result "));

area = new JTextArea(400, 100);
area.setBackground(Color.green);
area.setEnabled(true);
scroll = new JScrollPane(area);
scroll.setPreferredSize(new Dimension(420, 140));
second_panel.add(scroll);

// only used with the HashInstaller applet
third_panel = new JPanel();
third_panel.setBounds(25, 280, 450, 80);
third_panel.setBackground(Color.green);
third_panel.setBorder(BorderFactory.createTitledBorder
                    (BorderFactory.createLineBorder(Color.red), " Update Hash Value "));

// only used with the HashInstaller applet
md = new JTextField();
md.setPreferredSize(new Dimension(280, 20));
third_panel.add(md);

// only used with the HashInstaller applet
update_md = new JButton(" Update ",
                        new ImageIcon("c:/demo/sourcecode/gif/updateMD.gif"));
update_md.setBackground(Color.orange);
update_md.setSize(100, 40);
third_panel.add(update_md);

// to extract a current mileage from the AirMile applet.
display_mile.addActionListener(new ActionListener() {
    public void actionPerformed (ActionEvent e) {

        md.setEnabled(true);
        md.setText(null);

        // to create a script file that can extract a current mileage from the AirMile applet
        try {

            scr_file = new File("apdu.scr");

```

```

BufferedWriter out = new BufferedWriter(new FileWriter(scr_file));

out.write("powerup;\n\n");
out.write("// Select AirMile applet\n");
out.write
    ("0x00 0xA4 0x04 0x00 0x08 0x00 0x00 0x00 0x00 0x0B 0x00 0x00 0x01 0x7F;\n\n");
out.write("// Display milage\n");
out.write("0xB0 0x01 0x00 0x00 0x00 0x7F;\n\n");
out.write("powerdown;");
out.close();

} catch (Exception ee) {
}

// to create a batch file to run the script file
try {

    batch = new File("execute.bat");
    result = new File("result");
    BufferedWriter out = new BufferedWriter(new FileWriter(batch));
    out.write("@echo off\n");
    out.write("apdutool -o " + result.getAbsolutePath() + " " + scr_file.getAbsolutePath() + "\n");
    out.close();

    String batch_path = batch.getAbsolutePath();
    batch_path = batch_path.replace("\\", "/");
    Process child = Runtime.getRuntime().exec(batch_path);
    child.waitFor();

} catch (Exception e2) {
}

String text;

// to display current mileage in the AirMile applet
try {
    BufferedReader reader = new BufferedReader(new FileReader(result));
    while ((text = reader.readLine()) != null) {
        area.append(text + "\n");
    }
    area.append("\n");
    reader.close();

} catch (Exception ee) {
} finally {
    scr_file.delete();
    scr_file = null;
    batch.delete();
    batch = null;
    result.delete();
    result = null;
}

}
});

```

```

// only used with the HashInstaller applet
// to update a hash value of a CAP file
update_md.addActionListener(new ActionListener() {
    public void actionPerformed (ActionEvent e) {

        md.setEnabled(false);

        String hash = md.getText();
        String hash_value = new String();
        StringTokenizer token = new StringTokenizer(hash);
        while (token.hasMoreTokens())
            hash_value = hash_value + "0x" + token.nextToken().toUpperCase() + " ";

        // to create a script file that can update a hash value in HashInstaller applet
        try {

            scr_file = new File("apdu.scr");
            BufferedWriter out = new BufferedWriter(new FileWriter(scr_file));

            out.write("powerup;\n\n");
            out.write("// Select AirMile applet\n");
            out.write
                ("0x00 0xA4 0x04 0x00 0x08 0x00 0x00 0x00 0x0B 0x00 0x00 0x01 0x7F;\n\n");
            out.write("// Update MD\n");
            out.write("0xB0 0x02 0x00 0x10 " + hash_value + "0x7F;\n\n");
            out.write("powerdown;");
            out.close();

        } catch (Exception ee) {
        }

        // to create a batch file to run the script file
        try {

            batch = new File("execute.bat");
            result = new File("result");
            BufferedWriter out = new BufferedWriter(new FileWriter(batch));
            out.write("@echo off\n");
            out.write("apdutool -o " + result.getAbsolutePath() + " " + scr_file.getAbsolutePath() + "\n");
            out.close();

            String batch_path = batch.getAbsolutePath();
            batch_path = batch_path.replace("\\", "/");
            Process child = Runtime.getRuntime().exec(batch_path);
            child.waitFor();

        } catch (Exception e2) {
        }

        String text;

        // to display current mileage in AirMile applet
        try {
            BufferedReader reader = new BufferedReader(new FileReader(result));
            while ((text = reader.readLine()) != null) {
                area.append(text + "\n");
            }
        }
    }
}

```

```

    }
    area.append("\n");
    reader.close();

    } catch (Exception ee) {
    } finally {
        scr_file.delete();
        scr_file = null;
        batch.delete();
        batch = null;
        result.delete();
        result = null;
    }

    }
});

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

} // end of constructor

public static void main (String[] args) {

    JFrame frame = new HostAirMile();
    frame.setBounds(0, 0, 510, 420);
    frame.setVisible(true);

    } // end of method main

} // end of class HostAirMile

```

## HostEPurse.java

```

/*****
The class HostEPurse is a host application for the EPurse applet resided on the terminals.
It has an ability to update and extract a current balance in the EPurse applet.
*****/

```

```

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

class HostEPurse extends JFrame {

    private JPanel first_panel;
    private JButton balance;

    private JPanel second_panel;
    private JButton pay;
    private JTextField money;

    private JPanel third_panel;

```

```

private JScrollPane scroll;
private JTextArea area;

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

HostEPurse () {

    setTitle("EPurse Host Application");
    setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    getContentPane().setLayout(null);
    getContentPane().setBackground(Color.green);

    first_panel = new JPanel();
    first_panel.setBounds(25, 25, 450, 55);
    first_panel.setBackground(Color.green);

    balance = new JButton("    Current Balance    ", new
        ImageIcon("c:/demo/sourcecode/gif/display.gif"));
    balance.setBackground(Color.orange);
    balance.setSize(100, 40);
    first_panel.add(balance);

    second_panel = new JPanel();
    second_panel.setBounds(25, 80, 450, 55);
    second_panel.setBackground(Color.green);

    money = new JTextField();
    money.setPreferredSize(new Dimension(50, 35));
    money.setBackground(Color.green);
    second_panel.add(money);

    pay = new JButton("    Debit    ", new ImageIcon("c:/demo/sourcecode/gif/pay.gif"));
    pay.setVerticalTextPosition(AbstractButton.CENTER);
    pay.setHorizontalTextPosition(AbstractButton.LEFT);
    pay.setBackground(Color.orange);
    pay.setSize(100, 40);
    second_panel.add(pay);

    third_panel = new JPanel();
    third_panel.setBounds(25, 140, 450, 190);
    third_panel.setBackground(Color.green);
    third_panel.setBorder(BorderFactory.createTitledBorder
        (BorderFactory.createLineBorder(Color.blue), " Result "));

    area = new JTextArea(250, 100);
    area.setBackground(Color.green);
    area.setForeground(Color.black);
    area.setEnabled(true);
    scroll = new JScrollPane(area);
    scroll.setPreferredSize(new Dimension(420, 150));
    third_panel.add(scroll);

    balance.addActionListener(new ActionListener() {
        public void actionPerformed (ActionEvent e) {

```



```

money.setText(null);
money.setEnabled(true);

// to create a script file that can extract a current balance from the EPurse applet
try {

    scr_file = new File("apdu.scr");
    BufferedWriter out = new BufferedWriter(new FileWriter(scr_file));

    out.write("powerup;\n\n");
    out.write("// Select EPurse applet\n");
    out.write("0x00 0xA4 0x04 0x00 0x08 0x00 0x00 0x00 0x00 0x0C 0x00 0x00 0x01 0x7F;\n\n");
    out.write("// Display balance\n");
    out.write("0xB0 0x02 0x00 0x00 0x00 0x7F;\n\n");
    out.write("powerdown;");
    out.close();

} catch (Exception ee) {
}

// to create a batch file to run the script file
try {

    batch = new File("execute.bat");
    result = new File("result");
    BufferedWriter out = new BufferedWriter(new FileWriter(batch));
    out.write("@echo off\n");
    out.write("apdutool -o " + result.getAbsolutePath() + " " + scr_file.getAbsolutePath() + "\n");
    out.close();

    String batch_path = batch.getAbsolutePath();
    batch_path = batch_path.replace("\\", "/");
    Process child = Runtime.getRuntime().exec(batch_path);
    child.waitFor();

} catch (Exception e2) {
}

String text;

// to display current balance in the EPurse applet
try {
    BufferedReader reader = new BufferedReader(new FileReader(result));
    while ((text = reader.readLine()) != null) {
        area.append(text + "\n");
    }
    area.append("\n");
    reader.close();

} catch (Exception e3) {
} finally {
    scr_file.delete();
    scr_file = null;
    batch.delete();
    batch = null;
}

```

```

    result.delete();
    result = null;
}

}

});

pay.addActionListener(new ActionListener() {
    public void actionPerformed (ActionEvent e) {

        // to create a script file that can update a current balance from the EPurse applet
        try {

            scr_file = new File("apdu.scr");
            BufferedWriter out = new BufferedWriter(new FileWriter(scr_file));

            int amount = Integer.valueOf(money.getText()).intValue();
            String amount16 = Integer.toString(amount, 16);
            System.out.println(amount16);
            if (amount16.length() == 1)
                amount16 = "0" + amount16;

            out.write("powerup;\n\n");
            out.write("// Select EPurse applet\n");
            out.write("0x00 0xA4 0x04 0x00 0x08 0x00 0x00 0x00 0x00 0x0C 0x00 0x00 0x01 0x7F;\n\n");
            out.write("// Debit\n");
            out.write("0xB0 0x01 0x00 0x00 0x01 0x" + amount16 + " 0x7F;\n\n");
            out.write("powerdown;");
            out.close();

            money.setEnabled(false);

        } catch (Exception ee) {
        }

        // to create a batch file to run the script file
        try {

            batch = new File("execute.bat");
            result = new File("result");
            BufferedWriter out = new BufferedWriter(new FileWriter(batch));
            out.write("@echo off\n");
            out.write("apdutool -o " + result.getAbsolutePath() + " " + scr_file.getAbsolutePath() + "\n");
            out.close();

            String batch_path = batch.getAbsolutePath();
            batch_path = batch_path.replace("\\", "/");
            Process child = Runtime.getRuntime().exec(batch_path);
            child.waitFor();

        } catch (Exception e2) {
        }

        String text;

        // to display current balance in the EPurse applet

```

```

try {
    BufferedReader reader = new BufferedReader(new FileReader(result));
    while ((text = reader.readLine()) != null) {
        area.append(text + "\n");
    }
    area.append("\n");
    reader.close();

} catch (Exception e3) {
} finally {
    scr_file.delete();
    scr_file = null;
    batch.delete();
    batch = null;
    result.delete();
    result = null;
}

}
});

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

} // end of constructor

public static void main (String[] args) {

    JFrame frame = new HostEPurse();
    frame.setBounds(0, 0, 510, 400);
    frame.setVisible(true);

} // end of method main

} // end of class HostAirMile

```

## HashInstaller.java

```

/*****
    The HashInstaller generates a hash value of a CAP file with the MD5 class when the CAP file
    is read in and it registers the hash value in its instance. Therefore, the HashInstaller applet and
    the existing Java Card installer residing on the Java Card platform together perform
    as the proposed on-card installer. The AppletInfo class has an AID and hash value of each CAP
    file. The AppletInfoManager implements a linked list with a list AppletInfo.
    *****/

package installersupport;

import javacard.framework.*;

public class HashInstaller extends Applet implements HashInstallerInterface {

    // codes of CLA byte in the command APDUs

```

```

private final static byte HASH_CLA = (byte)0xB0;

// codes of INS byte in the command APDUs
private final static byte APPLET_AID = (byte)0x01;
private final static byte CAP_START = (byte)0x02;
private final static byte CAP_CONTENT = (byte)0x03;
private final static byte CAP_END = (byte)0x04;
/** For Test **/
private final static byte CHECK = (byte)0x05;

// Applet-specific status words
private final static short SW_INVALID_AID_LENGTH = 0x6A01;
private final static short SW_INVALID_MB_LENGTH = 0x6A02;
private final static short SW_UNAUTHORIZED_CLIENT = 0x6A03;

private final byte[] own_aid = {0x00, 0x00, 0x00, 0x00, 0x0A, 0x00, 0x00, 0x01};
private final byte[] airmile_aid = {0x00, 0x00, 0x00, 0x00, 0x0B, 0x00, 0x00, 0x01};

private final byte password = (byte)0x0A;

private AppletInfoManager manager;
private MD5 md5;

private byte[] temp_message;
private byte[] temp_aid;
private byte[] temp_md;

private HashInstaller () {

    manager = new AppletInfoManager();
    md5 = new MD5();

    // the length of an message block is from 1 to 64 bytes
    temp_message = new byte[64];
    temp_aid = new byte[16];
    temp_md = new byte[16];

    register(own_aid, (short)0, (byte)(own_aid.length));
} // end of constructor

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

    new HashInstaller();
} // end of method install

public void process (APDU apdu) {

    byte[] apdu_buffer = apdu.getBuffer();

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

    // verify the CLA byte

```

```

if (apdu_buffer[ISO7816.OFFSET_CLA] != HASH_CLA)
    ISOException.throwIt(ISO7816.SW_CLA_NOT_SUPPORTED);

// check the INS byte to decide which service method to call
switch (apdu_buffer[ISO7816.OFFSET_INS]) {
    case APPLET_AID: saveAID(apdu);
                    return;
    case CAP_START: initMD();
                    return;
    case CAP_CONTENT: updateMD(apdu);
                    return;
    case CAP_END: saveMD();
                 return;
    case CHECK: check(apdu);
               return;
    default: ISOException.throwIt(ISO7816.SW_INS_NOT_SUPPORTED);
}

} // end of method process

private void saveAID (APDU apdu) {

    byte[] apdu_buffer = apdu.getBuffer();

    // set the JCRE into the data_receiving mode
    apdu.setIncomingAndReceive();

    // data field = AID length (1 byte) + AID + parameter length (1 byte)
    short aid_length = (short)(apdu_buffer[ISO7816.OFFSET_LC] & 0x00FF);

    if (aid_length < 5 || aid_length > 16)
        ISOException.throwIt(SW_INVALID_AID_LENGTH);

    manager.add(new AppletInfo(apdu_buffer));

} // end of method saveAID

private void initMD () {

    md5.initialize();

} // end of method updateMD

private void updateMD (APDU apdu) {

    byte[] apdu_buffer = apdu.getBuffer();

    // set the JCRE into the data_receiving mode
    apdu.setIncomingAndReceive();

    // the length of a message block is from 1 to 64 bytes
    short mb_length = (short)(apdu_buffer[ISO7816.OFFSET_IC] & 0x00FF);
    if (mb_length < 1 || mb_length > 64)
        ISOException.throwIt(SW_INVALID_MB_LENGTH);

    // copy a message that is contained from index 5

```

```

arrayCopy(apdu_buffer, (short)(ISO7816.OFFSET_CDATA & 0x00FF),
          temp_message, (short)0, mb_length);

md5.update(temp_message, mb_length);
} // end of method updateMD

private void saveMD () {

    md5.update(temp_message, (short)0);

    // save a MD in an AppletInfo object
    md5.generate(manager.getCurrent().getHash());

} // end of method saveMD

public Shareable getShareableInterfaceObject (AID client_aid, byte parameter) {

    if (client_aid.equals(airmile_aid, (short)0, (byte)(airmile_aid.length)) == false)
        return null;

    if (parameter != password)
        return null;

    // grant the SIO
    return (this);

} // end of method getShareableInterfaceObject

public boolean verifyHashValue (byte aid_length, byte aid1, byte aid2,
    byte aid3, byte aid4, byte aid5, byte aid6, byte aid7, byte aid8,
    byte aid9, byte aid10, byte aid11, byte aid12, byte aid13, byte aid14,
    byte aid15, byte aid16, byte md1, byte md2, byte md3, byte md4,
    byte md5, byte md6, byte md7, byte md8, byte md9, byte md10, byte md11,
    byte md12, byte md13, byte md14, byte md15, byte md16) {

    // get the caller's AID
    AID client_aid = JCSystem.getPreviousContextAID();

    // check if the actual caller is the airmile applet
    if (client_aid.equals(airmile_aid, (short)0, (byte)(airmile_aid.length)) == false)
        ISOException.throwIt(SW_UNAUTHORIZED_CLIENT);

    temp_aid[0] = aid1;
    temp_aid[1] = aid2;
    temp_aid[2] = aid3;
    temp_aid[3] = aid4;
    temp_aid[4] = aid5;
    temp_aid[5] = aid6;
    temp_aid[6] = aid7;
    temp_aid[7] = aid8;
    temp_aid[8] = aid9;
    temp_aid[9] = aid10;
    temp_aid[10] = aid11;
    temp_aid[11] = aid12;
    temp_aid[12] = aid13;

```

```
temp_aid[13] = aid14;
temp_aid[14] = aid15;
temp_aid[15] = aid16;
```

```
temp_md[0] = md1;
temp_md[1] = md2;
temp_md[2] = md3;
temp_md[3] = md4;
temp_md[4] = md5;
temp_md[5] = md6;
temp_md[6] = md7;
temp_md[7] = md8;
temp_md[8] = md9;
temp_md[9] = md10;
temp_md[10] = md11;
temp_md[11] = md12;
temp_md[12] = md13;
temp_md[13] = md14;
temp_md[14] = md15;
temp_md[15] = md16;
```

```
AppletInfo applet = manager.getHead();
boolean result = false;
```

```
while (applet != null) {
```

```
    result = arrayCompare(applet.getAid(), (short)0, temp_aid, (short)0, (short)(aid_length & 0x00FF));
```

```
    if (result == true) {
```

```
        result = arrayCompare(applet.getHash(), (short)0, temp_md, (short)0, (short)16);
```

```
        if (result == true)
```

```
            return true;
```

```
        else
```

```
            return false;
```

```
    }
```

```
    applet = applet.getNext();
```

```
}
```

```
return false;
```

```
} // end of method getHashValue
```

```
private void check (APDU apdu) {
```

```
    byte[] buffer = apdu.getBuffer();
```

```
    // inform the JCRE that the applet has data to return
    apdu.setOutgoing();
```

```
    // set the actual number of the outgoing data bytes
    apdu.setOutgoingLength((byte)48);
```

```
    AppletInfo applet = manager.getHead();
```

```
    Util.arrayCopy(applet.getAid(), (short)0, buffer, (short)0, (short)8);
```

```
    Util.arrayCopy(applet.getHash(), (short)0, buffer, (short)8, (short)16);
```

```

    applet = applet.getNext();
    Util.arrayCopy(applet.getAid(), (short)0, buffer, (short)24, (short)8);
    Util.arrayCopy(applet.getHash(), (short)0, buffer, (short)32, (short)16);

    apdu.sendBytes((short)0, (short)48);

} // end of method check

private void arrayCopy (byte[] source, short s_start, byte[] destination, short d_start, short length) {

    short perform = 0;

    for (perform = length; perform > 0; --perform) {
        destination[d_start] = source[s_start];
        s_start++;
        d_start++;
    }

} // end of method arrayCopy

private boolean arrayCompare (byte[] source, short s_start, byte[] destination, short d_start, short length)
{

    short perform = 0;

    for (perform = length; perform > 0; --perform) {
        if (source[s_start] != destination[d_start])
            return false;
        s_start++;
        d_start++;
    }

    return true;

} // end of method arrayCompare

} // end of class HashInstalier

class MD5 {

    // for S table
    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;

// for padding at the end of the message
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};

private boolean pre_padding = false;

// for a 64 bits hash value
private byte[] word_a = new byte[4];
private byte[] word_b = new byte[4];
private byte[] word_c = new byte[4];
private byte[] word_d = new byte[4];

private byte[] word_aa = new byte[4];
private byte[] word_bb = new byte[4];
private byte[] word_cc = new byte[4];
private byte[] word_dd = new byte[4];

private byte[] message_length = new byte[8];
private byte[] x_table = new byte[64];

private byte[] support_temp1 = new byte[4];
private byte[] support_temp2 = new byte[4];

void initialize () {

    // initialized a message digest buffer
    // word A = {0x67, 0x45, 0x23, 0x01};
    // word B = {0xEF, 0xCD, 0xAB, 0x89};
    // word C = {0x98, 0xBA, 0xDC, 0xFE};
    // word D = {0x10, 0x32, 0x54, 0x76};
    word_a[0] = (byte)0x67;
    word_a[1] = (byte)0x45;
    word_a[2] = (byte)0x23;
    word_a[3] = (byte)0x01;
    word_b[0] = (byte)0xEF;
    word_b[1] = (byte)0xCD;
    word_b[2] = (byte)0xAB;
    word_b[3] = (byte)0x89;
    word_c[0] = (byte)0x98;
    word_c[1] = (byte)0xBA;
    word_c[2] = (byte)0xDC;
    word_c[3] = (byte)0xFE;
    word_d[0] = (byte)0x10;
    word_d[1] = (byte)0x32;
    word_d[2] = (byte)0x54;
    word_d[3] = (byte)0x76;

    for (short i = 0; i < 8; i++)

```

```

    message_length[i] = 0x00;

    pre_padding = false;
} // end of method initialize

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

    short operation = 0;

    switch (length) {

        // during transmit a message block (each 64 bytes)
        case 64: operation = 1;
                break;
        // after transmit the last message block
        case 0: // when the length of total message is 0 module 512
                if (pre_padding == false) {
                    operation = -1;
                    break;
                }
                else
                    return;
        // when transmit the last message block
        default: operation = 3;
                pre_padding = true;
    }

    // add the length of a message block to the total length of message
    addLength(message_length, length);

    // save word_a as word_aa, word_b as word_bb,
    // word_c as word_cc, word_d as word_dd
    arrayCopy(word_a, (short)0, word_aa, (short)0, (short)4);
    arrayCopy(word_b, (short)0, word_bb, (short)0, (short)4);
    arrayCopy(word_c, (short)0, word_cc, (short)0, (short)4);
    arrayCopy(word_d, (short)0, word_dd, (short)0, (short)4);

    do {

        // build x_table
        switch (operation) {

            // padding 56 bytes (starts with 1) followed by total length of message (8 bytes)
            case -1: arrayCopy(padding, (short)0, x_table, (short)0, (short)56);
                    arrayCopy(message_length, (short)0, x_table, (short)56, (short)8);
                    operation = 0;
                    break;
            // save a message block (64 bytes) to x_table
            case 1: arrayCopy(buffer, (short)0, x_table, (short)0, (short)64);
                    operation = 0;
                    break;

            // padding 56 bytes (start with 0) followed by total length of message (8 bytes)
            case 2: arrayCopy(padding, (short)8, x_table, (short)0, (short)56);
                    arrayCopy(message_length, (short)0, x_table, (short)56, (short)8);

```

```

        operation = 0;
        break;

// the last message block followed by at least one byte or at most 64 bytes
// padding and total length of message (8 bytes)
case 3: if (length < 56) {
        short required_pad = (short)((short)56 - length);
        arrayCopy(buffer, (short)0, x_table, (short)0, length);
        arrayCopy(padding, (short)0, x_table, (short)length, required_pad);
        arrayCopy(message_length, (short)0, x_table, (short)56, (short)8);
        operation = 0;
    }
    else {
        short required_pad = (short)((short)64 - length);
        arrayCopy(buffer, (short)0, x_table, (short)0, length);
        arrayCopy(padding, (short)0, x_table, (short)length, required_pad);
        operation = 2;
    }
    break;
}

// Round 1: do the following 16 operations
firstRound (word_a, word_b, word_c, word_d, x_table,
            (short)0, S11, (byte)0xD7, (byte)0x6A, (byte)0xA4, (byte)0x78); /* 1 */
firstRound (word_d, word_a, word_b, word_c, x_table,
            (short)4, S12, (byte)0xE8, (byte)0xC7, (byte)0xB7, (byte)0x56); /* 2 */
firstRound (word_c, word_d, word_a, word_b, x_table,
            (short)8, S13, (byte)0x24, (byte)0x20, (byte)0x70, (byte)0xDB); /* 3 */
firstRound (word_b, word_c, word_d, word_a, x_table,
            (short)12, S14, (byte)0xC1, (byte)0xBD, (byte)0xCE, (byte)0xEE); /* 4 */
firstRound (word_a, word_b, word_c, word_d, x_table,
            (short)16, S11, (byte)0xF5, (byte)0x7C, (byte)0x0F, (byte)0xAF); /* 5 */
firstRound (word_d, word_a, word_b, word_c, x_table,
            (short)20, S12, (byte)0x47, (byte)0x87, (byte)0xC6, (byte)0x2A); /* 6 */
firstRound (word_c, word_d, word_a, word_b, x_table,
            (short)24, S13, (byte)0xA8, (byte)0x30, (byte)0x46, (byte)0x13); /* 7 */
firstRound (word_b, word_d, word_d, word_a, x_table,
            (short)28, S14, (byte)0xFD, (byte)0x46, (byte)0x95, (byte)0x01); /* 8 */
firstRound (word_a, word_b, word_c, word_d, x_table,
            (short)32, S11, (byte)0x69, (byte)0x80, (byte)0x98, (byte)0xD8); /* 9 */
firstRound (word_d, word_a, word_b, word_c, x_table,
            (short)36, S12, (byte)0x8B, (byte)0x44, (byte)0xF7, (byte)0xAF); /* 10 */
firstRound (word_c, word_d, word_a, word_b, x_table,
            (short)40, S13, (byte)0xFF, (byte)0xFF, (byte)0x5B, (byte)0xB1); /* 11 */
firstRound (word_b, word_c, word_d, word_a, x_table,
            (short)44, S14, (byte)0x89, (byte)0x5C, (byte)0xD7, (byte)0xBE); /* 12 */
firstRound (word_a, word_b, word_c, word_d, x_table,
            (short)48, S11, (byte)0x6B, (byte)0x90, (byte)0x11, (byte)0x22); /* 13 */
firstRound (word_d, word_a, word_b, word_c, x_table,
            (short)52, S12, (byte)0xFD, (byte)0x98, (byte)0x71, (byte)0x93); /* 14 */
firstRound (word_c, word_d, word_a, word_b, x_table,
            (short)56, S13, (byte)0xA6, (byte)0x79, (byte)0x43, (byte)0x8E); /* 15 */
firstRound (word_b, word_c, word_d, word_a, x_table,
            (short)60, S14, (byte)0x49, (byte)0xB4, (byte)0x08, (byte)0x21); /* 16 */

// Round 2: do the following 16 operations

```

```

secondRound (word_a, word_b, word_c, word_d, x_table,
             (short)4, S21, (byte)0xF6, (byte)0x1E, (byte)0x25, (byte)0x62); /* 1 */
secondRound (word_d, word_a, word_b, word_c, x_table,
             (short)24, S22, (byte)0xC0, (byte)0x40, (byte)0xB3, (byte)0x40); /* 2 */
secondRound (word_c, word_d, word_a, word_b, x_table,
             (short)44, S23, (byte)0x26, (byte)0x5E, (byte)0x5A, (byte)0x51); /* 3 */
secondRound (word_b, word_c, word_d, word_a, x_table,
             (short)0, S24, (byte)0xE9, (byte)0xB6, (byte)0xC7, (byte)0xAA); /* 4 */
secondRound (word_a, word_b, word_c, word_d, x_table,
             (short)20, S21, (byte)0xD6, (byte)0x2F, (byte)0x10, (byte)0x5D); /* 5 */
secondRound (word_d, word_a, word_b, word_c, x_table,
             (short)40, S22, (byte)0x2, (byte)0x44, (byte)0x14, (byte)0x53); /* 6 */
secondRound (word_c, word_d, word_a, word_b, x_table,
             (short)60, S23, (byte)0xD8, (byte)0xA1, (byte)0xE6, (byte)0x81); /* 7 */
secondRound (word_b, word_d, word_d, word_a, x_table,
             (short)24, S24, (byte)0xE7, (byte)0xD3, (byte)0xFB, (byte)0xC8); /* 8 */
secondRound (word_a, word_b, word_c, word_d, x_table,
             (short)36, S21, (byte)0x21, (byte)0xE1, (byte)0xCD, (byte)0xE6); /* 9 */
secondRound (word_d, word_a, word_b, word_c, x_table,
             (short)56, S22, (byte)0xC3, (byte)0x37, (byte)0x07, (byte)0xD6); /* 10 */
secondRound (word_c, word_d, word_a, word_b, x_table,
             (short)12, S23, (byte)0xF4, (byte)0xD5, (byte)0x0D, (byte)0x87); /* 11 */
secondRound (word_b, word_c, word_d, word_a, x_table,
             (short)32, S24, (byte)0x45, (byte)0x5A, (byte)0x14, (byte)0xED); /* 12 */
secondRound (word_a, word_b, word_c, word_d, x_table,
             (short)52, S21, (byte)0xA9, (byte)0xE3, (byte)0xE9, (byte)0x05); /* 13 */
secondRound (word_d, word_a, word_b, word_c, x_table,
             (short)8, S22, (byte)0xFC, (byte)0xEF, (byte)0xA3, (byte)0xF8); /* 14 */
secondRound (word_c, word_d, word_a, word_b, x_table,
             (short)28, S23, (byte)0x67, (byte)0x6F, (byte)0x02, (byte)0xD9); /* 15 */
secondRound (word_b, word_c, word_d, word_a, x_table,
             (short)48, S24, (byte)0x8D, (byte)0x2A, (byte)0x4C, (byte)0x8A); /* 16 */

// Round 3: do the following 16 operations
thirdRound (word_a, word_b, word_c, word_d, x_table,
            (short)20, S31, (byte)0xFF, (byte)0xFA, (byte)0x39, (byte)0x42); /* 1 */
thirdRound (word_d, word_a, word_b, word_c, x_table,
            (short)32, S32, (byte)0x87, (byte)0x71, (byte)0xF6, (byte)0x81); /* 2 */
thirdRound (word_c, word_d, word_a, word_b, x_table,
            (short)44, S33, (byte)0x6D, (byte)0x9D, (byte)0x61, (byte)0x22); /* 3 */
thirdRound (word_b, word_c, word_d, word_a, x_table,
            (short)56, S34, (byte)0xFD, (byte)0xE5, (byte)0x38, (byte)0x0C); /* 4 */
thirdRound (word_a, word_b, word_c, word_d, x_table,
            (short)4, S31, (byte)0xA4, (byte)0xBE, (byte)0xEA, (byte)0x44); /* 5 */
thirdRound (word_d, word_a, word_b, word_c, x_table,
            (short)16, S32, (byte)0x4B, (byte)0xDE, (byte)0xCF, (byte)0xA9); /* 6 */
thirdRound (word_c, word_d, word_a, word_b, x_table,
            (short)28, S33, (byte)0xF6, (byte)0xBB, (byte)0x4B, (byte)0x80); /* 7 */
thirdRound (word_b, word_d, word_d, word_a, x_table,
            (short)40, S34, (byte)0xBE, (byte)0xBF, (byte)0xBC, (byte)0x70); /* 8 */
thirdRound (word_a, word_b, word_c, word_d, x_table,
            (short)52, S31, (byte)0x28, (byte)0x9B, (byte)0x7E, (byte)0xC6); /* 9 */
thirdRound (word_d, word_a, word_b, word_c, x_table,
            (short)0, S32, (byte)0xEA, (byte)0xA1, (byte)0x27, (byte)0xFA); /* 10 */
thirdRound (word_c, word_d, word_a, word_b, x_table,
            (short)12, S33, (byte)0xD4, (byte)0xEF, (byte)0x30, (byte)0x85); /* 11 */

```

```

thirdRound (word_b, word_c, word_d, word_a, x_table,
            (short)24, S34, (byte)0x4, (byte)0x88, (byte)0x1D, (byte)0x05); /* 12 */
thirdRound (word_a, word_b, word_c, word_d, x_table,
            (short)36, S31, (byte)0xD9, (byte)0xD4, (byte)0xD0, (byte)0x39); /* 13 */
thirdRound (word_d, word_a, word_b, word_c, x_table,
            (short)48, S32, (byte)0xE6, (byte)0xDB, (byte)0x99, (byte)0xE5); /* 14 */
thirdRound (word_c, word_d, word_a, word_b, x_table,
            (short)60, S33, (byte)0x1F, (byte)0xA2, (byte)0x7C, (byte)0xF8); /* 15 */
thirdRound (word_b, word_c, word_d, word_a, x_table,
            (short)8, S34, (byte)0xC4, (byte)0xAC, (byte)0x56, (byte)0x65); /* 16 */

// Round 4: do the following 16 operations
fourthRound (word_a, word_b, word_c, word_d, x_table,
            (short)0, S41, (byte)0xF4, (byte)0x29, (byte)0x22, (byte)0x44); /* 1 */
fourthRound (word_d, word_a, word_b, word_c, x_table,
            (short)28, S42, (byte)0x43, (byte)0x2A, (byte)0xFF, (byte)0x97); /* 2 */
fourthRound (word_d, word_d, word_a, word_b, x_table,
            (short)56, S43, (byte)0xAB, (byte)0x94, (byte)0x23, (byte)0xA7); /* 3 */
fourthRound (word_b, word_c, word_d, word_a, x_table,
            (short)20, S44, (byte)0xFC, (byte)0x93, (byte)0xA0, (byte)0x39); /* 4 */
fourthRound (word_a, word_b, word_c, word_d, x_table,
            (short)48, S41, (byte)0x65, (byte)0x5B, (byte)0x59, (byte)0xC3); /* 5 */
fourthRound (word_d, word_a, word_b, word_c, x_table,
            (short)12, S42, (byte)0x8F, (byte)0x0C, (byte)0xCC, (byte)0x92); /* 6 */
fourthRound (word_c, word_d, word_a, word_b, x_table,
            (short)40, S43, (byte)0xFF, (byte)0xEF, (byte)0xF4, (byte)0x7D); /* 7 */
fourthRound (word_b, word_d, word_d, word_a, x_table,
            (short)4, S44, (byte)0x85, (byte)0x84, (byte)0x5D, (byte)0xD1); /* 8 */
fourthRound (word_a, word_b, word_c, word_d, x_table,
            (short)32, S41, (byte)0x6F, (byte)0xA8, (byte)0x7E, (byte)0x4F); /* 9 */
fourthRound (word_d, word_a, word_b, word_c, x_table,
            (short)60, S42, (byte)0xFE, (byte)0x2C, (byte)0xE6, (byte)0xE0); /* 10 */
fourthRound (word_c, word_d, word_a, word_b, x_table,
            (short)24, S43, (byte)0xA3, (byte)0x01, (byte)0x43, (byte)0x14); /* 11 */
fourthRound (word_b, word_c, word_d, word_a, x_table,
            (short)52, S44, (byte)0x4E, (byte)0x08, (byte)0x11, (byte)0xA1); /* 12 */
fourthRound (word_a, word_b, word_c, word_d, x_table,
            (short)16, S41, (byte)0xF7, (byte)0x53, (byte)0x7E, (byte)0x82); /* 13 */
fourthRound (word_d, word_a, word_b, word_c, x_table,
            (short)44, S42, (byte)0xBD, (byte)0x3A, (byte)0xF2, (byte)0x35); /* 14 */
fourthRound (word_c, word_d, word_a, word_b, x_table,
            (short)8, S43, (byte)0x2A, (byte)0xD7, (byte)0xD2, (byte)0xBB); /* 15 */
fourthRound (word_b, word_c, word_d, word_a, x_table,
            (short)36, S44, (byte)0xEB, (byte)0x86, (byte)0xD3, (byte)0x91); /* 16 */

// perform the increment each of the four words
// by the value it had before this block was started
// word_a = word_a + word_aa
// word_b = word_b + word_bb
// word_c = word_c + word_cc
// word_d = word_d + word_dd
addBytes(word_a, word_aa, (short)0, (short)3);
addBytes(word_b, word_bb, (short)0, (short)3);
addBytes(word_c, word_cc, (short)0, (short)3);
addBytes(word_d, word_dd, (short)0, (short)3);

```

```

    } while (operation != 0);
} // end of method update

void generate (byte[] md) {
    // the message digest produced as output is A, B, C, D.
    // That is, we begin with the low-order byte of A, and end with the
    // high-order byte of D.

    for (short i = 0, j = 3; j >= 0; ++i, --j)
        md[i] = word_a[j];

    for (short i = 4, j = 3; j >= 0; ++i, --j)
        md[i] = word_b[j];

    for (short i = 8, j = 3; j >= 0; ++i, --j)
        md[i] = word_c[j];

    for (short i = 12, j = 3; j >= 0; ++i, --j)
        md[i] = word_d[j];
} // end of method generate

// Round 1:
// When  $F(X, Y, Z) = (X \& Y) | (\sim X \& Z)$ ,
// let [abcd k s i] denote the operation
//  $a = b + ((a + F(b,c,d) + X[k] + T[i]) \lll s)$ .
private void firstRound (byte[] a, byte[] b, byte[] c, byte[] d, byte[] x,
                        short offset, byte s, byte t1, byte t2, byte t3, byte t4) {

    //  $F(X, Y, Z) = (X \& Y) | (\sim X \& Z)$ 
    bitAnd(b, c, support_temp1);
    bitComplement(b, support_temp2);
    bitAnd(support_temp2, d, support_temp2);
    bitOr(support_temp1, support_temp2, support_temp2);

    //  $a + F(b,c,d) + X[k] + T[i]$ 
    arrayCopy(x, (short)offset, support_temp1, (short)0, (short)4);
    addBytes(support_temp2, support_temp1, (short)0, (short)3);
    support_temp1[0] = t1;
    support_temp1[1] = t2;
    support_temp1[2] = t3;
    support_temp1[3] = t4;
    addBytes (support_temp2, support_temp1, (short)0, (short)3);
    addBytes (support_temp2, a, (short)0, (short)3);

    //  $b + ((a + F(b,c,d) + X[k] + T[i]) \lll s)$ 
    rotateLeft(support_temp2, s, support_temp2);
    addBytes(support_temp2, b, (short)0, (short)3);

    //  $a = b + ((a + F(b,c,d) + X[k] + T[i]) \lll s)$ 
    arrayCopy(support_temp2, (short)0, a, (short)0, (short)4);
} // end of method firstRound

```

```

// Round 2:
// When  $G(X, Y, Z) = (X \& Z) | (Y \& \sim Z)$ ,
// let [abcd k s i] denote the operation
//  $a = b + ((a + G(b,c,d) + X[k] + T[i]) \lll s)$ .
private void secondRound (byte[] a, byte[] b, byte[] c, byte[] d, byte[] x,
                          short offset, byte s, byte t1, byte t2, byte t3, byte t4) {

//  $G(X, Y, Z) = (X \& Z) | (Y \& \sim Z)$ 
bitAnd(b, d, support_temp1);
bitComplement(d, support_temp2);
bitAnd(c, support_temp2, support_temp2);
bitOr(support_temp1, support_temp2, support_temp2);

//  $a + G(b,c,d) + X[k] + T[i]$ 
arrayCopy(x, (short)offset, support_temp1, (short)0, (short)4);
addBytes(support_temp2, support_temp1, (short)0, (short)3);
support_temp1[0] = t1;
support_temp1[1] = t2;
support_temp1[2] = t3;
support_temp1[3] = t4;
addBytes (support_temp2, support_temp1, (short)0, (short)3);
addBytes (support_temp2, a, (short)0, (short)3);

//  $b + ((a + G(b,c,d) + X[k] + T[i]) \lll s)$ 
rotateLeft(support_temp2, s, support_temp2);
addBytes(support_temp2, b, (short)0, (short)3);

//  $a = b + ((a + G(b,c,d) + X[k] + T[i]) \lll s)$ 
arrayCopy(support_temp2, (short)0, a, (short)0, (short)4);

} // end of method secondRound

// Round 3:
// When  $H(X, Y, Z) = X \wedge Y \wedge Z$ ,
// let [abcd k s i] denote the operation
//  $a = b + ((a + H(b,c,d) + X[k] + T[i]) \lll s)$ .
private void thirdRound (byte[] a, byte[] b, byte[] c, byte[] d, byte[] x,
                          short offset, byte s, byte t1, byte t2, byte t3, byte t4) {

//  $H(X, Y, Z) = X \wedge Y \wedge Z$ 
bitXor(b, c, support_temp1);
bitXor(support_temp1, d, support_temp2);

//  $a + H(b,c,d) + X[k] + T[i]$ 
arrayCopy(x, (short)offset, support_temp1, (short)0, (short)4);
addBytes(support_temp2, support_temp1, (short)0, (short)3);
support_temp1[0] = t1;
support_temp1[1] = t2;
support_temp1[2] = t3;
support_temp1[3] = t4;
addBytes (support_temp2, support_temp1, (short)0, (short)3);
addBytes (support_temp2, a, (short)0, (short)3);

//  $b + ((a + H(b,c,d) + X[k] + T[i]) \lll s)$ 
rotateLeft(support_temp2, s, support_temp2);
addBytes(support_temp2, b, (short)0, (short)3);

```

```

// a = b + ((a + H(b,c,d) + X[k] + T[i]) <<< s)
arrayCopy(support_temp2, (short)0, a, (short)0, (short)4);

} // end of method thirdRound

// Round 4:
// When I(X, Y, Z) = Y ^ (X | ~Z),
// let [abcd k s i] denote the operation
// a = b + ((a + I(b,c,d) + X[k] + T[i]) <<< s).
private void fourthRound (byte[] a, byte[] b, byte[] c, byte[] d, byte[] x,
                          short offset, byte s, byte t1, byte t2, byte t3, byte t4) {

// I(X, Y, Z) = Y ^ (X | ~Z)
bitComplement(d, support_temp1);
bitOr(a, support_temp1, support_temp1);
bitXor(c, support_temp1, support_temp2);

// a + I(b,c,d) + X[k] + T[i]
arrayCopy(x, (short)offset, support_temp1, (short)0, (short)4);
addBytes(support_temp2, support_temp1, (short)0, (short)3);
support_temp1[0] = t1;
support_temp1[1] = t2;
support_temp1[2] = t3;
support_temp1[3] = t4;
addBytes (support_temp2, support_temp1, (short)0, (short)3);
addBytes (support_temp2, a, (short)0, (short)3);

// b + ((a + I(b,c,d) + X[k] + T[i]) <<< s)
rotateLeft(support_temp2, s, support_temp2);
addBytes(support_temp2, b, (short)0, (short)3);

// a = b + ((a + I(b,c,d) + X[k] + T[i]) <<< s)
arrayCopy(support_temp2, (short)0, a, (short)0, (short)4);

} // end of method fourthRound

// two bytes are added and result will be saved in the first byte
private void addBytes (byte[] a, byte[] b, short overflow, short length) {

if (length == -1)
    return;

short one_byte = (short)((short)(a[length]&0x00FF) + (short)(b[length]&0x00FF) + overflow);
a[length] = (byte)(one_byte & (short)0x00FF);

if (overflow > 0)
    addBytes(a, b, (short)1, --length);
else
    addBytes(a, b, (short)0, --length);

return;

} // end of method addBytes

// a short value will be added to the byte array

```



```

private void addLength (byte[] total_length, short length) {

    byte[] part_length = new byte[5];
    short offset = -1;

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

    // call the roundEachLength method recursively
    for (short array_index = 7; offset >= 0; --offset, array_index = 7) {
        short one_byte = (short)((short)(total_length[array_index]&0x00ff)
            + (short)(part_length[offset]&0x00ff));
        total_length[array_index] = (byte)(one_byte & (short)0x00FF);
        if (overFlow(one_byte))
            total_length = roundEachLength(total_length, --array_index, (short)1);
    }

} // end of method addLength

private byte[] roundEachLength(byte[] total, short index, short overflow) {

    if (index < 0)
        return total;

    short one_byte = (short)((short)(total[index]&0x00ff) + overflow);
    total[index] = (byte)(one_byte & (short)0x00FF);

    if (overFlow(one_byte))
        total = roundEachLength(total, --index, (short)1);

    return total;

} // end of method roundEachLength

private boolean overFlow (short num) {

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

} // end of method overFlow

private void bitComplement (byte[] array, byte[] result) {

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

}

```

```

} // end of method bitComplement

private void bitOr (byte[] array1, byte[] array2, byte[] result) {

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

} // end of method bitOr

private void bitAnd (byte[] array1, byte[] array2, byte[] result) {

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

} // end of method bitAnd

private void bitXor (byte[] array1, byte[] array2, byte[] result) {

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

} // end of method bitAnd

// left shift in a byte array with rotation
private void rotateLeft (byte[] array, byte n_shift, byte[] result) {

    byte offset = 0x00;
    byte shift = 0x00;
    byte reverse_shift = 0x00;
    byte for_current = 0x00;
    short for_next = 0x00;

    byte first = 0x00;
    byte second = 0x00;
    byte third = 0x00;
    byte fourth = 0x00;

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

```

```

shift = (byte)(n_shift % (byte)8);
reverse_shift = (byte)((byte)0x08 - shift);

if (shift == 0) {
    shift = 8;
    reverse_shift = 0;
}

if (n_shift <= 8) {
    first = array[0];
    second = array[1];
    third = array[2];
    fourth = array[3];
} else if (n_shift <= 16) {
    first = array[1];
    second = array[2];
    third = array[3];
    fourth = array[0];
} else if (n_shift <= 24) {
    first = array[2];
    second = array[3];
    third = array[0];
    fourth = array[1];
}

for_next = (short)((fourth & offset) & 0x00FF);
result[3] = (byte)(fourth << shift);

for_current = (byte)(for_next >>> reverse_shift);
for_next = (short)((third & offset) & 0x00FF);
result[2] = (byte)((third << shift) | for_current);

for_current = (byte)(for_next >>> reverse_shift);
for_next = (short)((second & offset) & 0x00FF);
result[1] = (byte)((second << shift) | for_current);

for_current = (byte)(for_next >>> reverse_shift);
for_next = (short)((first & offset) & 0x00FF);
result[0] = (byte)((first << shift) | for_current);

for_current = (byte)(for_next >>> reverse_shift);
result[3] = (byte)(result[3] | (for_current));
} // end of method rotateLeft

private void arrayCopy (byte[] source, short s_start,
                       byte[] destination, short d_start, short length) {

    short perform = 0;

    for (perform = length; perform > 0; --perform) {
        destination[d_start] = source[s_start];
        s_start++;
        d_start++;
    }
}

```

```

    } // end of method arrayCopy

} // end of class MD5

class AppletInfo {

    private byte[] aid;
    private byte aid_length;
    private byte[] hash;
    private AppletInfo next;

    AppletInfo (byte[] aid_buffer) {

        aid = new byte[aid_buffer[ISO7816.OFFSET_LC]];

        // save a AID of each CAP file
        arrayCopy(aid_buffer, (short)(ISO7816.OFFSET_CDATA & 0x00FF),
            aid, (short)0, (short)(aid_buffer[ISO7816.OFFSET_LC] & 0x00FF));

        aid_length = aid_buffer[ISO7816.OFFSET_LC];

        hash = new byte[16];
        next = null;

    } // end of constructor

    void saveNext (AppletInfo applet) {

        next = applet;

    } // end of mehtod

    byte[] getAid () {

        return aid;

    } // end of method getAid

    byte getAidLength () {

        return aid_length;

    } // end of method getLength

    byte[] getHash () {

        return hash;

    } // end of method getHash

    AppletInfo getNext () {

        return next;

    } // end of method getNext

```

```

private void arrayCopy (byte[] source, short s_start, byte[] destination, short d_start, short length) {

    short perform = 0;

    for (perform = length; perform > 0; --perform) {
        destination[d_start] = source[s_start];
        s_start++;
        d_start++;
    }

} // end of method arrayCopy

} // end of class AppletInfo

class AppletInfoManager {

    private AppletInfo head;
    private AppletInfo temp;

    void add (AppletInfo applet) {

        if (head == null)
            head = applet;

        else {
            temp = head;
            while (temp.getNext() != null) {
                temp = temp.getNext();
            }
            temp.saveNext(applet);
        }

    } // end of method add

    // to save a hash value in an AppletInfo object
    AppletInfo getCurrent () {

        temp = head;

        while (temp.getNext() != null) {
            temp = temp.getNext();
        }

        return temp;

    } // end of method getCurrent

    AppletInfo getHead () {

        return head;

    } // end of method getHead

} // end of class AppletManager

```

## HashInstallerInterface.java

```
/**
 * The interface HashInstallerInterface contains the method verifyHashValue that is an abstract
 * method used for verifying a hash value of a CAP file.
 */

package installersupport;

import javacard.framework.Shareable;
import javacard.framework.AID;

public interface HashInstallerInterface extends Shareable {

    public boolean verifyHashValue
        (byte aid_length, byte aid1, byte aid2, byte aid3, byte aid4, byte aid5,
         byte aid6, byte aid7, byte aid8, byte aid9, byte aid10, byte aid11,
         byte aid12, byte aid13, byte aid14, byte aid15, byte aid16, byte md1,
         byte md2, byte md3, byte md4, byte md5, byte md6, byte md7, byte md8,
         byte md9, byte md10, byte md11, byte md12, byte md13, byte md14, byte md15,
         byte md16);

}

```

## HVGenerator.java

```
/**
 * The HVGenerator class sends a CAP file content, the output of the capdump tool, to the MD5
 * class, and the MD5 class that implement the MD5 algorithm generate a hash value of the content
 * of the CAP file. The MD5 class is the same class used in the proposed installer.
 */

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

class HVGenerator extends JFrame {

    private JButton select_file;
    private JButton generate;
    private JLabel label;
    private JTextField result;
    private JFileChooser fc;
    private File cap_file;
    private File cap_dump;
    private File batch;
    private MD5 md5;

    HVGenerator () {

        // for the upper panel
        setTitle("Hash Value Generator");
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        getContentPane().setLayout(null);
    }
}

```

```

getContentPane().setBackground(Color.green);

select_file = new JButton("Select a CAP file");
select_file.setBackground(Color.orange);
select_file.setBounds(25,25,275,30);

// for the bottom panel
generate = new JButton("Generate a hash value");
generate.setBackground(Color.orange);
generate.setBounds(25, 75, 275, 30);

label = new JLabel("The hash value of this CAP file");
label.setBounds(25, 150, 240, 30);
result = new JTextField();
result.setEditable(false);
result.setBounds(25, 180, 275, 25);

fc = new JFileChooser();
md5 = new MD5();

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

        result.setText(null);

        fc = new JFileChooser("c:\\demo");
        int returnVal = fc.showOpenDialog (HVGenerator.this);

        // a cap file must be selected
        if(returnVal == JFileChooser.APPROVE_OPTION)
            cap_file = fc.getSelectedFile();
        else {
            JOptionPane.showMessageDialog (HVGenerator.this,
                "You must choose a CAP file", "Error",JOptionPane.ERROR_MESSAGE);
            cap_file = null;
            return;
        }

        String temp = cap_file.getName();
        if (temp.indexOf(".cap") == -1) {
            JOptionPane.showMessageDialog (HVGenerator.this,
                "You must select a CAP file is", "Error",JOptionPane.ERROR_MESSAGE);
            cap_file = null;
        }
    }
});

// get a text representation of a CAP file to generate a hash value
generate.addActionListener(new ActionListener() {
    public void actionPerformed (ActionEvent e) {

        // create a batch file to run the capdump tool
        try {

```

```

batch = new File("dump.bat");
BufferedWriter out = new BufferedWriter(new FileWriter(batch));
out.write("capdump " + cap_file + "\n");
out.close();

Process child = Runtime.getRuntime().exec("dump.bat");
cap_dump = new File("capdump");

// standard output of the capdump tool will be recorded in a file
BufferedReader in =
    new BufferedReader(new InputStreamReader(child.getInputStream()));
BufferedWriter out2 = new BufferedWriter(new FileWriter(cap_dump));

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 {

    // a valid CAP file must be selected
    if (cap_dump.length() == 0) {
        batch.delete();
        batch = null;
        cap_dump.delete();
        cap_dump = null;
        JOptionPane.showMessageDialog (HVGenerator.this,
            "Please, select a valid CAP file", "Error",JOptionPane.ERROR_MESSAGE);
        cap_file = null;
        return;
    }
}

short length = 0;
byte[] data = new byte[64];
byte[] hv = new byte[16];
String message_digest = new String();

md5.initialize();

try {

    BufferedInputStream bufferin =
        new BufferedInputStream(new FileInputStream(cap_dump));

    // call the update method in the MD5 class to update a hash value
    while (true) {
        length = (short)(bufferin.read(data));

```



```

        if (length == -1)
            break;
        md5.update(data, length);
    }

    bufferin.close();

    // call the generate method in the MD5 class to generate a hash value
    md5.update(data, (short)0);
    md5.generate(hv);

    String temp;

    // display a message digest
    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;
    }
    result.setText(message_digest);

    batch.delete();
    batch = null;
    cap_dump.delete();
    cap_dump = null;

    } catch (FileNotFoundException ee) {
    } catch (IOException eee) {
    }

    }
});

getContentPane().add(select_file);
getContentPane().add(generate);
getContentPane().add(label);
getContentPane().add(result);

} // end of constructor

public static void main (String[] args) {

    JFrame mask = new HVGenerator();
    mask.setBounds(0, 0, 335, 250);
    mask.setVisible(true);

    } // end of method main

} // end of class HVGenerator

class MD5 {

```

```

// for S table
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;

// for padding at the end of the message
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};

private boolean pre_padding = false;

// for a 64 bits hash value
private byte[] word_a = new byte[4];
private byte[] word_b = new byte[4];
private byte[] word_c = new byte[4];
private byte[] word_d = new byte[4];

private byte[] word_aa = new byte[4];
private byte[] word_bb = new byte[4];
private byte[] word_cc = new byte[4];
private byte[] word_dd = new byte[4];

private byte[] message_length = new byte[8];
private byte[] x_table = new byte[64];

private byte[] support_temp1 = new byte[4];
private byte[] support_temp2 = new byte[4];

void initialize () {

    // initialized a message digest buffer
    // word A = {0x67, 0x45, 0x23, 0x01};
    // word B = {0xEF, 0xCD, 0xAB, 0x89};
    // word C = {0x98, 0xBA, 0xDC, 0xFE};
    // word D = {0x10, 0x32, 0x54, 0x76};
    word_a[0] = (byte)0x67;
    word_a[1] = (byte)0x45;

```

```

word_a[2] = (byte)0x23;
word_a[3] = (byte)0x01;
word_b[0] = (byte)0xEF;
word_b[1] = (byte)0xCD;
word_b[2] = (byte)0xAB;
word_b[3] = (byte)0x89;
word_c[0] = (byte)0x98;
word_c[1] = (byte)0xBA;
word_c[2] = (byte)0xDC;
word_c[3] = (byte)0xFE;
word_d[0] = (byte)0x10;
word_d[1] = (byte)0x32;
word_d[2] = (byte)0x54;
word_d[3] = (byte)0x76;

for (short i = 0; i < 8; i++)
    message_length[i] = 0x00;

pre_padding = false;
} // end of method initialize

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

    short operation = 0;

    switch (length) {

        // during transmit a message block (each 64 bytes)
        case 64: operation = 1;
                break;
        // after transmit the last message block
        case 0: // when the length of total message is 0 module 512
                if (pre_padding == false) {
                    operation = -1;
                    break;
                }
                else
                    return;
        // when transmit the last message block
        default: operation = 3;
                pre_padding = true;
    }

    // add the length of a message block to the total length of message
    addLength(message_length, length);

    // save word_a as word_aa, word_b as word_bb,
    // word_c as word_cc, word_d as word_dd
    arrayCopy(word_a, (short)0, word_aa, (short)0, (short)4);
    arrayCopy(word_b, (short)0, word_bb, (short)0, (short)4);
    arrayCopy(word_c, (short)0, word_cc, (short)0, (short)4);
    arrayCopy(word_d, (short)0, word_dd, (short)0, (short)4);

do {

```

```

// build x_table
switch (operation) {

// padding 56 bytes (starts with 1) followed by total length of message (8 bytes)
case -1: arrayCopy(padding, (short)0, x_table, (short)0, (short)56);
        arrayCopy(message_length, (short)0, x_table, (short)56, (short)8);
        operation = 0;
        break;
// save a message block (64 bytes) to x_table
case 1: arrayCopy(buffer, (short)0, x_table, (short)0, (short)64);
        operation = 0;
        break;

// padding 56 bytes (start with 0) followed by total length of message (8 bytes)
case 2: arrayCopy(padding, (short)8, x_table, (short)0, (short)56);
        arrayCopy(message_length, (short)0, x_table, (short)56, (short)8);
        operation = 0;
        break;

// the last message block followed by at least one byte or at most 64 bytes
// padding and total length of message (8 bytes)
case 3: if (length < 56) {
        short required_pad = (short)((short)56 - length);
        arrayCopy(buffer, (short)0, x_table, (short)0, length);
        arrayCopy(padding, (short)0, x_table, (short)length, required_pad);
        arrayCopy(message_length, (short)0, x_table, (short)56, (short)8);
        operation = 0;
    }
    else {
        short required_pad = (short)((short)64 - length);
        arrayCopy(buffer, (short)0, x_table, (short)0, length);
        arrayCopy(padding, (short)0, x_table, (short)length, required_pad);
        operation = 2;
    }
    break;
}

// Round I: do the following 16 operations
firstRound (word_a, word_b, word_c, word_d, x_table,
            (short)0, S11, (byte)0xD7, (byte)0x6A, (byte)0xA4, (byte)0x78); /* 1 */
firstRound (word_d, word_a, word_b, word_c, x_table,
            (short)4, S12, (byte)0xE8, (byte)0xC7, (byte)0xB7, (byte)0x56); /* 2 */
firstRound (word_c, word_d, word_a, word_b, x_table,
            (short)8, S13, (byte)0x24, (byte)0x20, (byte)0x70, (byte)0xDB); /* 3 */
firstRound (word_b, word_c, word_d, word_a, x_table,
            (short)12, S14, (byte)0xC1, (byte)0xBD, (byte)0xCE, (byte)0xEE); /* 4 */
firstRound (word_a, word_b, word_c, word_d, x_table,
            (short)16, S11, (byte)0xF5, (byte)0x7C, (byte)0x0F, (byte)0xAF); /* 5 */
firstRound (word_d, word_a, word_b, word_c, x_table,
            (short)20, S12, (byte)0x47, (byte)0x87, (byte)0xC6, (byte)0x2A); /* 6 */
firstRound (word_c, word_d, word_a, word_b, x_table,
            (short)24, S13, (byte)0xA8, (byte)0x30, (byte)0x46, (byte)0x13); /* 7 */
firstRound (word_b, word_d, word_d, word_a, x_table,
            (short)28, S14, (byte)0xFD, (byte)0x46, (byte)0x95, (byte)0x01); /* 8 */
firstRound (word_a, word_b, word_c, word_d, x_table,
            (short)32, S11, (byte)0x69, (byte)0x80, (byte)0x98, (byte)0xD8); /* 9 */

```

```

firstRound (word_d, word_a, word_b, word_c, x_table,
            (short)36, S12, (byte)0x8B, (byte)0x44, (byte)0xF7, (byte)0xAF); /* 10 */
firstRound (word_c, word_d, word_a, word_b, x_table,
            (short)40, S13, (byte)0xFF, (byte)0xFF, (byte)0x5B, (byte)0xB1); /* 11 */
firstRound (word_b, word_c, word_d, word_a, x_table,
            (short)44, S14, (byte)0x89, (byte)0x5C, (byte)0xD7, (byte)0xBE); /* 12 */
firstRound (word_a, word_b, word_c, word_d, x_table,
            (short)48, S11, (byte)0x6B, (byte)0x90, (byte)0x11, (byte)0x22); /* 13 */
firstRound (word_d, word_a, word_b, word_c, x_table,
            (short)52, S12, (byte)0xFD, (byte)0x98, (byte)0x71, (byte)0x93); /* 14 */
firstRound (word_c, word_d, word_a, word_b, x_table,
            (short)56, S13, (byte)0xA6, (byte)0x79, (byte)0x43, (byte)0x8E); /* 15 */
firstRound (word_b, word_c, word_d, word_a, x_table,
            (short)60, S14, (byte)0x49, (byte)0xB4, (byte)0x08, (byte)0x21); /* 16 */

// Round 2: do the following 16 operations
secondRound (word_a, word_b, word_c, word_d, x_table,
            (short)4, S21, (byte)0xF6, (byte)0x1E, (byte)0x25, (byte)0x62); /* 1 */
secondRound (word_d, word_a, word_b, word_c, x_table,
            (short)24, S22, (byte)0xC0, (byte)0x40, (byte)0xB3, (byte)0x40); /* 2 */
secondRound (word_c, word_d, word_a, word_b, x_table,
            (short)44, S23, (byte)0x26, (byte)0x5E, (byte)0x5A, (byte)0x51); /* 3 */
secondRound (word_b, word_c, word_d, word_a, x_table,
            (short)0, S24, (byte)0xE9, (byte)0xB6, (byte)0xC7, (byte)0xAA); /* 4 */
secondRound (word_a, word_b, word_c, word_d, x_table,
            (short)20, S21, (byte)0xD6, (byte)0x2F, (byte)0x10, (byte)0x5D); /* 5 */
secondRound (word_d, word_a, word_b, word_c, x_table,
            (short)40, S22, (byte)0x2, (byte)0x44, (byte)0x14, (byte)0x53); /* 6 */
secondRound (word_c, word_d, word_a, word_b, x_table,
            (short)60, S23, (byte)0xD8, (byte)0xA1, (byte)0xE6, (byte)0x81); /* 7 */
secondRound (word_b, word_d, word_d, word_a, x_table,
            (short)24, S24, (byte)0xE7, (byte)0xD3, (byte)0xFB, (byte)0xC8); /* 8 */
secondRound (word_a, word_b, word_c, word_d, x_table,
            (short)36, S21, (byte)0x21, (byte)0xE1, (byte)0xCD, (byte)0xE6); /* 9 */
secondRound (word_d, word_a, word_b, word_c, x_table,
            (short)56, S22, (byte)0xC3, (byte)0x37, (byte)0x07, (byte)0xD6); /* 10 */
secondRound (word_c, word_d, word_a, word_b, x_table,
            (short)12, S23, (byte)0xF4, (byte)0xD5, (byte)0x0D, (byte)0x87); /* 11 */
secondRound (word_b, word_c, word_d, word_a, x_table,
            (short)32, S24, (byte)0x45, (byte)0x5A, (byte)0x14, (byte)0xED); /* 12 */
secondRound (word_a, word_b, word_c, word_d, x_table,
            (short)52, S21, (byte)0xA9, (byte)0xE3, (byte)0xF9, (byte)0x05); /* 13 */
secondRound (word_d, word_a, word_b, word_c, x_table,
            (short)8, S22, (byte)0xFC, (byte)0xEF, (byte)0xA3, (byte)0xF8); /* 14 */
secondRound (word_c, word_d, word_a, word_b, x_table,
            (short)28, S23, (byte)0x67, (byte)0x6F, (byte)0x02, (byte)0xD9); /* 15 */
secondRound (word_b, word_c, word_d, word_a, x_table,
            (short)48, S24, (byte)0x8D, (byte)0x2A, (byte)0x4C, (byte)0x8A); /* 16 */

// Round 3: do the following 16 operations
thirdRound (word_a, word_b, word_c, word_d, x_table,
            (short)20, S31, (byte)0xFF, (byte)0xFA, (byte)0x39, (byte)0x42); /* 1 */
thirdRound (word_d, word_a, word_b, word_c, x_table,
            (short)32, S32, (byte)0x87, (byte)0x71, (byte)0xF6, (byte)0x81); /* 2 */
thirdRound (word_c, word_d, word_a, word_b, x_table,
            (short)44, S33, (byte)0x6D, (byte)0x9D, (byte)0x61, (byte)0x22); /* 3 */

```

```

thirdRound (word_b, word_c, word_d, word_a, x_table,
            (short)56, S34, (byte)0xFD, (byte)0xE5, (byte)0x38, (byte)0x0C); /* 4 */
thirdRound (word_a, word_b, word_c, word_d, x_table,
            (short)4, S31, (byte)0xA4, (byte)0xBE, (byte)0xEA, (byte)0x44); /* 5 */
thirdRound (word_d, word_a, word_b, word_c, x_table,
            (short)16, S32, (byte)0x4B, (byte)0xDE, (byte)0xCF, (byte)0xA9); /* 6 */
thirdRound (word_c, word_d, word_a, word_b, x_table,
            (short)28, S33, (byte)0xF6, (byte)0xBB, (byte)0x4B, (byte)0x80); /* 7 */
thirdRound (word_b, word_d, word_d, word_a, x_table,
            (short)40, S34, (byte)0xBE, (byte)0xBF, (byte)0xBC, (byte)0x70); /* 8 */
thirdRound (word_a, word_b, word_c, word_d, x_table,
            (short)52, S31, (byte)0x28, (byte)0x9B, (byte)0x7E, (byte)0xC6); /* 9 */
thirdRound (word_d, word_a, word_b, word_c, x_table,
            (short)0, S32, (byte)0xEA, (byte)0xA1, (byte)0x27, (byte)0xFA); /* 10 */
thirdRound (word_c, word_d, word_a, word_b, x_table,
            (short)12, S33, (byte)0xD4, (byte)0xEF, (byte)0x30, (byte)0x85); /* 11 */
thirdRound (word_b, word_c, word_d, word_a, x_table,
            (short)24, S34, (byte)0x4, (byte)0x88, (byte)0x1D, (byte)0x05); /* 12 */
thirdRound (word_a, word_b, word_c, word_d, x_table,
            (short)36, S31, (byte)0xD9, (byte)0xD4, (byte)0xD0, (byte)0x39); /* 13 */
thirdRound (word_d, word_a, word_b, word_c, x_table,
            (short)48, S32, (byte)0xE6, (byte)0xDB, (byte)0x99, (byte)0xE5); /* 14 */
thirdRound (word_c, word_d, word_a, word_b, x_table,
            (short)60, S33, (byte)0x1F, (byte)0xA2, (byte)0x7C, (byte)0xF8); /* 15 */
thirdRound (word_b, word_c, word_d, word_a, x_table,
            (short)8, S34, (byte)0xC4, (byte)0xAC, (byte)0x56, (byte)0x65); /* 16 */

// Round 4: do the following 16 operations
fourthRound (word_a, word_b, word_c, word_d, x_table,
            (short)0, S41, (byte)0xF4, (byte)0x29, (byte)0x22, (byte)0x44); /* 1 */
fourthRound (word_d, word_a, word_b, word_c, x_table,
            (short)28, S42, (byte)0x43, (byte)0x2A, (byte)0xFF, (byte)0x97); /* 2 */
fourthRound (word_c, word_d, word_a, word_b, x_table,
            (short)56, S43, (byte)0xAB, (byte)0x94, (byte)0x23, (byte)0xA7); /* 3 */
fourthRound (word_b, word_c, word_d, word_a, x_table,
            (short)20, S44, (byte)0xFC, (byte)0x93, (byte)0xA0, (byte)0x39); /* 4 */
fourthRound (word_a, word_b, word_c, word_d, x_table,
            (short)48, S41, (byte)0x65, (byte)0x5B, (byte)0x59, (byte)0xC3); /* 5 */
fourthRound (word_d, word_a, word_b, word_c, x_table,
            (short)12, S42, (byte)0x8F, (byte)0x0C, (byte)0xCC, (byte)0x92); /* 6 */
fourthRound (word_c, word_d, word_a, word_b, x_table,
            (short)40, S43, (byte)0xFF, (byte)0xEF, (byte)0xF4, (byte)0x7D); /* 7 */
fourthRound (word_b, word_d, word_d, word_a, x_table,
            (short)4, S44, (byte)0x85, (byte)0x84, (byte)0x5D, (byte)0xD1); /* 8 */
fourthRound (word_a, word_b, word_c, word_d, x_table,
            (short)32, S41, (byte)0x6F, (byte)0xA8, (byte)0x7E, (byte)0x4F); /* 9 */
fourthRound (word_d, word_a, word_b, word_c, x_table,
            (short)60, S42, (byte)0xFE, (byte)0x2C, (byte)0xE6, (byte)0xE0); /* 10 */
fourthRound (word_c, word_d, word_a, word_b, x_table,
            (short)24, S43, (byte)0xA3, (byte)0x01, (byte)0x43, (byte)0x14); /* 11 */
fourthRound (word_b, word_d, word_a, word_c, x_table,
            (short)52, S44, (byte)0x4E, (byte)0x08, (byte)0x11, (byte)0xA1); /* 12 */
fourthRound (word_a, word_b, word_c, word_d, x_table,
            (short)16, S41, (byte)0xF7, (byte)0x53, (byte)0x7E, (byte)0x82); /* 13 */
fourthRound (word_d, word_a, word_b, word_c, x_table,
            (short)44, S42, (byte)0xBD, (byte)0x3A, (byte)0xF2, (byte)0x35); /* 14 */

```

```

fourthRound (word_c, word_d, word_a, word_b, x_table,
             (short)8, S43, (byte)0x2A, (byte)0xD7, (byte)0xD2, (byte)0xBB); /* 15 */
fourthRound (word_b, word_c, word_d, word_a, x_table,
             (short)36, S44, (byte)0xEB, (byte)0x86, (byte)0xD3, (byte)0x91); /* 16 */

// perform the increment each of the four words
// by the value it had before this block was started
// word_a = word_a + word_aa
// word_b = word_b + word_bb
// word_c = word_c + word_cc
// word_d = word_d + word_dd
addBytes(word_a, word_aa, (short)0, (short)3);
addBytes(word_b, word_bb, (short)0, (short)3);
addBytes(word_c, word_cc, (short)0, (short)3);
addBytes(word_d, word_dd, (short)0, (short)3);

} while (operation != 0);

} // end of method update

void generate (byte[] md) {

// the message digest produced as output is A, B, C, D.
// That is, we begin with the low-order byte of A, and end with the
// high-order byte of D.

for (short i = 0, j = 3; j >= 0; ++i, --j)
    md[i] = word_a[j];

for (short i = 4, j = 3; j >= 0; ++i, --j)
    md[i] = word_b[j];

for (short i = 8, j = 3; j >= 0; ++i, --j)
    md[i] = word_c[j];

for (short i = 12, j = 3; j >= 0; ++i, --j)
    md[i] = word_d[j];

} // end of method generate

// Round 1:
// When  $F(X, Y, Z) = (X \& Y) | (\sim X \& Z)$ ,
// let [abcd k s i] denote the operation
//  $a = b + ((a + F(b,c,d) + X[k] + T[i]) \lll s)$ .
private void firstRound (byte[] a, byte[] b, byte[] c, byte[] d, byte[] x,
                        short offset, byte s, byte t1, byte t2, byte t3, byte t4) {

//  $F(X, Y, Z) = (X \& Y) | (\sim X \& Z)$ 
bitAnd(b, c, support_temp1);
bitComplement(b, support_temp2);
bitAnd(support_temp2, d, support_temp2);
bitOr(support_temp1, support_temp2, support_temp2);

//  $a + F(b,c,d) + X[k] + T[i]$ 
arrayCopy(x, (short)offset, support_temp1, (short)0, (short)4);
addBytes(support_temp2, support_temp1, (short)0, (short)3);

```

```

support_temp1[0] = t1;
support_temp1[1] = t2;
support_temp1[2] = t3;
support_temp1[3] = t4;
addBytes (support_temp2, support_temp1, (short)0, (short)3);
addBytes (support_temp2, a, (short)0, (short)3);

//  $b + ((a + F(b,c,d) + X[k] + T[i]) \lll s)$ 
rotateLeft(support_temp2, s, support_temp2);
addBytes(support_temp2, b, (short)0, (short)3);

//  $a = b + ((a + F(b,c,d) + X[k] + T[i]) \lll s)$ 
arrayCopy(support_temp2, (short)0, a, (short)0, (short)4);
} // end of method firstRound

// Round 2:
// When  $G(X, Y, Z) = (X \& Z) | (Y \& \sim Z)$ ,
// let [abcd k s i] denote the operation
//  $a = b + ((a + G(b,c,d) + X[k] + T[i]) \lll s)$ .
private void secondRound (byte[] a, byte[] b, byte[] c, byte[] d, byte[] x,
                          short offset, byte s, byte t1, byte t2, byte t3, byte t4) {

//  $G(X, Y, Z) = (X \& Z) | (Y \& \sim Z)$ 
bitAnd(b, d, support_temp1);
bitComplement(d, support_temp2);
bitAnd(c, support_temp2, support_temp2);
bitOr(support_temp1, support_temp2, support_temp2);

//  $a + G(b,c,d) + X[k] + T[i]$ 
arrayCopy(x, (short)offset, support_temp1, (short)0, (short)4);
addBytes(support_temp2, support_temp1, (short)0, (short)3);
support_temp1[0] = t1;
support_temp1[1] = t2;
support_temp1[2] = t3;
support_temp1[3] = t4;
addBytes (support_temp2, support_temp1, (short)0, (short)3);
addBytes (support_temp2, a, (short)0, (short)3);

//  $b + ((a + G(b,c,d) + X[k] + T[i]) \lll s)$ 
rotateLeft(support_temp2, s, support_temp2);
addBytes(support_temp2, b, (short)0, (short)3);

//  $a = b + ((a + G(b,c,d) + X[k] + T[i]) \lll s)$ 
arrayCopy(support_temp2, (short)0, a, (short)0, (short)4);

} // end of method secondRound

// Round 3:
// When  $H(X, Y, Z) = X \wedge Y \wedge Z$ ,
// let [abcd k s i] denote the operation
//  $a = b + ((a + H(b,c,d) + X[k] + T[i]) \lll s)$ .
private void thirdRound (byte[] a, byte[] b, byte[] c, byte[] d, byte[] x,
                          short offset, byte s, byte t1, byte t2, byte t3, byte t4) {

//  $H(X, Y, Z) = X \wedge Y \wedge Z$ .

```



```

bitXor(b, c, support_temp1);
bitXor(support_temp1, d, support_temp2);

// a + H(b,c,d) + X[k] + T[i]
arrayCopy(x, (short)offset, support_temp1, (short)0, (short)4);
addBytes(support_temp2, support_temp1, (short)0, (short)3);
support_temp1[0] = t1;
support_temp1[1] = t2;
support_temp1[2] = t3;
support_temp1[3] = t4;
addBytes (support_temp2, support_temp1, (short)0, (short)3);
addBytes (support_temp2, a, (short)0, (short)3);

// b + ((a + H(b,c,d) + X[k] + T[i]) <<< s)
rotateLeft(support_temp2, s, support_temp2);
addBytes(support_temp2, b, (short)0, (short)3);

// a = b + ((a + H(b,c,d) + X[k] + T[i]) <<< s)
arrayCopy(support_temp2, (short)0, a, (short)0, (short)4);

} // end of method thirdRound

// Round 4:
// When I(X, Y, Z) = Y ^ (X | ~Z),
// let [abcd k s i] denote the operation
// a = b + ((a + I(b,c,d) + X[k] + T[i]) <<< s).
private void fourthRound (byte[] a, byte[] b, byte[] c, byte[] d, byte[] x,
                          short offset, byte s, byte t1, byte t2, byte t3, byte t4) {

// I(X, Y, Z) = Y ^ (X | ~Z)
bitComplement(d, support_temp1);
bitOr(a, support_temp1, support_temp1);
bitXor(c, support_temp1, support_temp2);

// a + I(b,c,d) + X[k] + T[i]
arrayCopy(x, (short)offset, support_temp1, (short)0, (short)4);
addBytes(support_temp2, support_temp1, (short)0, (short)3);
support_temp1[0] = t1;
support_temp1[1] = t2;
support_temp1[2] = t3;
support_temp1[3] = t4;
addBytes (support_temp2, support_temp1, (short)0, (short)3);
addBytes (support_temp2, a, (short)0, (short)3);

// b + ((a + I(b,c,d) + X[k] + T[i]) <<< s)
rotateLeft(support_temp2, s, support_temp2);
addBytes(support_temp2, b, (short)0, (short)3);

// a = b + ((a + I(b,c,d) + X[k] + T[i]) <<< s)
arrayCopy(support_temp2, (short)0, a, (short)0, (short)4);

} // end of method fourthRound

// two bytes are added and result will be saved in the first byte
private void addBytes (byte[] a, byte[] b, short overflow, short length) {

```

```

if (length == -1)
    return;

short one_byte = (short)((short)(a[length]&0x00FF) + (short)(b[length]&0x00FF) + overflow);
a[length] = (byte)(one_byte & (short)0x00FF);

if (overflow(one_byte))
    addBytes(a, b, (short)1, --length);
else
    addBytes(a, b, (short)0, --length);

return;
} // end of method addBytes

// a short value will be added to the byte array
private void addLength (byte[] total_length, short length) {

byte[] part_length = new byte[5];
short offset = -1;

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

// call the roundEachLength method recursively
for (short array_index = 7; offset >= 0; --offset, array_index = 7) {
    short one_byte = (short)((short)(total_length[array_index]&0x00ff)
        + (short)(part_length[offset]&0x00ff));
    total_length[array_index] = (byte)(one_byte & (short)0x00FF);
    if (overflow(one_byte))
        total_length = roundEachLength(total_length, --array_index, (short)1);
}

} // end of method addLength

private byte[] roundEachLength(byte[] total, short index, short overflow) {

if (index < 0)
    return total;

short one_byte = (short)((short)(total[index]&0x00ff) + overflow);
total[index] = (byte)(one_byte & (short)0x00FF);

if (overflow(one_byte))
    total = roundEachLength(total, --index, (short)1);

return total;
}

```

```

} // end of method roundEachLength

private boolean overFlow (short num) {
    if ((short)(num & (short)0xFF00) >= (short)0x0100)
        return true;
    else
        return false;
} // end of method overFlow

private void bitComplement (byte[] array, byte[] result) {
    for (short i = 0; i < (short)4; i++)
        result[i] = (byte)~(array[i]);
} // end of method bitComplement

private void bitOr (byte[] array1, byte[] array2, byte[] result) {
    for (short i = 0; i < (short)4; i++)
        result[i] = (byte)(array1[i] | array2[i]);
} // end of method bitOr

private void bitAnd (byte[] array1, byte[] array2, byte[] result) {
    for (short i = 0; i < (short)4; i++)
        result[i] = (byte)(array1[i] & array2[i]);
} // end of method bitAnd

private void bitXor (byte[] array1, byte[] array2, byte[] result) {
    for (short i = 0; i < (short)4; i++)
        result[i] = (byte)(array1[i] ^ array2[i]);
} // end of method bitAnd

// left shift in a byte array with rotation
private void rotateLeft (byte[] array, byte n_shift, byte[] result) {
    byte offset = 0x00;
    byte shift = 0x00;
    byte reverse_shift = 0x00;
    byte for_current = 0x00;
    short for_next = 0x00;

    byte first = 0x00;
    byte second = 0x00;
    byte third = 0x00;
    byte fourth = 0x00;

    switch (n_shift % (byte)8) {
    case 0: offset = (byte)0xFF;
            break;

```

```

case 1: offset = (byte)0x80;
        break;
case 2: offset = (byte)0xC0;
        break;
case 3: offset = (byte)0xE0;
        break;
case 4: offset = (byte)0xF0;
        break;
case 5: offset = (byte)0xF8;
        break;
case 6: offset = (byte)0xFC;
        break;
case 7: offset = (byte)0xFE;
        break;
}

shift = (byte)(n_shift % (byte)8);
reverse_shift = (byte)((byte)0x08 - shift);

```

```

if (shift == 0) {
    shift = 8;
    reverse_shift = 0;
}

```

```

if (n_shift <= 8) {
    first = array[0];
    second = array[1];
    third = array[2];
    fourth = array[3];
} else if (n_shift <= 16) {
    first = array[1];
    second = array[2];
    third = array[3];
    fourth = array[0];
} else if (n_shift <= 24) {
    first = array[2];
    second = array[3];
    third = array[0];
    fourth = array[1];
}

```

```

for_next = (short)((fourth & offset) & 0x00FF);
result[3] = (byte)(fourth << shift);

```

```

for_current = (byte)(for_next >>> reverse_shift);
for_next = (short)((third & offset) & 0x00FF);
result[2] = (byte)((third << shift) | for_current);

```

```

for_current = (byte)(for_next >>> reverse_shift);
for_next = (short)((second & offset) & 0x00FF);
result[1] = (byte)((second << shift) | for_current);

```

```

for_current = (byte)(for_next >>> reverse_shift);
for_next = (short)((first & offset) & 0x00FF);
result[0] = (byte)((first << shift) | for_current);

```

```

    for_current = (byte)(for_next >>> reverse_shift);
    result[3] = (byte)(result[3] | (for_current));

} // end of method rotateLeft

private void arrayCopy (byte[] source, short s_start,
                       byte[] destination, short d_start, short length) {

    short perform = 0;

    for (perform = length; perform > 0; --perform) {
        destination[d_start] = source[s_start];
        s_start++;
        d_start++;
    }

} // end of method arrayCopy

} // end of class MD5

```

## Terminal.java

```

/*****
The class Terminal contains the off-card installation program that transmits a CAP file not
only to the Java Card installer but also to the HashInstaller installed onto a card before other
applets are installed. Also, this class enables for a user to run the converter tool, the scriptgen
tool, and the apdutool tool in the Java Card 2.2 Development Kit with pre-created batch files.
*****/

```

```

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

class Terminal extends JFrame {

    // for the top panel
    private JPanel download;
    private JButton download_files;
    private JRadioButton is_server;
    private JRadioButton is_client;
    private ButtonGroup group;

    // for the middle panel
    private JPanel convert;
    private JButton converter;

    // for the bottom panel
    private JPanel off_card;
    private JButton select_cap;
    private JButton install;

    private JFileChooser fc;
    private File cap_file;
    private File batch;

```

```

private File scr_file;
private File out_file;
private File cap_dump;

Terminal () {

setTitle("Terminal");
setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
getContentPane().setLayout(null);
getContentPane().setBackground(Color.green);

// for the top panel
download = new JPanel();
download.setBounds(25, 25, 250, 105);
download.setBackground(Color.green);
download.setBorder(BorderFactory.createTitledBorder
(BorderFactory.createLineBorder(Color.black), " Download "));

download_files = new JButton("    Download    ");
download_files.setBackground(Color.orange);
download_files.setSize(100, 40);

is_server = new JRadioButton("Server Applet");
is_server.setBackground(Color.green);
is_client = new JRadioButton("Client Applet");
is_client.setBackground(Color.green);
group = new ButtonGroup();
group.add(is_server);
group.add(is_client);

download.add(is_server);
download.add(is_client);
download.add(download_files);

// for the middle panel
convert = new JPanel();
convert.setBounds(25, 145, 250, 75);
convert.setBackground(Color.green);
convert.setBorder(BorderFactory.createTitledBorder
(BorderFactory.createLineBorder(Color.black), " Converter "));

converter = new JButton("    Convert    ");
converter.setBackground(Color.orange);
converter.setSize(150, 40);
convert.add(converter);

// for the bottom panel
off_card = new JPanel();
off_card.setBounds(25, 245, 250, 75);
off_card.setBackground(Color.green);
off_card.setBorder(BorderFactory.createTitledBorder
(BorderFactory.createLineBorder(Color.black), " Off-card installation program "));

install = new JButton("    Transmit    ");
install.setBackground(Color.orange);
install.setSize(200, 40);

```

```

off_card.add(install);

// to simulate download applets' classfiles via Internet
download_files.addActionListener(new ActionListener() {
    public void actionPerformed (ActionEvent e) {

        // to run a specific batch file
        if (!(is_server.isSelected() || is_client.isSelected())) {
            JOptionPane.showMessageDialog
                (Terminal.this, "Select one of following buttons: Server Applet or Client Applet",
                 "Error",JOptionPane.ERROR_MESSAGE);
            return;
        }

        // s_down_1(2).bat is a batch file contains commands for copying class files
        // from other directory
        if (is_server.isSelected()) {
            try {
                String command = "c:/demo/sourcecode/systemfile/s_down_1.bat";
                Process child = Runtime.getRuntime().exec(command);
                child.waitFor();
            } catch (Exception e2) {
            }

            try {
                String command = "c:/demo/sourcecode/systemfile/s_down_2.bat";
                Process child = Runtime.getRuntime().exec(command);
            } catch (Exception e3) {
            }
        }

        // c_down.bat is a batch file contains commands for copying class files
        // from other directory
        if (is_client.isSelected()) {
            try {
                String command = "c:/demo/sourcecode/systemfile/c_down.bat";
                Process child = Runtime.getRuntime().exec(command);
            } catch (Exception e1) {
            }
        }
    }
});

// to convert class files and export files to a CAP file and a export file
converter.addActionListener(new ActionListener() {
    public void actionPerformed (ActionEvent e) {

        // s_convert.bat is a batch file contains commands to run the convert tool
        if (is_server.isSelected()) {
            try {
                String command = "c:/demo/sourcecode/systemfile/s_convert.bat";
                Process child = Runtime.getRuntime().exec(command);
            } catch (Exception ee) {
            }
        }
    }
});

```

```

// c_convert.bat is a batch file contains commands to run the convert tool
if (is_client.isSelected()) {
    try {
        String command = "c:/demo/sourcecode/systemfile/c_convert.bat";
        Process child = Runtime.getRuntime().exec(command);
    } catch (Exception ee) {
    }
}

});

// to transmits a CAP file to the Java Card installer and the HashInstaller
// to generate a hash value of the CAP file
install.addActionListener(new ActionListener() {
    public void actionPerformed (ActionEvent e) {

        fc = new JFileChooser("c:\\demo\\terminal");
        fc.setDialogTitle("Select a CAP file");
        int returnVal = fc.showOpenDialog(Terminal.this);

        // a CAP file must be selected for conversion process
        if (returnVal == JFileChooser.APPROVE_OPTION)
            cap_file = fc.getSelectedFile();
        else {
            JOptionPane.showMessageDialog (Terminal.this, "You must choose a CAP file",
                "Error",JOptionPane.ERROR_MESSAGE);
            cap_file = null;
            return;
        }

        String temp = cap_file.getName();
        if (temp.indexOf(".cap") == -1) {
            JOptionPane.showMessageDialog (Terminal.this, "You must select a CAP file is",
                "Error",JOptionPane.ERROR_MESSAGE);
            cap_file = null;
            return;
        }

        // create a batch file and run the scriptgen tool
        try {

            batch = new File("script.bat");
            scr_file = new File("apdu.scr");
            BufferedWriter out = new BufferedWriter(new FileWriter(batch));
            out.write("@echo off\n");
            out.write("scriptgen -o " + scr_file.getAbsolutePath() + " "
                + cap_file.getAbsolutePath() + "\n");
            out.close();

            String batch_path = batch.getAbsolutePath();
            batch_path = batch_path.replace("\\", "/");
            Process child = Runtime.getRuntime().exec(batch_path);
            child.waitFor();
        }
    }
});

```



```

} catch (Exception e1) {
} finally {
    // after run the scriptgen tool, the batch file will be deleted
    batch.delete();
    batch = null;
}

String dumps = new String();
cap_dump = new File("dump");

// create a batch file and run the capdump tool
try {

    batch = new File("gendump.bat");
    BufferedWriter out = new BufferedWriter(new FileWriter(batch));
    out.write("@echo off\n");
    out.write("capdump " + cap_file.getAbsolutePath() + "\n");
    out.close();

    String batch_path = batch.getAbsolutePath();
    batch_path = batch_path.replace("\\", "/");
    Process child = Runtime.getRuntime().exec(batch_path);

    // standard output of the capdump tool will be recorded in a file
    BufferedReader in = new BufferedReader
        (new InputStreamReader(child.getInputStream()));
    BufferedWriter out2 = new BufferedWriter(new FileWriter(cap_dump));

    while ((dumps = in.readLine()) != null)
        out2.write(dumps + "\n");
    out2.close();

} catch (Exception ee) {
} finally {
    // after run the capdump tool, the batch file will be deleted
    batch.delete();
    batch = null;
    cap_file = null;
}

out_file = new File("updated_s_apdu.scr");
String input = new String();

byte[] data = new byte[64];
int length = 0;
byte apdu_length = 0;
String temp_byte = new String();

// to update the previously generated script file by the scriptgen tool to transmits
// a CAP file to the Java Card installer and the HashInstaller
try {

    BufferedReader reader = new BufferedReader(new FileReader(scr_file));
    BufferedWriter writer = new BufferedWriter(new FileWriter(out_file));

    writer.write("powerup;\n\n");

```

```

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

while ((input = reader.readLine()) != null) {
  writer.write(input + "\n");
}

// create an instance of this applet
writer.write("\n");
writer.write("//Install this applet\n");
if (is_server.isSelected()) {
  writer.write
    ("0x80 0xB8 0x00 0x00 0x0A 0x08 0x00 0x00 0x00 0x00 0x0B 0x00 0x00 0x01 0x00
0x7F;\n\n");
}
else {
  writer.write
    ("0x80 0xB8 0x00 0x00 0x0A 0x08 0x00 0x00 0x00 0x00 0x0C 0x00 0x00 0x01 0x00
0x7F;\n\n");
}
reader.close();
BufferedInputStream bufferin = new BufferedInputStream(new FileInputStream(cap_dump));

// select hashinstaller applet
writer.write("//Select hash installer\n");
writer.write("0x00 0xA4 0x04 0x00 0x08 0x00 0x00 0x00 0x00 0x0A 0x00 0x00 0x01
0x7F;\n\n");

// for save AID
writer.write("//Save AID\n");
if (is_server.isSelected())
  writer.write("0xB0 0x01 0x00 0x00 0x08 0x00 0x00 0x00 0x00 0x0B 0x00 0x00 0x01
0x7F;\n\n");
else
  writer.write("0xB0 0x01 0x00 0x00 0x08 0x00 0x00 0x00 0x00 0x0C 0x00 0x00 0x01
0x7F;\n\n");

writer.write("//Start of CAP content\n");
writer.write("0xB0 0x02 0x00 0x00 0x00 0x7F;\n\n");

// write a CAP content into script file
while (true) {

  length = (bufferin.read(data));

  if (length == -1)
    break;

  // to change to hex from decimal
  apdu_length = (byte)(length & 0x000000FF);
  temp_byte = (Integer.toString(apdu_length & 0x000000FF, 16)).toUpperCase();

  if (temp_byte.length() == 1)
    temp_byte = "0" + temp_byte;
}

```

```

writer.write("0xB0 0x03 0x00 0x00 " + "0x" + temp_byte + " ");

for (int i = 0; i < length; i++) {
    temp_byte = (Integer.toString(data[i] & 0x000000FF, 16)).toUpperCase();
    if (temp_byte.length() == 1)
        temp_byte = "0" + temp_byte;
    writer.write("0x" + temp_byte + " ");
}

writer.write("0x7F;\n");

} // end of while

// generate a message digest
writer.write("\n");
writer.write("// End of CAP content\n");
writer.write("0xB0 0x04 0x00 0x00 0x00 0x7F;\n\n");
writer.write("powerdown;");

bufferin.close();
writer.close();

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

// create a batch file and run the apdutil tool
try {

    batch = new File("install.bat");
    BufferedWriter out = new BufferedWriter(new FileWriter(batch));
    out.write("@echo off\n");
    out.write("apdutil -o result " + out_file.getAbsolutePath() + "\n");
    out.close();

    String batch_path = batch.getAbsolutePath();
    batch_path = batch_path.replace("\\", "/");
    Process child = Runtime.getRuntime().exec(batch_path);
    child.waitFor();

} catch (Exception e1) {
} finally {
    batch.delete();
    batch = null;
    out_file.delete();
    out_file = null;
}

}
});

```

```
getContentPane().add(download);
getContentPane().add(convert);
getContentPane().add(off_card);

} // end of constructor

public static void main (String[] args) {

    JFrame frame = new Terminal();
    frame.setBounds(0, 0, 310, 380);
    frame.setVisible(true);

} // end of method main

} // end of class OffInstaller
```

VITA #2

SYENG HO JANG

Candidate for the Degree of  
Master of Science

Thesis: SECURE OBJECT SHARING ON JAVA CARD

Major Field: Computer Science

Biographical:

Personal: The husband of Seon Kyung Kim, and the father of Lauren Sunny Jang and Jamie Austin Jang.

Education: Received Bachelor of Science degree in Computer Science from Oklahoma State University, Stillwater, Oklahoma in July 2000.  
Completed the requirements for the Master of Science degree with a major in Computer Science at Oklahoma State University in May, 2003.

Experience: Employed by Oklahoma State University, Center for Laser And Photonics Research as a system administrator; Employed by Oklahoma State University, Department of Computer Science as a teaching assistant.