2012

Smart Card Based User Authentication

Brandon James
Wright State University

Follow this and additional works at: http://corescholar.libraries.wright.edu/etd_all

Part of the Computer Sciences Commons

Repository Citation
Smart Card Based User Authentication

A thesis submitted in partial fulfilment
of the requirements for the degree of
Master of Science

By

BRANDON JAMES
B.S., Wright State University, June 2010

2012
Wright State University
I HEREBY RECOMMEND THAT THE THESIS PREPARED UNDER MY SUPERVISION
BY Brandon James ENTITLED Smart Card Based User Authentication BE ACCEPTED
IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR THE DEGREE OF
Master of Science.

Meilin Liu, Ph. D.
Thesis Advisor

Mateen Rizki, Ph.D., Chair
Department of Computer Science
College of Engineering and Computer Science

Committee on
Final Examination

Meilin Liu, Ph. D.

Travis Doom, Ph. D.

Jack Jean, Ph. D.

Andrew Hsu, Ph.D.
Dean, Graduate School
User authentication plays a very important role in building a secure computing system. In this thesis, we first studied and investigated existing user authentication schemes and some basic concepts of smart cards. We then proposed and implemented an improved user authentication scheme based on the smart card, specifically, Java card. We simulated a web server, and implemented the user authentication scheme with the Java card by programming a Java card applet and a Java program to send commands and receive responses from the card.

The proposed user authentication scheme has two phases: the registration phase and the user authentication phase. In the registration phase, the server triggers the Java card to generate a secret value and store it. Then the Java card uses this secret value and hashes the username and password combination, which is stored in the web server’s authentication database. The user’s login request to the web server will start the user authentication phase, where the smart card will compute the hash, using the username and password entered, and return it to the web server for user verification. The implementation of the proposed user authentication scheme proved the correctness and effectiveness of the scheme. Compared with the previous user authentication schemes, our proposed authentication scheme is more secure because it implements a two-factor authentication. Even if the user’s password is compromised, a user would still need the smart card to log into the system.
TABLE OF CONTENTS

CHAPTER 1. INTRODUCTION ......................................................... 1
  1.1 Background ............................................................... 1
  1.2 Related Work ............................................................ 2
  1.3 Problem Statement ....................................................... 3
  1.4 Organization .............................................................. 4

CHAPTER 2. BASIC CONCEPTS ..................................................... 6
  2.1 Overview of Security ...................................................... 6
  2.2 Smart Cards ............................................................... 8
  2.3 JavaCard ................................................................. 10
    2.3.1 History ............................................................. 11
  2.4 ISO-7816 Standards ...................................................... 11
    2.4.1 Contact Points ..................................................... 12
    2.4.2 Transmission Protocols .......................................... 13
    2.4.3 Industry Standard Commands for Communicating with Smart Cards ..... 14

CHAPTER 3. SMART CARD APPLET DEVELOPMENT .......................... 15
  3.1 Add a New Virtual Smart Card ......................................... 15
  3.2 Add a New Java Card Applet ........................................... 16
  3.3 Application Protocol Data Unit (APDU) ............................... 21

CHAPTER 4. USER AUTHENTICATION SCHEME WITH JAVA CARD APPLET . 23
  4.1 Add a New Java Card Applet ........................................... 23
  4.2 Sending Commands to a JavaCard: Java Program for Sending Commands to JavaCard 30
  4.3 User Authentication Scheme and Implementation ..................... 34
    4.3.1 Registration Phase ............................................... 36
    4.3.2 User Authentication Phase ...................................... 37
LIST OF FIGURES

2.1 Smart Card ................................................................. 9
2.2 Smart Card Pin Layout.................................................. 12

3.1 Add a new smart card. Step 1 .......................................... 15
3.2 Add a new smart card. Step 2 .......................................... 16
3.3 Creating a new Java Card Applet– Step 1 .......................... 18
3.4 Creating a new Java Card Applet– Step 2 .......................... 18
3.5 Creating a new Java Card Applet– Step 3 .......................... 19
3.6 Creating a new Java Card Applet– Step 4 .......................... 19

4.1 Registration Phase ....................................................... 37
4.2 User Authentication Phase ............................................. 39
LIST OF TABLES

3.1  Application Identifier (AID) Format ............................................ 17
3.2  Project AID Values................................................................. 17
3.3  Command APDU Structure ....................................................... 22
3.4  Response APDU Structure ......................................................... 22
CHAPTER 1

INTRODUCTION

1.1 Background

Information security has become important in recent years mainly because of the increase in availability of information. As computer systems become more complex and the internet allows for interconnectivity of systems, protecting access to information becomes crucial in order to preserve an individual’s privacy and keep confidential information from being compromised. Computer systems now store credit card and banking information, social security numbers, health records, and various other data that may be very personal to an individual. Due to the private nature of information today, measures to improve confidentiality, data integrity, availability, and access control must be implemented in order to ensure that malicious users are prevented from accessing or altering data, yet information is still available to those who have rightfully been granted access to it.

One main way of protecting data access is through authentication, “the binding of an identity to a subject,” as defined by Matt Bishop in Computer Security Art and Science [2]. One of the most common forms of authentication is having a username and password. A user must know its username and password in order to log into a system. However, if a password is compromised, a malicious user can gain access to a system in which they do not have access rights. The second method of authenticating users is to use something the users have, such as a badge or card. The third method of authenticating is to use the user’s biometrics, or the biometrics together with a password. The objective of this Master’s thesis is to study and
investigate the existing user authenticating schemes and some basic concepts of smart cards.

Using a smart card for authentication requires a user to have a physical object in their possession and also know a password (in a smart card often referred to as a PIN) in order to authenticate themselves. Requiring both to verify a person’s identity makes it more difficult for a user’s identity to be compromised because one’s adversary could still compromise a password, but he/she needs to have the smart card in their possession in order to gain access to the system. Compared with the previous user authentication schemes, our proposed authentication scheme using a smart card, is more secure because it implements a two-factor authentication. Even if the user’s password is compromised, a user would still need the smart card to log into the system.

1.2 Related Work

Many researchers have investigated how to utilize smart cards in a security solution. As far back as 1997, Guthery stated that Java Cards would become useful for secure internet computing [6]. His basis was primarily that by putting the Java language on a smart card, it would lend itself well for internet computing, since the Java language is used for internet programming. Also, by putting Java on a smart card, more developers would have the knowledge and ability to develop on the Java Card, since it is a level above the microcontroller level programming.

Hiltgen, Kramp, and Weigold proposed two different authentication solutions for use in internet banking [8]. In [18], an authentication scheme for internet banking using smart cards was proposed too.

Bichsel, Camenisch, Gro and Shoup [1], designed a scheme to use a smart card, a Java Card in particular, to build an electronic identity token to access buildings, in which the smart card was used to generate keys and random numbers that can be used to authenticate a user. Their research was primarily built around generating keys quickly and efficiently to prove that smart cards could in reality be used as secure identity tokens.
After studying and investigating the smart card, and the existing user authentication schemes, we proposed and implemented an improved user authentication scheme using a smart card. Our design is different, however, because many of the works cited here proposed solutions and proved that a certain authentication scheme would work and that using smart cards to perform authentication was feasible. In our case, by coding a Java applet, we proved that user authentication schemes using smart cards are indeed realizable and feasible.

1.3 Problem Statement

As systems become more complex and hold more personal data for individuals, user authentication becomes important so that the data is protected. In this thesis we aim to develop an improved user authentication scheme based on smart card. A smart card is capable of storing confidential information (such as security keys) that can be used to authenticate a user. Smart cards lend themselves well to being used for user authentication. Furthermore, we decided on using the Java Card to develop our solution. MULTOS and Java Card are the two platforms that are considered leaders in smart card solutions, mostly in the banking industry. MULTOS is used by Mastercard, a major financial services corporation, and Java, which was developed by Sun Microsystems, is the standard for Visa's smart credit cards [9]. Since the Java language and its platform have been used extensively in real-world applications, this became the chosen platform for the development of this research project.

As noted in [6], using Java on a smart card opens up more possibilities for developers, since Java is used quite extensively in various applications. For this reason, we opted to use the Java Card libraries to simulate our solution to the user authentication problem. We studied the syntax and requirements for developing a Java Card applet first. Then once we had some basic understanding of java card, we studied and investigated user authentication schemes and some different ways these schemes were implemented in the security applications. Based on
these knowledge, we then proposed and implemented an improved user authentication scheme based on the smart card, specifically, Java card. We simulated a web server, and implemented the user authentication scheme with Java card by programming a Java card applet and a Java program to send commands and receive responses from the card. The proposed user authentication scheme has two phases: the registration phase and the user authentication phase. In the registration phase, the server triggers the Java card to generate a secret value and store it. Then the Java card uses this secret value and hashes the username and password combination, which is stored in the web server’s authentication database. The user’s login request to the web server will trigger the user authentication phase. In the user authentication phase, once the username and password is entered, the username and the password will automatically be sent to the smart card.

The smart card will use the username, the password, and the stored secret value to compute a hash value. Then the username and the computed hash value will be sent to the web server. The web server will compare the received username and the hash value with the username and hash value stored in the database. If both match, then the user is authenticated, the user will have certain access control privileges on the web server. The implementation of the proposed user authentication scheme proved the correctness and effectiveness of the scheme.

Compared with the previous user authentication schemes, our proposed authentication scheme is more secure because it implements a two-factor authentication. Even if the user’s password is compromised, a user would still need the smart card to log in to the system.

1.4 Organization

Chapter 2 presents an overview of information security and defines the basic concepts related to security. This section also includes an overview of the Java Card and smart cards, in general, along with the ISO-7816 standards that govern the definitions and functionality of a smart
card. In Chapter 3, we describe the basic set-up for our development environment and how to create a Java Card applet. Chapter 4 describes the specific applet that we have implemented and the user authentication scheme that we have implemented. Chapter 5, concludes the thesis and describes the future work.
CHAPTER 2

BASIC CONCEPTS

2.1 Overview of Security

As the internet has progressed and computing systems have become more and more complex, security and privacy have become essential. Information security must be able to provide confidentiality to the user, ensure data integrity, as well as make sure that data is available whenever an authorized user needs to access information.

Security, in the context of computing, can be divided into five areas, or domains: Confidentiality, Integrity, Authentication, Non-Repudiation, and Reliability [2].

Confidentiality

Confidentiality can be defined as ensuring that any data is only shared with those users that are authorized to access it. According to the ISO (International Organization for Standardization), confidentiality is considered one of the cornerstones and a fundamental basis that any information security policy should contain. Without confidentiality, data can be compromised by those who should not have access to it, and thus could reveal information about a user that could be used in a malicious way. For most schemes, confidentiality is achieved by encrypting data during transmission so that an adversary cannot read the data during transmission easily even they can intercept the data during their transmission.
**Integrity**

Integrity refers to being able to trust the data and ensuring that it is accurate and consistent. To ensure data integrity, checks must be done to verify that the data has not been altered by any unauthorized individuals. An authorized user performing a transaction must be able to trust the data that is being used so that all transactions are completed correctly and successfully.

**Authentication**

Authentication is confirmation that an individual is who they say they are. Essentially there are two sides to authentication:

1. A user must be able to verify that the host they are communicating is guaranteed to be the one they are intending to communicate with.

2. The host, or server, on the other hand also needs to be able to verify that the user is in fact who they say they are.

The process of authentication proving who they are is often done using a shared secret that is communicated and verified.

**Non-Repudiation**

Non-Repudiation is the impossibility that an entity who performed a transaction could deny the transaction at a later time. To establish non-repudiation, a digital signature is often used. Digital signatures are done using a one-way hash function, such as SHA-1. By signing the data, the entity verifies that they were indeed the ones that performed the transaction, which prevents them from denying or challenging the action.
Reliability

Reliability, or availability, describes the overall functionality of the security system. A transaction must be performed at any time under certain conditions and also ensure that the security and privacy is preserved. Privacy and security could be fully preserved if information was not available, however, this also would not be very useful. There needs to be a careful balance to ensure that data is available to any authorized users whenever they may need it.

2.2 Smart Cards

The first patent for a smart card was filed in 1968 [7] which was for a card that included an integrated circuit chip. Since then, much progress has been made in the design of smart cards, and they are being used in various areas for providing security. The most familiar form of a smart card is the credit card-sized cards which can be used for electronic cash payments, but in more recent years, the SIM (Subscriber Identity Module) cards used in mobile phones are also used for smart cards.

Smart cards are portable microcomputers that have the ability to perform operations and hold authentication data for a user. Inside of the card is a series of integrated circuit chips (ICCs) that form a microprocessor that is capable of storing data to identify the user and perform cryptographic operations. Because of the circuit chips they contain, smart cards are also known as integrated circuit cards. These cards are used in a variety of industries including: mobile phones, banking, and healthcare [16] for user identification. In addition, smart cards are increasingly being used to store loyalty cards, electronic ticketing, toll road passes, and electronic cash, such as VisaCash [7]. Figure 2.1 shows an example of a common Java smart card [4].

Having the capability to store user data and perform operations such as encryption, smart cards can be used to verify an user, thereby preventing an unauthorized user from altering
While the data and computation abilities of smart cards make them qualified to be used for authentication verification, their portability and tamper-resistant design also make them a viable option for integrity checks. They also come in two forms, contact and contactless. Depending on the type of smart card, the platform varies. Java, .Net, Multos, Windows for Smart Card, and Linux platforms are widely used for smart card development. Smart Card standards must conform to the ISO 7816 requirements [14].

**Memory Limitations**

Smart cards contain three different types of memory: Read Only Memory (ROM), Electrical Erasable Programmable Read Only Memory (EEPROM), and Random Access Memory (RAM).

- **ROM** - contains the operating system for the smart card
- **EEPROM** - non-volatile memory which holds values even when the power is removed; holds values such as the user’s PIN and passwords
- **RAM** - holds temporary values (variables), divided among the Java stack, Java heap, APDU buffer, and atomic transaction buffer [1]

The memory on a smart card is quite small. In some cases the memory capacities may
be as small as: 24 K of ROM, 16 K of EEPROM, and 512 bytes of RAM [11].

2.3 JavaCard

There are multiple operating systems that can be used on a smart card, including: MultOS, Smart Card for Windows, and Java Card. For our research we opted to use the Java Card to perform security functions. The Java Card allowed us to develop the applet for our smart card using the Java language, which was used extensively. Technically, Java Card is a platform, or more specifically a framework, that is used to develop smart card applets, rather than an operating system. The Java card framework allows us to develop code that is compiled down to byte-code and then interpreted, and it has been specifically designed for mobility as well as security.

**Primitive Types Supported / Unsupported**

Due to the memory limitations of a smart card, only a subset of the primitive types available in the Java language are available in the Java Card framework.

Supported Primitive Types:

- boolean - 8 bits
- byte - 8 bits
- short - 16 bits

Unsupported Primitive Types:

- char - 16 bits
- float - 32 bits
• int - 32 bits
• double - 64 bits
• long - 64 bits

2.3.1 History

Engineers working for Schlumberger in Austin, TX, one manufacturer of smart cards, came up with the idea of putting Java on a smart card in March 1996 [6]. Internet development was growing rapidly, and the Java language was being used quite extensively as an internet language. For this reason, engineers decided that putting Java on a smart card would allow the cards to be used to authenticate users into internet systems. Because both components were using the same language, integrating the two would be much easier. Even today the Java Card is still being used and allows developers to code in a more familiar language rather than coding at the byte or assembly language level.

2.4 ISO-7816 Standards

The standards that govern the design and functionality of smart cards have been established by the ISO. The ISO-7816 standards have been established by the organization to provide a basis and guideline for how a smart card, or integrated circuit card, with contacts should be designed. These standards describe requirements for the physical aspects of the smart cards as well as how the cards should communicate with terminals and off-card applications. When a framework, such as the JavaCard framework, is created, it must be able to operate within the bounds set forth by the ISO standards including how the commands on the smart card are processed and which transmission protocols can be used [14]. Part 1 of the ISO 7816 standards defines the physical design of the smart card, including the materials that the chips should be made out of and the flexibility/bendability of the card.
2.4.1 Contact Points

Part 2 of the ISO 7816 standards defines the card contact-point dimensions and the contact-point placement. The minimum size of a contact-point of a smart card must be 1.7mm in height and 2.0mm in width [15].

A smart card has 8 contacts, or pins. Only 6 of these pins, however, are currently used. The other two are reserved for future expansion. Figure 2.2 shows the pin layout for a smart card according to the ISO 7816 standards [13].

![Smart Card Pin Layout](image)

Figure 2.2. Smart Card Pin Layout

- Vcc (1) - The Vcc pin is where the supply voltage is connected to the microchip [15]. This supply voltage is generally 5 volts [13].

- GND (5) - The GND pin is where the common ground is connected to the smart card.

- RST (2) - The RST (Reset) pin is used either by the card itself or the reader/interface to send a reset command to the smart card. Based on the “answer-to-reset” (ATR), the interface can determine which protocol (T=0 or T=1) the card is using to communicate commands and responses.
• Vpp (6) - The Vpp pin is where the programming voltage is connected to the smart card. This pin was used in older smart cards to program the high voltage signal to initialize the EEPROM but is now an optional pin.

• CLK (3) - The CLK (clock) pin is used to give the card a clock or timing signal input.

• I/O (7) - The I/O pin is used for basic input/output to and from the card reader.

• 2 RFU pins (4 and 8) - The RFU pins are not currently used and are reserved for future use.

2.4.2 Transmission Protocols

Part 3 of the ISO 7816 standards defines the transmission protocols that can be used by smart cards to communicate commands and responses to and from the card. There are currently two protocols, T=0 and T=1, defined in the ISO standards. Other protocols: T=2, T=3, T=4, T=5, T=13, T=14, and T=15, are reserved for future use and expansion.

• T=0: The T=0 protocol is an asynchronous, half-duplex protocol where each character is transmitted separately [15]. This protocol issues a start signal followed by a character and then a stop signal. Being that it is a half-duplex protocol, it offers both sending and receiving communication, but not simultaneously.

• T=1: The T=1 protocol as defined in the ISO 7816 standards is an asynchronous, half-duplex protocol where blocks of data can be communicated instead of a single character as in the T=0 protocol.
2.4.3 Industry Standard Commands for Communicating with Smart Cards

Part 4 of the ISO 7816 standards defines the commands and response structure for communicating with the smart cards. These application protocol data units (APDU’s) are illustrated in section 3.3 in detail.
3.1 Add a New Virtual Smart Card

To begin programming a Java card applet, the first thing that was done was adding Java Card libraries to NetBeans IDE. For the purpose of our research, NetBeans version 7.0 is being used. Then a virtual smart card was created that would be used for testing purposes. This was added through the “Services” menu.

![Figure 3.1. Add a new smart card. Step 1](image)

As shown in the Figure 3.2, the new virtual smart card can be assigned a RAM size, EEPROM size, the HTTP port, and other attributes. The HTTP port is assigned to be used as the “server” for processing the smart card commands.
3.2 Add a New Java Card Applet

Once the smart card is added, a new applet can be created to be used for processing the commands given to the smart card. The type of project that was created was a “Classic Applet Project”.

The applet is added with an applet ID and a package ID, that must be assigned according to the requirements of ISO 7816. The ID’s are application identifiers (AID). When a Java Card reads a command, it chooses the applet based on the bytes received. This command is then executed with the data that was also passed to the smart card. Each AID must be between 5 and 16 bytes long. The format must conform to format shown in the table 3.1 and table 3.2.
Table 3.1. Application Identifier (AID) Format

<table>
<thead>
<tr>
<th>Registered Application Provider (RID)</th>
<th>Proprietary Application Identifier Extension (PIX)</th>
</tr>
</thead>
<tbody>
<tr>
<td>5 bytes</td>
<td>0-11 bytes</td>
</tr>
</tbody>
</table>

Table 3.2. Project AID Values

<table>
<thead>
<tr>
<th></th>
<th>RID*</th>
<th>PIX</th>
</tr>
</thead>
<tbody>
<tr>
<td>Applet AID</td>
<td>0x1000AABB00</td>
<td>0x9B</td>
</tr>
<tr>
<td>Package AID</td>
<td>0x1000AABB00</td>
<td>0x1001</td>
</tr>
</tbody>
</table>

*Note that the RID for both the applet and the package are the same value.

The procedure of creating a new Java card applet is illustrated as follows:

- Step1 is to create a new project as shown in Figure 3.3.
- Step2 is to choose the project to be a classic applet project as shown in Figure 3.4.
- Step3 is to choose the project to specify the application identification for this project as shown in Figure 3.5.
- Step4 is to specify the package application identification for this project as shown in Figure 3.6.
Figure 3.3. Creating a new Java Card Applet – Step 1

Figure 3.4. Creating a new Java Card Applet – Step 2
Figure 3.5. Creating a new Java Card Applet– Step 3

Figure 3.6. Creating a new Java Card Applet– Step 4
After the preceding steps, an applet template is then generated by the Java Card Runtime Environment (JCRE):

```java
package smartcardapplet;

import javacard.framework.*;
/**
 * @author BRANDON JAMES
 */
public class SmartCardApplet extends Applet {

/**
 * Installs this applet.
 * @param bArray the array containing installation parameters
 * @param bOffset the starting offset in bArray
 * @param bLength the length in bytes of the parameter data in bArray
 */
public static void install(byte[] bArray, short bOffset, byte bLength) {
    new SmartCardApplet();
}

/**
 * Only this class’s install method should create the applet object.
 */
protected SmartCardApplet() {
    register();
}

/**
 * Processes an incoming APDU.
 * @see APDU
 * @param apdu the incoming APDU
 */
public void process(APDU apdu) {
    //Insert your code here
}
}
```

Each of these methods is required for the Java Card to work properly. The required methods for the Java Card applet are install(), the applet constructor, and process(), which
processes the commands sent to the smart card.

- **install()** - creates a new instance of the applet by calling the applet constructor; ultimately, this method also registers the newly created instance with the JCRE, which is done through the applets constructor.

- **SmartCardApplet()** - the constructor for the applet calls the register() method which is a part of the Applet class. As messages are passed to the smart card the JCRE has a list of registered applets that can be searched so that the messages are sent and received correctly.

- **process()** - accepts a parameter of type APDU (application protocol data unit). This method accepts a buffer from the card reader and processes the command. Each function that this applet performs will be included in this process method. The applet uses the `process()` method to “interpret and perform the task specified by the command” APDU [3].

### 3.3 Application Protocol Data Unit (APDU)

The `process()` function takes an argument in the form of an application protocol data unit (APDU). This is the communication format that is used by the Java Card to communicate between the card and the terminal [12]. This format is specified in section 4 of the ISO-7816 standards. These messages are passed through the card reader that is connected to the terminal.

The APDU messages come in pairs. There is a command APDU that issues a request or *command* to the Java Card, and then the card produces a *response* to the command. Each of these messages has a specific format or structure that needs to be followed so that the messages can be properly processed by and sent from the smart card. Both of the structures are shown in the tables below.

The *command* APDU is formatted into 7 parts as shown in Table 3.3.

- **CLA** - the *Class Of Instruction*, which indicates the type of command.
Table 3.3. Command APDU Structure

<table>
<thead>
<tr>
<th>Required</th>
<th>Optional</th>
</tr>
</thead>
<tbody>
<tr>
<td>CLA</td>
<td>INS</td>
</tr>
<tr>
<td>1 byte</td>
<td>1 byte</td>
</tr>
<tr>
<td>Param1</td>
<td>Param2</td>
</tr>
<tr>
<td>1 byte</td>
<td>1 byte</td>
</tr>
<tr>
<td>L_{data}</td>
<td>Data</td>
</tr>
<tr>
<td>1 byte</td>
<td>variable</td>
</tr>
<tr>
<td>L_{expected}</td>
<td></td>
</tr>
<tr>
<td>1 byte</td>
<td></td>
</tr>
</tbody>
</table>

- **INS** - the Instruction Number, which indicates the specific command (instruction) being called.
- **Param1** - the first parameter of the instruction.
- **Param2** - the second parameter of the instruction.
- **L_{data}** - the length (in bytes) the data field contains.
- **Data** - the command data.
- **L_{expected}** - the length (in bytes) expected in the response APDU.

Table 3.4. Response APDU Structure

<table>
<thead>
<tr>
<th>Optional</th>
<th>Required</th>
</tr>
</thead>
<tbody>
<tr>
<td>Data</td>
<td>StatusWord1</td>
</tr>
<tr>
<td>variable</td>
<td>1 byte</td>
</tr>
<tr>
<td></td>
<td>StatusWord2</td>
</tr>
<tr>
<td></td>
<td>1 byte</td>
</tr>
</tbody>
</table>

The response APDU is formatted into 3 parts as shown in Table 3.4.

- **Data** - the data returned in response to the command APDU should not exceed the length specified in the command L_{expected} field.
- **StatusWord1**.
- **StatusWord2**.

*StatusWord1 and StatusWord2 together indicate the status of the APDU command, e.g. Successful Processing, Execution Error, Execution Warning, etc.
4.1 Add a New Java Card Applet

The basic JavaCard template for our applet was generated in Section 3.2. To this we first need to added some final static byte variables that will be used in the applet. These bytes will determine which methods get called. Recall from the discussion of the APDU format from 3.3 that the first byte of the APDU is the class of instruction and the second byte of each command APDU is the instruction byte. These bytes need to be defined so that when the APDU is sent to the JavaCard, it is able to be processed correctly. As globals inside the class for our applet we define the bytes that we will need for processing incoming APDU’s.

```java
final static byte CLA = (byte) 0xA0; //defines the CLA byte used for instructions
final static byte generateRandomData = (byte) 0x20;
final static byte generateNewStoredSecret = (byte) 0x40;
final static byte generateHash = (byte) 0x60;
```

Following the definitions for the instruction bytes in our applet, we can define any other global values that we may need for our applet. A couple examples here would be any status words that we want to return which inform the user the outcome of a specific method. These status words are crucial, especially in the testing process, for the functionality of the smart card because they allow us to inform the user of any exceptions that were thrown or what the result of the function was. We can define a specific value for our applet that gets returned if a NULL pointer exception was thrown or if an Array Out-Of-Bounds exception was thrown. This helps
to improve the communication to the smart card because it makes the response APDUs more meaningful.

```java
final static short SW_CryptoException = 0x6900;
final static short SW_ARRAY_OUT_OF_BOUNDS = 0x6902;
final static short SW_NULLPOINTER = 0x6904;
```

The standard template was created for the Java Card applet back in section 3.2. Here the process() method can be defined to call the various functions defined in our Java Card applet. Starting out we define some global values that we can use for our applet. Then we first have a constructor that registers the applet.

The process() method takes any incoming APDU and decides what function needs to be performed. Each method is chosen based on the given instruction (INS) byte in the APDU that corresponds to the final static byte defined. Through the process() function the java card knows how to take the incoming command APDU and send it to a method in order to get the response that the user expects.

```java
private byte[] storedSecret;

public static void install(byte[] bArray, short bOffset, byte bLength) {
    new SmartCardApplet(bArray, bOffset, bLength);
}

protected SmartCardApplet(byte[] bArray, short bOffset, byte bLength) {
    register();
}

public boolean select()
{
    return true;
}

public void process(APDU apdu) {
```
byte[] buffer = apdu.getBuffer();

/*
 * if statement executes the SELECT to select the applet functionality for the Java Card
 */
if ((buffer[ISO7816.OFFSET_CLA] == (byte) 0x0) && (buffer[ISO7816.OFFSET_INS] == (byte) (0xA4)))
    {return;}

switch (buffer[ISO7816.OFFSET_INS])
{
    case generateRandomData: generateRandomData(apdu);
        return;
    case generateNewStoredSecret: generateNewStoredSecret(apdu);
        return;
    case generateHash: generateHash(apdu);
        return;
    default: ISOException.throwIt(ISO7816.SW_INS_NOT_SUPPORTED);
}

Once the process() method has been defined, we need to define methods for each of the possible instructions. For this research project, we will be using the smart card to generate a new secret value that will be stored on the Java card. Using this secret, a hash function can be used to generate a value that will be unique to the user who owns the card, which contains the secret value. There will be two methods used here. The first will be to generate a new secret. This method will be called by the card issuing authority in the registration phase of the user authentication scheme. Then another method will be needed to compute the hash of the stored secret with the data input from the APDU. A third method has been added for generating random data. This will be used for testing and will be used in future research to generate random values based on an input seed. The Java card Applet code is shown below.

public void generateRandomData(APDU apdu)
{
    byte[] buffer = apdu.getBuffer();
    short dataLength = (short) buffer[ISO7816.OFFSET_LC];

    byte[] outputBuffer = new byte[16];
    RandomData data = RandomData.getInstance(RandomData.ALG_PSEUDORANDOM);
data.setSeed(buffer, (short) ISO7816.OFFSET_CDATA, (short) dataLength);
data.generateData(outputBuffer, (short) 0, (short) outputBuffer.length);
apdu.setOutgoing();
apdu.setOutgoingLength((short) outputBuffer.length);
apdu.sendBytesLong(outputBuffer, (short) 0, (short) outputBuffer.length);
}

//generates a new stored secret of length 16 bytes
//the seed is given through the data portion of the command APDU
public void generateNewStoredSecret(APDU apdu) {
    byte[] buffer = apdu.getBuffer();
    short dataLength = (short) buffer[ISO7816.OFFSET_LC];

    byte[] outputBuffer = new byte[16];
    storedSecret = new byte[16];
    RandomData data = RandomData.getInstance(RandomData.ALG_PSEUDO_RANDOM);
data.setSeed(buffer, (short) ISO7816.OFFSET_CDATA, (short) dataLength);
data.generateData(storedSecret, (short) 0, (short) storedSecret.length);
data.generateData(outputBuffer, (short) 0, (short) outputBuffer.length);
apdu.setOutgoing();
apdu.setOutgoingLength((short) outputBuffer.length);
apdu.sendBytesLong(outputBuffer, (short) 0, (short) outputBuffer.length);
}

//generates and returns the hash
//the data input in the command APDU will be the username and password
//an XOR operation will be performed on the username and password
//then the result will be XOR-ed with the stored secret on the card
//then a SHA1 hash will be performed on the result
//this result will be returned

//input APDU
//P1 - contains the length of the username
//P2 - contains the length of the password
//data - the username followed by the password
public void generateHash(APDU apdu) {
    byte[] Hash = new byte[20];

    byte[] buffer = apdu.getBuffer();
    byte usernameLength = buffer[ISO7816.OFFSET_P1];
    byte passwordLength = buffer[ISO7816.OFFSET_P2];
    short dataLength = (short) 0;

    //data is the data from the buffer passed to the smart card
    //it contains the username and password
    byte[] username = null;
    byte[] password = null;
    byte[] data = null;

    //if the username is longer then pad the password with 0's
    //to make the same length for XOR operation
    if(usernameLength > passwordLength)
{ 
dataLength = (short) usernameLength;
username = new byte[usernameLength];
password = new byte[usernameLength];

for(short j = 0; j < usernameLength; j++)
{
    username[(short) j] = buffer[(short)(ISO7816.OFFSET_CDATA + j)];

    if(j < passwordLength)
    {
        password[(short) j] = buffer[(short)(ISO7816.OFFSET_CDATA + j + (short) usernameLength)];
    } else
    {
        password[(short) j] = (byte) 0x00;//pad end with 0's if shorter
    }
}

else if(passwordLength > usernameLength)
{
    dataLength = (short) passwordLength;
username = new byte[passwordLength];
password = new byte[passwordLength];

for(short k = 0; k < passwordLength; k++)
{
    password[(short) k] = buffer[(short)(ISO7816.OFFSET_CDATA + k + (short) usernameLength)];

    if(k < usernameLength)
    {
        username[(short) k] = buffer[(short)(ISO7816.OFFSET_CDATA + k)];
    } else
    {
        username[(short) k] = (byte) 0x00;//pad end with 0's if shorter
    }
}

else
{
    dataLength = (short) usernameLength;
username = new byte[usernameLength];
password = new byte[usernameLength];

for(short j = 0; j < usernameLength; j++)
{
    username[(short) j] = buffer[(short)(ISO7816.OFFSET_CDATA + j)];
}

for(short j = 0; j < passwordLength; j++)
{
    password[(short) j] = buffer[(short)(ISO7816.OFFSET_CDATA + j + (short) username.length)];
}
data = new byte[dataLength];

//XOR the username and password
for(short k = 0; k < dataLength; k++)
{
    data[(short) k] = (byte) (username[(short) k] ^ password[(short) k]);
}

//data now contains the XOR output of the username and password
//now need to XOR this value with the stored secret
//make combination to hash equal to the XOR output of the data in and the stored secret
byte[] combinationToHash;
byte[] tempData;
byte[] tempStoredSecret;

if(data.length > storedSecret.length)
{
    combinationToHash = new byte[data.length];
tempData = new byte[data.length];
tempData = data;
tempStoredSecret = new byte[data.length];

    for(short j = 0; j < data.length; j++)
    {
        if(j < storedSecret.length)
        {
            tempStoredSecret[(short) j] = storedSecret[(short) j];
        }
        else
        {
            tempStoredSecret[(short) j] = (byte) 0x00;//pad end with 0’s if shorter
        }
    }
    } else if(storedSecret.length > data.length)
    {
    combinationToHash = new byte[storedSecret.length];
tempStoredSecret = new byte[storedSecret.length];
tempStoredSecret = storedSecret;
tempData = new byte[storedSecret.length];

    for(short k = 0; k < storedSecret.length; k++)
    {
        if(k < data.length)
        {
            tempData[(short) k] = data[(short) k];
        }
        else
        {
            tempData[(short) k] = (byte) 0x00;//pad end with 0’s if shorter
        }
    }
}
else
{
  combinationToHash = new byte[storedSecret.length];
tempData = new byte[data.length];
tempData = data;
tempStoredSecret = new byte[storedSecret.length];
tempStoredSecret = storedSecret;
}

//perform XOR operations
for(short x = 0; x < data.length; x++)
{
  combinationToHash[(short) x] = (byte) (tempData[(short) x] ^ tempStoredSecret[(short) x]);
}

MessageDigest SHA1Engine = MessageDigest.getInstance(MessageDigest.ALG_SHA, false);
SHA1Engine.reset();

try
{
  SHA1Engine.doFinal(combinationToHash, (short) 0, (short) combinationToHash.length, Hash, (short) 0);
}
catch(CryptoException e)
{
  ISOException.throwIt((short) e.getReason());
}
catch (ArrayIndexOutOfBoundsException ex)
{
  ISOException.throwIt(SW_ARRAY_OUT_OF_BOUNDS);
}
catch (NullPointerException ex)
{
  ISOException.throwIt(SW_NULLPOINTER);
}

apdu.setOutgoing();
apdu.setOutgoingLength((short) Hash.length);
apdu.sendBytesLong(Hash, (short) 0, (short) Hash.length);
}

The code for generating the hash first parses the input APDU data. P1 of the input APDU is equal to the length of the username, and P2 gives the length of the password. Using these parameters, the data portion of the APDU is parsed to extract the username and password. Then an XOR operation is performed on the username and password. The XOR operation can only be performed on byte arrays of equal length. In order to perform a successful XOR operation, whichever is shorter, the username or password, must be padded with zeroes to make both the same length.
Once the XOR operation on the username and password is complete, the result is then XOR-ed with the stored secret on the card. Finally, a SHA1 hash operation is done on the final result of the XOR operations. This hash value is then returned in the response APDU.

4.2 Sending Commands to a JavaCard: Java Program for Sending Commands to JavaCard

A Java program was created to send commands to the Java card and get a response. This program will be used for sending a command to the Java card and receiving a response that includes a hashed value that will be used for authentication. For this research project, the Java card is a service acting as a virtual card, which is running on the local computer port 9025. The main purpose of the SendCommandsToJavaCard program is to allow a system to execute this Java program which will handle all of the commands and responses to and from the Java card.

To first get a connection to the virtual smart card, a socket is open to allow for communication to and from the smart card. The CadClientInterface method is called to create a new interface using communication protocol T1 and containing input and output streams.

```java
CadClientInterface clientInterface;
Socket socket = null;
InputStream input = null;
OutputStream output = null;

socket = new Socket("localhost", 9025);
input = socket.getInputStream();
output = socket.getOutputStream();
clientInterface = CadDevice.getCadClientInstance(CadDevice.PROTOCOL_T1, input, output);
```

Smart cards can use a couple different protocols for communication between the card and the reader. T0 and T1 protocols are defined in the ISO 7816-3 standards.

- T0 is a byte oriented protocol. This communication is asynchronous and for each character (byte) that is sent, an acknowledgement must be received. Using the T=0 protocol, each
character is transmitted individually [7].

- T1 is a block oriented protocol, where data is transmitted through blocks of data. A number of bytes can be sent before an acknowledgement must be received.

A smart card contains an “Answer To Reset” (ATR) which is a message sent to the card reader when the reader sends a command to electrically reset the smart card. This ATR contains various information about the card and its current state. With this message, a card reader is also able to determine which protocol the smart card is using so that it knows how to communicate data.

For our purposes we are communicating to the local port using the T1 protocol because blocks (APDUs) of data will be transmitted to the smart card. When the virtual smart card was added (see figure 3.2), the protocol was set to be T=1. The first step then is to connect to the port in which the virtual smart card is running. The getCadClientInstance method throws an IOException if it is unable to connect to the given port. By surrounding this in a try/catch statement we can catch this exception and print the error message if the clientInterface is unable to connect.

```java
public void connectToSimulator()
{
    try {
        socket = new Socket("localhost", 9025);
        input = socket.getInputStream();
        output = socket.getOutputStream();
        clientInterface = CadDevice.getCadClientInstance(CadDevice.PROTOCOL_T1, input, output);
    } catch (IOException ex) {
        System.out.println("IO Exception\n" + ex.getMessage() + "\n");
    }
}
```

Note that our Java program is successfully connected to our simulation port. We can begin to exchange APDUs with the smart card. To do this we begin by creating a new instance
of the APDU object. Then we can begin setting the fields of our APDU. The APDU object contains a command array which holds the command that will be sent to the smart card. We access the various indices by using the offset which are constants that are built into the APDU class. We use this method to set the CLA, INS, P1, and P2 portions of the command. The data input, however, is set by calling the class method `setDataIn` and passing in a byte array containing the data that should be sent.

```java
Apdu apdu = new Apdu();
apdu.command[Apdu.CLA] = (byte) 0x0;
apdu.command[Apdu.INS] = (byte) 0xA4;
apdu.command[Apdu.P1] = (byte) 0x04;
apdu.command[Apdu.P2] = (byte) 0x0;
byte[] appletAID = {(byte)0x10, (byte)0x0, (byte)0xAA, (byte)0xBB, (byte)0x00, (byte)0x9C};

apdu.setDataIn(appletAID);

try{
    instance.clientInterface.exchangeApdu(apdu);
} catch (CadTransportException ex) {
    System.out.println("Cad Transport Exception\n" + ex.getMessage().toString());
} catch (IOException ex) {
    System.out.println("IO Exception\n" + ex.getMessage() + "\n");
}
```

Being that smart cards (also known as multi-application smart cards) can have multiple applets on them, it is important for a host to first send a command to the smart card to let it know which applet it wants to use. ISO-7816 standards dictate that this initial select command have a CLA of 0x0, an INS of 0xA4, and P1 equals 0x04. Then the data portion of the command contains the applet AID of the applet that the host wishes to use.

Then to test that the send commands Java program, along with the Java Card simulator, a command was sent to the virtual card to call the `generateRandomData()` method on the card.
apdu.command[Apdu.CLA] = (byte) 0xA0;
apdu.command[Apdu.INS] = (byte) 0x20;
apdu.command[Apdu.P1] = (byte) 0x0;
apdu.command[Apdu.P2] = (byte) 0x0;
byte[] data = {(byte)0xAA, (byte)0xBB, (byte)0xCC, (byte)0xDD};
apdu.setDataIn(data);

try{
    instance.clientInterface.exchangeApdu(apdu);
}
catch (CadTransportException ex) {
    System.out.println("Cad Transport Exception\n" + ex.getMessage().toString());
}
catch (IOException ex) {
    System.out.println("IO Exception\n" + ex.getMessage() + "\n");
}

Here the INS byte is set to 0x20 which informs the applet which instruction to call. The method in the Java Card applet which generates random data uses the date portion of the APDU as a seed value for the random data engine. In both instances, the select APDU and the command to generate random data, the bytes received in response from the smart card were printed using apdu.getResponseApduBytes(). A sample run is shown below.

run:
0x0 0xA4 0x2 0x0 0x10 0x0 0xAA 0xBB 0x0 0x9C 0x0 0x90 0x0 0xA0 0x40 0x0 0x0 0x4 0xAA 0xBB 0xCC 0xDD 0x10 0xB7 0xD3 0x3B 0x96 0x99 0x75 0x53 0x5A...
0x7E 0xED 0x4C 0x34 0xA5 0xAC 0x13 0x3B 0x90 0x0
BUILD SUCCESSFUL (total time: 0 seconds)

The first command was the select command, which successfully gave the status bytes SW1 and SW2 as 0x90 0x0. The second command also was successful and the random data was returned. The first portion of each APDU response shows the command that was sent. Then the next byte gives the length of the response data. For the first command, no response data was returned, so this length is 0x0. The status words then follow this length byte. For the second command, however, there is some random data returned. Because of the coding of the applet method, sixteen bytes of random data should be returned. The length byte 0x10 in the response
shows that this is indeed the case. Then the sixteen bytes of random data are shown, as follows,

\[ 0xB7 \ 0xD3 \ 0x3B \ 0x96 \ 0x99 \ 0x75 \ 0x53 \ 0x5A \ 0x7E \ 0xED \ 0x4C \ 0x34 \ 0xA5 \ 0xAC \ 0x13 \ 0x3B, \]

followed by the status words, \( 0x90 \ 0x0 \), which indicate that this command was executed successfully.

4.3 User Authentication Scheme and Implementation

In this section, we will propose and implement an user authentication scheme based on the
smart card, specifically, Java card. We will simulate a web server, and implement the user au-
thentication scheme with Java card by programming a Java card applet and a Java program to
send commands and receive responses from the card. The proposed user authentication scheme
has two phases: the registration phase and the user authentication phase. In the registration
phase, the server (the card issuing authority) uses the Java card to generate a secret value and
store it. Then the Java card uses this secret value and hashes the username and password com-
bination, which is stored in the web server’s authentication database. The user’s login request
to the web server will trigger the user authentication phase. In the user authentication phase,
once the username and password is entered, the username and the password will automatically
be sent to the smart card.

The smart card will use the username, the password, and the stored secret value to
compute a hash value. Then the username and the computed hash value will be sent to the web
server. The web server will compare the received username and the hash value with the user-
name and hash value stored in the database. If both match, then the user is authenticated, the
user will have certain access control privileges on the web server. The detailed implementation
is described as follows.
Once the Java card applet and the send commands to Java card program were implemented, methods needed to be set-up to create a new stored secret and return the user’s hash. Creating a new stored secret would be done by the card issuing authority. Once this stored secret was created, it would persist throughout the use of the card. A method is also needed to perform and return the hash value for a user, so that a system can verify that a person is who they say they are.

The smart card issuing authority will need to create a new login. The SendCommandsToJavaCard program will be called with three parameters: username, password, and “C”. This will create a new login. The username and password will be passed to the smart card, and then the username, password, and stored secret will be hashed to create a value to store in the system database, not on the smart card itself. This will be the hashed value that is compared when the user attempts to log into a system. A string (in bytes) containing the username and password will be sent to the smart card. Also in the command APDU, parameters P1 and P2 will be set with the lengths of the username and password, respectively. Then this command APDU will be sent to the Java card. The Java card will then XOR the username and password. The result will then be XOR-ed with the stored secret, and then the SHA1 hash will be computed.

$$\text{hashedValue} = \text{SHA1}( \text{username} \oplus \text{password} \oplus \text{storedSecret} )$$
4.3.1 Registration Phase

The registration phase of the user authentication scheme is described as follows. The process is also illustrated in the diagram shown in Figure 4.1.

1. User requests to register with the server and give its username and password to the server

2. The server sends the command to the Java card to generate a new secret value and store it in Java card permanently

3. The server calls the SendCommandsToJavaCard to send the username and password to the smart card

4. Smart card performs XOR on the username and password
   \[(\text{username} \oplus \text{password})\]

5. Smart card performs XOR on the previous XOR output and the storedSecret
   \[((\text{username} \oplus \text{password}) \oplus \text{storedSecret})\]

6. Smart card takes the result of the XOR operation and performs a SHA1 hash
   \[\text{SHA1}((\text{username} \oplus \text{password}) \oplus \text{storedSecret})\]

7. SendCommandsToJavaCard returns the username and SHA1 hash value from the smart card to the server

8. The server stores the username and corresponding SHA1 hash value in the user authentication database

9. The server (the card issuing authority) issues a new smart card to user
4.3.2 User Authentication Phase

The user authentication phase of the user authentication scheme is described as follows. The process is also illustrated in the diagram shown in Figure 4.2.

1. User sends request to server to login
2. The server then triggers the smart card
3. The SendCommandsToJavaCard program, which runs on the client, is started and prompts the user to enter the username and password
4. SendCommandsToJavaCard sends the username and password input to the smart card
5. Smart card performs XOR on the username and password

\[(\text{username} \oplus \text{password})\]

6. Smart card performs XOR on the previous XOR output and the storedSecret

\[((\text{username} \oplus \text{password}) \oplus \text{storedSecret})\]

7. Smart card takes the result of the XOR operation and performs a SHA1 hash

\[\text{SHA1}((\text{username} \oplus \text{password}) \oplus \text{storedSecret})\]

8. Smart card returns the SHA1 hash value to the SendCommandsToJavaCard program

9. Server queries the database based on the username and pulls the stored hash value and compares to the data sent back from smart card

10. If the two hashed values match, then the user is authenticated; otherwise, the user is invalid and is not authenticated into the system

In this user authentication scheme, the user’s password is never transmitted across the network. The username and password are gathered on the client side using the SendCommandsToJavaCard program. Then these values are sent to the smart card. The only values that get passed across the network for verification are the username and the hashed output from the smart card. The system can then authenticate the user by comparing these two values to the username and the hash value stored in the user authentication database. The database values come from the registration phase, which are generated and stored in the user authentication database by the card issuing authority.

The first step in setting up the Java Card would be to set a new stored secret that would persist throughout the life of the smart card. This step would be done by the card issuing authority. For demonstration and testing purposes a method was created to reset this stored secret value, however, in practice these methods should be kept separate from a program that creates logins and return hash values because a malicious user could run this function and reset
the stored secret so that the hash value returned is different and users would not be able to use the smart card to authenticate themselves.

This method takes the args passed to the SendCommandsToJavaCard and passes the string to the Java card. The Java card uses this passed in data as a seed to generate a new random stored secret.

4.3.3 Method to create new stored secret on Java card

The following Java card Applet code segment is the method to create new stored secret value on Java Card.
//generate new stored secret
if(args.length == 2 && "N".equals(args[1]))
{
    byte[] SecretSeed = new byte[args[0].getBytes().length];
    SecretSeed = args[0].getBytes();

    //use Username to seed for the random stored secret
    instance.createNewStoredSecret(SecretSeed);

    try {
        instance.clientInterface.exchangeApdu(instance.apdu);
    } catch (CadTransportException ex) {
        System.out.println("Cad Transport Exception\n" + ex.getMessage().toString());
    } catch (IOException ex) {
        System.out.println("IO Exception\n" + ex.getMessage() + "\n");
    }
    System.out.println("New Stored Secret: ");
    //instance.printBytes(instance.apdu.getResponseApduBytes());
    instance.printDataOutBytes(instance.apdu);
}

4.3.4 Method to Create New Login Session

The following Java card Applet code segment is the method to create a new login session while
a username, password are provided and the user’s smart card is present.

//if last index of args is "C" means create a new login
if(args.length == 3 && "C".equals(args[2]))
{
    "CreateNewLogin"
    "command the java card to compute a hash using
    "its stored secret, the username, and password
    args[0] contains the username ; args[1] contains the password
    byte[] data = new byte[args[0].getBytes().length + args[1].getBytes().length];

    for(int i=0; i < data.length; i++)
    {
        if(i < args[0].length())
        {
            data[i] = args[0].getBytes()[i];
        }
        else
        {
            data[i] = args[1].getBytes()[i-args[0].length()];
        }
    }
instance.getHash(data, args[0].length(), args[1].length());

try {
    instance.clientInterface.exchangeApdu(instance.apdu);
} catch (CadTransportException ex) {
    System.out.println("Cad Transport Exception\n" + ex.getMessage().toString());
} catch (IOException ex) {
    System.out.println("IO Exception\n" + ex.getMessage() + "\n");
}

//Print HASH value
instance.printDataOutBytes(instance.apdu);
}

4.3.5 Method to return the user’s hash value

The following Java card Applet code segment is the method for the Java card to return the user’s hash value while a username, password are provided to the Java card and the user’s smart card is stored inside the Java card.

To just get the hash value for comparison, only the username and password are required. The SendCommandsToJavaCard program sends a command to the Java card to return SHA1 hash value to be sent to the server.

//command the java card to compute a hash using its stored secret, the username, and password
else
{
    //System.out.println("JustGetTheHash");

    //command the java card to compute a hash using
type stored secret, the username, and password
//args[0] contains the username ; args[1] contains the password
byte[] username = new byte[usersUsername.getBytes().length];
username = usersUsername.getBytes();
byte[] password = new byte[usersPassword.getBytes().length];
password = usersPassword.getBytes();
byte[] data = new byte[usersUsername.getBytes().length + usersPassword.getBytes().length];

    for(int i=0; i < data.length; i++)
    {
        if(i < username.length)
        {
            data[i] = username[i];
else {
    data[i] = password[i-username.length];
}

System.out.println("Data: " + data.toString());
System.out.println("Data2: ");
instance.printBytes(data);

instance.getHash(data, username.length, password.length);

try {
    instance.clientInterface.exchangeApdu(instance.apdu);
} catch (CadTransportException ex) {
    System.out.println("Cad Transport Exception\n" + ex.getMessage().toString());
} catch (IOException ex) {
    System.out.println("IO Exception\n" + ex.getMessage() + "\n");
}

//Print HASH value
instance.printDataOutBytes(instance.apdu);
}

These methods form the basis for user authentication. The SendCommandsToJavaCard is responsible for getting the username and password from the user. Since this program runs on the client’s system, the user’s password is never transmitted to the server over the network. Once the program takes in the username and password, it runs the necessary function to get the hash value from the Java card and return it. This hashed value and the user’s username are passed through the network to the server. The server can then compare the username and hash value to verify that the user is who they say they are. If the checks are valid, then the user is authenticated and is granted access to the system.

By computing this hash value by the Java card and sending the username and the computed hash value to the server for comparison, the authentication process requires the smart card to be present. If a malicious user manages to steal a user’s username and password, they still will not be able to log into the system because they do not have the smart card which contains the stored secret value and can compute the hash value. This authentication scheme requires the user to have his or her smart card in possession in order to log into the
server. The stored secret value on the smart card is needed to calculate the hash value correctly and authenticate the user into the system. Compared with the previous user authentication schemes, our proposed authentication scheme is more secure because it implements a two-factor authentication. Even if the user’s password is compromised, a user would still need the smart card to log into the system.

### 4.3.6 Server Side Implementation

**Registration Phase Server Code**

The server sided code was also implemented, using the C\# language, as part of this master thesis research project. When a registration request is received through the web application, the server side method gets the first name and last name that were entered through the web application text boxes. Then the web server calls the `SendCommandsToJavaCard` program which is on the client machine. A method is called which generates a new stored secret on the smart card. Then the method to get the username and the password on the client side is called. Using the new username and password entered and the new stored secret, the smart card computes the hash. The username and the hash value are returned to the web server to be stored in the authentication database.

```csharp
protected void CreateAccountButton_Clicked(object sender, EventArgs e)
{
    string usersFirstName = firstName.Text;
    string usersLastName = lastName.Text;
    string usersUsername = "";
    string hashedValue = "";

    //generate new stored secret
    string commandStoredSecret = "java -jar \"C:\Documents and Settings\BRANDON JAMES\My Documents\NetBeansProjects \SendCommandsToSmartCard\dist\SendCommandsToSmartCard.jar\"";
    commandStoredSecret = commandStoredSecret + " ";
    commandStoredSecret = System.DateTime.Now.ToString().Substring(0, 20) + " " + "N";

    new System.Diagnostics.ProcessStartInfo("cmd.exe", "/C " + commandStoredSecret);
```
newSecret.CreateNoWindow = true;
newSecret.UseShellExecute = false;
newSecret.RedirectStandardOutput = true;

javaProcessNewSecret.WaitForExit();
javaProcessNewSecret.Close();

//add login
string command =
"java -jar "C:\Documents and Settings\BRANDON JAMES\My Documents\NetBeansProjects\SendCommandsToSmartCard\dist\SendCommandsToSmartCard.jar"";
command = command + "C";

System.Diagnostics.ProcessStartInfo startInfo =
new System.Diagnostics.ProcessStartInfo("cmd.exe", "/C " + command);
startInfo.CreateNoWindow = true;
startInfo.UseShellExecute = false;
startInfo.RedirectStandardOutput = true;
System.IO.StreamReader outputReader = javaProcess.StandardOutput;
javaProcess.WaitForExit();
javaProcess.Close();

string returnAPDU = outputReader.ReadToEnd();
returnAPDU = returnAPDU.Replace("\r", "");
returnAPDU = returnAPDU.Replace("\n", "");

//returnAPDU contains string formatted as follows:
// username////////hashedvalue
int index = returnAPDU.LastIndexOf("////////");
usersUsername = returnAPDU.Substring(0, index);
hashedValue = returnAPDU.Remove(0, usersUsername.Length + 7);

OleDbConnection conn =
new OleDbConnection
("Provider=Microsoft.Jet.OLEDB.4.0;Data Source=C:/Documents and Settings/BRANDON JAMES
/my documents/visual studio 2010/Projects/TestAuth/TestAuth/App_Data/Users.mdb;");
conn.Open();
OleDbCommand insertCommand = new OleDbCommand();
insertCommand.Connection = conn;

string insertQuery =
"INSERT INTO Users([firstName], [lastName], [username], [hashedUN_PW_SS])
VALUES(@usersFirstName, @usersLastName, @usersUserName, @hashedValue)";
insertCommand.CommandText = insertQuery;
insertCommand.Parameters.Add("@usersFirstName", OleDbType.VarChar).Value = usersFirstName;
insertCommand.Parameters.Add("@usersLastName", OleDbType.VarChar).Value = usersLastName;
insertCommand.Parameters.Add("@usersUserName", OleDbType.VarChar).Value = usersUsername;
insertCommand.Parameters.Add("@hashedValue", OleDbType.VarChar).Value = hashedValue;
insertCommand.ExecuteNonQuery();
conn.Close();
firstName.Text = "";
lastName.Text = "";
hashedValue = "";

//clears any current authentication cookie
FormsAuthentication.SignOut();
string returnURL = "~/Default.aspx";
Response.Redirect(returnURL);
}

User Authentication Phase Server Code

The server side code of the web application for the user authentication phase calls the client program SendCommandsToJavaCard. This program gets the username and password from the user and sends the values to the smart card. The smart card computes and returns the hash value using the stored secret that was generated during the registration phase. This hash value along with the username is sent from the client to the web server. Using the username and hash value from the client, the web server verifies that the data matches the values in the authentication database.

protected void LoginButton_Clicked(object sender, EventArgs e)
{
    bool userIsValid = false;
    LoginError.Visible = false;
    LoginError.Text = "Login Error";
    string usersUsername = "";

    string command =
    "java -jar "C:\Documents and Settings\BRANDON JAMES\My Documents\NetBeansProjects\\SendCommandsToSmartCard\dist\SendCommandsToSmartCard.jar"
    System.Diagnostics.ProcessStartInfo startInfo =
    new System.Diagnostics.ProcessStartInfo("cmd.exe", "/C " + command);
    startInfo.CreateNoWindow = true;
    startInfo.UseShellExecute = false;
    startInfo.RedirectStandardOutput = true;
    System.IO.StreamReader outputReader = javaProcess.StandardOutput;
    javaProcess.WaitForExit();
    javaProcess.Close();

    string returnAPDU = outputReader.ReadToEnd();

returnAPDU = returnAPDU.Replace("\r", "");
returnAPDU = returnAPDU.Replace("\n", "");

//return APDU contains the username and hash value formatted as follows:
// username://hashValue
int index = returnAPDU.LastIndexOf("///////");

if (index > 0)
{
    usersUsername = returnAPDU.Substring(0, index);

    returnAPDU = returnAPDU.Remove(0, usersUsername.Length + 7);
}

//Get Hashed Value from database to compare
OleDbConnection conn = new OleDbConnection("Provider=Microsoft.Jet.OLEDB.4.0;
Data Source=C:/Documents and Settings/BRANDON JAMES
/my documents/visual studio 2010/Projects/TestAuth/TestAuth/App_Data/Users.mdb;")){
    conn.Open();
    OleDbCommand queryCommand = new OleDbCommand();
    queryCommand.CommandText = selectQuery;
    queryCommand.Parameters.Add("@username", OleDbType.VarChar).Value = usersUsername;
    OleDbDataReader result = queryCommand.ExecuteReader();
    string hashedValue = "";
    if (result.HasRows)
    {
        while (result.HasRows)
        {
            string colName = result.GetName(0);

            while (result.Read())
            {
                hashedValue = result.GetValue(0).ToString();
            }
            result.NextResult();
        }
    }

    else
    {
        LoginError.Visible = true;
    }

    conn.Close();

    if (returnAPDU == hashedValue)
    {
        userIsValid = true;
    }
else
{
    userIsValid = false;
    LoginError.Visible = true;
}

if (returnAPDU.Contains("No Smart Card present."))
{
    userIsValid = false;
    LoginError.Text = "Login Error. Please insert smart card.";
    LoginError.Visible = true;
}

if (userIsValid)
{
    FormsAuthentication.SetAuthCookie(usersUsername, false /* createPersistentCookie */);
    string returnURL = "~/Default.aspx";
    Response.Redirect(returnURL);
}

Together these two methods implement the server-side functionality for the registration and user authentication phase of the proposed authentication scheme. Both of these methods execute a client side program which sends the commands to the smart card. The client-side program then processes the response from the smart card and returns the data to the web server.
CHAPTER 5

CONCLUSION & FUTURE WORK

5.1 Conclusion

In this thesis, we have proposed and implemented an user authentication scheme based on the smart card. The smart card based user authentication scheme presented in this thesis showed that it is feasible to use a smart card to ensure that data is accessed only by authorized individuals. By combining two methods of authentication, “what you have” and “what you know,” we provide a two-factor solution which will authenticate a user to a system. The implementation of the proposed user authentication scheme proved the correctness and effectiveness of the scheme. Compared with the previous user authentication schemes, our proposed authentication scheme is more secure because it implements a two-factor authentication. Even if the user’s password is compromised, a user would still need the smart card to log into the system. By implementing this scheme and writing the code for the Java card, client side, and server side, we have constructed a base that can be used in future research.

5.2 Future Work

There is a weakness existing in our proposed user authentication scheme. It cannot withstand replay attack. When the adversary impersonates a legal user to login the specified server by replaying the transmitted messages between the legal user and that server, then we say that this protocol is vulnerable to the replay attack [17].
The first direction of our future work of this research project is to expand our authentication scheme to protect against replay attacks. The current scheme is susceptible to replay attacks because the username and hash value could be hacked from the web server database or intercepted during transmission from the client to the server. To protect against the replay attack, we will propose a modified user authentication scheme that uses a random number generated by the web server and sent to the client for each login session. This random number will be unique to each login session. The hash value will then be re-computed, by the smart card, using this random number specific to the current session. This modified user authentication scheme will certainly prevent against the replay attack, but the computation complexity of the scheme will be increased. More messages need to be exchanged between the server and the user.

This modified user authentication scheme can be further improved with public key encryption. The random number used in the modified user-authentication scheme could be intercepted when sent to the client. To further enhance the security of the modified user authentication scheme, we propose to use public key encryption to protect the messages exchanged between the server and the user. The web server and the smart card all have their own pairs of public key and private key. The web server and the smart card will exchange their public keys during the registration phase. When the random message is sent to the smart card, it will be encrypted by the smart card’s public key and the smart card can then decrypt the random number using its own private key, and then compute the hash. The computed hash value can then be encrypted by the web server’s public key, and transmitted to the web server, who will then decrypt it using its own private key. The web server can then send the username and hash value to the user authentication database to verify the user’s identity. This user authentication scheme will be more secure than the modified user authentication scheme using random number. However, the computation complexity of the scheme will be increased compared to the previous scheme.

The second direction of our future work of this research project is to extend our re-
search to enhance the functionality and security of the computing system by coupling the smart card with another hardware security device, such as a TPM (Trusted Platform Module). In particular, the work done in [5] and [10] showed there is a lot of potential for using smart cards in various ways and coupling them with other security devices to design a more trusted computing platform.

The foundation of the research presented in this thesis can be used in other ways to enhance information security by utilizing more of the smart card’s features/functions. Also, combining smart card with other secure hardware components, for example, TPM (Trusted Platform Module), to further increase the security of a computing system.

By coupling a smart card with a TPM and designing an authentication scheme using both of these components, our goal is to exploit the features and functions of both components to build a trusted computing system with higher level of security. Smart cards can perform cryptographic functions and other security protocols, while still providing portability. A TPM is installed on the motherboard, which adds an encryption barrier and a key hierarchy that is used to access the system. Both of these hardware devices can store keys and hashed values which can be used to ensure a system has not been altered. This coupling will make the system more secure because only when the smart card is present, the system will be able to operate normally and make changes. We can also utilize the features of the TPM to provide integrity checks on the operating system.

In the future we also plan to move beyond implementing our user authentication solution in a simulation environment, and we will test our user authentication scheme using a physical smart card. By showing the user authentication scheme is realizable on a physical card, we will further prove that our authentication scheme with a smart card is feasible for use in real-world systems.
REFERENCES


