Security and E-payment System
2001, Spring Semester
Term Project Final Report

“Developing and Applying to SET with Java-card”

Team Name: Enigma
Members: Cho Won Chul
         Kim Hee Dae
         Lee Jung Hwan
         Yoon Won Jung
Submit Date: June 14, 2001
## List of Contents

1. Introduction
   
   Objective

2. E-payment System
   
   2.1 What is E-payment system
   
   2.2 Comparison of E-payment system protocols
   
   2.2.1 SSL
   
   2.2.2 SET
   
   2.2.3 C-SET
   
   2.2.4 Interoperability: SET/C-SET, SSL/SET
   
   2.2.5 Others: algorithm, biometric

3. SET
   
   3.1 Definition
   
   3.2 Background of SET
   
   3.3 History of SET
   
   3.4 Future of SET
   
   3.5 Certificates
   
   3.6 Transactions
   
   3.7 Encryption Process
   
   3.8 Three Parts to the SET System
   
   3.9 Secure Transmission Schemes in SET Protocol
   
   3.10 Essential Security Requirements in SET
   
   3.11 Entities of SET protocol in Cyber shopping
   
   3.12 Overview of main Messages in SET

4. JAVACARD
   
   4.1 Smart Card Hardware
   
   4.1.1 Smart Card Contact Points
   
   4.1.2 Smart Card Central Processing Unit
4.1.3 Smart Card Coprocessors

4.2 Smart Card Communication data units

4.2.1 APDU Protocol

4.3 Smart Card Operating Systems

4.4 Java Card Technology

4.4.1 java Card Virtual Machine(JCVM)

4.4.2 Java card runtime environment(JCRE)

4.4.3 Java Card API

4.4.4 Applet Program Installation

5. Implementation of Java Card

5.1 Scenario

5.2 Java Card Program

6. E-payment Application in SET

7. Conclusion and further more

Appendix (Source code)

1. Server.java

2. Crytostest.java

3. Client.java

Reference
List of Figure

< Figure 2-1> C-SET/SET ....................................................................................................................12
< Figure 2-2> C-SET/SET..................................................................................................................13
< Figure 2-3> Hybrid SET/SSL model ...............................................................................................14
< Figure 2-4> Exchanging message in Hybrid SET/SSL model ............................................................15
< Figure 3-1> Entities of SET protocol in Cyber shopping .................................................................21
< Figure 3-2> Overview of main Messages in SET ..............................................................................22
< Figure 4-1> Java Card Contact Point ..............................................................................................24
< Figure 4-2> Send APDU ....................................................................................................................25
< Figure 4-3> Response APDU ..........................................................................................................25
< Figure 4-4> Java Card File System ..................................................................................................26
< Figure 4-5> JCVM ...............................................................................................................................27
< Figure 4-6> JCRE Structure .............................................................................................................28
< Figure 4-7> Java Card Program Installation Process .........................................................................29
< Figure 5-1> Dual Signature Process ................................................................................................33
< Figure 5-2> Payment Information Encryption ....................................................................................34
< Figure 5-3> Verify Dual Signature ....................................................................................................35
1. Introduction

As we enter the second millennium we experience one of the most important changes in our lives—the move to an Internet-based society. Almost everything will be changed at home, in school, at work, in the government—even in our leisure activities. Some changes are already here and they are spreading around the globe. Others are just beginning. One of the most significant changes is in the manner we conduct business especially in how we manage the marketplace and commerce.

Electronic commerce describes the manner in which transactions take place over networks, mostly the Internet. It is the process of electronically buying and selling goods, services, and information.

How about the payment in EC? We have face on many difficult problems to solve this kind of transaction. Now the importance of E-payment system is one of hot issues in Electronic Commerce. So our team wants to introduce the overall E-payment system and some implementation using Smart-card.

Our main objective this project is following:
- Describing typical electronic payment systems for EC
- Comparing the relationship between SSL and SET protocols
- Classifying and describing the types of Smart Card used for payments
- Describing the characteristics of Java Card
- Implementing Java Smart Card applying to SET

To achieve these objectives we have studied many related topics especially Java-card encryption. So here we would like to introduce our compositions. The topic is like following:
- E-payment system and it's protocol; SSL, SET, C-SET, SET/ C-SET, SSL/SET, etc.
- More SET
- JAVA CARD
- Implementation of Java Card
- E-payment Application in SET
2. E-payment System

2.1 What is E-payment system?

E-payment system is the new payment methods with the emergence of electronic commerce on the Internet. Secure payment systems are critical to the success of EC.
There are four essential security requirements for safe electronic payments. (Authentication, Encryption, Integrity, Non-repudiation)
The key security schemes adopted for electronic payment systems are encryption.
Security schemes are adopted in protocols like SSL and SET.

2.2 Comparison of E-payment system protocols

2.2.1. SSL

Concept
Short for Secure Sockets Layer, a protocol developed by Netscape for transmitting private documents via the Internet. SSL works by using a private key to encrypt data that is transferred over the SSL connection. Both Netscape Navigator and Internet Explorer support SSL, and many Web sites use the protocol to obtain confidential user information, such as credit card numbers. SSL is an open, nonproprietary protocol. It has been submitted to the W3 Consortium (W3C) working group on security for consideration as a standard security approach for World Wide Web browsers and servers on the Internet.
Taher Elgamal was one of the developers of Secure Sockets Layer. Currently, he is president of the Information Security Group for The Kroll-O'Gara Company. "The primary use of SSL is in virtually all the encrypted e-commerce credit-card transactions today. SSL consists of both the handshake – the negotiation that convinces the browser and the server that they both support the methods that they agreed on -- and the secure pipe to make sure that people cannot attack SSL itself," he says.
By convention, Web pages that require an SSL connection start with https: instead of http:, and provide data encryption, server authentication, message integrity, and optional client authentication for a TCP/IP connection. The secure environment for SSL is created through the use of public key cryptography, which consists of the encryption (scrambling) and decryption (unscrambling) of information. Public key cryptography allows anyone to send an encrypted message to a designated recipient, using what is known as a public key. The recipient then uses a private key to decrypt the message. Therefore, only the designated recipient has the ability to read the message.
SSL comes in two strengths, 40-bit and 128-bit, which refer to the length of the "session key" generated by every encrypted transaction. The longer the key, the more difficult it is to break the encryption code. Most browsers support 40-bit SSL sessions, and the latest browsers, including Netscape Communicator 4.0, enable users to encrypt transactions in 128-bit sessions – trillions of times stronger than 40-bit sessions. Global companies that require international transactions over the web can use global server certificates program to offer strong encryption to their customers.

**Characteristics**

The primary goal of the SSL Protocol is to provide privacy and reliability between two communicating applications. The protocol is composed of two layers. At the lowest level, layered on top of some reliable transport protocol (e.g., TCP[TCP]), is the SSL Record Protocol. The SSL Record Protocol is used for encapsulation of various higher level protocols. One such encapsulated protocol, the SSL Handshake Protocol, allows the server and client to authenticate each other and to negotiate an encryption algorithm and cryptographic keys before the application protocol transmits or receives its first byte of data. One advantage of SSL is that it is application protocol independent. A higher level protocol can layer on top of the SSL Protocol transparently.

The SSL protocol provides **connection security** that has three basic properties:

- The connection is private. Encryption is used after an initial handshake to define a secret key. Symmetric cryptography is used for data encryption (e.g., DES[DES], RC4[RC4], etc.).
- The peer's identity can be authenticated using asymmetric, or public key, cryptography (e.g., RSA[RSA], DSS[DSS], etc.).
- The connection is reliable. Message transport includes a message integrity check using a keyed MAC. Secure hash functions (e.g., SHA, MD5, etc.) are used for MAC computations.

**working and Handshake**

**Working:**

SSL works through encryption, specifically, public key encryption. This is a technique that uses a pair of asymmetric keys for encryption and decryption. Each pair of keys consists of a public key and a private key. The public key is made public by distributing it widely. The private key is never distributed; it is always kept secret.

Data that is encrypted with the public key can be decrypted only with the private key. Conversely, data encrypted with the private key can be decrypted only with the public key. This asymmetry is the property that makes public key cryptography so useful. The primary goal of the SSL Protocol is to provide privacy and reliability between two communicating applications. The protocol is composed of two layers. At the lowest level, layered on top of some reliable transport protocol (e.g., TCP[TCP]), is the SSL Record Protocol. The SSL Record Protocol is used for encapsulation of various higher level protocols. One such encapsulated protocol, the SSL Handshake Protocol, allows the server and client to authenticate each other and to negotiate an encryption algorithm and cryptographic keys before
the application protocol transmits or receives its first byte of data. One advantage of SSL is that it is application protocol independent. A higher level protocol can layer on top of the SSL Protocol transparently. Privacy is guaranteed through encryption. Although information can still be intercepted by a third party, they will be unable to read them as they have no access to the encryption key. Integrity is also ensured through encryption. If information is received that will not decrypt properly then the recipient knows that the information has been tampered with during transmission. Authentication is provided through digital certificates. Digital certificates provide the basis for secure electronic transactions as they enable all participants in a transaction to quickly and easily verify the identity of the other participants.

**Handshaking:**

The SSL protocol uses a combination of public-key and symmetric key encryption. Symmetric key encryption is much faster than public-key encryption, but public-key encryption provides better authentication techniques. An SSL session always begins with an exchange of messages called the SSL handshake. The handshake allows the server to authenticate itself to the client using public-key techniques, then allows the client and the server to cooperate in the creation of symmetric keys used for rapid encryption, decryption, and tamper detection during the session that follows. Optionally, the handshake also allows the client to authenticate itself to the server. Here in detail are the steps taken during a SSL transaction:

1. The client sends a request for a document to be transmitted using the HTTPS protocol by prefixing the URL with "https".
2. The server sends its certificate to the client.
3. The client checks if the certificate was issued by a Certificate Authority (CA) it trusts. If not, it gives the user the option to continue or to terminate the transaction.
4. The client compares the information in the certificate with the information it just received concerning the site: its domain name and its public key. If the information matches, the client accepts the site as authenticated.
5. The client tells the server what ciphers, or encryption algorithms, it can communicate with.
6. The server chooses the strongest common cipher and informs the client.
7. The client generates a private (or session) key using the agreed cipher.
8. The client then encrypts the session key using the server's public key and sends it to the server.
9. The server receives the encrypted session key and decrypts it with its private key.
10. The client and the server then use the session key for the rest of the transaction.

It is important to note that both client and server authentication involve encrypting some piece of data with one key of a public-private key pair and decrypting it with the other key. In the case of server authentication, the client encrypts the premaster secret with the server’s public key. Only the corresponding private key can correctly decrypt the secret, so the client has some assurance that the identity associated with the public key is in fact the server with which the
client is connected. Otherwise, the server cannot decrypt the premaster secret and cannot generate the symmetric keys required for the session, and the session will be terminated.

In the case of client authentication, the client encrypts some random data with the client's private key— that is, it creates a digital signature. The public key in the client's certificate can correctly validate the digital signature only if the corresponding private key was used. Otherwise, the server cannot validate the digital signature and the session is terminated.

The SSL protocol runs above TCP/IP and below higher-level protocols such as HTTP or IMAP. It uses TCP/IP on behalf of the higher-level protocols, and in the process allows an SSL-enabled server to authenticate itself to an SSL-enabled client, allows the client to authenticate itself to the server, and allows both machines to establish an encrypted connection.

2.2.2. SET (more specific description in next chapter)

**Concept**
SET is the Secure Electronic Transaction protocol developed by Visa and MasterCard specifically for enabling secure credit card transactions on the Internet. Like SSL, SET allows for the merchant's identity to be authenticated via digital certificates. However, SET also allows for the merchant to request users authenticate themselves through digital certificates. This makes it much more difficult for someone to use a stolen credit card. A further advantage of SET is that the merchant has no access to credit card numbers and thus another source of fraud is eliminated.

SET is exempt from the US cryptographic export restrictions and unlike SSL can therefore use strong, 128 bits encryption for credit card numbers world-wide. In order to gain this exemption, the use of strong encryption has to be limited to the financial information only and does not include other elements of the transaction, for example details of the goods being bought and the delivery address.

It uses digital certificates to ensure the identities of all parties involved in a purchase and encrypts credit card information before sending it across the Internet.

**Difference with SSL and advantage**
To begin with, it's an entirely different kind of transaction. Once you initiate an SSL session with a commerce site, the transaction isn't any different than when you give your credit card to a clerk in a department store. They tell you the price, you say OK, and you wait while the clerk attempts to validate your card and the purchase amount with the bank card company. If it's approved, the clerk prints out a receipt, which you sign, keeping a copy for your records.

With SET, the bank card company gets involved in the middle of the transaction, essentially acting as middleman. Theoretically, when you enter your card in an online SET transaction, the commerce site might never see your actual credit card number. Instead, that info would go to the financial institution, who would verify the card and the amount, then make payment to the commerce site, all in exchange for a fee, of course.
What are the advantages of SET? Well, we're speaking theoretically here, since SSL is actually up and running and SET isn't. With that proviso out of the way, SET could offer more security for online transactions, by adding the mandatory authentication between buyer and seller, using digital certificates. It could also speed up the payment settlement process, since the financial institutions are involved up front, rather than after the fact. It might also increase privacy, since you wouldn't necessarily be sharing your credit card number with online retailers, but with your financial institution (which presumably already knows your card.)

2.2.3. C-SET

C-SET stands for "Chip-Secure Electronic Transaction" which means "system for providing secure electronic payment transactions over the Internet using Smart-Cards". The name was chosen in February 1996, for several reasons:
- to emphasize that the protocol is based on the SET (Secure Electronic Transaction) protocol defined by MasterCard and VISA, and incorporates all of its principles,
- to indicate that the SET protocol has been adapted to Smart-Card technology, or rather, to show that it has been extended to include a physical level of security (something that SET had expressly left out). Only the use of card readers and black boxes can provide real security.

The Anglo-Saxon abbreviation C-SET was then adopted by the French banking community.

Various fraud risks

- **Invalid card carrier**: the customer provides an invalid card number, or a real number which isn't his. The hacker can usually be traced by means of the name and delivery address provided (assuming they are correct!), but in doing so the merchant might incur substantial investigation costs, perhaps even legal costs. In some circumstances, this can result in a loss for the merchant. Should a real number be used, this is also bound to result in a contestation by the genuine card holder, with ensuing damage to the reputation of the card, the bank and the merchant.

- **Invalid merchant**: here, a hacker sets up an Internet server allegedly to sell goods, but delivers nothing and vanishes after a few weeks, having recorded all the card numbers, with the intention of using them fraudulently elsewhere. A variation on this involves a hacker tapping into the Web site of a genuine merchant and reading off the card numbers as they are uploaded by unwitting customers. The genuine merchant delivers the goods ordered to its customers, so there are no immediate complaints, and this form of fraud can therefore last for much longer before anyone notices it. In both cases, there is a risk of loss to the bank of the "defrauded" customer, and a risk of losing credibility in the eyes of customers.

- **"Trojan horse" viruses**: viruses are programs in their own right which can be transmitted by means of an infected diskette, through a local network, an FTP download, a message received with an attachment, or simply by
viewing an Internet Web page (via Java applets or Active X). Some viruses, known as "Trojan horses" (which are luckily still extremely rare, though this might not last), can lie dormant in your system, awaiting specific events such as the inputting of a sequence of numbers from the keyboard, which might constitute a card number, or the shifting of a non-encrypted card number along the computer's internal bus, or the detection of an X509-standard file which might constitute a customer certificate, etc. Once detected, the data in question is recorded pending a period of inactivity of the host computer (lunch-time, or even the middle of the night) whereupon the virus can trigger and establish an Internet connection (by switching on the computer if need be, as is possible on the latest models, and by disabling the "firewall" protection systems of corporate networks which often only filter incoming accesses), and send an e-mail to the hacker, containing the intercepted data. The hacker can then pretend to be the real customer. The financial risk is the same as with the "invalid merchant" case.

"non-WYSIWYG" viruses: refers to viruses which are capable of displaying on-screen different information from that used in the actual payment under way. For example, the true price of the item ordered is displayed, but a sum ten times larger is sent to the merchant. These viruses could be used to generate illicit profits for fraudulent merchants (who would then disappear less than a fortnight later, before customers start complaining), or more likely cause disruptions in dealings with bona fide merchants, leading to a loss of credibility in the system among users. To date, there have been no known cases of such a virus.

SET security provides some protection against the above risks. C-SET security, on the other hand, provides total protection against them.

**Distinguishing features of C-SET relative to SET**

**SET** is a protocol which was devised by MasterCard, VISA and various US parties with a strong vested interest in the Internet, in order to secure payments made by cards over the Internet. Customers do not physically use their cards; instead, they make use of a pre-allocated "SET certificate". This certificate is recorded on the hard drive of a customer's computer, or on a diskette. C-SET is a protocol, but primarily an architecture, defined by Groupement des Cartes Bancaires. The C-SET protocol is an adaptation of the SET protocol, enabling the integration of a physical level of security (something which SET itself does not provide), using Smart-Cards, secure Smart-Card readers, and "Electronic Commerce Black Boxes":

- The key distinguishing feature of C-SET is that the bank card and its chip are used in a physical sense.
- The secure card readers come in the form of a box shaped like a calculator, and incorporating a dual-position Smart-Card reader: AFNOR for reading BO'-type cards (such as the bank cards currently issued in France) and ISO for reading EMV cards (for future applications). They also incorporate a numerical keypad and a display. Finally, they come with internal software, and an RSA encryption module to verify or issue signatures, and check the authenticity of the software used. The internal software is uploaded in a secure manner, which will open the way for the integration of other applications, such as Home Banking, access control, the secure downloading of information into Smart-Cards, etc. It will also enable a smooth transition towards the use of EMV-standard cards. Finally, a system checks that the
carrier software is authentic (i.e. certified by the bank) and offers a facility to trace any attempts to subvert the system. Such card readers may eventually be integrated into the standard design of micro-computers by manufacturers (into keyboards, for example). In order to be operational, the card readers will have to be approved by Groupement des Cartes Bancaires.

The **Electronic Commerce Black Boxes** are secure encryption units, based on the banking Black Boxes currently in use in France, and adapted for use with TCP/IP networks and with RSA encryption.

Finally, we should point out that the C-SET architecture incorporates 2 fundamental categories of equipment: "Internet Remote Payment Controllers" and "Translators".

The Internet Remote Payment Controllers are devices that act as an Internet Payment Gateway and Remote Payment Controllers. The Internet Payment Gateway is a firewall and manages the C-SET protocol like does a Payment Gateway in a SET architecture but while controlling the chip-data. The Remote Payment Controllers are the same than those used to secure payment via Minitel (a semi-graphical terminal very well used in France for remote retailing and on-line services).

The **Translators** control international flows by acting as Secure Bridges between France and abroad to convert C-SET transactions in SET and reverse if necessary.

### 2.2.4. Interoperability

<table>
<thead>
<tr>
<th>SET/C-SET: C-SET for Cardholder and SET for Merchant / SET for Cardholder, C-SET for Merchant</th>
</tr>
</thead>
</table>

**Is the C-SET secure system compatible with the SET secure system?** Yes. The Translator located at Europay France will ensure full inter-operability with SET, both for international payments made by foreigners using SET on French Web sites (see Figure 2-1) and for international payments made by French nationals equipped with a card reader and a Smart-Card on Web sites abroad (see Figure 2-2).
Furthermore, the Translator acts as a Secure Bridge enabling the use in France of French keys, even for international payments (where the merchant's private RSA key is abroad), thus complying with French law. C-SET is THE ONLY solution offering this facility.

SSL/SET: Hybrid SSL/SET architecture

Concept
This architecture would combine the benefits of SET with the simplicity of SSL. Clients and merchants could continue to use their SSL applications, while the SET services are assured on the bank side. This solution necessarily relies on a payment intermediary to manage the conversions between SET and SSL and to act as the proxy and guarantor of its clients (consumers and merchants) to SET certification authorities. For example, the payment intermediary could manage the SET certificates for customers and save them from the concomitant administrative loads. At the same time, the intermediary would guarantee the authenticity of those that it represents with respect to each other as well as with respect to the SET authorities. The intermediary could also play the role of a gateway SET/SSL. In this case, the client and merchant would talk to the payment intermediary on a channel secured by SSL. However, the intermediary would use the protocol SET to communicate with the payment gateway.

Hybrid SET/SSL model
<Figure 2-3> presents the hybrid SET/SSL model. In this configuration, the payment intermediary plays the roles of SSL server with respect to the client and the merchant, Web host for the merchant, and SSL certification authority for both.
From the SET viewpoint, the payment intermediary acts as the merchant’s agent and will be the merchant as far as the SET payment gateway is concerned. The intermediary will have two pairs of private/public keys, one pair for the exchange of data encryption keys and the other for signing the transmitted data. Since it has the role of SSL certification authority for the merchant and the client, it will own a third pair of private/public keys and an SSL certificate.

Transaction Flows

A transaction includes four parts

1. SSL Session between the client and the intermediary

The client visits the merchant’s site and chooses the desired article. He or she then clicks on the link to the merchant’s Web page and ends up on the server of the payment intermediary. After establishing an SSL session with the intermediary server, the client fills out an order form, which includes the client name, order details, name of the merchant, transaction amount, etc., and particularly the number of the bank card to be charged. The merchant does not have access to the client’s card number.

The certificate from the payment intermediary to the client contains the SSL certificate of the intermediary. The return Certificate message from the client includes the client’s SSL certificate that the intermediary has issued.
The intermediary next establishes another SSL session with the merchant. Once the communication has been secured, the intermediary transfers to the merchant a payment order without financial details or possibly without the purchaser identity. The merchant analyzes this request; if it is accepted, the merchant informs the intermediary of its acceptance, according to the application protocol established between the intermediary and the merchant. Once the intermediary is satisfied with the validity of the response, it initiates a SET communication with the SET payment gateway.

2. Payment Authorization

In this phase of the transaction, the intermediary behaves exactly as a merchant in the SET protocol, exchanging the messages shown in <Figure 2-4>.

Using the data collected from both the client and the merchant, the payment intermediary forms the authorization request AuthReq, and signs it with the private key of the merchant. If the client is SET-certified the dual signature of SET is constructed with the client private key; otherwise, the intermediary could use its own signature key. If the response is contained in AuthRes, the intermediary alerts the merchant that payment authorization has been obtained for delivery of goods to the client. Simultaneously, it signals to the client that the financial transaction has been authorized.

3. Notification of the Merchant and the Client

In this phase, the payment intermediary informs the client and the merchant of the authorization results using the SSL session already established.

4. Financial Settlement

Once the goods have been shipped, the merchant sends the intermediary a request for settlement (clearance).
exchanges follow the application protocol established between the intermediary and the merchant and are transported over a session secured by SSL. Having received the merchant’s request, the payment intermediary reconfigures it into the SET CapReq Message, which is directed toward the SET payment gateway. The response in the CapRes is converted back to the format common with the merchant and sent in a session by SSL.

2.2.5. Others : algorithm, biometric

There are some implementation Constraints like memory and limited computing power. To develop the high capacity products takes high costs. The solution about these constraints is meeting with Elliptic Curve Cryptosystem. It provide less EEPROM and shorter transmission time, scalability, no coprocessor, and on card key generation. Another technique is using biometric method.

Fingertip sensor on smart card provide authentication by sensor, self – activate and self – destroy system. Fingertip sensor on Card Reader provide smart card stores finger print information and finger print verification on card reader.

The SSL protocol, widely deployed today on the Internet, has helped create a basic level of security sufficient for most to conduct business over the Web. SSL is implemented in most major Web browsers used by consumers, as well as in merchant server software, which supports seller's virtual storefront in cyberspace. Hundreds of millions of dollars are already changing hands when cybershoppers enter their credit card numbers on Web pages secured with SSL technology. The SSL protocol is a powerful tool for the secure distribution of information, but does not address all of the risks associated with sending and accepting transactions over the Internet. For example, SSL establishes a secure session between a browser and a server. During the period when the browser is logged onto an SSL server, authentication between the browser and the server takes place. However, SSL does not authenticate the parties who are using that software. Thus, while cardholders using SSL can submit payment information free from the prying eyes of a third party, there is no way of verifying the identity of the online storefront that they are visiting. There are many pilot schemes running using the SET protocol, but mainstream adoption has been slower than predicted. The main reasons behind this are the growing acceptance of SSL for secure credit card transactions and the complexity and cost of the SET system.

In this project we will mainly deal with SET protocol because this SET is not only the basic concept to transact with multiple entities with, but also our object of this project is implementing SET with javacard. And we want to suggest more simple improved SET in the later chapter.
3. SET

3.1 Definition

SET is the Secure Electronic Transaction protocol developed by Visa and MasterCard specifically for enabling secure credit card transactions on the Internet. Like SSL, SET allows for the merchant's identity to be authenticated via digital certificates. However, SET also allows for the merchant to request users authenticate themselves through digital certificates. This makes it much more difficult for someone to use a stolen credit card. A further advantage of SET is that the merchant has no access to credit card numbers and thus another source of fraud is eliminated.

SET is exempt from the US cryptographic export restrictions and unlike SSL can therefore use strong, 128 bit encryption for credit card numbers world-wide. In order to gain this exemption, the use of strong encryption has to be limited to the financial information only and does not include other elements of the transaction, for example details of the goods being bought and the delivery address.

It uses digital certificates to ensure the identities of all parties involved in a purchase and encrypts credit card information before sending it across the Internet.

3.2 Background of SET

Visa and MasterCard joined forces to develop a single standard to secure payment card transactions over insecure networks. They released the first draft of the standards last summer, and since then they have been working feverishly to educate potential users and software developers. The final version of the specification came out May 31, and Secure Electronic Transactions (SET)-compliant applications should appear by the end of the year. Other major card brands, including American Express and NOVUS (Discover), have also endorsed the SET standard. SET's goals are to provide confidentiality, authentication, and integrity of payment card transmissions. To do this, SET uses a variety of cryptographic techniques, including encryption, digital signatures, and certificates.

3.3 History of SET

1995.2 SEPP(Secure Electronic Payment Protocol) announcement by MasterCard, Netscape, IBM, GTE
1995.2 STT(Secure Transaction Technology) announce by VISA, Microsoft
1996.1 SET announcement by VISA, MasterCard, GTE, IBM,Microsoft, Netscape, SAIC, Terisa, VeriSign
1996.8 SET 0.0 Spec. announcement
1997.5 SET 1.0 Spec. announce
1998.1 SET 1.0 I14Y(Interoperability, 14 char. between I and Y) is going on
3.4 Future of SET
As you can see, SET messages can be quite complex, and the specified key sizes (1,024- and 2,048-bit) are quite secure for now; but that in itself is not enough to guarantee security. As we've learned from the experiences of Netscape and others, the devil is in the details when it comes to implementing cryptographic algorithms. On paper, SET has gone a long way toward making payment card purchases more secure than they've ever been. For more information on SET, see either MasterCard's Web site, at

3.5 Certificates
The first thing all SET participants need -- cardholders, card issuers, merchants, and "acquirers" (like an issuer, but for merchants) -- is a set of public/private key pairs. Once you've generated these, you must get your public key guaranteed in a certificate issued by a trusted third party or certificate authority. The certificate authority's certificate is in turn guaranteed by another certificate authority, up a hierarchy of trust that culminates in a "root" certificate authority. Should any certificate become compromised (including the root), the standard has procedures in place to revoke and reissue certificates.

To help maintain security, cardholder certificates contain a one-way hashed value of the card number, expiration date, and a secret value generated by the cardholder's software. The information cannot be recovered directly from the certificate, but the cardholder can prove the certificate is his or hers by providing account information. For most cardholders, their certificate authority will be the same company that issues their card. How a certificate authority verifies the identity of the certificate holder is outside the scope of the standard, but the process is likely to be similar to opening an account at a bank.

The developers of the SET standard are aware that most cardholders don't know, or care, about certificates right now, and educating them and building the infrastructure to issue and maintain large numbers of certificates takes time. As a result, they have allowed "interim" implementations to temporarily forgo issuing cardholder certificates (all other certificates are required). In this case, messages back to the cardholder (order status, for example) are not encrypted, but the integrity of messages from the cardholder is still ensured by the use of a message digest.

3.6 Transactions
Now armed with the necessary keys and certificates, a cardholder can start adding to his or her debt. Most SET transactions consist of two rounds of requests and responses. First, the requester and responder exchange certificates and verify each other's authentication information. Then, if the first round succeeds, the actual request and response take place.

Let's say you've decided what you want to buy, your software has completed the round of certificate exchanges successfully, and now you're ready to send your purchase on its way. From the certificate exchange, you have the
merchant's public key, the payment processor's key, and a unique transaction identifier issued by the merchant. How does SET deliver your purchase securely?

First, you create the necessary order information (OI) and payment instructions (PI). Both include the merchant-assigned transaction identifier. Next, you apply a one-way hashing function to make digests of both the order information and payment instructions. Then you create a "dual signature." This allows both the merchant and payment processor to independently verify that your payment instructions and order information belong together without requiring either processor to see the other's data. SET's dual signature is a digest of the concatenation of the OI and PI message digests, encrypted with your private key. For better encryption/decryption performance, you select a random symmetric key and encrypt the payment instructions with it; and, finally, you encrypt the random symmetric key and your account number with the payment processor's public key. When you're done, you have a message containing:

- OI, including the merchant's transaction identifier
- A digest of the order information
- PI, including merchant's transaction identifier, encrypted with a random symmetric key
- A digest of the payment instructions
- A dual signature: digest (OI digest + PI digest) encrypted with your private key
- Your account number plus the random symmetric key encrypted with the payment processor's public key.

### 3.7 Encryption Process

In a typical SET transaction, there is information that is private between the customer and the merchant (such as the items being ordered) and other information that is private between the customer and the bank (such as the customer's credit card number). SET allows both kinds of private information to be included in a single, digitally signed transaction.

Information intended for the bank is encrypted using the bank's public key whilst information for the merchant is encrypted with the merchant's public key. This means that the merchant has no access to the credit card details and thus a source of fraud is eliminated.

In addition to this encryption, both sets of information are digitally signed. Finally these two signatures are combined to produce one signature that covers the whole transaction.

In detail, the steps that are taken are:

1. The customer selects the items to be purchased, and pushes the "purchase" button.
2. The browser will send an initial message to the webserver, a SET-PDU.
3. The server will respond, and will contain the merchant’s digital certificate, assuring the customer of the merchant’s authentication.
4. The customer’s software examines the message, and then the customer creates a purchase order which will include the sensitive credit card information. The cardholder encrypts the purchase order with a single new key that is generated. Then the key is encrypted with the gateway’s public key.
Now only the gateway can read the single key and therefore decrypt the purchasing information.

5. Merchant receives the purchase order, and decrypts it, and authenticates the customer. Now the merchant sends an authorization request message to the gateway.

6. The gateway decrypts the information, and authenticates the merchant. Then it decrypts the purchase order, and authenticates the customer.

7. The gateway now checks the company that issued the credit card. This ensures the card’s validity and limit, and finalizes the purchase.

8. The gateway receives the authorization from the credit card company, and sends a response message to the merchant.

9. The merchant sends a confirmation of the customer’s order to the customer. The process is finished!!

3.8 Three Parts to the SET System
The three parts to the SET system are: the "wallet" on the user's computer; a commerce server that runs at the merchant's Web site; and the payment server that runs at the merchant's bank. Although future versions of Microsoft's Internet Explorer and Netscape's Navigator will come with SET wallets pre-installed, currently users need to download and install a wallet on their computers. During the installation process the user provides credit card details and obtains user name and PIN to provide secure access to the wallet. The installation process also produces a public and private key for the user. The user also has to obtain a digital certificate from their bank or certificate agency (CA) for each credit card.

To use SET, users select products from the merchant's Web site and then elect to pay via SET by pressing an on-screen button. This automatically starts the wallet program. The user then selects which credit card to use, and the wallet and the merchant's server exchange certificates. If these are accepted, the wallet then encrypts and transmits the purchase details to the merchant. Although the purchase details include the encrypted credit card information, the merchant can not read them. Instead he passes this on to the bank's payment server which debits the users credit card and passes the payment to the merchant.

3.9 Secure Transmission Schemes in SET Protocol

Sender’s Computer

1. The message is hashed to a prefixed length of message digest.

2. The message digest is encrypted with the sender’s private signature key, and a digital signature is created.

3. The composition of message, digital signature, and Sender’s certificate is encrypted with the symmetric key which is generated at sender’s computer for every transaction. The result is an encrypted message. SET protocol uses the DES algorithm instead of RSA for encryption because DES can be executed much faster than RSA.

4. The Symmetric key itself is encrypted with the receiver’s public key which was sent to the sender in advance. The result is a digital envelope.
Receiver’s Computer
5. The encrypted message and digital envelope are transmitted to receiver’s computer via the Internet.
6. The digital envelope is decrypted with receiver’s private exchange key.
7. Using the restored symmetric key, the encrypted message can be restored to the message, digital signature, and sender’s certificate.
8. To confirm the integrity, the digital signature is decrypted by sender’s public key, obtaining the message digest.
9. The delivered message is hashed to generate message digest.
10. The message digests obtained by steps 8 and 9 respectively, are compared by the receiver to confirm whether there was any change during the transmission. This step confirms the integrity.

3.10 Essential Security Requirements in SET
Authentication: A way to verify the buyer’s identity before payments are made
Integrity: Ensuring that information will not be accidentally or maliciously altered or destroyed, usually during transmission
Encryption: A process of making messages indecipherable except by those who have an authorized decryption key
Non-repudiation: Merchants need protection against the customer’s unjustifiable denial of placed orders, and customers need protection against the merchants’ unjustifiable denial of past payment

3.11 Entities of SET protocol in Cyber shopping

<Figure 3-1>
3.12 Overview of main Messages in SET

<Figure 3-2>
4. JAVACARD

Java Card® Sun® J ava ±â¼úÀ» ¸Þ¸ð¸®ÀÇ ÇѰ踦 °®´Â ½º¸¶Æ® Ä«µå³ª ´Ù¸¥ µð¹ÙÀ̽º»ó¿¡¼­ ±¸ÇöµÉ
¼ö ÀÖµµ·Ï ±Ô°ÝÀÌ Á¤ÇØÁ® ÀÖ´Ù. Java Card API ´Â ¾î¶² ½º¸¶Æ® Ä«µå Ç÷§Æû¿¡¼­ ÀÚ¹Ù Ä«µå ±â¼ú·Î
±¸ÇöµÈ ÀÀ¿ë ÇÁ·Î±×·¥À» ´Ù¸¥ Ç÷§Æû¿¡¼­ µ¿ÀÛÇϵµ·Ï ÇÑ´Ù. Java Card Application Environment (JCAE)´Â 90%ÀÇ ½º¸¶Æ® Ä«µå Á¦Á¶ ´É·ÂÀ» °®´Â ȸ»çµé¿¡°Ô OEM ¹æ½ÄÀ¸·Î ¶óÀ̼¾½ÌµÇ¾î ÀÖ´Ù.

?? Platform Independent - Java Card API® J ava Card ±ÔÁ¤µÇ¾î ÀÖ´Ù.
?? Multi- Application Capable - Multiple applications can run on a single card. ÇϳªÀÇ Ä«µå»ó¿¡¼­ ¿©·¯ ÀÀ¿ë ÇÁ·Î±×·¥À» µ¿ÀÛ½Ãų ¼ö ÀÖ´Ù.
?? Post-Issuance of Applications - Àû¿ëµÇ¾î ÇÁ·Î±×·¥À» ¼³Ä¡ÇÏ´Â ÇÏ´Â °ÍÀÌ °¡´ÉÇÏ´Ù. ÀÌ·¯ÇÑ ±â´ÉÀ¸·Î Ä«µå ¹ßÇàȸ»ç´Â °í°´ÀÇ º¯°æ ¿ä±¸¸¦ Àû±ØÀûÀ¸·Î ¼ö¿ëÇÒ ¼ö ÀÖ´Ù.
?? Flexible - Àû¿ëµÇ¾î ÇÁ·Î±×·¥À» ÀÛ¼ºÇϴµ¥ »ó´çÈ÷ À¯¿¬ ÇÏ´Ù.
?? Compatible with Existing Smart Card Standards - Å©±â¿Í ÀÖ´Ù API® ISO7816® Europay/Master Card/Visa (EMV)® ¾Ê¼ºÀÌ ÀÖ´Ù.

4.1 Smart Card Hardware

8 Contact Point Embedded CPU, ROM, RAM, EEPROM ÀÖ´Ù.

4.1.1 Smart Card Contact Points

8 Contact Point® ISO7816® ÀÌ¿ëµÈ Half-duplex mode.

- Vcc point : 3 5 7 9 11 13 15.
- RST point : 4 6 8 10 12 14 16.
- CLK point : 2 4 6 8 10 12 14 16.
- GND point : 1 3 5 7 9 11 13 15.
- Vpp point : Optional 2 4 6 8 10 12 14 16.
- I/O point : Optional 1 3 5 7 9 11 13 15 (Command) 1 3 5 7 9 11 13 15 Half-duplex mode.
RFU points: Reserved

4.1.2 Smart Card Central Processing Unit

CPU 8-bit microcontroller. Motorola 6805, Intel 8051, 5MHz. Clock multiplier 40MHz
16bit 32bit micro-controller

4.1.3 Smart Card Coprocessors

Cryptographic coprocessor. CPU modular arithmetic large-integer RSA

4.2 Smart Card Communication

CAD(Card Acceptance Device) Reader Terminal. Host Card half-duplex APDU

4.2.1 APDU Protocol

APDU Protocol ISO7816-4. APDU Host Card Command APDU Figure 4-2
Figure 4-3} Response APDU, send APDU Host.

<table>
<thead>
<tr>
<th>Mandatory Header</th>
<th>Optional Body</th>
</tr>
</thead>
<tbody>
<tr>
<td>CLA</td>
<td>INS</td>
</tr>
</tbody>
</table>

Figure 4-2} Send APDU

Optional Boay

Mandatory Trailer

Data field | SW1  | Sw2  |

4.3 Smart Card Operating Systems

O.S ISO7816- 4
Figure 4-4: Java Card File System

- **Master File (MF)**: Root file, dedicated files, elementary files. 
  - **Dedicated File (DF)**: DF, elementary file. 
  - **Elementary File (EF)**: EF, transparent file, linear fixed, linear variable, cyclic fixed.

4.4 Java Card Technology

- Java Card 2.1 Virtual Machine (JCVM) 
- Java Card 2.1 Runtime Environment (JCRE) 
- Java Card 2.1 Application Programming Interface (API)
4.4.1 java Card Virtual Machine (JCVM)

JCVM는 Java Virtual Machine의 상위 개념입니다. Off-card virtual machine과 on-card virtual machine으로 나뉩니다. Off-card VM는 Converter가 application class와 api file을 CAP (converted applet)로 변환하고, CAP는 Java Card의 EEPROM에 저장되어 on-card virtual machine의 (interpreter)로 실행됩니다.
4.4.2 Java card runtime environment (JCRE)

JCRE consists of Application Components and O.S.

![Figure 4-6] JCRE Structure

Figure 4-6 shows the JCRE structure, which includes O.S., Native Functions, Java Card Virtual Machine (JCVM), Java card application framework classes (APIs), and industry-specific extensions. Native Functions include low-level communication protocol, memory management, and cryptographic operations.

4.4.3 Java Card API

Java Card API includes classes 30, 31, 32, 33, 10, and 11.

- java.lang.
- javacard.framework
4.4.4 Applet Program Installation

Java Card Interpreter \(\text{CAP}\) \(\text{CAD}\) load\(\text{Component}\) \(\text{Installer}\) \(\text{On-card Installer}\) \(\text{Off-card Installation Program}\) \(\text{JCRE}\). \(\text{Figure 4-7}\) shows the Java Card Program Installation Process.

\(<\text{Figure 4-7}>\) Java Card Program Installation Process
5. Implementation of Java Card

5.1 Scenario

5.2 Java Card Program
private CryptoTest() {
    
                        
    } // end of the constructor

Install Method: install the security and e-payment system JCRE: install the Java Card Environment; Java Application: run the Java application.
public static void install(APDU apdu) {
    // create a CryptoTest applet instance
    new CryptoTest();
} // end of install method

public boolean select() {
    // reset validation flag in the PIN object to false
    pin.reset();
    // returns true to JCRE to indicate that the applet
    // is ready to accept incoming APDUs.
    return true;
} // end of select method

public void process(APDU apdu) {
    // APDU object carries a byte array (apduBuffer) to
    // transfer incoming and outgoing APDU header
    // and data bytes between card and CAD
    apduBuffer = apdu.getBuffer();
    switch (apduBuffer[ISO.OFFSET_INS]) {
        case SHATest:   RunTest9(apdu);     return;
        case RSAEnc:    RunTest10(apdu);    return;
        case RSADec:    RunTest11(apdu);    return;
        case PinCheck:  CheckThePin(apdu);  return;
        case SETTest:   RunSETTest(apdu);   return;
        case KeyEnc:    RunKeyEnc(apdu);    return;
        case SendCert:  RunSendCert(apdu);  return;
        case VerDual:   RunVerDual(apdu);   return;
        case VerPay:    RunVerPay(apdu);    return;
        case DecSym:    RunDecSym(apdu);    return;
        default: ISOException.throwIt (ISO.SW_INS_NOT_SUPPORTED);
    }
} // end of process method

private void CheckThePin(APDU apdu) {
    short byteRead = (short)(apdu.setIncomingAndReceive());
    if (byteRead != 4)
        ISOException.throwIt(ISO.SW_WRONG_LENGTH);
} // end of CheckThePIN method

// Takes an input of 4 bytes and checks to see if it is the actual 'mypin' set above
private void CheckThePin(APDU apdu) {
    short byteRead = (short)(apdu.setIncomingAndReceive());
    if (byteRead != 4)
        ISOException.throwIt(ISO.SW_WRONG_LENGTH);
if (pin.check(apduBuffer, (short)APDUDATA, (byte)4) == false){
    ISOException.throwIt(SW_WRONG_PIN);
}
} // end of CheckThePin method

RunSETTest method

// Takes an input message from the incoming APDU
// and make digital dual signature and concatenate to other information.
// It then sends the result back.
void RunSETTest(APDU apdu) {
    // access authentication
    if ( !pin.isValidated() )
        ISOException.throwIt(ISO.SW_PIN_REQUIRED);
    // read apdu buffer

    // Dual Signature Process
    <Figure 5-1> Dual Signature Process
short byteRead = (short)(apdu.setIncomingAndReceive());

// make new allocation digest message
// hash order information to 20 bytes digest
sha.generateDigest(apduBuffer, (short)APDUDATA, (short)30,
 apduBuffer, (short)APDUDATA);

// hash payment information to 20 byte digest
sha.generateDigest(apduBuffer, (short)(APDUDATA + 30), (short)30,
apduBuffer, (short)(APDUDATA +20));

// make new allocation op digest message buffer
sha.generateDigest(apduBuffer, (short)APDUDATA, (short)40,
apduBuffer, (short)(APDUDATA+40));

// perform the encryption
RSAEncryptPrivateKey.cryptoUpdate(apduBuffer, (short)(APDUDATA+40),
(short)20, apduBuffer, (short)(APDUDATA+40));

// Send results
apdu.setOutgoing();
// indicate the number of bytes in the data field
apdu.setOutgoingLength((short)168);
apdu.sendBytesLong(apduBuffer, (short)APDUDATA, (short)168);

}

RunKeyEnc method[] Card Holder[] Payment Information[] tripleDEC key[] . [] [] [] Payment Gateway
[] Public Key[] Encrypt[][] Host[] . [] [] [] [] [] Figure <5-2> [] .

<Figure 5-2> Payment Information Encryption
private void RunKeyEnc(APDU apdu) {
    // access authentication
    if (!pin.isValidated())
        ISOException.throwIt(ISO.SW_PIN_REQUIRED);

    // read apdu buffer
    short byteRead = (short)(apdu.setIncomingAndReceive());
    // Takes Payment Information and encrypts it using triple DES Key.
    des3key.setKey(tripleDESKey, (short)0);
    des3key.encryptECB(apduBuffer, (short)APDUDATA, (short)32,
                       apduBuffer, (short)APDUDATA);
    short i;
    for(i = 0; i < (short)16; i++)
        apduBuffer[(short)APDUDATA + (short)32 + i] = (byte)i;
    // encrypt Symmetric information and Cardholder's account Information
    // by using payment gateway's public key.
    RSAPublicKey.cryptoUpdate(apduBuffer, (short)(APDUDATA + 32), (short)16,
                              apduBuffer, (short)(APDUDATA + 32));

    // Send results
    apdu.setOutgoing();
    // indicate the number of bytes in the data field
    apdu.setOutgoingLength((short)160);
    apdu.sendBytesLong(apduBuffer, (short)APDUDATA, (short)160);
}

RunVerDual method  RunVerPay method  Figure 5-3  Merchant Server  Java Card
private void RunVerDual(APDU apdu) {
    // access authentication
    if (!pin.isValidate())
        ISOException.throwIt(ISO.SW_PIN_REQUIRED);

    short byteRead = (short)(apdu.setIncomingAndReceive());

    // perform the decryption
    RSApublicKey.cryptoUpdate(apduBuffer, (short)APDU_DATA, (short)128,
        apduBuffer, (short)(APDU_DATA));

    // Send results
    apdu.setOutgoing();
    // indicate the number of bytes in the data field
    apdu.setOutgoingLength((short)128);
    apdu.sendBytesLong(apduBuffer, (short)APDU_DATA, (short)128);
}

private void RunVerPay(APDU apdu) {
    // access authentication
    if (!pin.isValidate())
        ISOException.throwIt(ISO.SW_PIN_REQUIRED);

    short byteRead = (short)(apdu.setIncomingAndReceive());

    // perform the decryption
    RSAPrivateKey.cryptoUpdate(apduBuffer, (short)(APDU_DATA),
        (short)128, apduBuffer, (short)(APDU_DATA));

    // Send results
    apdu.setOutgoing();
    // indicate the number of bytes in the data field
    apdu.setOutgoingLength((short)128);
    apdu.sendBytesLong(apduBuffer, (short)APDU_DATA, (short)128);
}
6. E-payment Application in SET

We want to here to show how the developed program work at the SET. Here we would like to introduce simplified SET. In improved protocol there aren’t issue. We combined the issue company and bank. It is more simple previous SET protocol, and the goal by using this mechanism is to achieve multiple purposes. By using this improved mechanism the smartcard can be used independently with no certification, it can be used within just stored value. Also, it provide some certification and authentication through the bank. Of course there needed dual signature for hiding order information or payment information.

The main implementation is like following:

<table>
<thead>
<tr>
<th>Cardholder Registration</th>
<th>Request to Verify the Certificate</th>
</tr>
</thead>
<tbody>
<tr>
<td>Client</td>
<td>CA</td>
</tr>
<tr>
<td>- Certificate Request to CA</td>
<td>- Response from Card Holder</td>
</tr>
<tr>
<td>Read from Card reader (Authentication)</td>
<td>- Request to Issue</td>
</tr>
<tr>
<td>Purchase Request (simultaneously)</td>
<td>Merchant Registration</td>
</tr>
<tr>
<td>- to Issuer to get the certificate</td>
<td>- Authorization Request (to Acquirer)</td>
</tr>
<tr>
<td>- to Merchant</td>
<td>- response to Merchant</td>
</tr>
<tr>
<td>Design the protocol using DES-3,RSA,etc.</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Purchase Response to Client</th>
<th>Verify Card Holder information</th>
</tr>
</thead>
<tbody>
<tr>
<td>Merchant</td>
<td>ISSUE</td>
</tr>
<tr>
<td>Authorization Request to gateway</td>
<td>- send confirm to CA</td>
</tr>
<tr>
<td>Capture Request to gateway</td>
<td>Authorization process</td>
</tr>
<tr>
<td>Response processing</td>
<td>- response to Gateway</td>
</tr>
<tr>
<td>- to Client, Payment, CA</td>
<td>Capture process</td>
</tr>
<tr>
<td></td>
<td>- clearing request from gateway</td>
</tr>
<tr>
<td></td>
<td>- bill to client</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Authorization process</th>
<th>Merchant Authorization process</th>
</tr>
</thead>
<tbody>
<tr>
<td>PG</td>
<td>Acquirer</td>
</tr>
<tr>
<td>- request to Issue</td>
<td>- response to CA</td>
</tr>
<tr>
<td>- response to Merchant</td>
<td></td>
</tr>
<tr>
<td>Capture process</td>
<td>- response to CA</td>
</tr>
</tbody>
</table>
7. Conclusion and further more

We have introduced here the overview of E-payment system and main protocol:

E-payment system is the new payment methods with the emergence of electronic commerce on the Internet. Secure payment systems are critical to the success of EC. There are four essential security requirements for safe electronic payments. (Authentication, Encryption, Integrity, Non-repudiation) The key security schemes adopted for electronic payment systems are encryption. Security schemes are adopted in protocols like SSL and SET.

Secure Sockets Layer, a protocol developed by Netscape for transmitting private documents via the Internet. SSL works by using a private key to encrypt data that is transferred over the SSL connection. Both Netscape Navigator and Internet Explorer support SSL, and many Web sites use the protocol to obtain confidential user information, such as credit card numbers. SSL is an open, nonproprietary protocol. It has been submitted to the W3 Consortium (W3C) working group on security for consideration as a standard security approach for World Wide Web browsers and servers on the Internet. The primary goal of the SSL Protocol is to provide privacy and reliability between two communicating applications. The protocol is composed of two layers. At the lowest level, layered on top of some reliable transport protocol (e.g., TCP), is the SSL Record Protocol. The SSL Record Protocol is used for encapsulation of various higher level protocols.

SET is the Secure Electronic Transaction protocol developed by Visa and MasterCard specifically for enabling secure credit card transactions on the Internet. Like SSL, SET allows for the merchant's identity to be authenticated via digital certificates. However, SET also allows for the merchant to request users authenticate themselves through digital certificates. This makes it much more difficult for someone to use a stolen credit card. A further advantage of SET is that the merchant has no access to credit card numbers and thus another source of fraud is eliminated.

SET is exempt from the US cryptographic export restrictions and unlike SSL can therefore use strong, 128 bit encryption for credit card numbers world-wide. With SET, the bank card company gets involved in the middle of the transaction, essentially acting as middleman. Theoretically, when you enter your card card in an online SET transaction, the commerce site might never see your actual credit card number. Instead, that info would go to the financial institution, who would verify the card and the amount, then make payment to the commerce site, all in exchange for a fee, of course.

C-SET stands for "Chip-Secure Electronic Transaction" which means "system for providing secure electronic payment transactions over the Internet using Smart-Cards". SET is a protocol which was devised by MasterCard, VISA and various US parties with a strong vested interest in the Internet, in order to secure payments made by cards over the Internet. Customers do not physically use their cards; instead, they make use of a pre-allocated "SET certificate". This certificate is recorded on the hard drive of a customer's computer, or on a diskette. C-SET is a protocol, but primarily an architecture, defined by Groupement des Cartes Bancaires. The C-SET protocol is an adaptation of the SET protocol, enabling the integration of a physical level of security (something which SET itself does not provide), using Smart-Cards, secure Smart-Card readers, and "Electronic Commerce Black Boxes".
And we also have introduced the interoperability like SET/ C-SET, SSL/SET and Others method like algorithm, biometric.

Our project focused on the SET, so we introduced SET protocol, more deeply, and suggest more improved SET protocol. In improved protocol there aren’t issue. We combined the issue company and bank. It is more simple previous SET protocol, and the goal by using this mechanism is to achieve multiple purposes. By using this improved mechanism the smart-card can be used independently with no certification, it can be used within just stored value. Also, it provides some certification and authentication through the bank. Of course there needed dual signature for hiding order information or payment information.

Lastly we implemented with Java-Card encryption algorithm so as to apply it to the improved SET. The source code that we have developed in appendix.

In this project we have leaned following:

. Understanding Smart card and SET protocol using Java-card
. Learning about encryption and decryption techniques of Java-card
. Understanding E-payment system using encryption and decryption of order and signature information between each entity under the assumption of server-client network based on SET protocol.

If further more studying is asked to us, we want to program various encryption techniques, and to design more improved model based on SET.
Appendix (Source code)

1. Server.java

// Set up a Server that will receive packets from a
// client and send packets to a client.
import java.io.*;
import java.net.*;
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
import javacard.framework.*;
import javacardx.framework.*;
import javacardx.crypto.*;

public class Server extends JFrame {
    private JTextArea display;
    private DatagramPacket sendPacket, receivePacket;
    private DatagramSocket socket;

    public Server() {
        super( "Server" );
        display = new JTextArea();
        getContentPane().add( new JScrollPane( display),
            BorderLayout.CENTER );
        setSize( 400, 300 );
        show();

        try {
            socket = new DatagramSocket( 5000 );
            display = new JTextArea();
            getContentPane().add( new JScrollPane( display),
                BorderLayout.CENTER );
            setSize( 400, 300 );
            show();

            try {
                socket = new DatagramSocket( 5000 );
            }
            catch( SocketException se ) {
                se.printStackTrace();
                System.exit( 1 );
            }
        }
    }
}
public void waitForPackets()
{
    while ( true ) {
        try {
            // set up packet
            byte data[] = new byte[ 1000 ];
            receivePacket =
                new DatagramPacket( data, data.length );

            // wait for packet
            socket.receive( receivePacket );

            // process packet
            display.append( "nPacket received:" +
                "nFrom host: " + receivePacket.getAddress() +
                "nHost port: " + receivePacket.getPort() +
                "nLength: " + receivePacket.getLength() +
                "nContaining: init\n" );
            //new String( receivePacket.getData(), 0,
            // receivePacket.getLength() ) );
            byte[] receiveData = receivePacket.getData();
            for(int i=0; i < receivePacket.getLength(); i++){
                display.append(Integer.toHexString(receiveData[i] & 0xff));
                if(i%20 == 19) display.append("n");
            }
            // echo information from packet back to client
            display.append("nEcho data to client...\n");
            sendPacket =
                new DatagramPacket( receivePacket.getData(),
                    receivePacket.getLength(),
                    receivePacket.getAddress(),
                    receivePacket.getPort() );
            socket.send( sendPacket );
            display.append("nPacket sent\n");
            display.setCaretPosition( display.getText().length() );
        } catch( IOException io ) {
            display.append( io.toString() + "n");
            io.printStackTrace();
        }
    }
}

public static void main( String args[] )
{
    Server app = new Server();
    app.addWindowListener(
        new WindowAdapter() {
            public void windowClosing( WindowEvent e )
            {
                System.exit( 0 );
            }
        });
    app.waitForPackets();
}
2. **CryptoTest.java**

```java
// Author    : Nevil le Pattinson
// Copyright : Schlumberger APC, Austin, Texas.
// email     : Neville.Pattinson@slb.com
// Date      : October 21st 1999
// Version   : 2.1
// Applet for exercising DES, Triple DES and RSA in Cyberflex Access applets.
// Should be used in conjunction with CryptoTest.fcn in APDU Manager.
// Triple DES, DES Enc/dec only supported on cards with GetData return byte 9 = 0x0C.
// Tests 2,4,5,6,8 all fail on cards with GetData return byte 9 = 0x0B.
// "Requires a PIN of 12345678 to be presented before any test will run"
// Test1 is a DES Encryption using a Byte array Key cast to a DES Key.
// Test2 is a DES Encryption/Decryption using a Byte array Key cast to a DES Key.
// Test3 is a DES Encryption using DES Key (01) in file 0011.
// Test4 is a DES Encryption/Decryption using DES Key (01) in file 0011.
// Test5 is a triple DES Encryption/Decryption using a Byte array Key cast to a DES3 Key.
// Test6 is a triple DES MAC/VerifyMAC using a Byte array Key cast to a DES3 Key.
// Test7 is a DES MAC/VerifyMAC on 24 bytes using a Byte array Key cast to a DES Key.
// Test8 is a DES encryption/Decryption in CBC mode of a 24 byte message.
// Test9 is a SHA digest from XX bytes to 20. Where XX is in the range 1 to 256
// Test10 is a 1024bit RSA Private Key Encryption operation on message input
// Test11 is a 1024bit RSA Public Key Decryption on cryptogram

import javacard.framework.*;
import javacardx.framework.*;
import javacardx.crypto.*;

public class CryptoTest extends javacard.framework.Applet {

    /* constants declaration */
    // code of CLA byte in the command APDU header
    final byte CryptoTest_CLA = (byte) 0x03;

    // Codes of INS byte in the command APDU header
    // N.B. Odd values are not allowed under ISO spec.
    final byte Test1 =     (byte) 0x10;
    final byte Test2 =     (byte) 0x20;
    final byte Test3 =     (byte) 0x30;
    final byte Test4 =     (byte) 0x33;
    final byte Test5 =     (byte) 0x50;
    final byte TDMACTest = (byte) 0x52;
    final byte DMACTest =  (byte) 0x54;
    final byte DEncTEst =  (byte) 0x56;
    final byte SHATe st =   (byte) 0x58;
    final byte RSAEnc =    (byte) 0x32;
    final byte RSADec =    (byte) 0x34;
    final byte SETTest =   (byte) 0x36;
    final byte KeyEnc =    (byte) 0x38;
    final byte SendCert =  (byte) 0x3a;
    final byte VerDual =   (byte) 0x3b;
    final byte VerPay  =    (byte) 0x3c;
    final byte DecSym  =   (byte) 0x3d;
    final byte PinCheck =  (byte) 0x40;
    final byte CLA_F0 =    (byte) 0xF0;

    final byte PinTryLimit = (byte) 0x00; // maximum number of incorrect tries before the PIN is blocked
    final byte MaxPinSize = (byte) 0x04; // maximum size of PIN allowed

    final short SW_WRONG_PIN = (short) 0x6BBB; // Error return if PIN not correctly presented
    final short VERIFY_BAD =   (short) 0x6CCC; // Error return for MAC or Signature Verify failure
    final short APDUDATA =     (short) ISO.OFFSET_CDATA; // Define apdu buffet offset for data.

    //Private Key definition
```
(byte)0xC0, (byte)0x04, (byte)0x00,
(byte)0x01, (byte)0x00, (byte)0x01,
];

//PIN Key definition
static byte[] pinString = //Note this is the PIN
{ (byte)0x12, (byte)0x34, (byte)0x56, (byte)0x78
};

//DES Key definition
static byte[] singleDESKey = //Note this is the DES key definition
{ (byte)0x38, (byte)0x12, (byte)0xA4, (byte)0x19, (byte)0xC6, (byte)0xE7, (byte)0x71
};

//3DES Key definition
static byte[] tripleDESKey = / /Note this is the triple DES key definition
{ (byte)0x38, (byte)0x12, (byte)0xA4, (byte)0x19, (byte)0xC6, (byte)0x71, (byte)0x71,
(byte)0xAD, (byte)0x9F, (byte)0x61, (byte)0xFE, (byte)0xFA, (byte)0x20, (byte)0xCE, (byte)0x63
};

static byte[] sendDataBuf = new byte[180];
/* instance variables declaration */
byte apduBuffer[];   // APDU buffer
byte sha1buf[];     // sha1 output buffer
byte ICV[];      // Initial Chaining Vector for CBC operations DES
byte ICV3[];     // Initial Chaining Vector for CBC operations 3DES
OwnerPIN pin;    // Pin Object
DES_Key deskey;    // DES key object from byte array DES
DES_Key deskey2;   // DES Key object from byte array DES
DES3_Key des3key;   // Make a new triple DES object
// Sha1MessageDigest sha;  // SHA1 object (Sha1MessageDigest broken)
MessageDigest sha;   // Alternative SHA1 object
RSA_CRT_PrivateKey RSAPrivateKey;//Private Key object
RSA_PublicKey RSAPublicKey;      // Public Key object
private CryptoTest() {
    pin = new OwnerPIN(PinTryLimit, MaxPinSize);// Initialise the PIN data
    pin.updateAndUnblock(pinString, (short)0, (byte)4);// Load and make the PIN active.

    // Initialise objects
    sha1buf = new byte[20];   // Result of SHA operations go here
    ICV = new byte[8];   // ICV for DES is an 8 byte array
    ICV3 = new byte[16];  // ICV for 3DES must be declared at 16 bytes (bug in card)
    deskey = new DES_Key(); // Make a new Des Key object
    deskey2 = new DES_Key((short)1); // Reference DES Key in file 0011
    des3key = new DES3_Key();   // Make a new triple DES object
    sha = new Sha1MessageDigest(); // instantiate sha object
    RSAPublicKey = new RSA_PublicKey((short)1024);  // make a new RSA public key object
    RSAPrivateKey = new RSA_CRT_PrivateKey((short)1024);//allocate private key

    //Set the Private key in manageable chunks as we cannot setkey of 338 all at once due to ram limitations.
    RSAPrivateKey.setKey(rsaPrivateKey, (short)0, (short)0, (short)70);
    RSAPrivateKey.setKey(rsaPrivateKey, (short)70, (short)70, (short)67);
    RSAPrivateKey.setKey(rsaPrivateKey, (short)137, (short)137, (short)67);
    RSAPrivateKey.setKey(rsaPrivateKey, (short)204, (short)204, (short)67);
    RSAPrivateKey.setKey(rsaPrivateKey, (short)271, (short)271, (short)67);

    //Set the Public key in manageable chunks
    RSAPublicKey.setKey(rsaPublicKey, (short)0, (short)0, (short)6);
    RSAPublicKey.setKey(rsaPublicKey, (short)6, (short)6, (short)128);
    RSAPublicKey.setKey(rsaPublicKey, (short)134, (short)134, (short)6);
    register();
} // end of the constructor

// Install method
public static void install(APDU apdu){
public boolean select() {
    // reset validation flag in the PIN object to false
    pin.reset();
    // returns true to JCRE to indicate that the applet
    // is ready to accept incoming APDUs.
    return true;
} // end of select method

public void process(APDU apdu) {
    // APDU object carries a byte array (apduBuffer) to
    // transfer incoming and outgoing APDU header
    // and data bytes between card and CAD
    apduBuffer = apdu.getBuffer();

    // Filter out the initial Select APDU for CLA=00 card
    if ((apduBuffer[ISO.OFFSET_CLA] == ISO.CLA_ISO) &&
        (apduBuffer[ISO.OFFSET_INS] == ISO.INS_SELECT)) {
        ISOException.throwIt(ISO.SW_NO_ERROR);
    }

    // Filter out the initial Select APDU for CLA=F0 card
    if ((apduBuffer[ISO.OFFSET_CLA] == CLA_F0) &&
        (apduBuffer[ISO.OFFSET_INS] == ISO.INS_SELECT)) {
        ISOException.throwIt(ISO.SW_NO_ERROR);
    }

    switch (apduBuffer[ISO.OFFSET_INS]) {
        case Test1:     RunTest1(apdu);     return;
        case Test2:     RunTest2(apdu);     return;
        case Test3:     RunTest3(apdu);     return;
        case Test4:     RunTest4(apdu);     return;
        case Test5:     RunTest5(apdu);     return;
        case TDMACTest: RunTest6(apdu);     return;
        case DMACTest:  RunTest7(apdu);     return;
        case DEncTTest: RunTest8(apdu);     return;
        case SHA1Test:  RunTest9(apdu);     return;
        case RSAEnc:    RunTest10(apdu);    return;
        case RSADec:    RunTest11(apdu);    return;
        case PinCheck:  CheckThePin(apdu);  return;
        case SETTest:   RunSETTest(apdu);   return;
        case KeyEnc:    RunKeyEnc(apdu);    return;
        case SendCert:  RunSendCert(apdu);  return;
        case VerDual:   RunVerDual(apdu);   return;
        case VerPay:    RunVerPay(apdu);    return;
        case DecSym:    RunDecSym(apdu);    return;
        default:    ISOException.throwIt (ISO.SW_INS_NOT_SUPPORTED);
        } // end of process method
    
    // CheckThePin method
    // Takes an input of 4 bytes and checks to see if it is the actual 'mypin' set above
    private void CheckThePin(APDU apdu) {
        short byteRead = (short)(apdu.setIncomingAndReceive());
        if (byteRead != 4)
            ISOException.throwIt(ISO.SW_WRONG_LENGTH);

        if ((apdu.getBuffer().get((short)APDUDATA, (byte)4) == false))
            ISOException.throwIt(ISO.SW_WRONG_PIN);
    } // end of CheckThePin method

    // RunTest1 method
    // Takes an input 8 byte message from the incoming APDU and encrypts it using a DES key.
    // Response APDU sends the original message (8 bytes)plus the encrypted block(8 bytes)
    private void RunTest1(APDU apdu) {
/* Security and E-payment System */
Confidential

/* Security and E-payment System */

Term Project Final Report
2001 Spring Semester

// access authentication
if ( ! pin.isValidate() )
    ISOException.throwIt (ISO.SW_PIN_REQUIRED);

short byteRead = (short)(apdu.setIncomingAndReceive());
if (byteRead != 8)
    ISOException.throwIt(ISO.SW_WRONG_LENGTH);

// DES encrypt
// Original message (0-7); Encrypted message (8-15)
deskey.setKey(singleDESKey, (short)0);
deskey.encryptECB(apduBuffer, (short)APDUDATA, (short)8, apduBuffer, (short)(APDUDATA+8));

// DES decrypt/encrypt test
// Original message (0-7); Encrypted message (8-15); Decrypted message (16-23)
deskey.setKey(singleDESKey, (short)0);
deskey.decryptECB(apduBuffer, (short)(APDUDATA), (short)8, apduBuffer, (short)(APDUDATA+8));
deskey.encryptECB(apduBuffer, (short)(APDUDATA), (short)8, apduBuffer, (short)(APDUDATA+8));

// Send results
apdu.setOutgoing();
// indicate the number of bytes in the data field
apdu.setOutgoingLength((short)16);
// at offset 0 send 16 byte of data in the buffer
apdu.sendBytesLong(apduBuffer, (short)APDUDATA, (short)16);

} // end of RunTest1 method

// RunTest2 method
// Takes an input 8 byte message from the incoming APDU and encrypts it using
// a DES Key. It then decrypts it using the same DES Key.
// Response APDU sends the original message (8 bytes)plus the encrypted block(8 bytes) plus the decrypted Message (8 bytes)
private void RunTest2(APDU apdu) {

    // access authentication
    if ( ! pin.isValidate() )
        ISOException.throwIt (ISO.SW_PIN_REQUIRED);

    short byteRead = (short)(apdu.setIncomingAndReceive());
    if (byteRead != 8)
        ISOException.throwIt(ISO.SW_WRONG_LENGTH);

    // DES encrypt/decrypt test
    // Original message (0-7); Encrypted message (8-15); Decrypted message (16-23)
deskey.setKey(singleDESKey, (short)0);
deskey.encryptECB(apduBuffer, (short)APDUDATA, (short)8, apduBuffer, (short)(APDUDATA+8));
deskey.decryptECB(apduBuffer, (short)(APDUDATA), (short)8, apduBuffer, (short)(APDUDATA+8));

deskey2.encryptECB(apduBuffer, (short)(APDUDATA), (short)8, apduBuffer, (short)(APDUDATA+8));

    // Send results
    apdu.setOutgoing();
    // indicate the number of bytes in the data field
    apdu.setOutgoingLength((short)24);
    // at offset 0 send 24 byte of data in the buffer
    apdu.sendBytesLong(apduBuffer, (short)APDUDATA, (short)24);
} // end of RunTest2 method

// RunTest3 method
// Takes an input 8 byte message from the incoming APDU and encrypts it using
// DES Key 01 in file 0011. It then decrypts it using the same DES Key.
// Response APDU sends the original message (8 bytes)plus the encrypted block(8 bytes)
private void RunTest3(APDU apdu) {

    // access authentication
    if ( ! pin.isValidate() )
        ISOException.throwIt (ISO.SW_PIN_REQUIRED);

    short byteRead = (short)(apdu.setIncomingAndReceive());
    if (byteRead != 8)
        ISOException.throwIt(ISO.SW_WRONG_LENGTH);

    // DES encrypt
    // Original message (0-7); Encrypted message (8-15)
deskey2.encryptECB(apduBuffer, (short)(APDUDATA), (short)8, apduBuffer, (short)(APDUDATA+8));
private void RunTest4(APDU apdu) {
    // access authentication
    if (!pin.isValidated())
        ISOException.throwIt(ISO.SW_PIN_REQUIRED);
    short byteRead = (short)(apdu.setIncomingAndReceive());
    if (byteRead != 8)
        ISOException.throwIt(ISO.SW_WRONG_LENGTH);
    // DES encrypt/decrypt test
    // Original message (0-7); Encrypted message (8-15); Decrypted message (16-23)
    deskey2.encryptECB(apduBuffer, (short)APDUDATA, (short)8, apduBuffer, (short)(APDUDATA+8));
    deskey2.decryptECB(apduBuffer, (short)(APDUDATA+8), (short)8, apduBuffer, (short)(APDUDATA+16));
    // Send results
    apdu.setOutgoing();
    // indicate the number of bytes in the data field
    apdu.setOutgoingLength((short)24);
    // at offset 0 send 24 byte of data in the buffer
    apdu.sendBytesLong(apduBuffer, (short)APDUDATA, (short)24);
} // end of RunTest4 method

private void RunTest5(APDU apdu) {
    // access authentication
    if (!pin.isValidated())
        ISOException.throwIt(ISO.SW_PIN_REQUIRED);
    short byteRead = (short)(apdu.setIncomingAndReceive());
    if (byteRead != 8)
        ISOException.throwIt(ISO.SW_WRONG_LENGTH);
    // triple DES encrypt/decrypt test
    // Original message (0-7); Encrypted message (8-15); Decrypted message (16-23)
    des3key.setKey(tripleDESKey, (short)0);
    des3key.encryptECB(apduBuffer, (short)APDUDATA, (short)8, apduBuffer, (short)(APDUDATA+8));
    des3key.decryptECB(apduBuffer, (short)(APDUDATA+8), (short)8, apduBuffer, (short)(APDUDATA+16));
    // Send results
    apdu.setOutgoing();
    // indicate the number of bytes in the data field
    apdu.setOutgoingLength((short)24);
    // at offset 0 send 24 byte of data in the buffer
    apdu.sendBytesLong(apduBuffer, (short)APDUDATA, (short)24);
} // end of RunTest5 method

private void RunTest6(APDU apdu) {
    // access authentication
    if (!pin.isValidated())
        ISOException.throwIt(ISO.SW_PIN_REQUIRED);
    short byteRead = (short)(apdu.setIncomingAndReceive());
    if (byteRead != 8)
        ISOException.throwIt(ISO.SW_WRONG_LENGTH);
    // triple DES encrypt/decrypt test
    // Original message (0-7); Encrypted message (8-15); Decrypted message (16-23)
    des3key.setKey(tripleDESKey, (short)0);
    des3key.encryptECB(apduBuffer, (short)APDUDATA, (short)8, apduBuffer, (short)(APDUDATA+8));
    des3key.decryptECB(apduBuffer, (short)(APDUDATA+8), (short)8, apduBuffer, (short)(APDUDATA+16));
    // Send results
    apdu.setOutgoing();
    // indicate the number of bytes in the data field
    apdu.setOutgoingLength((short)24);
    // at offset 0 send 24 byte of data in the buffer
    apdu.sendBytesLong(apduBuffer, (short)APDUDATA, (short)24);
} // end of RunTest6 method

private void RunTest7(APDU apdu) {
    // access authentication
    if (!pin.isValidated())
        ISOException.throwIt(ISO.SW_PIN_REQUIRED);
    short byteRead = (short)(apdu.setIncomingAndReceive());
    if (byteRead != 8)
        ISOException.throwIt(ISO.SW_WRONG_LENGTH);
    // triple DES encrypt/decrypt test
    // Original message (0-7); Encrypted message (8-15); Decrypted message (16-23)
    des3key.setKey(tripleDESKey, (short)0);
    des3key.encryptECB(apduBuffer, (short)APDUDATA, (short)8, apduBuffer, (short)(APDUDATA+8));
    des3key.decryptECB(apduBuffer, (short)(APDUDATA+8), (short)8, apduBuffer, (short)(APDUDATA+16));
    // Send results
    apdu.setOutgoing();
    // indicate the number of bytes in the data field
    apdu.setOutgoingLength((short)24);
    // at offset 0 send 24 byte of data in the buffer
    apdu.sendBytesLong(apduBuffer, (short)APDUDATA, (short)24);
} // end of RunTest7 method

private void RunTest8(APDU apdu) {
    // access authentication
    if (!pin.isValidated())
        ISOException.throwIt(ISO.SW_PIN_REQUIRED);
    short byteRead = (short)(apdu.setIncomingAndReceive());
    if (byteRead != 8)
        ISOException.throwIt(ISO.SW_WRONG_LENGTH);
    // triple DES encrypt/decrypt test
    // Original message (0-7); Encrypted message (8-15); Decrypted message (16-23)
    des3key.setKey(tripleDESKey, (short)0);
    des3key.encryptECB(apduBuffer, (short)APDUDATA, (short)8, apduBuffer, (short)(APDUDATA+8));
    des3key.decryptECB(apduBuffer, (short)(APDUDATA+8), (short)8, apduBuffer, (short)(APDUDATA+16));
    // Send results
    apdu.setOutgoing();
    // indicate the number of bytes in the data field
    apdu.setOutgoingLength((short)24);
    // at offset 0 send 24 byte of data in the buffer
    apdu.sendBytesLong(apduBuffer, (short)APDUDATA, (short)24);
} // end of RunTest8 method

private void RunTest9(APDU apdu) {
    // access authentication
    if (!pin.isValidated())
        ISOException.throwIt(ISO.SW_PIN_REQUIRED);
    short byteRead = (short)(apdu.setIncomingAndReceive());
    if (byteRead != 8)
        ISOException.throwIt(ISO.SW_WRONG_LENGTH);
    // triple DES encrypt/decrypt test
    // Original message (0-7); Encrypted message (8-15); Decrypted message (16-23)
    des3key.setKey(tripleDESKey, (short)0);
    des3key.encryptECB(apduBuffer, (short)APDUDATA, (short)8, apduBuffer, (short)(APDUDATA+8));
    des3key.decryptECB(apduBuffer, (short)(APDUDATA+8), (short)8, apduBuffer, (short)(APDUDATA+16));
    // Send results
    apdu.setOutgoing();
    // indicate the number of bytes in the data field
    apdu.setOutgoingLength((short)24);
    // at offset 0 send 24 byte of data in the buffer
    apdu.sendBytesLong(apduBuffer, (short)APDUDATA, (short)24);
} // end of RunTest9 method

private void RunTest10(APDU apdu) {
    // access authentication
    if (!pin.isValidated())
        ISOException.throwIt(ISO.SW_PIN_REQUIRED);
    short byteRead = (short)(apdu.setIncomingAndReceive());
    if (byteRead != 8)
        ISOException.throwIt(ISO.SW_WRONG_LENGTH);
    // triple DES encrypt/decrypt test
    // Original message (0-7); Encrypted message (8-15); Decrypted message (16-23)
    des3key.setKey(tripleDESKey, (short)0);
    des3key.encryptECB(apduBuffer, (short)APDUDATA, (short)8, apduBuffer, (short)(APDUDATA+8));
    des3key.decryptECB(apduBuffer, (short)(APDUDATA+8), (short)8, apduBuffer, (short)(APDUDATA+16));
    // Send results
    apdu.setOutgoing();
    // indicate the number of bytes in the data field
    apdu.setOutgoingLength((short)24);
    // at offset 0 send 24 byte of data in the buffer
    apdu.sendBytesLong(apduBuffer, (short)APDUDATA, (short)24);
} // end of RunTest10 method
if ( ! pin.isValidated() )
    throwIt(ISO.SW_PIN_REQUIRED);

    short byteRead = (short)apdu.setIncomingAndReceive();
if (byteRead != 24)
    ISOException.throwIt(ISO.SW_WRONG_LENGTH);

des3key.setKey(tripleDESKey, (short)0);
des3key.setICV(ICV3, (short)0); // Allocate the ICV location (8 or 16 bytes actually used!)
    des3key.clearICV(); // Make the ICV for CBC operations = 0
    apdu.waitExtension();
    des3key.generateMAC(apduBuffer, APDUDATA, (short)24, apduBuffer, (short)(APDUDATA+24), (short)8);
    apdu.waitExtension();
    if (des3key.verifyMAC(apduBuffer, (short)(APDUDATA+24), (short)8, apduBuffer, (short)(APDUDATA), (short)24))
    {
        apduBuffer[APDUDATA+32] = 1; // Mac verified
    }
else
    {
        apduBuffer[APDUDATA+32] = 0; // MAC failed
    }

    // Send results
    apdu.setOutgoing();
    // indicate the number of bytes in the data field
    apdu.setOutgoingLength((short)9);
    // at offset 0 send 9 byte of data in the buffer
    apdu.sendBytesLong(apduBuffer, APDUDATA+24, 9);
} // end of RunTest6 method

// RunTest7 method
// Takes an input 24 byte message from the incoming APDU and MACs it using
// a DES Key. It then Verifys the MAC using the same DES Key.
// 9000 for good and 6CCC for bad
private void RunTest7(APDU apdu) {

    // access authentication
    if ( ! pin.isValidated() )
        throwIt(ISO.SW_PIN_REQUIRED);

short byteRead = (short)apdu.setIncomingAndReceive();
if (byteRead != 24)
    ISOException.throwIt(ISO.SW_WRONG_LENGTH);

deskey.setKey(singleDESKey, (short)0); deskey.setICV(ICV, (short)0); // Allocate the ICV location
    deskey.clearICV(); // Make the ICV for CBC operations = 0
    apdu.waitExtension();
    deskey.generateMAC(apduBuffer, APDUDATA, (short)24, apduBuffer, (short)(APDUDATA+24), (short)8);
    apdu.waitExtension();
    boolean result = deskey.verifyMAC(apduBuffer, (short)(APDUDATA+24), (short)8, apduBuffer, (short)(APDUDATA), (short)24); // MAC failed

    if (result == false)
        ISOException.throwIt(VERIFY_BAD); // MAC failed
} // end of RunTest7 method

// RunTest8 method
// Takes an input 24 byte message from the incoming APDU and Encrypts it using
// a DES Key. It then Decrypts the the cryptogram using the same DES Key.
// Response APDU sends the message (24 bytes) plus Cryptogram (24 bytes) plus decrypted message (24 bytes)
private void RunTest8(APDU apdu) {

    // access authentication
    if ( ! pin.isValidated() )
        throwIt(ISO.SW_PIN_REQUIRED);

short byteRead = (short)apdu.setIncomingAndReceive();
if (byteRead != 24)
    ISOException.throwIt(ISO.SW_WRONG_LENGTH);
Security and E-payment System
Confidential
Term Project Final Report
2001 Spring Semester

```
deskey.setKey(singleDESKey, (short)0);
deskey.setICV(ICV, (short)0); // Allocate the ICV location
deskey.clearICV(); // Make the ICV for CBC operations = 0
apdu.waItExtension();
deskey.encryptCBC(apduBuffer, (short)APDUDATA, (short)24, apduBuffer, (short)(APDUDATA+24));
apdu.waItExtension();
deskey.decryptCBC(apduBuffer, (short)(APDUDATA+24), (short)24, apduBuffer, (short)(APDUDATA+48));
apdu.waItExtension();

// Send results
apdu.setOutgoing();
// indicate the number of bytes in the data field
apdu.setOutgoingLength((short)72);
// at offset 0 send 72 byte of data in the buffer
apdu.sendBytesLong(apduBuffer, (short)APDUDATA, (short)72);
} // end of RunTest8 method

// RunTest9 method
// Takes an input message from the incoming APDU and SHA digests it.
// It then sends the 20 byte digest back.
// Response APDU sends the digest (20 bytes)
private void RunTest9(APDU apdu) {

    // access authentication
    if (!pin.isValidated())
    ISOException.throwIt (ISO.SW_PIN_REQUIRED);
    short byteRead = (short)(apdu.setIncomingAndReceive());

    sha.generateDigest(apduBuffer, (short)APDUDATA, (short)byteRead, sha1buf, (short)0);
    //sha.sha1Block(mybuf, (short)0, (byte)0x80, sha1buf, (short)0);//broken in card!

    // Send results
    apdu.setOutgoing();
    // indicate the number of bytes in the data field
    apdu.setOutgoingLength((short)20);
    // at offset 0 send 20 byte of data in the buffer
    apdu.sendBytesLong(sha1buf, (short)0, (short)20);
} // end of RunTest9 method

// RunTest10 method
// Takes an input message from the incoming APDU and encrypts it using an RSA Private Key.
// It then sends the encryption back.
// Response APDU sends the encrypted result
private void RunTest10(APDU apdu) {

    // access authentication
    if (!pin.isValidated())
    ISOException.throwIt (ISO.SW_PIN_REQUIRED);
    short byteRead = (short)(apdu.setIncomingAndReceive());

    // perform the encryption
    RSAPrivateKey.cryptUpdate(apduBuffer, (short)APDUDATA, (short)byteRead, apduBuffer, (short)APDUDATA);

    // Send results
    apdu.setOutgoing();
    // indicate the number of bytes in the data field
    apdu.setOutgoingLength((short)128);
    // at offset 0 send 128 byte of data in the buffer
    apdu.sendBytesLong(apduBuffer, (short)APDUDATA, (short)128);
} // end of RunTest10 method

// RunTest11 method
// Takes an input message from the incoming APDU
// and decrypts it using an RSA Public Key.
// It then sends the result back.
private void RunTest11(APDU apdu) {

    // access authentication
```

© "Enigma": Cho Won Chul, Kim Hee Dae, Lee Jung Hwun, Yoon Won Jung
if (!pin.isValidated())
    ISOException.throwIt(ISO.SW_PIN_REQUIRED);

    short byteRead = (short)(apdu.setIncomingAndReceive());
    // perform the decryption
    RSAPublicKey.cryptoUpdate(apduBuffer, (short)APDUDATA, (short)byteRead, apduBuffer, (short)APDUDATA);

    // Send results
    apdu.setOutgoing();
    // indicate the number of bytes in the data field
    apdu.setOutgoingLength((short)128);
    // at offset 0 send x byte of data in the buffer
    apdu.sendBytesLong(apduBuffer, (short)APDUDATA, (short)128);

} // end of RunTest11 method

// RunSETTest method
// Takes an input message from the incoming APDU
// and make digital signature and concatenate to other information.
// It then sends the result back.
private void RunSETTest(APDU apdu) {

    // access authentication
    if (!pin.isValidated())
        ISOException.throwIt(ISO.SW_PIN_REQUIRED);

    // read apdu buffer
    short byteRead = (short)(apdu.setIncomingAndReceive());

    // make new allocation digest message
    // hash order information to 20 bytes digest
    sha.generateDigest(apduBuffer, (short)APDUDATA, (short)30, apduBuffer, (short)APDUDATA + 30);
    // hash payment information to 20 byte digest
    sha.generateDigest(apduBuffer, (short)(APDUDATA + 30), (short)30, apduBuffer, (short)(APDUDATA + 20));

    // make new allocation on digest message buffer
    sha.generateDigest(apduBuffer, (short)APDUDATA, (short)40, apduBuffer, (short)(APDUDATA + 40));

    // perform the encryption
    RSAPrivateKey.cryptoUpdate(apduBuffer, (short)(APDUDATA + 40), (short)20, apduBuffer, (short)(APDUDATA + 40));

    // Send results
    apdu.setOutgoing();
    // indicate the number of bytes in the data field
    apdu.setOutgoingLength((short)168);
    apdu.sendBytesLong(apduBuffer, (short)APDUDATA, (short)168);

} // end of RunTest11 method

private void RunKeyEnc(APDU apdu) {

    // access authentication
    if (!pin.isValidated())
        ISOException.throwIt(ISO.SW_PIN_REQUIRED);

    // read apdu buffer
    short byteRead = (short)(apdu.setIncomingAndReceive());
// Takes Payment Information and encrypts it using triple DES Key.
des3key.setKey(tripleDESKey, (short)0);
des3key.encryptECB(apduBuffer, (short)APDUDATA, (short)32, apduBuffer, (short)APDUDATA);
short i;
for(i = 0; i < (short)16; i++)
apduBuffer[(short)APDUDATA + (short)32 + i] = (byte)i;//tripleDESKey[i];
// for(i < (short)20; i++)
// apduBuffer[(short)APDUDATA + (short)32 + i] = (byte)i;
// encrypt Symmetric information and Cardholder's account Information
// by using payment gateway's public key.
RSAPublicKey.cryptoUpdate(apduBuffer, (short)(APDUDATA + 32), (short)16, apduBuffer, (short)(APDUDATA + 32));

// Send results
apdu.setOutgoing();
// indicate the number of bytes in the data field
apdu.setOutgoingLength((short)160);
apdu.sendBytesLong(apduBuffer, (short)APDUDATA, (short)160);
}

private void RunSendCert(APDU apdu) {
  // access authentication
  if ( ! pin.isValidated() )
    ISOException.throwIt (ISO.SW_PIN_REQUIRED);
  // read apdu buffer
  short i;
  for(i = 0; i < (short)140; i++)
apduBuffer[(short)APDUDATA + i] = rsaPublicKey[i];
  // Send results
  apdu.setOutgoing();
  // indicate the number of bytes in the data field
  apdu.setOutgoingLength((short)140);
apdu.sendBytesLong(apduBuffer, (short)APDUDATA, (short)140);
}

private void RunVerDual(APDU apdu) {
  // access authentication
  if ( ! pin.isValidated() )
    ISOException.throwIt (ISO.SW_PIN_REQUIRED);
  short byteRead = (short)(apdu.setIncomingAndReceive());
  // perform the decryption
  RSAPublicKey.cryptoUpdate(apduBuffer, (short)APDUDATA, (short)128, apduBuffer, (short)(APDUDATA));
  // Send results
  apdu.setOutgoing();
  // indicate the number of bytes in the data field
  apdu.setOutgoingLength((short)128);
apdu.sendBytesLong(apduBuffer, (short)APDUDATA, (short)128);
}

private void RunVerPay(APDU apdu) {
  // access authentication
  if ( ! pin.isValidated() )
    ISOException.throwIt (ISO.SW_PIN_REQUIRED);
  short byteRead = (short)(apdu.setIncomingAndReceive());
  // perform the decryption
RSAPrivateKey.getCryptoflatMap(apdu, (short)(APDUDATA), (short)(128), apduBuffer, (short)(APDUDATA));

// Send results
apdu.setOutgoing();
// indicate the number of bytes in the data field
apdu.setOutgoingLength((short)(128));
apdu.sendBytesLong(apduBuffer, (short)(APDUDATA), (short)(128));
}

private void RunDecSym(APDU apdu)
{
// access authentication
if ( ! pin.isValidated() )
    ISOException.throwIt(ISO.SW_PIN_REQUIRED);
short byteRead = (short)(apdu.setIncomingAndReceive());
    des3key.setKey(apduBuffer, (short)(APDUDATA));
    des3key.decryptECB(apduBuffer, (short)(APDUDATA+16), (short)(32), apduBuffer, (short)(APDUDATA));
}

} // end of class CryptoTest

3. Client.java

// Set up a Client that will send packets to a
// server and receive packets from a server.
import java.io.*;
import java.net.*;
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
import java.applet.*;
import java.lang.String;
import slb.iop.*;
import javacard.framework.*;
import javacardx.framework.*;
import javacardx.crypto.*;

public class Client extends JFrame implements ActionListener {
    private JTextField oI, pI;
    private JLabel loI, lpI, lpin;
    private JTextArea display;
    private JButton  connect, disconnect, enter, verdual, verpay;
    private JPasswordField pinField;
    private DatagramPacket sendPacket, receivePacket;
    private DatagramSocket socket;
    private byte receiveData[];
    SmartCard sCard = new SmartCard();
    IOP siOP = new IOP();

    public Client()
    {
        super( "Client" );
        Container c = getContentPane();
c.setLayout( new FlowLayout() );
loI = new JLabel("Order Information");
c.add(loI);
oI = new JTextField(30);
c.add(oI);
lpI = new JLabel("Payment Information");
c.add(lpI);
pl = new JTextField(30);
c.add(pl);
lpin = new JLabel("PIN");
c.add(lpin);
pinField = new JPasswordField(10);
c.add(pinField);

ButtonHandler handler = new ButtonHandler();
connect = new JButton("Connect");
c.add(connect);
connect.addActionListener(handler);
disconnect = new JButton("Disconnect");
c.add(disconnect);
disconnect.addActionListener(handler);
enter = new JButton("Enter");
c.add(enter);
enter.addActionListener(handler);
verdual = new JButton("VerDual");
c.add(verdual);
verdual.addActionListener(handler);

public void waitForPackets() {
    while (true) {
        try {
            // set up packet
            byte data[] = new byte[1000];
            receivePacket = new DatagramPacket(data, data.length);

            // wait for packet
            socket.receive(receivePacket);

            // process packet
            display.append("Packet received:
From host: "+receivePacket.getAddress()+
HOST port: "+receivePacket.getPort()+
Length: "+receivePacket.getLength()+
Containing:

t new String( receivePacket.getData(), 0, receivePacket.getLength() ) );
            receiveData = new byte[ receivePacket.getData() ];
            receiveData = receivePacket.getData();
            for (int i = 0; i < receivePacket.getLength(); i++) {
                display.append( Integer.toHexString( (i & 0xff) ) );
            }
        } catch (SocketException se) {
            se.printStackTrace();
            System.exit(1);
        }
    }
}
public static void main( String args[] )
{
    Client app = new Client();
    app.addWindowListener( new WindowAdapter() {
        public void windowClosing( WindowEvent e )
        {
            System.exit( 0 );
        }
    } );
    app.waitForPackets();
}

private class ButtonHandler implements ActionListener { public void actionPerformed( ActionEvent e ) {
    String s = "";
    if(e.getSource() == connect) {
        s = "Connect " + e.getActionCommand();
        display.append(s);
        setConnect();
    }
    else if(e.getSource() == disconnect) {
        connect();
        s = "Disconnect " + e.getActionCommand();
    }
    else if(e.getSource() == enter) {
        try {
            display.append( e.getActionCommand() + " 
" );
            String s = e.getActionCommand();
            byte data[] = s.getBytes();
            sendPacket = new DatagramPacket( data, data.length,
                    InetAddress.getLocalHost(), 5000 );
            socket.send( sendPacket );
            display.append( "Packet sent\n" );
            display.setCaretPosition( display.getText().length() );
        }
        catch ( IOException exception ) {
            display.append( exception.toString() + " \n" );
            exception.printStackTrace();
        }
        catch( IOException exception ) {
            display.append( exception.toString() + " \n" );
            exception.printStackTrace();
        }
    }
}
s = "Enter " + e.getActionCommand();
    setEnter();
} else if(e.getSource() == verdual) {
    s = "VerDual " + e.getActionCommand();
    verDual();
} else if(e.getSource() == verpay) {
    s = "VerPay " + e.getActionCommand();
    verPayment();
}

private void verDual(){
    short retvalue[];
    byte CLA,INS,P1,P2;
    int iArray[];
    String s;
    // Digital Signature
    try {
        CLA = (byte)0x03;
        INS = (byte)0x34;
        P1  = (byte)0;
        P2  = (byte)0;
        display.append("\n Decription Dual Digital Signature\n");
        int i;
        iArray = new int[128];

        for(i=0;i<128;i++){
            iArray[i] = receiveData[180+i] & 0xff;
            display.append(Integer.toHexString(iArray[i]));
            if(i%20 == 19) display.append("\n");
        }
        display.setCaretPosition(display.getText().length() );
        retvalue = new short[128];
        retvalue = sCard.SendCardAPDU(CLA,INS,P1,P2,iArray,128); //1);

        int iErrorCode = sCard.GetLastErrorCode();
        if (iErrorCode != 0x9000) {
            s=sCard.GetErrorMessage();
            display.append("SendCardAPDU: " + s + "\n");
            System.out.println(iErrorCode);
        } else {
            display.append("\n Data Good\n");
            if (retvalue != null) {
                System.out.println(retvalue.length);
                for(i=0;i<retvalue.length;i++){
                    display.append(Integer.toHexString((int)retvalue[i]));
                    if(i%20 == 19) display.append("\n");
                }
                display.setCaretPosition(display.getText().length() );
            }
        }
    }
}
} catch (slbException b) {s = b.getMessage();
    display.append("Connect: " + s + "
");}

// Order Information & Payment Information Hash
try {
    CLA = (byte)0x03;
    INS = (byte)0x58;
    P1  = (byte)0;
    P2  = (byte)0;

    display.append("\nOrder Info & Payment Info Hash\n");
    int i;
    iArray = new int[40];

    for(i=0;i<40;i++) {
        iArray[i] = receiveData[140+i] & 0xff;
        display.append(integer.toHexString(iArray[i]));
        if(i%20 == 19) display.append("\n");
    }

    retvalue = new short[20];
    retvalue = sCard.SendCardAPDU(CLA,INS,P1,P2,iArray,20); //1);
    int iErrorCode = sCard.GetLastErrorCode();
    if (iErrorCode != 0x9000) {
        s = sCard.GetErrorMessage();
        display.append("SendCardAPDU: " + s + "\n");
        System.out.println(iErrorCode);
    } else {
        display.append("\nHash Data Good\n");
        if (retvalue != null) {
            System.out.println(retvalue.length);

            for(i=0;i<retvalue.length;i++) {
                display.append(integer.toHexString((int)retvalue[i]));
                if(i%20 == 19) display.append("\n");
            }
        }
        display.setCaretPosition(display.getText().length());
    }
}

} catch (slbException b) {s = b.getMessage();
    display.append("Connect: " + s + "\n");}

private void verPayment(){
    short retvalue[];
    byte CLA,INS,P1,P2;
    int iArray[];
    String s;

    // Get RSA
    try {
        CLA = (byte)0x03;
        INS = (byte)0x32;
        P1  = (byte)0;
        P2  = (byte)0;

        display.append("\nRSA Private Key Decryption of 3DES Key\n");
    }
}
```java
int i;
int[] iArray = new int[128];
for(i=0;i<128;i++) {
    byte[] Array = receiveData[339+i] & 0xff;
display.append(Integer.toHexString(Array[i]));
    if(i%20 == 19) display.append("\n");
}
display.setCaretPosition(
    display.getText().length() );
short[] retvalue = new short[128];
retvalue = sCard.SendCardAPDU(CLA,INS,P1,P2,iArray,128);
int iErrorCode = sCard.GetLastError();
if (iErrorCode != 0x9000)
{
    s=sCard.GetErrorMessage();
display.append("SendCardAPDU: " + s + \n");
    System.out.println(iErrorCode);
} else
{
    display.append("Data Good\n");
    if (retvalue != null)
    {
        System.out.println(retvalue.length);
        display.append("Return Data: \n");
        for(i=0;i<128;i++)
        { display.append(Integer.toHexString((int)retvalue[i]));
            if(i%20 == 19) display.append("\n");
        }
display.append("\n");
    }
}
catch (slbException b) {s=b.getMessage();
display.append("Connect: + s+ \n");}
}
```

```java
private void setEnter(){
    short[] retvalue = new short[140];
    byte CLA,INS,P1,P2;
    int iArray[];
    byte bArray[];
    byte dataArray[] = new byte[468];
    String OrderString, PaymentString;
    String PINString, s;
    PINString = new String(pinField.getPassword());
    // Get Order
    try {
        CLA = (byte)0x03;
        INS = (byte)0x3a;
        P1  = (byte)0;
        P2  = (byte)0;
        display.append("Get Card Holder's Public Key" + \n");
        int i;
        iArray = new int[0];
        retvalue = new short[140];
        retvalue = sCard.SendCardAPDU(CLA,INS,P1,P2,iArray,140);
        int iErrorCode = sCard.GetLastError();
        if (iErrorCode != 0x9000)
        {
            s=sCard.GetErrorMessage();
display.append("SendCardAPDU: " + s + \n");
```

57

© "Enigma": Cho Won Chul, Kim Hee Dae, Lee Jung Hwun, Yoon Won Jung
System.out.println(iErrorCode);

try{
    display.append("Data Good\n");
    if (retvalue != null)
    {
        byte sBytes[] = new byte[retvalue.length];
        System.out.println(retvalue.length);
        for (i = 0; i < retvalue.length; ++i)
        {
            sBytes[i] = (byte)(retvalue[i] & 0xff);
            display.append(Integer.toHexString((int)retvalue[i]));
            dataArray[i] = sBytes[i];
            if(i%20 == 19) display.append("\n");
        }
    }
    display.setCaretPosition(display.getText().length());
}

} catch (slbException b) {s = b.getMessage();
    display.append("Connect: " + s+ "\n");}

//= try dual signature
try {
    CLA = (byte)0x03;
    INS = (byte)0x36;
    P1  = (byte)0;
    P2  = (byte)0;
    display.append("Dual Signature" + "\n");
    int i;
    iArray = new int[60];
    retvalue = new short[168];
    retvalue = new short[168];
    OrderString = oI.getText();
    int length = OrderString.length();
    if(length == 0)
    {
        display.append("Enter Oder Information\n");
        return;
    }
    iArray = new int[60];
    //byte bArray[];
    bArray = OrderString.getBytes();
    if(length < 30)
    {
        for(i=0;i<length;i++)
            iArray[i] = (int) bArray[i] & 0xff;
        if(i<30;i++)
            iArray[i] = 0;
    }
    else
    {
        for(i=0;i<30;i++)
            iArray[i] = (int) bArray[i] & 0xff;
    }
PaymentString = pl.getText();
length = PaymentString.length();

if(length == 0)
{
    display.append("Enter Payment Information\n");
    return;
}

bArray = PaymentString.getBytes();

if(length < 30)
{
    for(i=0;i<length;i++)
    { 
        iArray[30+i] = (int) bArray[i] & 0xff;
        for(;i<30;i++)
        { 
            iArray[30+i] = 0;
        }
    }
}
else
{
    for(i=0;i<30;i++)
    { 
        iArray[30+i] = (int) bArray[i] & 0xff;
    }
}

retvalue = sCard.SendCardAPDU(CLA,INS,P1,P2,iArray,168);

int iErrorCode = sCard.GetLastError();
if (iErrorCode != 0x9000)
{
    s = sCard.GetErrorMessage();
    display.append("SendCardAPDU: " + s + \n");
    System.out.println(iErrorCode);
}
else
{
    display.append("Data Good\n");
    if (retvalue != null)
    {
        byte sBytes[] = new byte[retvalue.length];
        System.out.println(retvalue.length);
        for (i = 0; i < retvalue.length; i++)
        {
            sBytes[i] = (byte)(retvalue[i] & 0xff);
            dataArray[i+140] = sBytes[i];
            display.append(Integer.toHexString((int)retvalue[i]));
            if(i%20 == 19) display.append("\n");
        }
    
    display.setCaretPosition(
                display.getText().length() );

    }
}

} catch (sibException b) {s = b.getMessage();
    display.append("Connect: " + s+ \n");}

// Key Encryption
try
{
    CLA = (byte)0x03;
    INS = (byte)0x38;
    P1 = (byte)0;
    P2 = (byte)0;

display.append("Key Encryption" + \n");
    int i;
    // Payment Information
    iArray = null;

retvalue = null;
int iArray = new int[30];
retvalue = new short[160];
for (i = 0; i < 30; i++) iArray[i] = 0xaa;
retvalue = sCard.SendCardAPDU(CLA, INS, P1, P2, iArray, 160);

int iErrorCode = sCard.GetLastErrorCode();
if (iErrorCode != 0x9000)
{
    s = sCard.GetErrorMessage();
    display.append("SendCardAPDU: " + s + \n");
    System.out.println(iErrorCode);
}
else
{
    display.append("Data Good\n");
    if (retvalue != null)
    {
        byte sBytes[] = new byte[retvalue.length];
        System.out.println(retvalue.length);
        for (i = 0; i < retvalue.length; i++)
        {
            sBytes[i] = (byte)(retvalue[i] & 0xff);
            dataArray[i + 140 + 168] = sBytes[i];
            display.append(Integer.toHexString((int)retvalue[i]));
            //System.out.print(i + ":" + sBytes[i] + " ");
        }
    }
    display.setCaretPosition(display.getText().length());
}
}

catch (slbException b) {s = b.getMessage();
    display.append("Connect: " + s + \n");
}

try {
    display.append("\nSending packet\n");
    //String s = e.getActionCommand();
    //byte data[] = s.getBytes();
    sendPacket = new DatagramPacket(dataArray, dataArray.length,
        InetAddress.getLocalHost(), 5000); 
    socket.send(sendPacket);
    display.append("Packet sent\n");
    display.setCaretPosition(display.getText().length());
}
catch (IOException exception) {
    display.append(exception.toString() + \n");
    exception.printStackTrace();
}

int iAID[] = {0x11,0x11,0x11,0x11,0x11,0x11};
private void setConnect() {
    short retvalue[];
    byte CLA, INS, P1, P2;
    int iArray[];
    byte bAArray[];

    String PINString, s;
    PINString = new String(pinField.getPassword());
if (connect())
{
    iArray = new int[4];
    try
    {
        CLA = (byte)0x03;
        INS = (byte)0x40;
        P1  = (byte)0;
        P2  = (byte)0;
        int length = PINString.length();
        if (length != 8)
            display.append("Length must be 8.\n");
        else
        {
            display.append("String PIN + PINString + \" n");
            for (int i = 0; i < 4; i++)
            {
                s = PINString.substring(i*2,(i*2)+2);
                iArray[i] = Integer.valueOf(s, 16).intValue();
                System.out.println(iArray[i]);
            }
            display.append("Validating PIN...\n");
            sCard.SendCardAPDU(CLA,INS,P1,P2,iArray,0);
            int iErrorCode = sCard.GetLastErrorCode();
            if (iErrorCode != 0x9000)
            {
                if (iErrorCode == 0x6BBB)
                    display.append("Wrong PIN\n");
                else
                {
                    s = sCard.GetErrorMessage();
                    display.append("SendCardAPDU: \" + s + \" n");
                    System.out.println(iErrorCode);
                }
            }
            else
            {
                display.append("PIN Validated...\n");
            }
        }
    }
    catch (slbException b) { s = b.getMessage();
        display.append("Connect: " + s+ \" n");
    }
}

public boolean connect() {
    boolean bRet;
    String s;
    int numReaders;
    String rs[] = null;
    rs = sIOP.ListReaders();
    if (!sIOP.Connect(sCard, rs[0],false)) {
        display.append("Unable to connect to card.\n");
        return false;
    }
    try
    {
        display.append("Connected. Selecting root\n");
        bRet = sCard.SelectCardlet(iAID);
        if (!bRet) {
            s = sCard.GetErrorMessage();
        }
    }
    catch (slbException b) { s = b.getMessage();
        display.append("Connect: " + s+ \" n");
    }
display.append("SelectCardlet: " + s + ", n");
        return false;
    } else {
        display.append("Cardlet selected!", n");
    }
}

    catch (slbException b)
    {
        s = b.getMessage();
        display.append("Connect: " + s + ", n");
    }

    return true;
}
Reference

1. Cyberflex Access Software Developer’s Kit Guide
3. A developer’s Guide to Schumberger Smart Card Middlware
7. Schlumberger User Discussion Forums http://www.flexfrum.com
8. Smart Card Application Development Using Java, Uwe Hansman, Martin S. Nicklous, Thomas Schack, Frank Seliger
10. Java Card technology for smart cards: architecture and programmer's guide, Chen Zhiqun, Addison-Wesley, 2000
11. http://ktwww.kotel.co.kr/smart/