

Design of Smart Card Enabled Protocols for Micro-Payment and Rapid Application Development Builder for E-Commerce

By

TSANG Hin Chung

Supervised By

Professor K. S. LEUNG

Professor K. H. LEE

A Thesis Submitted in Partial Fulfillment
of the Requirements for the Degree of
Master of Philosophy
in
Computer Science and Engineering

©The Chinese University of Hong Kong

August, 2001

The Chinese University of Hong Kong holds the copyright of this thesis. Any person(s) intending to use a part or the whole of the materials in this thesis in a proposed publication must seek copyright release from the Dean of the Graduate School.



概要

隨著電子商務迅速發展，傳統的安全認證和網上交易的安全措施已不能滿足現今越來越嚴峻的互聯網環境。傳統的電子商務系統的可信性和可靠性都受到先進的逆向工程技術和容易攻擊的作業系統威脅。因為黑客會透過應用邏輯襲擊用戶的電腦。

透過適當地運用防干擾的智能卡所產生的系統是不易受到攻擊。然而，許多已發表的智能卡系統仍然遇到不同的困難，如過份依賴「對稱鑰匙加密」技術。更糟的是，有些系統不可以在智能卡執行，或因不適當的設計而需要作進一步的改善。

本論文有幾項目的。首先是編寫一個有條理的智能系統、認證及交易協定。透過分析其他智能卡的系統工具，找出它們的不足之處，例如忽視互相認證等。本論文亦會提出嶄新的認證及交易協定，以補足其他智能卡的缺點，提高智能卡的效率。

本論文所提出的全面的認證協定不但支援近端認證協定、密碼確認或人體特徵認證，更支援獨特的遠端認證協定。除此之外，認證協定還可以執行相互認證及抵抗認證重複的攻擊。在認證協定完成之後，「安全三重資料加密標準鑰匙」會建立於用戶端和伺服器，使兩者能夠互相溝通。這項新穎的協定也能處理智能卡及伺服器的失效問題。至於，建基於認證協定的交易協定，本論文亦會作出介紹。本論文的交易協能締造一個可靠的、有效率的交易，以及使終斷的交易重新執行。

以往，很多發行者都沒有驗證交易協定的正確性，有見及此，本論文將會深入分析交易協定，以驗證它的正確性。而且，本論文亦會詳細介紹協定的實驗評估。

為使智能卡系統容易建立起來，在論文中也提出了一項為 B2C 與 P2P 電子商務軟件開發的革新方案，Smart RAD。這由「應用程式界面」及「公用建立工具」所組成。這個方案可幫助應用程式開發員按需要設計認證和交易協定。在功能方面，這個方案提供一個穩健的認證和可靠的交易、可描述性、原始化設定、個人化、公開密碼匙基礎建設。另外，在論文中，亦會將這項計劃與其他智能卡開發的工具作比較。

最後，透過 P2P 與 B2C 的應用，Multi-Max 程式，去示範及解釋 Smart RAD 的用法和功能。Multi-Max 程式容許用戶與商戶或其他用戶進行網上買賣。

Abstract

As a consequence of the burgeoning e-commerce industry, the conventional security measures for authentication and online transaction can no longer fulfill the current stringent Internet environment. The reliability and the credibility of the traditional e-commerce systems are at stake due to the advance of the reverse-engineering technology and the vulnerability of the operating system because the application logic residing in the client machine is susceptible to intrusion.

With the appropriate deployment of tamper-resistant smart cards, the resulting system is less likely to be attacked easily. Nevertheless, most of the published authentication protocols on smart card based system suffer from different tricky problems such as heavy dependence on symmetric key cryptography. Even worse, some of them are not feasible to be implemented on the ordinary smart cards, or need further enhancement due to inappropriate design.

This thesis has several objectives. It fashions a coherent study on the state-of-the-art of smart card systems, authentication and transaction protocols. By analyzing the related work, we identify potential critical defects such as the ignorance of mutual authentication. As a remedy, we propose efficient smart card based authentication and transaction protocols.

The proposed comprehensive scheme not only supports local authentication protocol, either password-based or biometric-based, but also supports unique remote authentication protocol for secure remote access. It performs

mutual authentication and resists replaying attack. A secret session key is also established between both client and server sides after the completion of the authentication protocol by the silent key distribution scheme. The session key can be used for subsequent communications. The devised protocol can handle both the smart card and server failures well. The transaction protocol for the micro-payment built on top of the authentication protocol is described. This protocol enables reliable transaction, efficient and faultless resumption from broken transactions.

An in-depth analysis based on BAN logic, which is not done in other similar work, to prove the correctness of the proposed authentication protocol is described. In addition, the experimental evaluation results of the proposed protocols are detailed to show the efficiency and the effectiveness of the protocols.

Lastly, to make ease of the development of smart card based system, we have designed and implemented an innovative builder, *Smart RAD*, for B2C & P2P e-commerce software development. The implementation is comprised of an application-programming interface and utility builder to facilitate the application developers to customize their own authentication and different services, say the transaction service. Functionally, it provides strong authentication, reliable transaction, accountability, personalization, customization and PKI systems. A comparison with other smart card development kits would be shown.

As our last contribution, a P2P and B2C application, *Multi-MAX*, is implemented to successfully demonstrate the feasibility and the flexibility of Smart RAD. It allows the user to purchase a product from a retail shop on the web or trade with other users through an application based interface.

Acknowledgments

This thesis would not have been accomplished without the guidance, technical and mental supports of my supervisors, fellow researchers and friends. I owe my warmest thanks to my supervisors Kwong-Sak Leung and Kin-Hong Lee.

I greatly appreciate Ng Man Lung and Lin Yuen Wun s' helps for designing a nice graphical user interfaces for this project. It is my pleasure to work with them.

Contents

1	Introduction	1
1.1	Authentication and Transaction Protocol	2
1.2	E-Commerce Enabler	3
2	Literature Review	4
2.1	Cryptographic Preliminaries	4
2.1.1	One-Way Hash Function	4
2.1.2	Triple DES	5
2.1.3	RSA	7
2.1.4	Elliptic Curve	8
2.2	Smart Cards	8
2.2.1	Smart Card Operating Systems	11
2.2.2	Java Card	12
2.3	Authentication Protocol	14
2.3.1	Properties	15
2.3.2	Survey	16
2.4	Transaction Protocol	19
2.5	BAN Logic	20
2.5.1	Notation	20
2.5.2	Logical Postulates	22
2.5.3	Protocol Analysis	25

3	Authentication Protocol	26
3.1	Formulation of Problem	26
3.2	The New Idea	27
3.3	Assumptions	29
3.4	Trust Model	29
3.5	Protocol	30
3.5.1	Registration	30
3.5.2	Local Authentication	31
3.5.3	Remote Authentication	33
3.5.4	Silent Key Distribution Scheme	35
3.5.5	Advantages	37
3.6	BAN Logic Analysis	38
3.7	Experimental Evaluation	43
3.7.1	Configuration	44
3.7.2	Performance Analysis	45
4	Transaction Protocol	51
4.1	Assumptions	52
4.2	Protocol	55
4.3	Conflict Resolution Policy	58
4.4	Justifications	58
4.5	Experimental Evaluation	59
4.5.1	Configuration	59
4.5.2	Performance Analysis	60
5	E-Commerce Builder	65
5.1	Overview	66
5.2	Design of Smart RAD	68
5.2.1	Mechanism	68
5.2.2	Java Card Layer	69

5.2.3	Host Layer	71
5.2.4	Server Layer	72
5.3	Implementation	73
5.3.1	Implementation Reflection	73
5.3.2	Implementation Issues	76
5.4	Evaluation	77
5.5	An Application Example: Multi-MAX	79
5.5.1	System Model	79
5.5.2	Design Issues	80
5.5.3	Implementation Issues	80
5.5.4	Evaluation	84
5.6	Future Work	89
6	Conclusion	91
A	Detail Experimental Result	93
A.1	Authentication Time Measurement	94
A.2	On-Card and Off-Card Computation Time in Authentication . .	95
A.3	Authentication Time with Different Servers	96
A.4	Transaction Time Measurement	97
A.5	On-card and Off-card Computation Time in Transaction	97
B	UML Diagram	99
B.1	Package cuhk.cse.demo.applet	99
B.2	Package cuhk.cse.demo.client	105
B.3	Package server	110
C	Glossary and Abbreviation	115
	Bibliography	118

List of Figures

2.1	Simple view of DES	5
2.2	Simple view of 3DES	7
2.3	Class hierarchy of smart card	9
2.4	Example of physical architecture of smart card	10
2.5	Format of Command APDU and Response APDU	11
2.6	Java Card Technology Architecture	13
2.7	General Payment Model	19
3.1	System Diagram of the Authentication System	28
3.2	Registration	32
3.3	Local authentication protocol	34
3.4	Remote authentication protocol	36
3.5	Testing environment	44
3.6	Authentication time with different hosts	46
3.7	On-card and off-card computation time in authentication	47
3.8	Authentication time with different servers	49
3.9	Authentication time with different servers	50
4.1	Testing environment	53
4.2	Class of Transaction	53
4.3	Flow diagram of the transaction protocol	54
4.4	Transaction time with different hosts	60
4.5	The On-card and the off-card computation time in the transaction	62

5.1	System Diagram of Smart RAD	69
5.2	UML diagram of the objects in Java Card layer	71
5.3	UML diagram of the objects in Host layer	73
5.4	Utility Services	74
5.5	System Model of Multi-MAX	80
5.6	UML diagram of class Install	81
5.7	UML diagram of class RunTradeServer and RunTradeServer- Thread	82
5.8	UML diagram of class CardUtility and ClientBase	83
5.9	UML diagram of class RunTimeServerThread and RunCAServer- Thread	83
5.10	Sample JNLP file	84
5.11	Snapshot of initialization form	85
5.12	Snapshot of web-based shopping portal	86
5.13	Snapshot of authentication form	86
5.14	Snapshot of banking form	87
5.15	Snapshot of transaction form	88
5.16	Snapshot of transaction log form	88
B.1	UML diagram of UFOApplet	99
B.2	UML diagram of Hash	100
B.3	UML diagram of HashGenerator	100
B.4	UML diagram of Record	101
B.5	UML diagram of SmartPurse	102
B.6	UML diagram of Transaction	103
B.7	UML diagram of TransactionManager	104
B.8	UML diagram of ClientBase	105
B.9	UML diagram of CardUtility	106
B.10	UML diagram of Install	107

B.11 UML diagram of BankRecord 107

B.12 UML diagram of Transaction 108

B.13 UML diagram of BigHash 108

B.14 UML diagram of TimeUtility 109

B.15 UML diagram of CAUtility 109

B.16 UML diagram of CustomCertAndKeyGen 109

B.17 UML diagram of TradePanel 110

B.18 UML diagram of CAPanel 110

B.19 UML diagram of TimePanel 111

B.20 UML diagram of ServerFrame 111

B.21 UML diagram of RunCAServer 112

B.22 UML diagram of RunCAServerThread 112

B.23 UML diagram of RunTimeServer 112

B.24 UML diagram of RunTimeServerThread 113

B.25 UML diagram of RunTradeServer 113

B.26 UML diagram of RunTradeServerThread 114

B.27 UML diagram of StopServer 114

List of Tables

2.1	Meaning of APDU header and trailer	11
2.2	Advantages and Disadvantages of one-factor authentication pro- tocol	14
2.3	Advantages and Disadvantages of two-factor authentication pro- tocol	15
2.4	Advantages and Disadvantages of biometric-based authentica- tion protocol	15
2.5	Characteristics of different remote authentication protocols . . .	18
3.1	Notations used in the proof	38
3.2	Statistical result of the authentication time with different hosts	46
3.3	Statistical result of the on-card and off-card authentication time	48
3.4	Statistical result of authentication time with different servers . .	49
3.5	Cases to be studied	50
4.1	Statistical result of the transaction time with different hosts . .	61
4.2	Statistical result of the on-card and the off-card computation time in the transaction	62
5.1	Brief description of smart card development products	78
5.2	Comparison of Smart RAD with other products	78
A.1	Authentication Time Measurement	94
A.2	On-card and off-card computation time in authentication	95

A.3 Authentication time measurement with different servers 96

A.4 Transaction time measurement 97

A.5 On-card and off-card computation time in transaction 98

Chapter 1

Introduction

Reverse engineering, Trojan horses and physical intrusion are the tricky issues in the security of the e-commerce software in recent years. With the new discovery on the vulnerability of the operating system, intruders are able to attack a remote host.

Usually, the application logic of e-commerce systems resides in the client host and the security of the client host is always undermined. With the rapid enhancement of the reverse engineering technology, hostile users can extract the high-level representation of the application and the data flow within the application logic. By understanding the application logic, hostile users can re-engineer the application logic and bypass the system security measure such that hostile users can perform restricted operations illegally or at the worst case, interrupt the regular operation of the application server.

In addition, the remote attack with Trojan horses is one of the common scenario in cyber attacks. It is not necessary for the intruder to have physical contact with the targeted computer. Instead, Trojan horses are installed in the targeted computer by other means in advance. The intruder can remotely monitor the user's action, read and write processes in the targeted computer. Hence, the application logic in the targeted computer can be attacked easily.

To cope with these serious problems, the application logic should be installed in a tamper-proof device such that no unauthorized local and remote

access is allowed without the knowledge of the computer owner. The solution is *Smart Cards*.

Tamper-resistant feature prevents the smart card from being attacked easily. The cost for hacking an advanced smart card is unreasonably high. Moreover, with the memory and processing chip, the smart card can store and manipulate the sensitive information without leaking out. The smart card can communicate with other parties secretly with the help of the cryptographic co-processor. These factors contribute to the trend that the smart card is very popular in e-commerce applications.

1.1 Authentication and Transaction Protocol

The smart card is particularly useful in the authentication and the transaction processes. The smart card helps to identify the valid card holder and performs secure transactions.

Today, most of the smart card based authentication protocols ignore the mutual authentication and perform only user authentication. Also, some of the published works suffer from critical problems that make them not realistic to be used practically. More importantly, some of them are not allowed to be implemented on different types of smart cards.

As a matter of fact, the biometric-based authentication system can protect the system for local access. However, it may not be true for remote access. Several studies [58] [30] [40] have shown that biometric-based systems cannot simply apply to remote access systems.

These concerns motivate us to design a new local and remote authentication protocol, which can be used in biometric-based or password-based local/remote access systems. The proposed protocol is designed to eliminate the critical defects in existing protocols.

1.2 E-Commerce Enabler

Development of smart card based applications is not trivial. It involves the understanding of smart card operating systems, underlying system of card terminal, cryptographic theory and its related development libraries. In addition, the time to familiarize oneself with the usage of smart card and smart card reader is considerably long compared with other application developments.

It drives us to propose an innovative solution for smart card enabled software developments particular to B2C and P2P e-commerce softwares, which are designed for speeding up the development time. Unlike [22] [19] [18] [20] [55], it is intended to encapsulate the underlying mechanism of smart cards so that the developer can develop a smart card based system easily without considering the complicated smart card architecture.

Chapter 2

Literature Review

In this chapter, we fashion a coherent study on the state-of-the-art of cryptographic techniques, smart card issues, authentication and transaction issues. The internal mechanism of the one-way hash function, DES, triple DES and elliptic curve will be described. The current issues of authentication and transaction protocols will be examined. Finally, the BAN logic is introduced for the analysis of authentication protocols.

2.1 Cryptographic Preliminaries

In this section, we will overview the most common used cryptographic algorithms and techniques in smart cards.

2.1.1 One-Way Hash Function

One-way hash function is commonly used on message authentication by producing a "fingerprint" of a message. MD5 and SHA1 are examples of well-known hash function. A hash function, H can be generalized in this form:

$h = H(M)$, where M is the input message

Typically, the hash function must have the following properties:

- H can be applied to any block of data M with any size.
- The hash value h always has a fixed length.
- For any given M , $H(M)$ can be computed easily.
- For any hash value h , it is computationally infeasible to find M such that $H(M) = h$. This is called *one-way* property.
- For any given data M , it is computational infeasible to find N such that $H(M) = H(N)$. This is called *weak collision resistance*.
- It is computational infeasible to find any pair (M, N) such that $H(M) = H(N)$. This is called *strong collision resistance*.

2.1.2 Triple DES

Triple DES is the variation of Data Encryption Standard (DES) [57] [64] which is adopted in 1977 by the National Bureau of Standards. In DES, data are encrypted in 64-bit blocks using a 56-bit key. The algorithm transforms a 64-bit input in a series of steps into a 64-bit output. The same steps, with the same key, are used to reverse the encryption. Figure 2.1 shows how DES is performed.

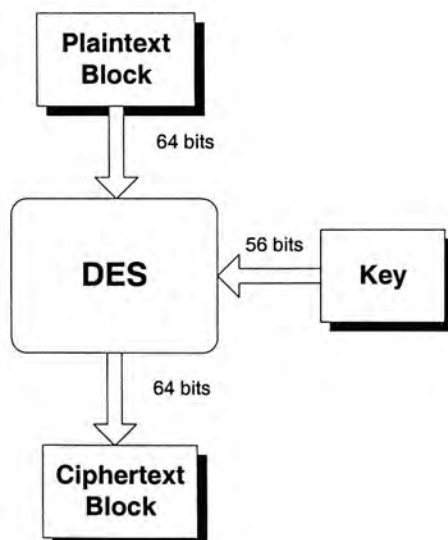


Figure 2.1: Simple view of DES

By and large, the design criteria of DES are:

1. Randomness

The output is changed randomly.

2. Nonlinearity

The encryption function is non-affine to any value of key.

3. Avalanche Property

It is a property of the ordinary encryption algorithm, that a small change of inputs (plaintext or key) results in a significant change in the ciphertext.

4. Correlation Immunity

The output bits are statically independent of any subset of input bits.

5. Completeness Property

Each bit of ciphertext is a complex function of ALL input bits.

However, DES suffers from brute-force attacks, and hence, a substitute for it, triple DES is found to make the best use of the existing DES devices. Triple DES is now adopted for the use in the key management standards of ANS X9.17 and ISO 8732. Figure 2.2 describes the mechanism of triple DES in encrypt-encrypt-encrypt (EEE) and encrypt-decrypt-encrypt (EDE) modes.

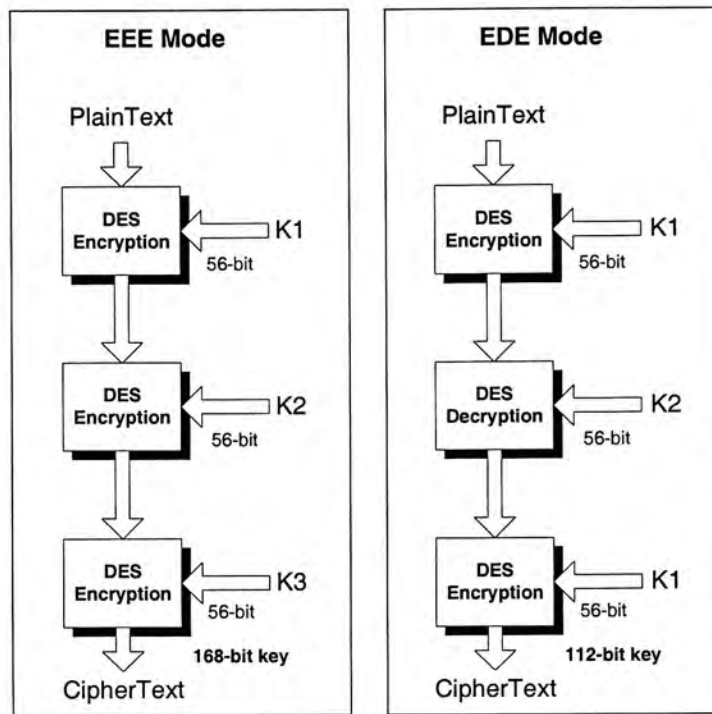


Figure 2.2: Simple view of 3DES

Currently, there is no practical cryptanalytic attack on triple DES. [12] assured that the cost of brute-force key search on triple DES is on the order of 2^{112} and it is estimated that the cost of the differential cryptanalysis grows exponentially, that is exceeding 10^{52} .

2.1.3 RSA

Ron Rivest, Adi Shamir, and Len Adleman at MIT designed the RSA scheme in 1978 [52]. It utilizes exponential functions as its expression and its security is based on the difficulty of factoring the product of two large prime numbers. RSA key pairs, where the public key consists of a modulus m and public exponent e while the private key consists of the same modulus m and a private exponent d .

The two keys are generated from two randomly chosen large prime numbers, p and q . To assure maximum security, the lengths of these numbers should be equal. The modulus m is computed as the product of the two primes:

$$m = p * q$$

Next, an encryption key e is chosen so that e and $(p-1)(q-1)$ are relatively prime. The decryption key d is chosen so that:

$$ed = 1 \pmod{(p-1)(q-1)}$$

$$d = e^{-1} \pmod{(p-1)(q-1)}$$

Let x be the plaintext with the same size as the modulus and y be the ciphertext. Then, the formulas for encryption and decryption are as follows:

$$y = x^e \pmod{m}$$

$$x = y^d \pmod{m}$$

For this algorithm to be satisfactory for public-key encryption, the following requirements must be met:

- It is possible to find the values of e , d , m such that $x^{ed} = x \pmod{m}$ for all $x < m$.
- It is relatively easy to compute x^e and y^d for all values of $x < y$.
- It is infeasible to determine d given e and m .

2.1.4 Elliptic Curve

The algorithm using the Elliptic curve is faster than RSA or DSA. With the same key size, the security level of Elliptic curve is higher. The obvious advantage of the elliptic curve in the implementation is that it does not require cryptographic co-processors and the smaller key size can save valuable memory space in the smart card. However, the elliptic curve is not widely applied in the smart card because RSA and DSA are more compatible with the existing PKI system such as SSL and certification authority.

2.2 Smart Cards

Smart Cards [60] [11] [6] [10] [1] are credit-card sized cards with processors built inside. Roughly speaking, smart cards can be categorized as memory

cards and microprocessor cards. Most of the earlier released smart cards are memory cards without processing power. They are widely used for the storage of data only. A Microprocessor card is a card with limited processing power and memory for the data storage. In general, it can be sub-classified as multi-functional or specific-functional microprocessor cards. Figure 2.3 shows the class hierarchy of smart cards. Nowadays, multi-functional microprocessor cards have become dominant in the market since their release because they allow sensitive data to be stored and manipulated inside the cards without leaking out.

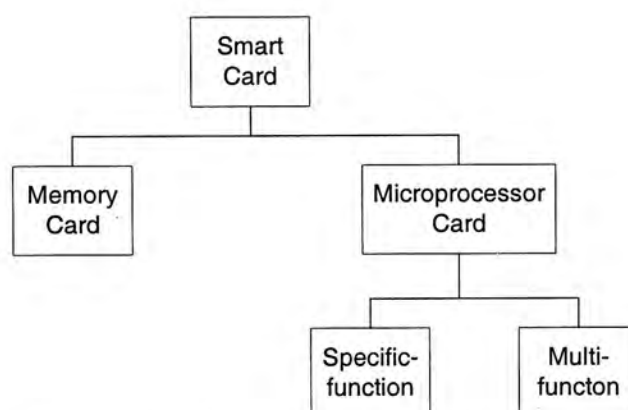


Figure 2.3: Class hierarchy of smart card

At the same time, we can also divide the smart cards into contact or contact-less type. A card that is of contact type needs to be placed in the card acceptance device for use. A card that is of contact-less type must be put within certain distance around the card acceptance device without being placed in a card acceptance device. Still, a contact-less card is more expensive and less common than a card of contact type.

In addition, the smart card is always equipped with the read-only memory (ROM), electrical erasable programmable read-only memory (EEPROM) and random access memory (RAM). In order to perform cryptographic operations, a crypto-coprocessor can be embedded in the card to enhance the performance of the computational intensive operation such as modular arithmetic and large integer calculation. Figure 2.4 shows the physical architecture of the smart

card.

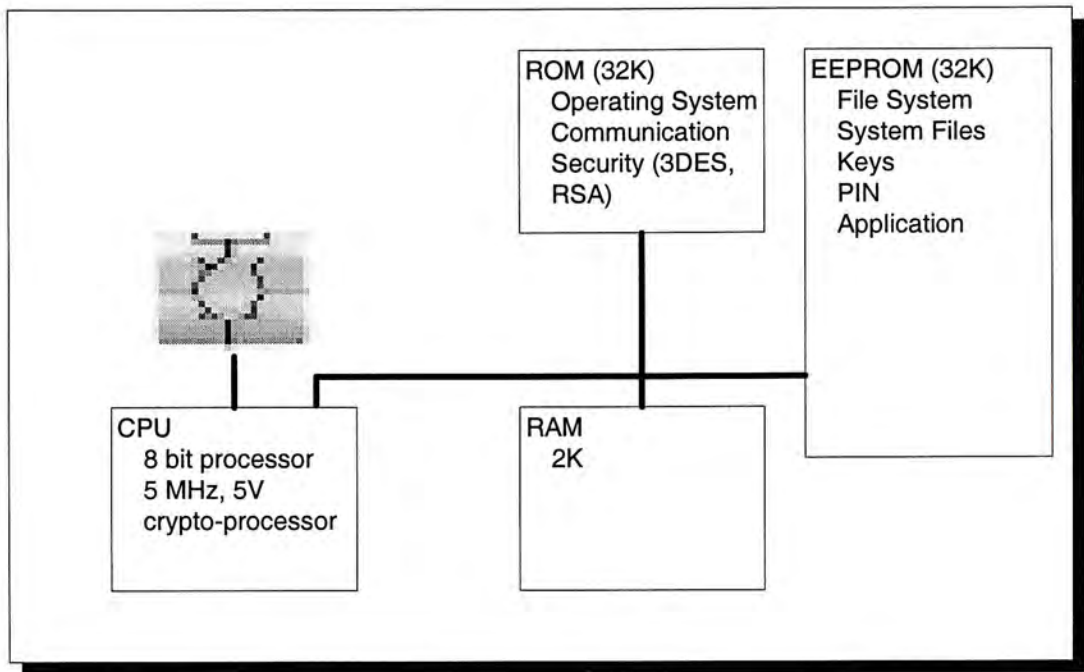


Figure 2.4: Example of physical architecture of smart card

Smart Cards communicate with the attached host according to ISO7816-4 standard, that defines the application protocol between the smart card and host application. The card and the host can send an *application protocol data unit* (APDU) to each other for communications. The APDU sent from the host to the card is called *command APDU* while the APDU sent from the card to the host is called *response APDU*. Figure 2.5 describes the format of the command APDU and response APDU. For the command APDU, *CLA*, *INS*, *P1* and *P2* are its header while the optional data field and *Le* form its body. For the response APDU, the optional data field forms its body while *SW1* and *SW2* form its trailer. Table 2.1 shows the physical meaning of different fields in the APDU.

Other than ISO7816, many standards for the smart cards have been drafted and released including GSM, EMV, Open platform, OpenCard and PC/SC for different purposes and platforms.

Command APDU

Mandatory Header					Optional Body	
CLA	INS	P1	P2	Lc	Optional Data	Le

Response APDU

Optional Body	Trailer	
Optional Data	SW1	SW2

Figure 2.5: Format of Command APDU and Response APDU

Field	Length (byte)	Meaning
CLA	1	identifies the class of an instruction
INS	1	identifies an instruction for a specific applet
P1	1	passes command specific parameters to the command
P2	1	passes command specific parameters to the command
Lc	1	specifies the length of the optional data
Le	1	specifies the length of the optional data in the corresponding response APDU
SW1	1	specifies the the status code
SW2	1	specifies the the meaning of the status code

Table 2.1: Meaning of APDU header and trailer

2.2.1 Smart Card Operating Systems

In general, smart cards can be classified in terms of their operating systems, namely, file system smart cards, Java Cards, Multos, smart cards for windows.

- File System Smart Card
It is composed of the Elementary file (EF), Dedicated file (DF) and Master file (MF). They form a hierarchical file system.

- Multos [45]

The smart card program written in Multos Executable Language (MEL) is run on top of the Application Abstraction Machine (AAM).

- Smart Card for Windows [42]

It is a combination of the traditional ISO 7816-4 compliant operating system and a programmable platform. The core provides a file system, access controls, cryptography services, an API and a selection of ISO commands.

- Java Card

The Java Card platform allows the application to be written in Java language.

2.2.2 Java Card

The Java Card [66] specification enables the on-card program to be written in Java and allows the on-card program to be run on smart cards and other devices with limited memory. Figure 2.6 shows the Java Card technology architecture. Nevertheless, there are many limitations on the Java Card platform as follows:

1. Dynamic class loading is not supported.
2. Garbage collection is not mandatory.
3. Customized security manager is not supported. Security policies are implemented directly.
4. Multiple threads are not supported in JVM.
5. An object cannot be cloned.
6. Package *java.lang* is not fully implemented on Java Card JVM.

- 7. Primitive types such as *double*, *float* and *long* are not available. The keyword *int* is optionally supported.
- 8. Only one dimensional array is supported.

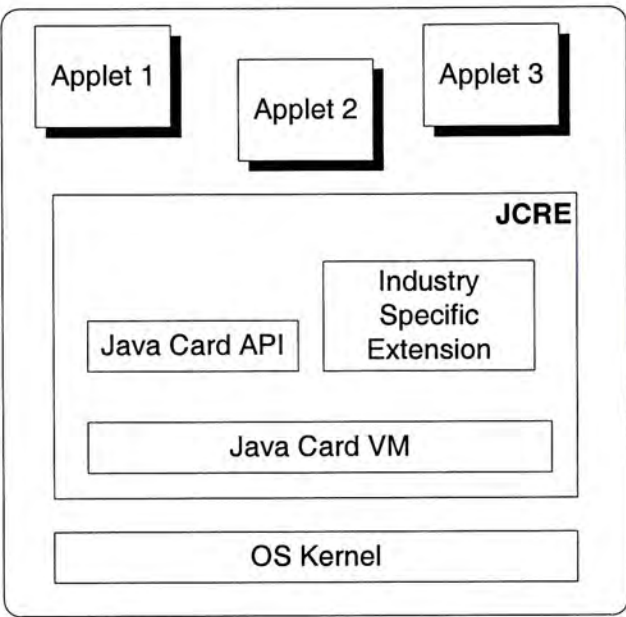


Figure 2.6: Java Card Technology Architecture

Advantages of a Java Card

Strictly speaking, the benefits of Java Card includes platform independence, multi-application capability, post-issuance, flexibility and compatibility with smart card standards.

Java Card technology is platform-independent, enabling developers to use Java Card technology-based applets to run applications on different vendors’ cards. Java Card technology has been designed fundamentally to be secure and multi-application capable. It provides a post-issuable feature, where applica- tions can be securely loaded after the cards are issued, allowing card issuers to dynamically respond to the preferences of the card holder.

Additionally, Java Card technology is compatible with existing smart card standards, such as formal international standards ISO7816, and industry- specific standards, such as Europay/MasterCard/Visa (EMV) and the Eu- ropean Telecommunications Standards Institute 03.19 (ETSI 03.19).

2.3 Authentication Protocol

When a registered system user logs into the system, the system initializes the authentication protocol with the login user. How does the system know that it is the registered user? There is a situation that a hostile user impersonates a registered user to login to the system. Authentication protocol is designed to resolve this problem. Typically, authentication can be performed with these approaches:

- what a person knows
- Some examples are the password [37] [44], pin and hand-written signature. It is called one-factor authentication.

Advantages	Disadvantages
<ul style="list-style-type: none">• Easy to remember• Easy to modify the secret	<ul style="list-style-type: none">• Easy to leak out the password• Easy to intrude such systems• Easy to be confused when there are many passwords memorized by a user• Dependency of the security with the length of passwords or the complexity of the signature

Table 2.2: Advantages and Disadvantages of one-factor authentication protocol

- what a person has
- Credit card systems and ATM machines are examples of it, in which a person holds a user’s specific password-protected card. It is called two-factor authentication.

Advantages	Disadvantages
<ul style="list-style-type: none">• Not able to be forged without expensive equipment• Being protected by a password so that only stealing the card cannot access the system	<ul style="list-style-type: none">• Only applicable to local access or secure network systems

Table 2.3: Advantages and Disadvantages of two-factor authentication protocol

- what a person is (Biometrics)
Human has some unique inborn features such as the retina pattern and fingerprint. These features can be used for the identification of a person.

Advantages	Disadvantages
<ul style="list-style-type: none">• Extremely difficult to be falsified	<ul style="list-style-type: none">• Expensive to equip with sophisticated retina and fingerprint readers• System failure when the retina or fingerprint data are lost

Table 2.4: Advantages and Disadvantages of biometric-based authentication protocol

2.3.1 Properties

Central to the authentication protocol, the following requirements [51] [27] must be achieved:

- Mutual Authentication
It is an important property that enables the involved parties to identify themselves mutually and establish a secret session key. Traditionally, the

authentication protocol only considers the user authentication. Nowadays, this design practice can lead to the potential intrusion into the user host by the forged authentication server.

- Confidentiality

All messages should be transmitted in the encrypted form to prevent the masquerade and compromise of secret session keys. Otherwise, a plaintext message can be grabbed with a simple sniffer program and interpreted directly.

- Timelessness

The protocol is able to resist message replay attacks, which could lead to the compromise of a session key or the impersonation of another user.

Replay attack can be classified in different types [26] as follows:

- Simple replay
- Repetition that can be logged
- Repetition that cannot be detected
- Backward replay without modification

2.3.2 Survey

Authentication protocols can be classified as local and remote authentications. Local authentication [54] [15] is used for local access within a secure network while remote authentication is used for remote access in an open network. We focus on remote authentication systems with the collaboration of the smart card and public key infrastructure.

Many smart card based remote authentication protocols have been proposed. In 1991, reference [9] described a remote authentication protocol making use of smart cards. This protocol allows the server to verify the password

without storing the password in the server. This protocol can also tackle the replaying attack by using time-stamps. Nonetheless, it was later discovered [8] that anyone who possesses the public information in the network could derive some of the secret keys of the password generation center. A hostile party can therefore impersonate a legal user in a subsequent login.

In 1993, Chang et al. [7] proposed another authentication protocol based on Shamir's algorithm for the signature scheme. All messages transmitted have included the time-stamps to resist replaying attacks. This protocol does not require the authentication server to store the password. However, this protocol does not allow the user to change the password unless the username is changed as well since the password is dependent on the username according to a predefined formula. It makes this protocol too impractical to apply to real systems.

Later, Lee and Kim [38] proposed a remote authentication protocol in 1998. More precisely, it is called mutual authentication protocol. This protocol incorporates the challenge-response protocol into public key techniques. The card generates a digest according to the username and smart card identification number. The digest is sent to the authentication server and compared with the digest computed in the server. Obviously, the digest may be sent to the fraudulent authentication server even though it is sent through a secure channel. After that, the fraudulent server can impersonate the legal user to access the authentication server successfully.

Recently, Hwang and Li [35] proposed a new remote authentication protocol based on ElGamal's public key crypto-system in 2000. The authentication server does not need to store the password for each user, and this protocol can withstand replaying attacks. However, this protocol does not support mutual authentication. At the same time, the user password cannot be changed unless the username is changed as [7] proposed in 1993.

On the other hand, a local collaborative authentication protocol by using

fingerprint verification was proposed by Moon [43] to verify the ownership of a smart card in 2000. It is considered as a highly secure measure to guarantee that only the valid user can activate the authorized card. However, this authentication system cannot be used for the remote authentication that is common in the e-commerce system. It provides no guarantees that only the correct user can login to the correct remote authentication server, ignores of the key distribution scheme and does not handle the failure of the system [58] [40] [30].

In general, we can characterize the remote authentication protocol according to the features listed as follows:

1. Resist time attacks
2. Store the user password for verification in the server
3. Use a formula to verify the user identity
4. Authenticate the client only
5. Authenticate the client and the server mutually
6. Password cannot be changed once it is assigned to a user

Table 2.5 summarizes the characteristics of these protocols.

Protocol	Resistance to replay attacks	Storage of the user password	Storage of the formula	User authentication	Mutual authentication	Fixed password
[9]	Yes	No	Yes	Yes	No	Yes
[7]	Yes	No	Yes	Yes	No	Yes
[38]	No	Yes	No	Yes	Yes	No
[35]	Yes	No	Yes	Yes	No	Yes

Table 2.5: Characteristics of different remote authentication protocols

2.4 Transaction Protocol

Transaction protocols describe the methodology for the exchange of payments and goods. The security concern of the transaction protocol will be examined.

Generally, different payment systems [2] [4] [34] [62] [63] [49] may have different requirements from the transaction protocol. Typically, they are always characterized by the properties such as integrity, authorization, confidentiality, availability and reliability.

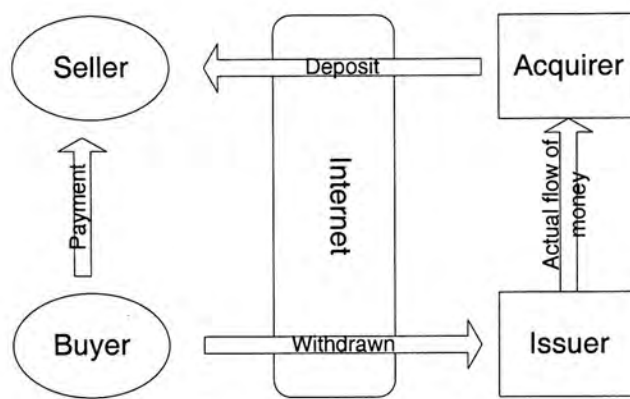


Figure 2.7: General Payment Model

Integrity

The protocol guarantees that the payment transferred cannot be modified and it can only be deposited to a dedicated party.

Authorization

It is of paramount importance that the protocol disallows the receipt of payments without explicit authorization. Reference [2] generalizes the method for authorization into three types.

- Out-band authorization
- Password authorization
- Signature authorization

Confidentiality

Confidentiality of a message ensures that the unrelated parties in the protocol would not be able to read the message. A message can only be read by the participants of the protocol system.

Availability

It ensures that the buyer and seller can always make or receive payments whenever necessary.

Reliability

Obviously, the protocol must be free from hanging in unknown or inconsistent state. Both the buyer and seller will not suffer a loss of property in case of protocol failures. On the other hand, the protocol can perform recovery from failures.

2.5 BAN Logic

BAN logic [5], named by its founders M. Burrows, M. Abadi and R. Needham, is popularly applied to the analysis of protocols for authentication in distributed systems. It provides the formalism [48] [16] that allows a protocol to prove the true presence of each party to the others, and to check the redundancy as well as the security problems. The notation and logical postulates of the BAN logic are outlined in the following sections.

2.5.1 Notation

The symbols A , B and S denote the specific principles. K_{ab} , K_{as} and K_{bs} denote the specific shared keys. K_a , K_b and K_s denote the specific public keys. K_a^{-1} , K_b^{-1} and K_s^{-1} denote the corresponding private key. N_a , N_b and

N_c denote the specific statements. The symbols P , Q and R represent the principles. The statements were represented by X and Y . K represents the encryption key. These notations are used as either meta-symbols or as the free variables with an implicit universal quantification.

The proposition is connected by the conjunction, denoted by a comma. Conjunction has the properties of associativity and commutativity. In addition to the conjunction, we use the following connective expression:

- $P \models X$

P believes X , or P would be entitled to believe X . In particular, the principal P may act as though X is true. This operator is central to logic.

- $P \triangleleft X$

P sees X . P receives a message from a principal and can read and repeat X in that message.

- $P \mid\sim X$

P has said X . P has sent X , but it does not contain the information of when this message was sent. However, it is known that P believed X when P sent X .

- $P \mid\Rightarrow X$

P has jurisdiction over X . In other words, P has the authority on X . Similar situation such as the certification authority has jurisdiction to say which certificate (public key) of a person is.

- $\#(X)$

The message X is *fresh*. This message has not appeared before the current round of the protocol. *Nonce* is one example of such a message. It usually includes a time-stamp or message sequence number.

- $P \stackrel{K}{\leftrightarrow} Q$

P and Q use the shared key K for the communication. The shared key K is only known to P and Q , or a principle trusted by either P or Q .

- $\stackrel{K}{|}\rightarrow P$:

P has a public key K . The corresponding private key K^{-1} will not be known by any principal except P , or principals trusted by P .

- $P \stackrel{X}{\rightleftharpoons} Q$

The formula X is a secret known only to P and Q . P and Q can use X to prove their identities to one another. An example of a shared secret is a password.

- $\{X\}_K$

This represents that X is encrypted under the key K .

- $< X >_Y$

This represents that X has combined with the formula Y and it is intended that Y is a secret, and its presence proves the identity of whoever says $< X >_Y$.

2.5.2 Logical Postulates

Now, we introduce the logical postulates to provide enough machinery to carry out the analysis of authentication protocols.

- Rule 1: Message-meaning rule

It concerns the interpretation of plaintext messages, encrypted messages and messages with secret. It is used to derive the beliefs about the origin of messages. The formula below is postulated for shared keys.

$$\frac{P \models Q \stackrel{K}{\leftrightarrow} P, P \triangleleft \{X\}_K}{P \models Q \sim X}$$

Similarly, the formula for public keys is postulated like this:

$$\frac{P \models \overset{K}{\rightarrow} Q, P \triangleleft \{X\}_{K-1}}{P \models Q \mid \sim X}$$

The formula for shared secrets is postulated like this:

$$\frac{P \models Q \overset{Y}{\rightleftharpoons} P, P \triangleleft \langle X \rangle_Y}{P \models Q \mid \sim X}$$

- Rule 2: Nonce-verification

It ensures that the message is fresh and the sender still believes it.

$$\frac{P \models \sharp(X), P \models Q \mid \sim X}{P \models Q \models X}$$

- Rule 3: Jurisdiction

It states that if P believes that Q has jurisdiction over X , then P trusts Q on the truth of X :

$$\frac{P \models Q \Rightarrow X, P \models Q \models X}{P \models X}$$

- Rule 4: A necessary property of the belief operator

P believes a set of statements if and only if P believes each individual statement separately.

$$\frac{P \models X, P \models Y}{P \models (X, Y)} \quad \frac{P \models (X, Y)}{P \models X} \quad \frac{P \models Q \models (X, Y)}{P \models Q \models X}$$

- Rule 5: Similar rule is applied to the operator $\mid \sim$

$$\frac{P \models Q \mid \sim (X, Y)}{P \models Q \mid \sim X}$$

- Rule 6: A principal sees a formula, then he also sees its components, provided that he knows the necessary keys.

$$\begin{array}{c}
 \frac{P \triangleleft (X, Y)}{P \triangleleft X} \quad \frac{P \triangleleft \langle X \rangle_Y}{P \triangleleft X} \quad \frac{P \models Q \xleftrightarrow{K} P, P \triangleleft \{X\}_K}{P \triangleleft X} \\
 \frac{P \models \xrightarrow{K} P, P \triangleleft \{X\}_K}{P \triangleleft X} \quad \frac{P \models \xrightarrow{K} Q, P \triangleleft \{X\}_{K-1}}{P \triangleleft X}
 \end{array}$$

- Rule 7: Freshness of a formula

If one part of a formula is known to be fresh, the entire formula must be fresh.

$$\frac{P \models \#(X)}{P \models \#(X, Y)}$$

- Rule 8: If a key is used between a pair of principals in both directions, the following formulas can describe this property.

$$\frac{P \models R \xleftrightarrow{K} R'}{P \models R' \xleftrightarrow{K} R} \quad \frac{P \models Q \equiv R \xleftrightarrow{K} R'}{P \models Q \models R' \xleftrightarrow{K} R}$$

- Rule 9: If a secret is used between a pair of principals in both directions, the following two rules can reflect this property.

$$\frac{P \models R \xrightarrow{X} R'}{P \models R' \xrightarrow{X} R} \quad \frac{P \models Q \models R \xrightarrow{X} R'}{P \models Q \models R' \xrightarrow{X} R}$$

With the postulates, we can construct the proofs in the logic. Starting from a formula X , we can prove the validity of Y if there is a sequence of formula Z_0, \dots, Z_n , where $Z_0 = X$, $Z_n = Y$, and each Z_{i+1} can be obtained from Z_i by the application of the mentioned logical postulates.

2.5.3 Protocol Analysis

Principally, a protocol is analyzed with the following procedures:

1. The protocol to be analyzed is converted from its informal form into the formalized logical statements.
2. The assumptions about the initial state are stated.
3. The assertions for each statement about the state of the system are devised.
4. The logical postulates are applied to the protocol step-by-step, in order to derive the conclusion.

Chapter 3

Authentication Protocol

With the advance of the smart card technology and growing demand for secure applications in the community, many researches are being done on smart-card based systems from in-house applications to the Internet applications.

In this section, we will propose our design of the smart-card based authentication protocol and silent key distribution scheme with the failure handling for the ordinary application service provider. Compared with other similar protocols, our protocol not only can resist replaying attacks, but also can perform a mutual authentication, and handle smart card or server failures properly. The protocol can be incorporated into biometric-based or password-based systems.

Formal analysis of our protocol with *BAN logic* will be detailed to show its correctness. Finally, the implementation with the Java card and the experimental result will also be described.

3.1 Formulation of Problem

When we have remote access to an application service provider, the choice of the authentication mechanism is very crucial to the security of the system. Without deliberate consideration to the design of the authentication protocol, serious defects of the authentication system would be induced and harmful to the service provider and other system users.

The choice of the password-based authentication system cannot simply resolve the challenges from the intruders. As mentioned before, the hostile party can always impersonate a user by the brute-force password search in the password domain. The password can also be stolen more easily compared with other approaches.

Another solution is the biometric based authentication system. It is generally applied to local access systems or secure networks. Obviously, it cannot be applied to remote access system because of the unrecoverable failure of biometric-based systems. Just imagine what would happen when the fingerprint data of a person is stolen, say in the remote authentication process. Still, the intruder can impersonate the user to log in the system with the stolen fingerprint data. Anyway, the user can replace the fingerprint data with another fingerprint, but what would happen when all ten fingerprint data are stolen? Unlike password-based or PKI systems, in which the password or the certificate can be regenerated easily, it is fatal that the biometric based system cannot handle data loss failures.

3.2 The New Idea

We propose the one-way function based authentication protocol with the collaboration of the smart cards and public key infrastructure.

Inspired by the idea of [17] [3], we use a pair of one-way hash function f and F to send a secret to another party for the verification without disclosing the secret. The authentication data are hashed with f and salted with a timestamp [29]. The generated hash value can be verified by another party who holds F .

Definition: f is of the form $f: G \rightarrow G$ where G is a domain where we can test the membership, compute the inverse and group operation, sample from a nearly uniform distribution efficiently. There is a constraint applied on f :

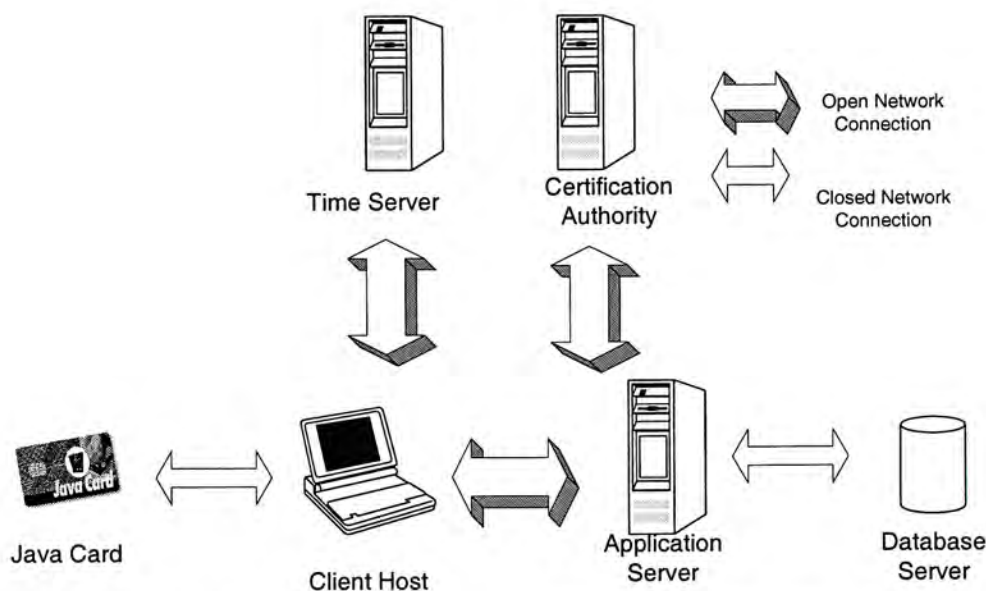


Figure 3.1: System Diagram of the Authentication System

$G \rightarrow G$, that is

$$F(x, f(y)) = f(x * y) \quad \text{for } x \neq 1 \quad (3.1)$$

Several parties are involved in this protocol as shown in figure 3.1. Consider the scenario that the client host attached with the card reader is trying to login to the remote application server to access some services. Certification authority and the time server will participate in the protocol.

- **Client Host**

It initializes the authentication protocol by activating the card applet and communicating with the application server.

- **Java Card**

It has a built-in finite state machine to handle the request from the client host and make a suitable response.

- **Application Server**

It handles the client request for accessing some services.

- **Database Server**

It stores user's information partly for the authentication.

- **Certification Authority**

It stores, distributes and issues the certificate of the service subscribers and the application service provider.

- **Time Server**

It distributes its current GMT time.

3.3 Assumptions

We hereby assume that the certification authority and the time server are trustable and can always give us the valid data according to the request. Also, the communication channel between the card and client host is protected by the PKI, which is commonly provided in the smart card system. The PKI protection is also available to the communication channel between the database server and application server. At last, we assume that the communication channel between the client host and application server is protected.

3.4 Trust Model

A trust model is used to describe how the entity in a protocol can trust another entity. With the trust model, we can determine who can be trusted by someone when some constraint(s) is/are satisfied. The trust model of the card, client and application are defined as follows:

- Trust 1

The card and server believe that only the one who can show the valid password/biometric data is the legal user.

- Trust 2

The client and server believe that only subscriber's card can sign the message with his private key.

- Trust 3

The card believes that only the application server can generate the correct digest with the card-specific F .

- Trust 4

The application server believes that only the card can generate the correct digest with the card-specific f .

3.5 Protocol

In general, our protocol is divided into three phases.

- **Registration Phase**

The subscriber is issued with an authorized smart card.

- **Local Authentication Phase**

The card and local host are mutually authenticated.

- **Remote Authentication Phase**

The card and application server are mutually authenticated and a shared session key is established on the card and application server without the knowledge of the client.

When the service subscriber initializes the protocol, the subscriber must be issued with the authorized smart card in advance. This can be done when the subscriber approaches to the card center to formally apply for a service and the card.

3.5.1 Registration

In the registration phase, the subscriber will be assigned an authorized smart card. The subscriber should go to the card center to prove his identity. Then

the following steps will be taken as shown in figure 3.2 to issue the smart card. The sequence of the protocol is indicated in the circle on the top left corner of each text box in the figure.

1. A card applet is loaded onto the card and customized with the password/biometric data.
2. A pair of RSA keys, functions f and F are generated on card.
3. The RSA public key and F are downloaded from the card.
4. The RSA public key downloaded from the card is used to generate a self-signed certificate, which is further signed by the certification authority while F is stored in the corresponding user-entry in the database.
5. The purse in the card is activated.

3.5.2 Local Authentication

As mentioned before, the subscriber can only access the service provided by the application server with an authorized card. Before the subscriber can access the desired service, the local authentication must be performed successfully. A client program runs on behalf of the subscriber to help the subscriber to execute the authentication procedure. Figure 3.3 illustrates how the local authentication is done. The card and client can identify each other after this phase.

We describe the protocol as a sequence of rounds, which consists of a number of messages. The notation $X \rightarrow Y : M$ is used to describe that message M is sent from X to Y . For the interest of brevity, the messages transmitted are assumed private and signed by the corresponding party.

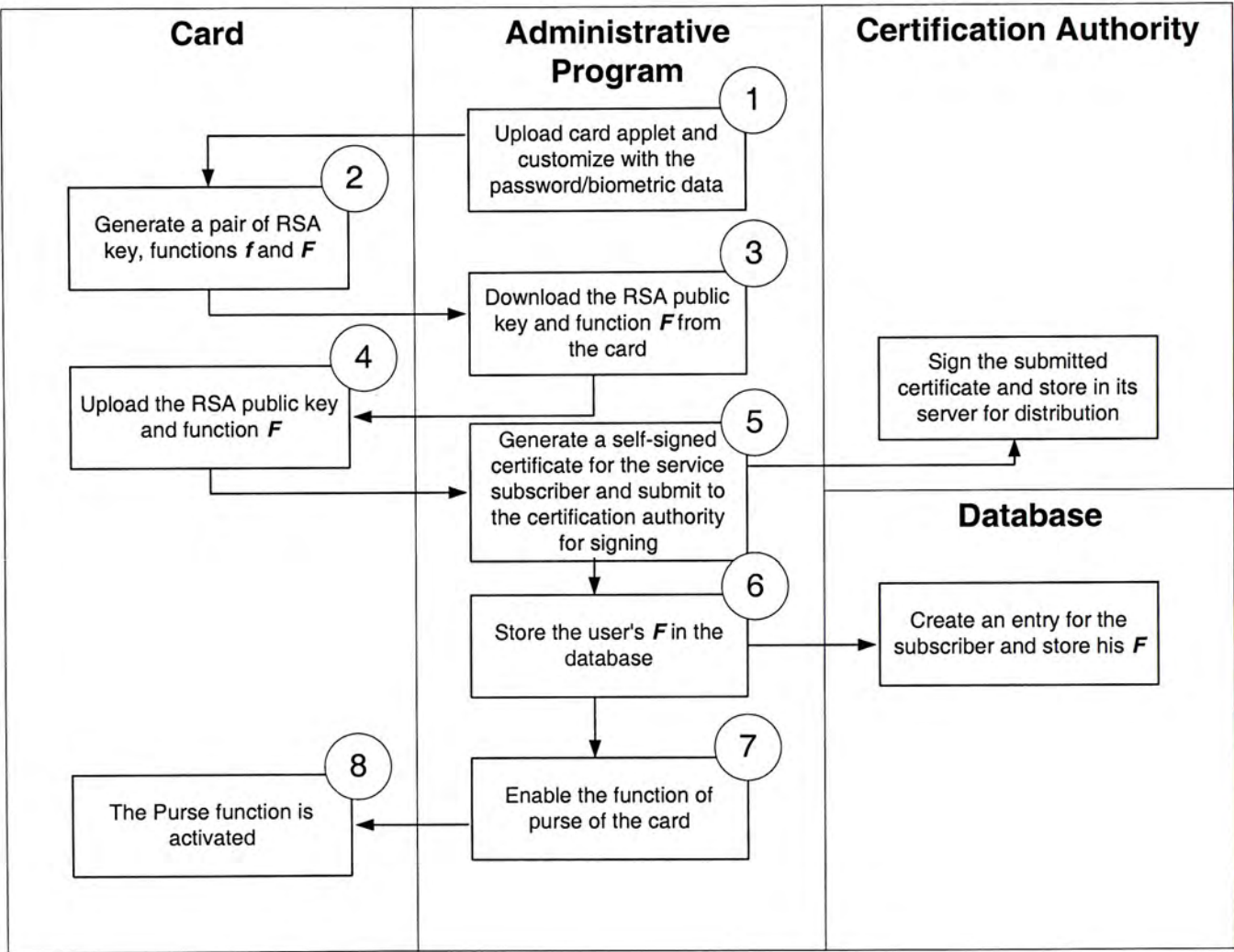


Figure 3.2: Registration

Formally, the subscriber supplies the login information *login* in M1, say password or biometric data, in step 5 to the host which further sends to the card.

$$Host \rightarrow Applet : username, login, time t \quad (M1)$$

The card receives the message and verifies its content in step 6. Upon the successful verification, the card generates the authentication data in step 7 and sends M2 to the host in step 8 for remote access of the application server. Otherwise, the card sends a failure signal to the host and halts the execution.

$$Applet \rightarrow Host : username, f(random\ number\ X), time t \quad (M2)$$

3.5.3 Remote Authentication

When the card and host are ready, the remote authentication can be carried out to enable the subscriber to access the services. Figure 3.4 illustrates the mechanism of the remote authentication protocol. The interactions with the certification authority and time server are not shown in the figure for simplicity. The card, client and application server would have been authenticated by each other after this phase.

The username, time t and hashed random number $f(random\ number\ X)$ attached in M3 (defined below) are sent to the remote application server in step 1. They are for the calculation of the hashed value by the application server.

$$Host \rightarrow Server : username, f(random\ number\ X), time t \quad (M3)$$

When the server has received the message, the user's F would be retrieved from the database and used for the computation of the hash value

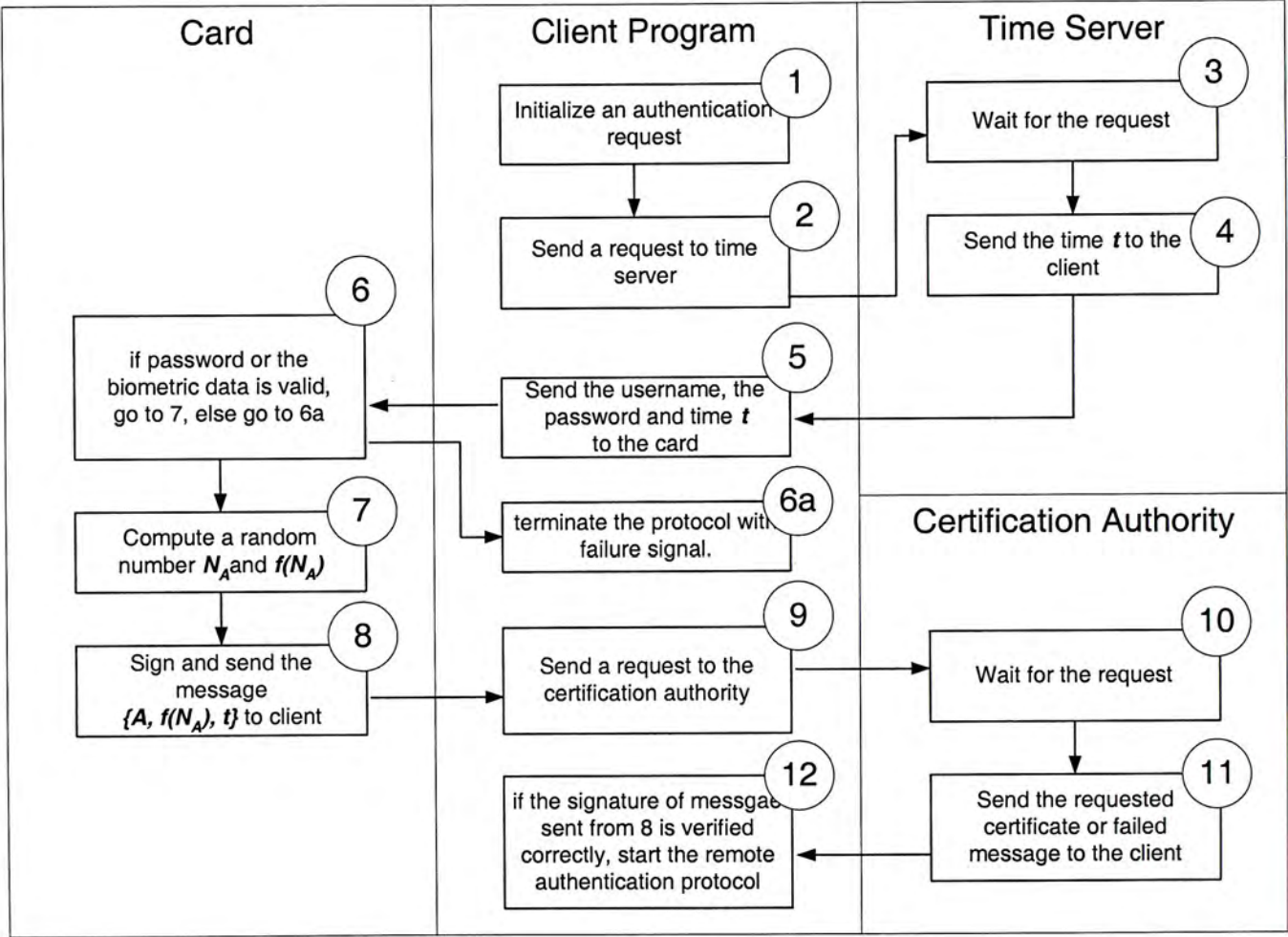


Figure 3.3: Local authentication protocol

$F(\text{time } t, f(\text{random number } X))$ in step 5. Time t' and the hash value are sent in M4 to the host for the user authentication.

$$\text{Server} \rightarrow \text{Host} : F(\text{time } t, f(\text{random number } X)), \text{time } t' \quad (\text{M4})$$

The host would forward the received message M5 to the card upon correct signature verification of the message in step 8.

$$\text{Host} \rightarrow \text{Applet} : F(\text{time } t, f(\text{random number } X)), \text{time } t' \quad (\text{M5})$$

The card applet checks the validity of the hashed value in step 10. After correct verification in step 11, the card computes the hashed value for the server. The computed hash value M6 is sent to the server in steps 12 and 13. Otherwise, a failure signal is sent and the protocol is halted.

$$\text{Applet} \rightarrow \text{Host} : f(\text{time } t' * \text{random number } X) \quad (\text{M6})$$

$$\text{Applet} \rightarrow \text{Host} : f(\text{time } t' * \text{random number } X) \quad (\text{M7})$$

When the server receives the forward message from the host, it checks the correctness of the hashed value in step 14. For the correct matching of the hash value, the remote authentication protocol is terminated successfully and a secret session key is formed in step 15.

3.5.4 Silent Key Distribution Scheme

Once the authentication protocol is completed successfully, a shared secret key would be built in the card and application server without the knowledge of the client program. Afterwards, this key can be used for the subsequent secure

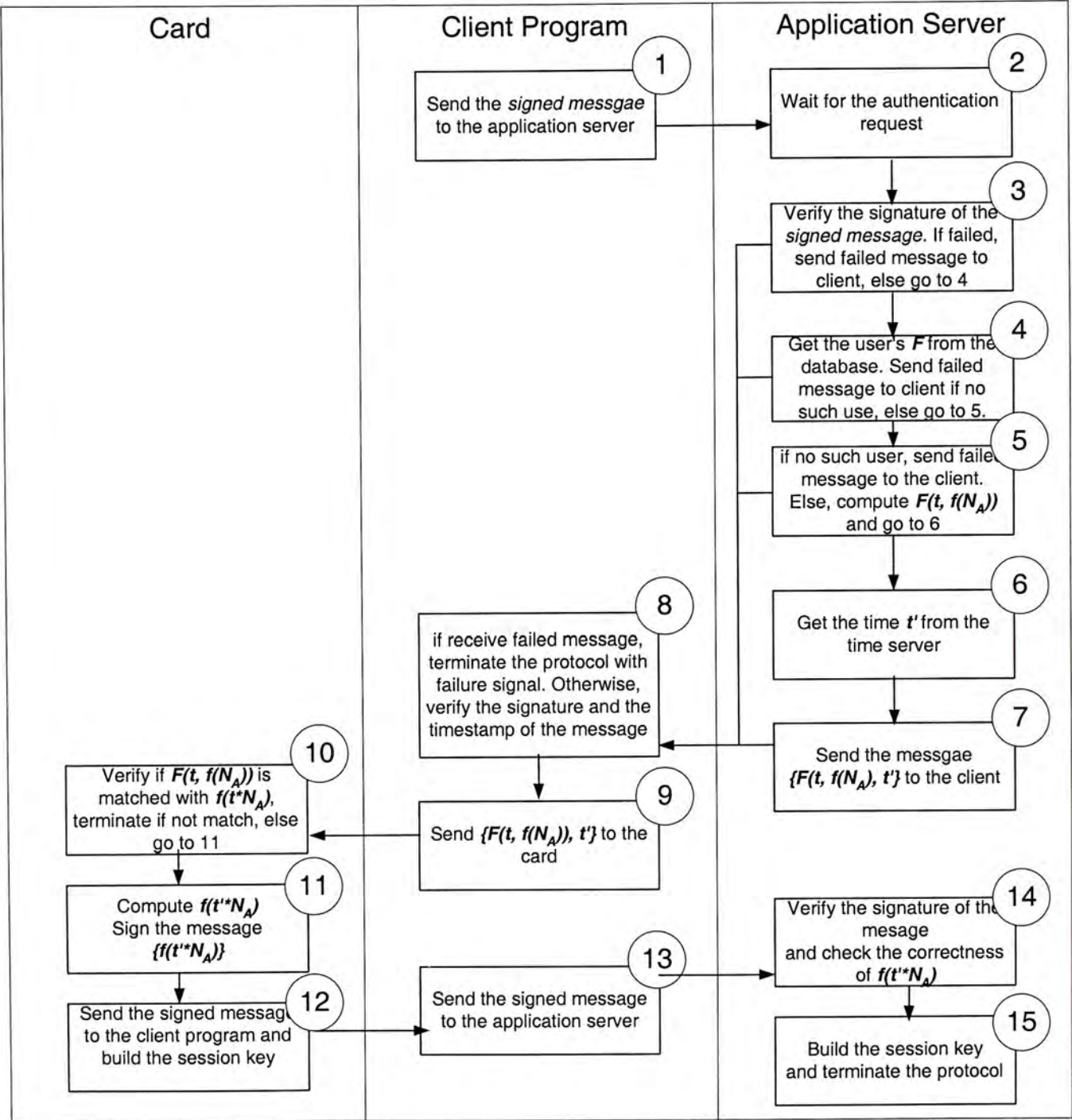


Figure 3.4: Remote authentication protocol

communication between the card and application server when the confidential information is being sent between them.

The card builds the secret key for triple DES algorithm [57] by using three data sets obtained from the protocol:

$$f(X), f((t' - 1) * X), f((t - 1) * X)$$

Meanwhile, the application server constructs the secret key according to three data sets collected from the protocol:

$$f(X), F((t' - 1), f(X)), F((t - 1), f(X))$$

3.5.5 Advantages

By and large, our protocol has the following features and improvements on the existing smart-card based remote authentication protocol.

1. Higher security measures by the mutual authentication
2. Complement the biometric-based protocol
3. Resistance to replaying attacks by using time-stamp token
4. Silent key distribution scheme for the applet and server
5. Resist the ordinary physical and remote attacks
6. Independence of the username from the password
7. Better smart card failure handling because different user has different f ;
better server failure handling because F cannot generate $f(x)$ given x
8. Suitableness to apply on password-based or biometric-based systems,
wireless and Internet-based e-commerce service servers

3.6 BAN Logic Analysis

Formally, the following notations are used in the proof. $\{M\}_{K_{A^{-1}}}$ denotes that the message M is signed with the private key of A . $\{M\}_{K_{AB}}$ denotes that the message M is encrypted with the shared key of A and B .

Symbol	Meaning
ID	Username
PWD	Password or biometric data
X	Random number
t	Time generated by the time server
t'	Time generated by the time server
K_{AH}	Shared key of Applet and Host
K_{HS}	Shared key of Host and Server
K_{AS}	Shared key of Applet and Server
$K_{A^{-1}}$	Private Key of Applet
$K_{S^{-1}}$	Private key of Server

Table 3.1: Notations used in the proof

Step 1 - Formalize Protocol

The messages transmitted in the protocol are listed as follows:

- M1: $Host \rightarrow Applet$

The message is encrypted with the shared secret key of the applet and the host. The password is included in this message as the critical login information that is only known by the applet and host.

$$\{ID, Applet \stackrel{PWD}{\Rightarrow} Host, t, \#t\}_{K_{AH}} \tag{M1}$$

- M2: $Applet \rightarrow Host$

The message is signed with the private key of the applet and is encrypted with the shared key of the applet and host.

$$\{ID, \{(\text{Applet} \mid \sim f(X)), t, \#t\}_{K_{A-1}}\}_{K_{AH}} \quad (\text{M2})$$

- M3: *Host* \rightarrow *Server*

Message M2 is decrypted with the shared secret key of the applet and host. The decrypted message is encrypted with the shared key of the host and the server. The encrypted message is then forwarded to the server.

$$\{ID, \{\text{Applet} \mid \sim f(X), t, \#t\}_{K_{A-1}}\}_{K_{HS}} \quad (\text{M3})$$

- M4: *Server* \rightarrow *Host*

The message contains the required hash value $F(t, f(X))$ for the server authentication and the generated parameter for the subsequent user authentication.

$$\{\{\text{Applet} \xrightarrow{F(t, f(X))} \text{Server}, t', \#t'\}_{K_{S-1}}\}_{K_{HS}} \quad (\text{M4})$$

- M5: *Host* \rightarrow *Applet*

The host re-encodes the message M4 with the shared key of the applet and host.

$$\{\{\text{Applet} \xrightarrow{F(t, f(X))} \text{Server}, t', \#t'\}_{K_{S-1}}\}_{K_{AH}} \quad (\text{M5})$$

- M6: *Applet* \rightarrow *Host*

The applet computes the required hash value from the server and signs it with the private key of the applet.

$$\{\{\text{Applet} \xrightarrow{f(t' * X)} \text{Server}\}_{K_{A-1}}\}_{K_{AH}} \quad (\text{M6})$$

- M7: $Host \rightarrow Server$

The host re-encodes the message M6 which is then forwarded to the server. After successful user authentication, a shared secret session key would be established on the server and applet.

$$\{\{Applet \xrightarrow{f(t' * X)} Server\}_{K_{A^{-1}}}\}_{K_{HS}} \quad (M7)$$

Step 2 - State Initial Assumptions

There are some axioms made before the initialization of the protocol. Shared secret key K_{AH} is established between the applet and the host (axiom 1) while K_{HS} is established between the host and the server (axiom 2). The applet has K_A as the public key (axiom 3) while the server has K_S as the public key (axiom 4). The applet, host and the server believes the certificate issued by the certification authority, i.e. the applet, host and server believe K_A is the public key of the applet (axioms 5, 6 and 7); the host and the server believe K_S is the public key of the server (axioms 8, 9).

$$Applet \xleftrightarrow{K_{AH}} Host \quad (Axiom\ 1)$$

$$Host \xleftrightarrow{K_{HS}} Server \quad (Axiom\ 2)$$

$$\begin{array}{c} K_A \\ \vdash \end{array} Applet \quad (Axiom\ 3)$$

$$\begin{array}{c} K_S \\ \vdash \end{array} Server \quad (Axiom\ 4)$$

$$Applet \models \begin{array}{c} K_A \\ \vdash \end{array} Applet \quad (Axiom\ 5)$$

$$Host \models \begin{array}{c} K_A \\ \vdash \end{array} Applet \quad (Axiom\ 6)$$

$$Server \models \begin{array}{c} K_A \\ \vdash \end{array} Applet \quad (Axiom\ 7)$$

$$Host \models \begin{array}{c} K_S \\ \vdash \end{array} Server \quad (Axiom\ 8)$$

$$Server \models \begin{array}{c} K_S \\ \vdash \end{array} Server \quad (Axiom\ 9)$$

As for the hash functions f and F , it is well-known that the server has the access to the hash value of $F(M)$ for any message M (axioms 12 and 13) while the applet has the access to the hash value of $f(M)$ for any message M (axioms 10 and 11). The applet and host have the password or biometric data as the shared secret (axiom 14). The server and applet believe the time server (axioms 15 and 16).

$$Applet \models \forall x.[Applet \Rightarrow f(x)] \quad (\text{Axiom 10})$$

$$Server \models \forall x.[Applet \Rightarrow f(x)] \quad (\text{Axiom 11})$$

$$Applet \models \forall x.[Server \Rightarrow F(x)] \quad (\text{Axiom 12})$$

$$Server \models \forall x.[Server \Rightarrow F(x)] \quad (\text{Axiom 13})$$

$$Applet \stackrel{PWD}{\rightleftharpoons} Host \quad (\text{Axiom 14})$$

$$Applet \models \#t \quad (\text{Axiom 15})$$

$$Server \models \#t' \quad (\text{Axiom 16})$$

Step 3 - Devise Assertion and Execute Proof

By the application of rule 1 (please refer to the logical postulates stated in chapter 2), axiom 1, the applet sees the valid password supplied by the host (A3), which is the secret known only by the applet and card holder. In other words, the applet believes the presence of the card holder according to the trust model. By the trust model and jurisdiction rule, the applet believes that the time t is fresh.

$$Host \sim PWD \quad (\text{A1})$$

$$Applet \triangleright PWD \quad (\text{A2})$$

$$Applet \models Host \sim PWD \quad (\text{by rule 1 and axiom 1}) \quad (\text{A3})$$

The applet generates a hash value from a random value and sends the signed message to the host, which sends to the server. When the server receives the message, it believes that it is the applet's login request (A7) since the correct hash value is signed with the applet's private key (A5) and its time-stamp is fresh (axiom 16). Then, the server computes the requested digested value for the applet and starts the user authentication. With the freshness of t and rule 7, the server is convinced that the request message is sent recently and is not a replay of an old message.

$$Server \triangleright f(x) \quad (\text{by rule 6 and axiom 2}) \quad (A4)$$

$$Server \models Applet \sim f(x) \quad (\text{by rule 1 and A4}) \quad (A5)$$

$$Server \models Applet \sim t \quad (\text{by rule 1 and axiom 2}) \quad (A6)$$

$$Server \models Applet \models f(x) \quad (\text{by axiom 16 and A15}) \quad (A7)$$

The host forwards the message with the digested value $F(t, f(x))$ to the applet. The applet finds that the hash value $F(t, f(x))$ is matched with $f(t * X)$ by the definitions of f and F . Similarly, by the application of the nonce-verification rule, the authentication request is fresh and not a replaying message.

Since the applet can see and verify the hash value $F(t, f(x))$ correctly, we can conclude that the applet believes the true identity of the server (C1) by the jurisdiction rule and corrected hash value. Then, the applet generates the requested hash value $f(t' * X)$ and sends to the server via the host for the user authentication.

$$Applet \triangleright F(t, f(x)) \quad (\text{by rule 6 and axiom 2}) \quad (A8)$$

$$Applet \mid \equiv Server \mid \sim F(t, f(x)) \quad (\text{by rule 1 and A7}) \quad (A9)$$

$$Applet \mid \equiv Server \mid \equiv F(t, f(x)) \quad (\text{by A8, rule 2 and axiom 15}) \quad (C1)$$

$$Applet \mid \equiv Server \mid \sim t' \quad (\text{by rule 1}) \quad (A10)$$

When the server receives the information, it checks the correctness of the hash value. Since $f(t' * X)$ is equal to $F(t', f(x))$ and the applet is convinced that the reply message is fresh, the user authentication is passed (C2) according to the jurisdiction rule and nonce-verification rule.

$$Server \triangleright F(t', f(x)) \quad (\text{by rule 6 and axiom 2}) \quad (A11)$$

$$Server \mid \equiv Applet \mid \sim F(t', f(x)) \quad (\text{by rule 1 and A11}) \quad (A12)$$

$$Server \mid \equiv Applet \mid \equiv F(t', f(x)) \quad (\text{by A12, rule 2 and axiom 16}) \quad (C2)$$

Step 4 - Draw Conclusions

Finally, a shared secret key is established between the server and applet (C3 and C4) since the server and host know the true presence of each other (C1 and C2) after validating the authentication data.

$$Applet \mid \equiv Applet \stackrel{K_{AS}}{\longleftrightarrow} Server \quad (\text{by C1}) \quad (C3)$$

$$Server \mid \equiv Applet \stackrel{K_{AS}}{\longleftrightarrow} Server \quad (\text{by C2}) \quad (C4)$$

3.7 Experimental Evaluation

This section describes the experimental result we have done for the evaluation of the proposed authentication protocol.

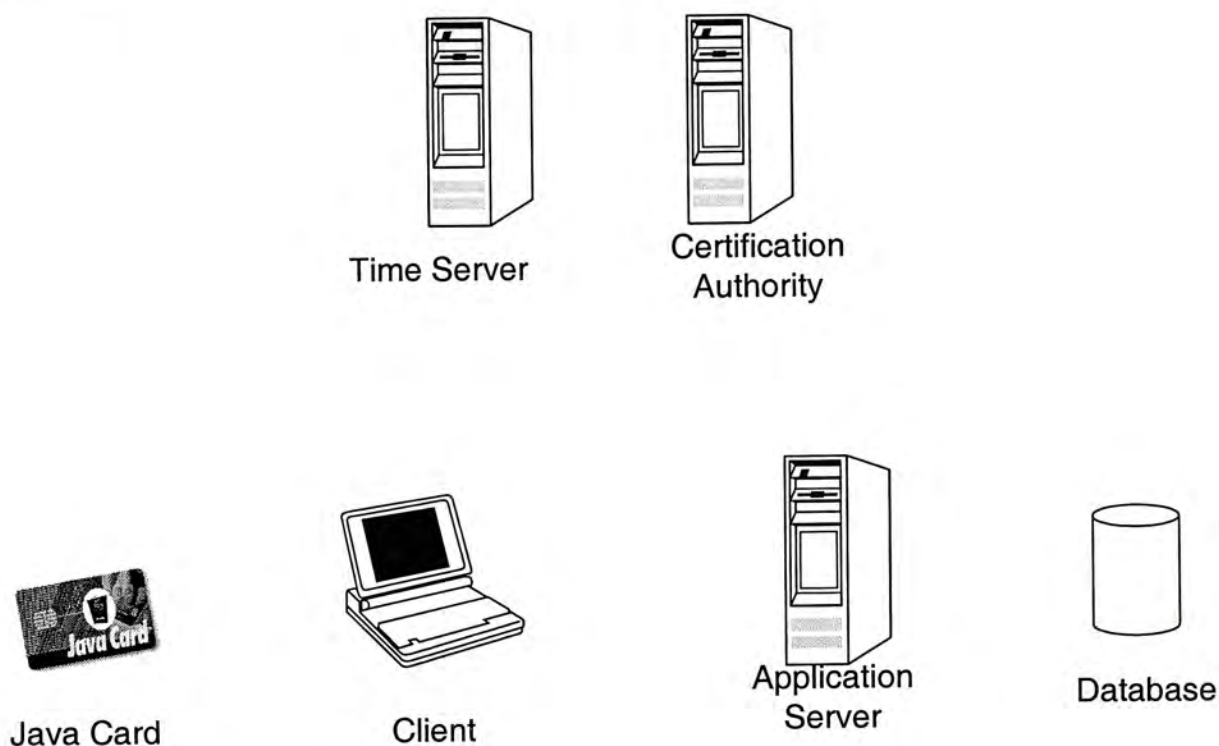


Figure 3.5: Testing environment

3.7.1 Configuration

The development environment is based on GemXpresso rapid application development toolkit 211 version 2.4 with the GemXpresso 211PKis Java card. This Java card is implemented according to Java Card 2.1 API and Java Card 2.1 VM [31] [11] with triple DES and RSA algorithms [57].

Various APIs are used for writing the test program. They include Java Core API 1.2.2, Java Card 2.1, Java Cryptography Extension 1.2.1, Java Secure Socket Extension 1.0.2, OpenCard 1.2, Gemplus Card Service and Open Platform.

The experimental environment is setup as shown in figure 3.5.

The implementation of hash functions f and F is based on the exponential expression as expressed in Hash 1 and Hash 2.

It is assumed that $G = Z_N^*$ where N is a product of two large distinct prime numbers. By definition, f and F have the following expressions:

$$f(x) = x^2 \mod N \quad (\text{Hash 1})$$

$$F(x, y) = x^2 * y \mod N \quad (\text{Hash 2})$$

It can be proved that $F(x, f(y)) = f(x * y)$.

3.7.2 Performance Analysis

In order to evaluate the authentication protocol in term of the speed, we have performed the following experiments for testing.

- Measurement of the authentication time with different PCs
- Measurement of the on-card and off-card computation times in the authentication
- Measurement of the authentication time with different servers
- Measurement of the scalability of the authentication server

Experiment 1

In this experiment, we have tested the authentication time in different machines. We have tested the authentication protocol with 333MHz PC, 500MHz PC and 667MHz PC. Figure 3.6 shows the corresponding results and table 3.2 shows its corresponding statistical results.

From the observation, the faster the speed of the CPU is, the faster the authentication speed is. In general, the authentication can always be completed within 3100 ms. However, the authentication time does not decrease linearly with the speed of the CPU. Therefore, we measure the on-card and off-card computation time in experiment 2 and measure the effect of the sever to the authentication time in experiment 3.

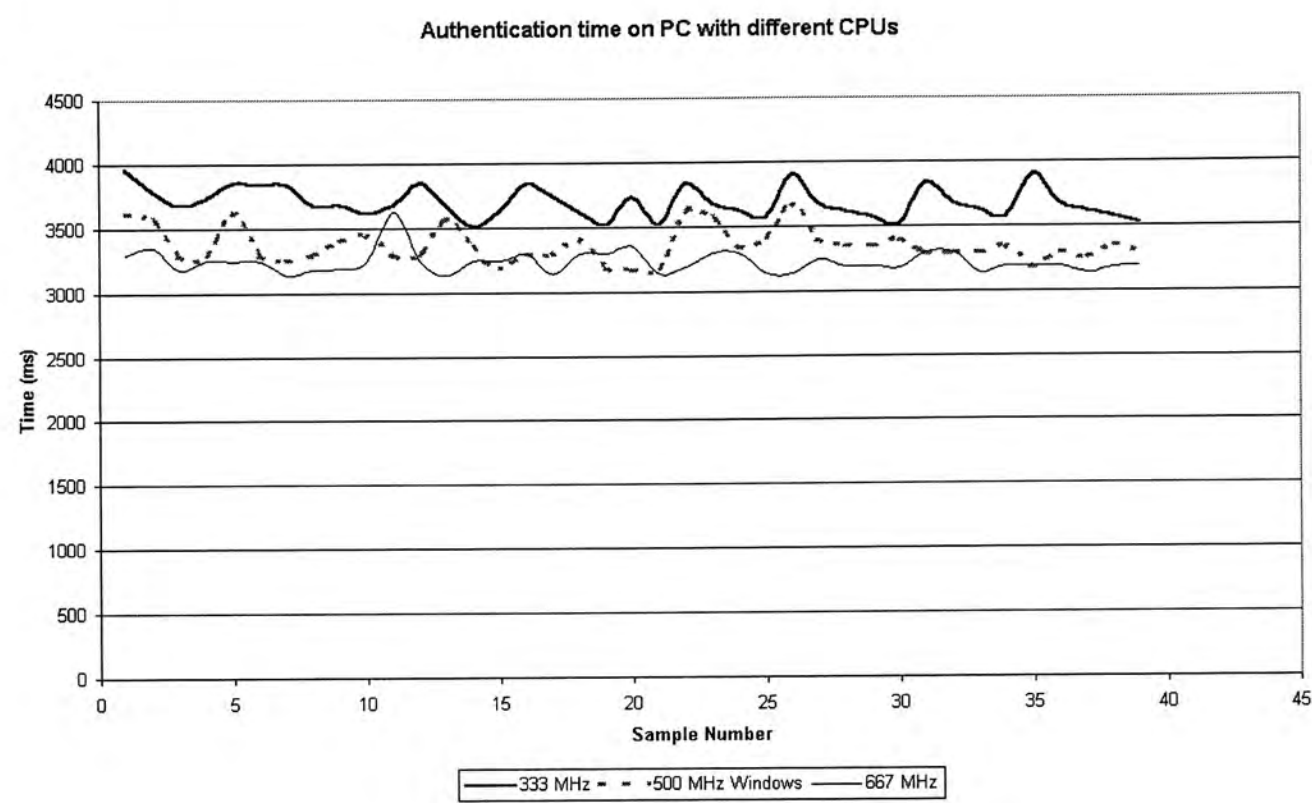


Figure 3.6: Authentication time with different hosts

Host CPU (MHz)	333MHz	500MHz	667MHz
OS	Windows 98 SE	Windows 98 SE	Windows 98 SE
Average Time (ms)	3934	3801	3229
Standard Deviation	283	73	93
Maximum Time (ms)	5060	3900	3630
Minimum Time (ms)	3680	3670	3130

Table 3.2: Statistical result of the authentication time with different hosts

Experiment 2

In this experiment, we have measured the on-card and off-card computation in each authentication. The client host has 667MHz CPU and 128M RAM running on windows 98 SE platform. Figure 3.7 shows the experimental result

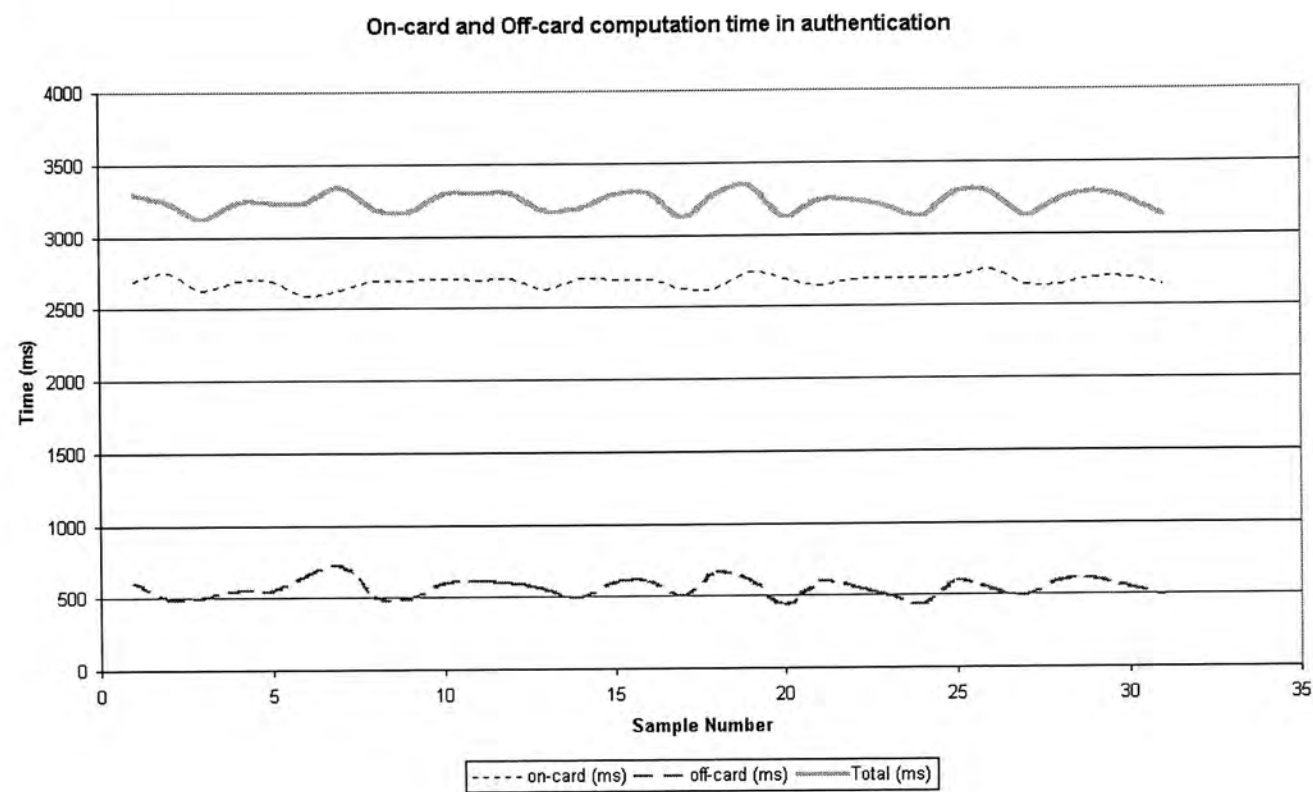


Figure 3.7: On-card and off-card computation time in authentication

in graph and table 3.3 shows the statistical result of this experiment.

Obviously, the off-card computation time only takes 17.2% of the total authentication time while the on-card computation time is 2677 ms, which takes 82.8% of the total authentication time. In other words, however fast the client and server CPUs are, the fastest time cannot be less than 2600 ms. The reason for the long on-card computation time is because of the relatively slow speed of the optimized RSA signature generation and verification. The software implementation of the one-way hash function f also introduces certain degree of time overhead.

	On-card	Off-card	Total
Average Time (ms)	2677	558	3235
Standard Deviation	38	67	69
Maximum Time (ms)	2750	720	3350
Minimum Time (ms)	2590	440	3130

Table 3.3: Statistical result of the on-card and off-card authentication time

Experiment 3

In this experiment, we have evaluated how the performance of the authentication protocol varies with the authentication server. Figure 3.8 shows the experimental results and table 3.4 shows the corresponding statistical results. The client host has 500MHz CPU and 128M RAM running on windows 98 SE platform.

It is observed that the performance of the server would not change the authentication time significantly and can be negligible when the server CPU has speed more than 333MHz as the authentication times on the 333MHz server and 12400MHz server are very close in average with less than 0.001% fluctuation.

There are two reasons for this observation. First, most of the time is spent on the card and the off-card computation is negligible. Second, the most time consuming operations are RSA signature generation and verification which have already been optimized. The performance results are similar on the server with CPUs faster than 333MHz.

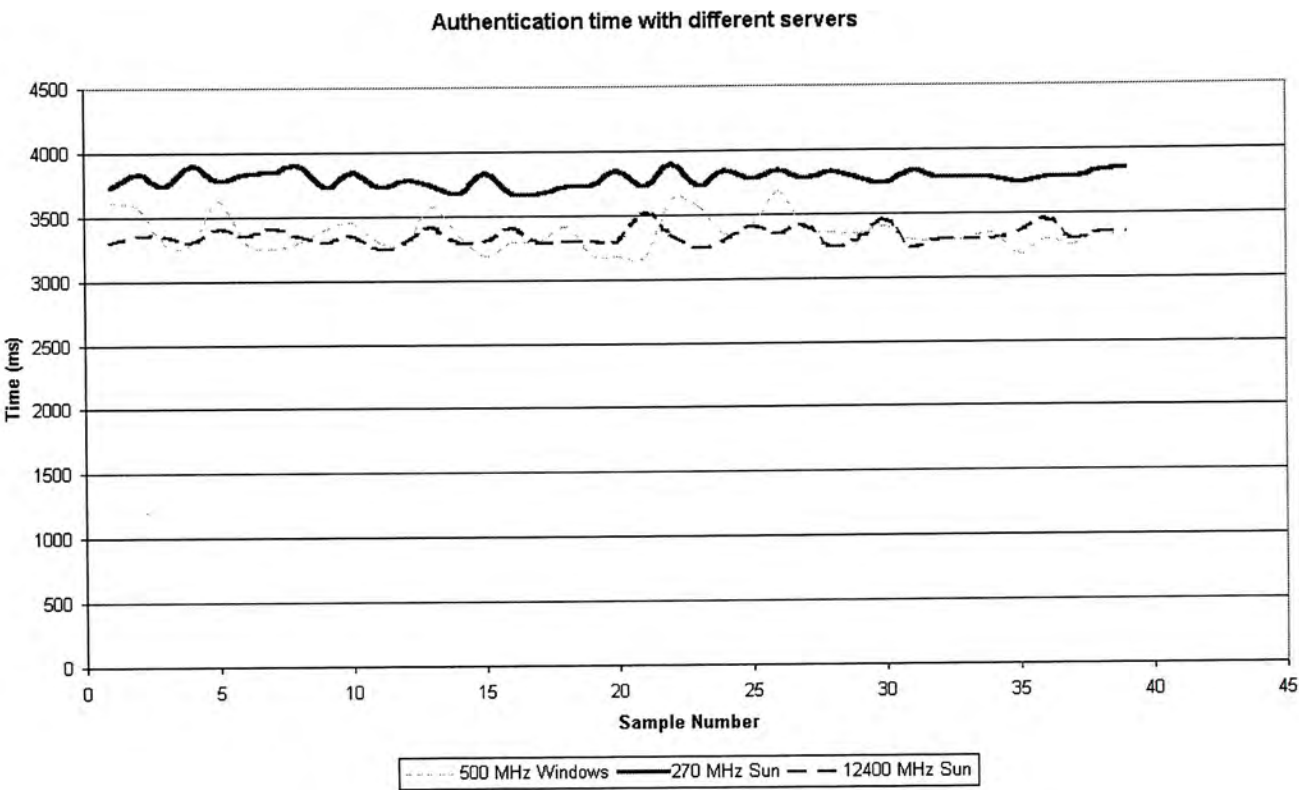


Figure 3.8: Authentication time with different servers

Server CPU (MHz)	333	270	12400
Server OS	Windows 98 SE	Solaris 2.6	Solaris 7.0
Average Time (ms)	3361	3791	3335
Standard Deviation	136	61	65
Maximum Time (ms)	3680	3900	3520
Minimum Time (ms)	3180	3680	3240

Table 3.4: Statistical result of authentication time with different servers

Experiment 4

In this experiment, we determine the scalability of the authentication server by simulating a number of authentication request from clients. A number of authentication request is sent to the server at the same time and we record

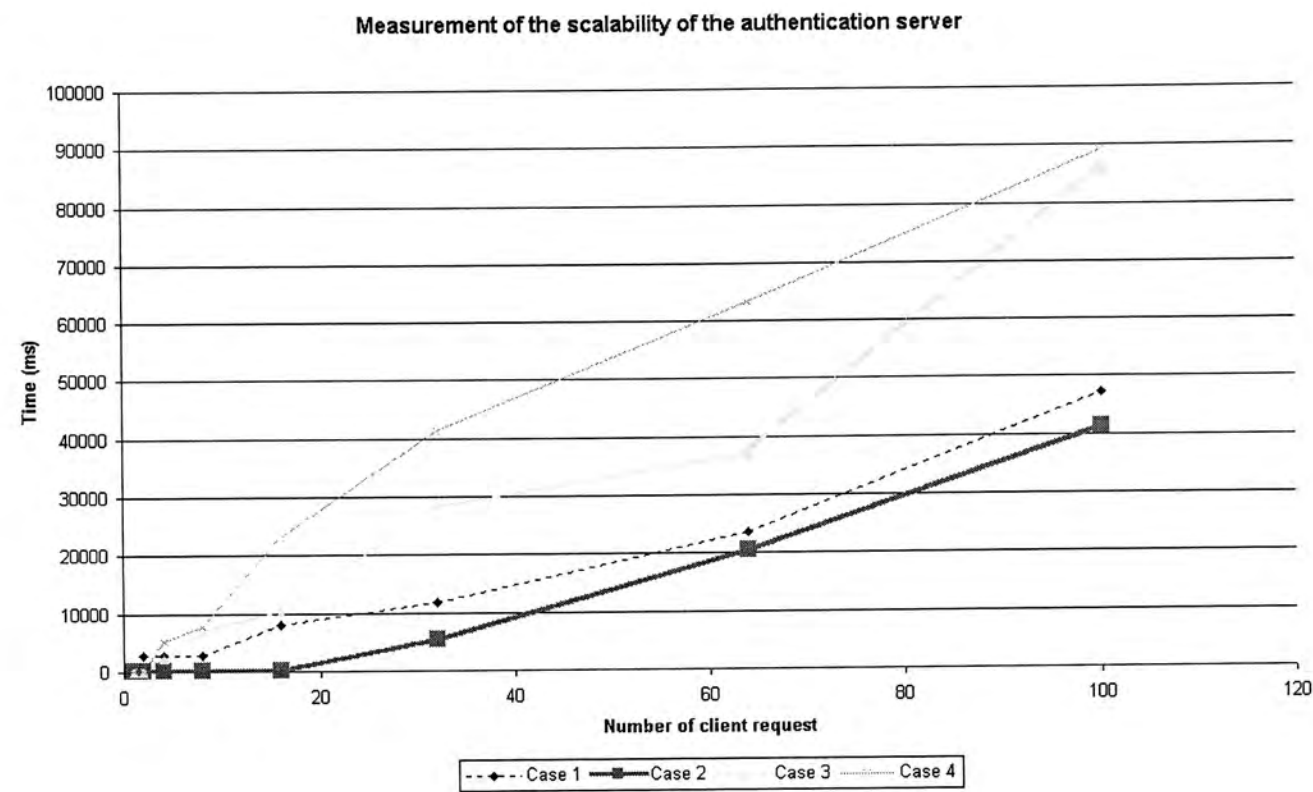


Figure 3.9: Authentication time with different servers

the time to complete all requests by the server. Four cases are considered as shown in table 3.5 (the row indicates the client while the column indicates the server):

	SunUltra 5/270	Sun Enterprise E4500
SunUltra 5/270	case 1	case 2
Sun Enterprise E4500	case 3	case 4

Table 3.5: Cases to be studied

In general, the time to complete all the authentication requests grows linearly with the number of the request.

Chapter 4

Transaction Protocol

Transaction protocols handle the exchange of payments and products. Before the transaction protocol can be initialized, the negotiation protocol should be completed for making the compromise on the payment and product between the involved parties. On the other hand, the product delivery protocol will be started when the transaction protocol is done.

The study of the micro-payment transaction is motivated by the high cost of credit card transactions. Practically, credit card transactions are only applicable when the amount of money involved is large because of the relatively high cost for each transaction. It is also inappropriate to perform credit card transactions on the web-based system in case that the network is insecure or the credit card information is misused by the seller.

The transaction protocol should have the following characteristics:

- Low cost for micro-payments
- Privacy and anonymity of the parties
- Recoverable transaction if it is broken
- Appropriate conflict resolution policy based on digital signatures
- The seller cannot find the identity of the buyer

- Money can be transferred amongst the buyer, seller, bank and application service provider
- Secured by smart cards and PKI

In this chapter, we introduce the proposed transaction protocol based on our authentication protocol. We first state the assumptions and list out the protocol in detail. The conflict resolution, justification and evaluation are outlined in the rest of this chapter.

4.1 Assumptions

As the transaction protocol is based on the authentication protocol, it is assumed that the authentication protocol is completed successfully and a triple DES key is established between the card and payment gateway, which is built on top of the authentication server. Hence, the transaction message is encrypted and authenticated.

The scenario for the transaction is shown in figure 4.1. The payment gateway handles the transaction request from the buyer. The seller server can handle the request from the payment gateway and buyers.

Additionally, we assume that the transaction record has different fields as shown in figure 4.2. *time* stores the time of the transaction. The identity of the seller is stored in *merchantCode* while the product ID is stored in *productCode*. *note* describes the product in detail. Moreover, *group* is used for the product classification. For handling recovery, *isComplete* is used to indicate whether this transaction has been completed or not while *isEmpty* is used to indicate whether this record is empty or not.

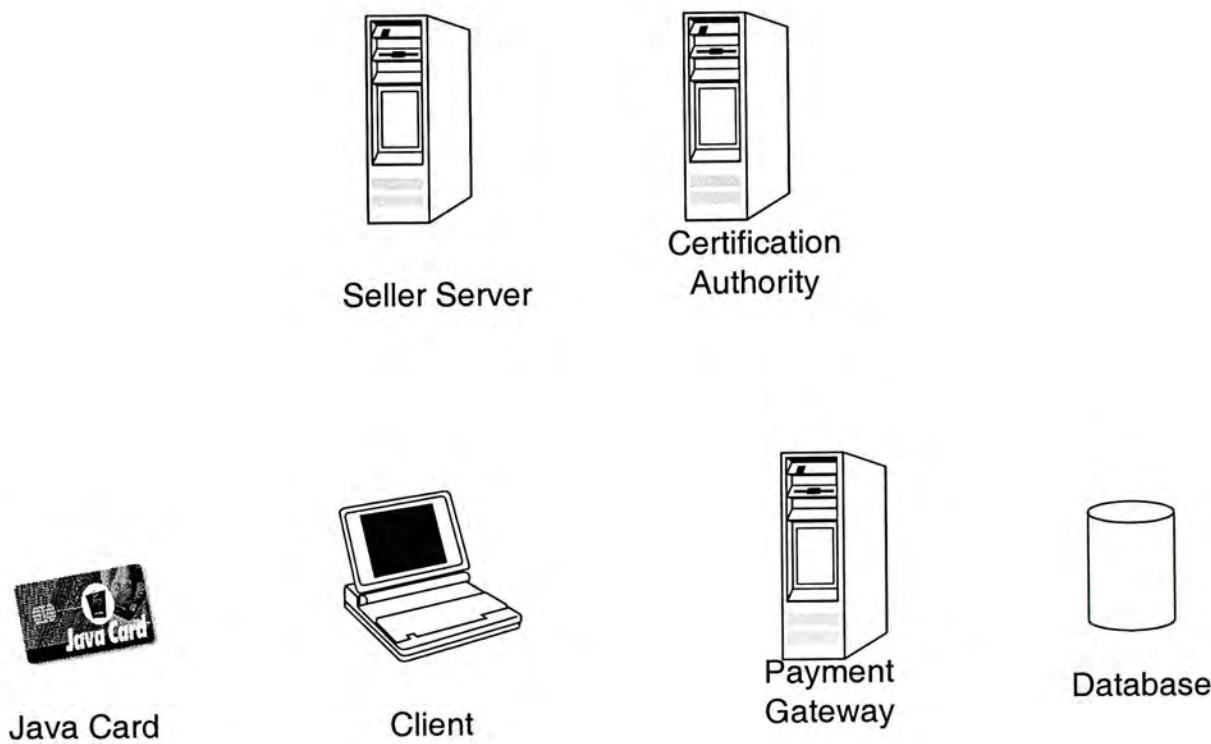
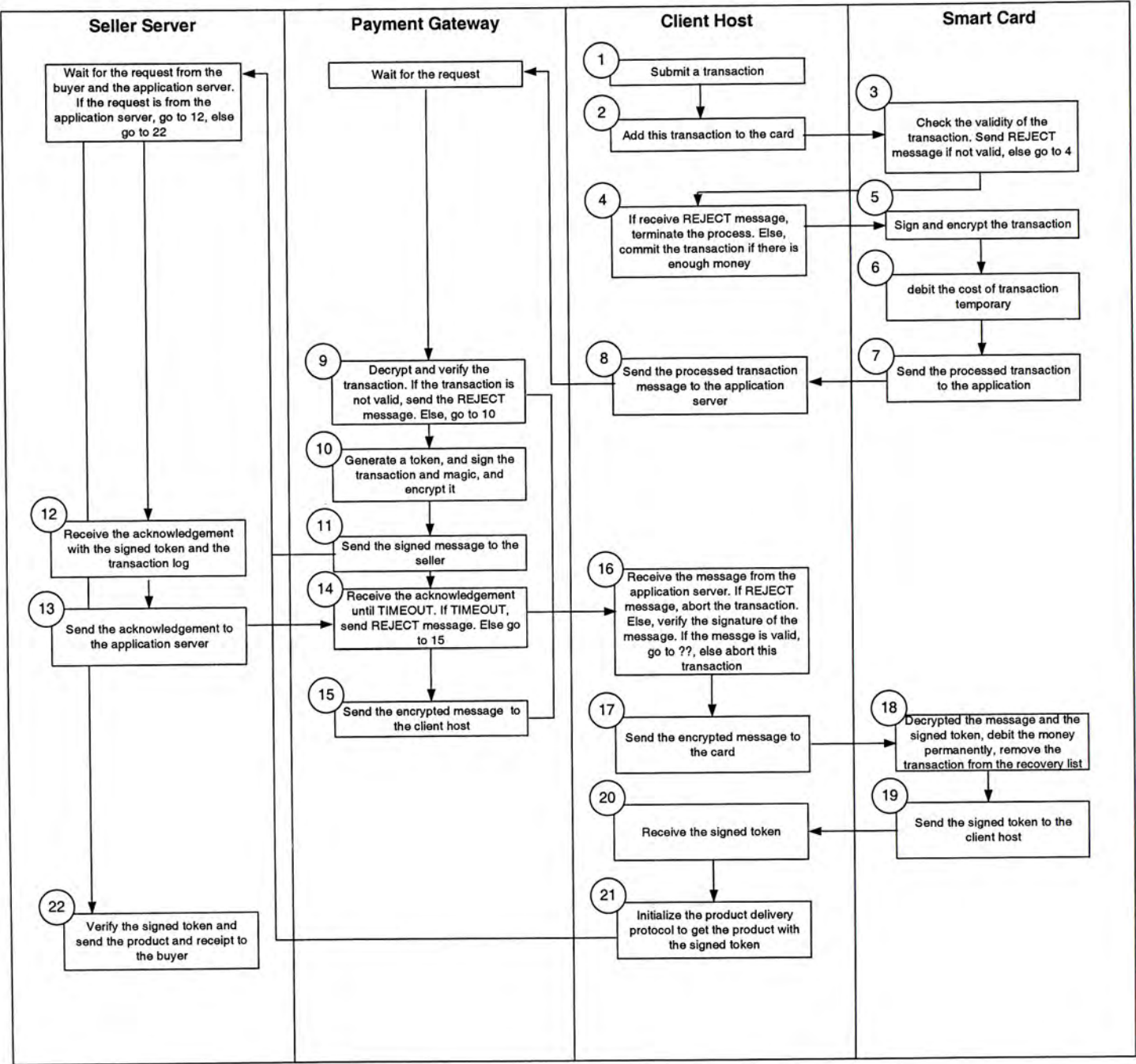


Figure 4.1: Testing environment

```
class Transaction
{
    private Calendar time;
    private String merchantCode;
    private String productCode;
    private int amount;
    private String note;
    private String group;
    private boolean isComplete;
    private boolean isEmpty;
}
```

Figure 4.2: Class of Transaction



Note:
1. It is assumed that the the local and remote authentication have been performed sucessfully.
2. Transaction record has fields TIME, SELLER, PRODUCT, DESCRIPTION, GROUP, COST

Figure 4.3: Flow diagram of the transaction protocol

4.2 Protocol

Intuitively, the buyer and seller compromise on the product and payments by the negotiation protocol. Then, the buyer initializes the transaction protocol and makes a request to the payment gateway. The gateway sends an acknowledgement to the seller, who then sends another acknowledgement to the gateway. After receiving the acknowledgement from the seller, the gateway sends the signed token to the buyer. The buyer can use the signed token to initialize the product delivery protocol for getting the product and receipt.

Figure 4.3 shows the flow diagram of the transaction protocol. The sequence of the transaction protocol is indicated in the circle on the top left corner of each text box in the figure. The following notations are deployed in this section:

- K_{AH} denotes the secret key between the applet and host.
- K_{HS} denotes the secret key between the host and payment gateway.
- K_{AS} denotes the secret key between the applet and payment gateway.
- K_{Seller} denotes the public key of the seller server.
- K_S denotes the public key of the payment gateway.

Transaction protocol is initialized by the client in the host. When the client wants to buy a desired item, the client host gets the transaction record for this item in step 1. Next, the host sends the transaction record $M1$ to the applet on the card in step 2.

$$Host \rightarrow Applet : M1 = \{Transaction\}_{K_{AH}}$$

After checking the validity of the transaction, the card sends the encrypted and signed transaction record $M2$ to the remote payment gateway via the

client host in step 8. At the same time, money is temporary deducted from the card in step 6.

$$Applet \rightarrow Gateway : M2 = \{\{Transaction\}_{K_A^{-1}}\}_{K_{AS}}$$

The payment gateway verifies the signature and the validity of the product record in the received transaction record. If the record is verified successfully, the gateway will generate a unique token *Token* which is a signed message with the transaction record, acknowledgement from the seller and a random generated number. Message *M3* is sent to the seller server without the seller acknowledgement. Then, the seller server sends the signed acknowledgement to the gateway. After receiving the acknowledgement from the seller server, the gateway sends message *M4* to the client host. The token is used to allow the seller server and client host to identify each other in the product delivery protocol. When the host receives the purchase acknowledgement from the gateway in step 16, the debit operation can be committed ultimately in step 18 and the applet can send the token to the host .

$$Gateway \rightarrow Seller Server : M3 = \{\{Transaction, Token\}_{K_{S-1}}\}_{K_{Seller}}$$

$$Gateway \rightarrow Applet : M4 = \{\{Transaction, Token\}_{K_{S-1}}\}_{K_{AS}}$$

Concurrently, the token will be verified and the local token database will be updated when the seller server receives the token from the gateway in step 12. Finally, the client host uses the received token to get the product from the seller in the product delivery phase.

Transaction Recovery

In case of any abnormal transaction termination before the commitment of transactions in the applet, money that is temporary deducted from the card would be restored if the client chooses to cancel the transaction. Otherwise, the transaction would be resumed and money would be debited from the card permanently once the transaction is completed successfully. Consider different scenarios listed as follows:

- Before the permanent deduction of money (before the execution of step 18)

In case of the abnormal terminal of the protocol, money that is *temporarily* deducted in the card is restored. For the protocol termination before step 18, the gateway and seller server can ignore the uncommitted transaction since the gateway cannot send *Token* to the applet. Moreover, as the seller server cannot show the signed *Token* by the buyer to the gateway, the seller cannot earn any money. Even before step 18, the client host still cannot get *Token* to retrieve the product from the seller server since it is encrypted with the secret session key. In other words, the buyer will not pay for the product and the seller does not deliver the product to the buyer.

- After the permanent deduction of money (after the execution of step 18)

Up to step 18, it is guaranteed that the seller server has received *Token* from the gateway and the applet has received *Token*. Money is deducted *permanently*. For any subsequent termination of the protocol, the buyer can restore the broken transaction later and get the product from the seller. It should be noted that the execution of step 18 is atomic, i.e. step 18 is either executed successfully or uncommitted.

4.3 Conflict Resolution Policy

In case there is a conflict among the seller, buyer and application service provider, the following conflict resolution policy can help to resolve the dispute.

Several conflicts are considered:

- Case 1: The buyer cannot deny a transaction if the gateway can show the request from the buyer and the seller can show the signed token from the buyer.
- Case 2: The seller cannot deny a transaction if the gateway can show the acknowledgement from the seller and the buyer can show the acknowledgement from the seller in the signed token.
- Case 3: The gateway cannot deny a transaction if the seller can show the acknowledgement from the gateway and the buyer can show the signed token from the gateway.

4.4 Justifications

Confidentiality

It is trivial to show the confidentiality of the transaction protocol since the message transmitted in the protocol is secured with the triple DES key to keep the message from being read by other parties and signed by the appropriate RSA key.

Non-repudiation

By utilizing the digital signature and PKI, the transaction is non-repudiated. Messages must be signed by the corresponding party for making it effective. Therefore, any parties can resolve the conflict according to the conflict resolution policy described before.

Privacy and Anonymity

To make the protocol practical in the real world, the transaction protocol supports *partial-anonymity*. During the transaction, the gateway does not send the identity of the buyer to the seller server. The seller would identify the buyer by the signed *Token* instead. On the other hand, only the buyer and the gateway knows all identities of the involved parties.

4.5 Experimental Evaluation

Several experiments are done to evaluate and analyze the performance of the transaction protocol so as to find the possible improvements.

- Measurement of the transaction time on different hosts
- Measurement of the on-card and the off-card computation time in the transaction
- Testing of the recoverability of the transaction protocol

4.5.1 Configuration

The development environment is based on GemXpresso rapid application development toolkit 211 version 2.4 with the GemXpresso 211PKis Java card. This Java card is implemented according to Java Card 2.1 API and Java Card 2.1 VM [31] [11] with triple DES and RSA algorithms [57].

Different APIs are used for writing the test programs. They include Java Core API 1.2.2, Java Card 2.1, Java Cryptography Extension 1.2.1, Java Secure Socket Extension 1.0.2, OpenCard 1.2, Gemplus Card Service and Open Platform.

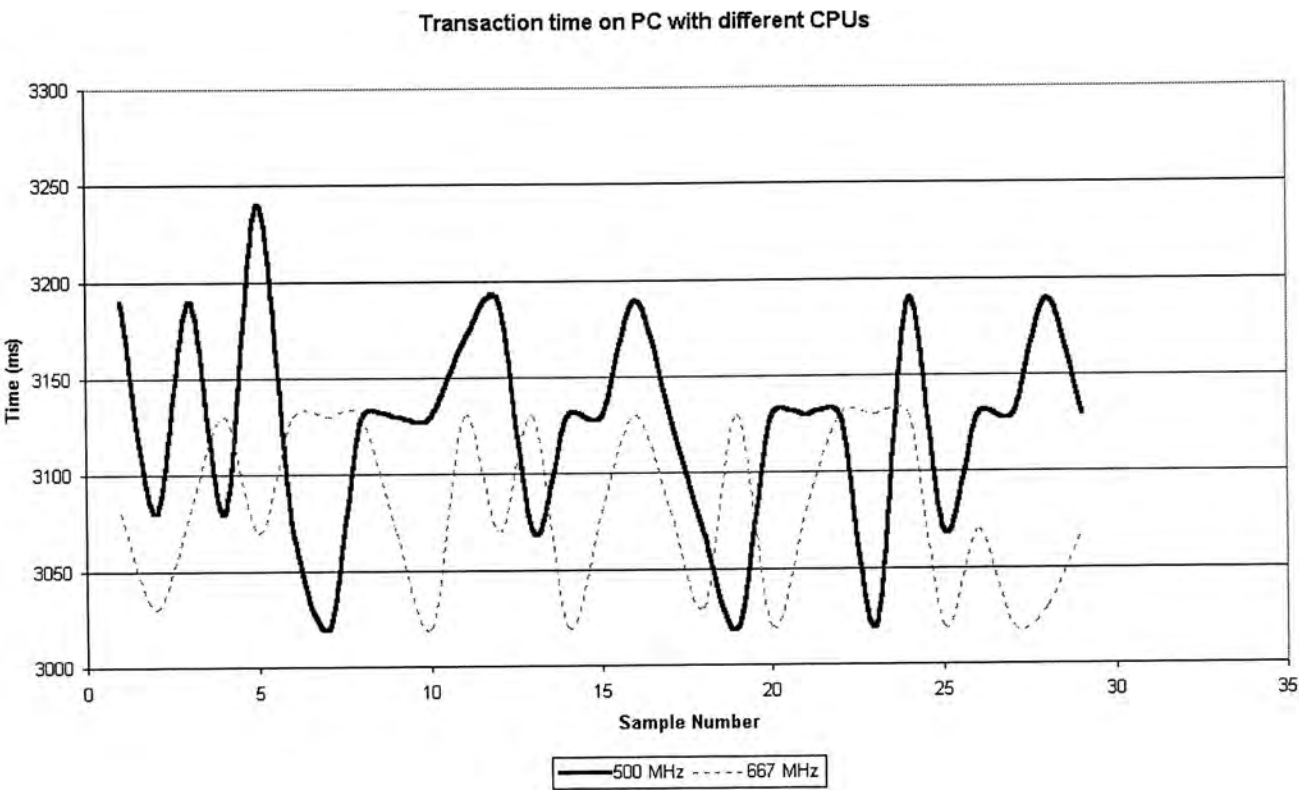


Figure 4.4: Transaction time with different hosts

4.5.2 Performance Analysis

Experiment 1

In this experiment, the transaction times are tested on the hosts of different speeds. The average transaction time is around 3000 ms which is similar for both 500MHz and 667MHz PCs. The performance of the client host and server cannot affect the transaction protocol significantly. Therefore, we evaluate the on-card and off-card processing times in the experiment 2.

Server Host CPU (MHz)	500MHz	667MHz
Server OS	Windows 98 SE	Windows 98 SE
Client Host CPU (MHz)	500MHz	667MHz
Client OS	Windows 98 SE	Windows 98 SE
Average Time (ms)	3124	3082
Standard Deviation	57	44
Maximum Time (ms)	3240	3130
Minimum Time (ms)	3020	3020

Table 4.1: Statistical result of the transaction time with different hosts

Experiment 2

In this experiment, the on-card and off-card computation times are measured. The testing machine has 667MHz and 128M ram running on the windows 98 SE platform.

It is found that the on-card processing time is longer than the off-card processing time taking almost 70% of the total. However, the ratio of the on-card to the off-card processing time is less than that in the authentication. The extra-long off-card processing time is induced by the communication cost of the payment gateway with the seller server and application server.

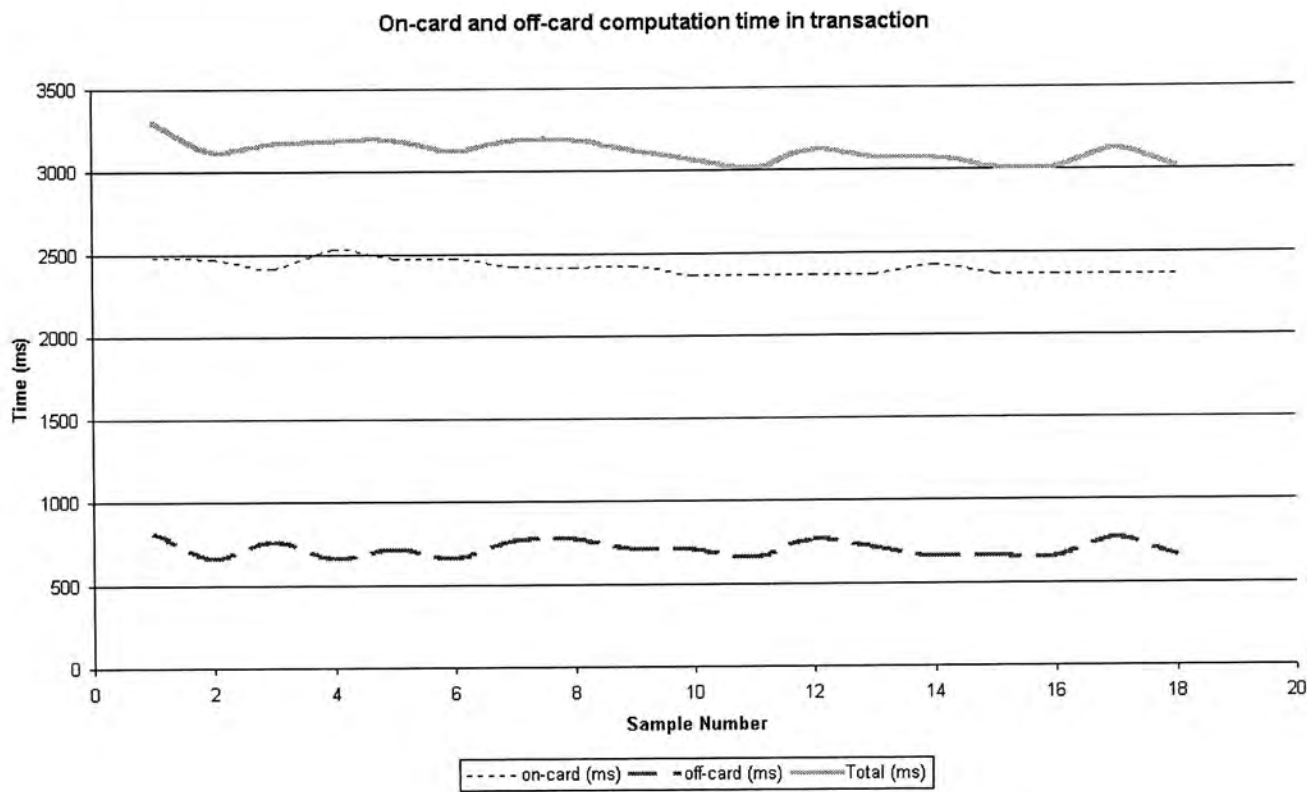


Figure 4.5: The On-card and the off-card computation time in the transaction

	On-card	Off-card	Total
Average Time (ms)	2410	712	3122
Standard Deviation	54	55	77
Maximum Time (ms)	2530	820	3300
Minimum Time (ms)	2360	660	3020

Table 4.2: Statistical result of the on-card and the off-card computation time in the transaction

Experiment 3

Experiments are done to test the recoverability of the transaction protocol. We break the running transaction and resume the broken transaction. Considering the single point of failure in the protocol, we terminate the normal operations

of the card, host, gateway and seller server and evaluate the recoverability of the protocol.

- Stop the card

Any termination in steps 3, 5, 6 and 7 enables the silent termination of the transaction. The broken transaction can be resumed in the next login session. Otherwise, the card holder can cancel the transaction without affecting the other parties and the temporary deducted money can be restored. After the execution of step 18, it is guaranteed that the card holder can get the product even the subsequent termination of the protocol. The card holder can request the delivery of the product later.

- Stop the host

If the transaction is broken in steps 1 or 2, it will not affect other parties since the transaction is not submitted to the card. If the transaction is broken in step 8, the card can restore to the original state or resume the broken transaction in the next login session. If the transaction is broken in steps 16 or 17, the card can do the same as before. The application server and seller server do not suffer from any loss for the storage of an invalid record. The card holder can execute steps 20 and 21 until the product is retrieved from the seller server.

- Stop the payment gateway

If the transaction is broken in steps 9, 10 or 11, the host and card can restore the transaction later; the seller server operates normally as the broken transaction record is not received. For the termination in steps 14 or 15, the host and card can restore the broken transaction as before. Alternatively, it is not harmful to the seller server since the card cannot obtain the *Token* for the retrieval of the product.

- Stop the seller server

Considering the broken transaction in steps 12 or 13, the payment gateway can detect this condition by using the acknowledgement from the seller server. The gateway can send the *REJECT* signal to the host and card to recover from this broken transaction. If the transaction is broken in step 22, the seller server can use the sophisticated product delivery protocol to provide a reliable product delivery. The buyer can execute steps 18 and 19 until he receives the desired product.

To conclude, the abnormal termination of the transaction can always be recovered successfully, and the payment can always flow correctly.

Chapter 5

E-Commerce Builder

Nowadays, smart cards are regarded as the promising enabler of the E-commerce system. Smart card systems have a stronger resistant to reverse engineering compared with the mandatory approach that the business logic resides in the client host. Nonetheless, the development of smart card based e-commerce systems is heavily dependent on the knowledge of smart card operating systems, security and different advanced application-programming interfaces. Understanding the smart card operating systems [32], namely file system smart card, Java Card, Multos and smart card for windows, is not enough for an application developer to develop a smart card based system. It is also essential to understand the OpenCard architecture [31] [41] [47] to interface with the smart card and card terminal. To familiarize oneself with the cryptographic theory [53] [61] [14] [13] and security-related application programming interfaces [39] [50] [69] is also a formidable task. Altogether, the acquisition of this knowledge definitely lengthens the development period and significantly raises the cost of the development. Otherwise, a fragile system may be built without carefully understanding them.

This chapter proposes the first innovative solution for the rapid application development (RAD) of smart-card based B2C & P2P E-commerce applications. We have developed a builder with all the necessary resources for these types of application developments. With our builder, it helps to speed up the

development time effectively. Our builder supports strong authentication services and reliable transaction services with service recovery. The flexibility of our builder also facilitates the application developer to customize their applications according to their requirements. Section 5.1 gives an overview of the builder. The underlying mechanism and the detailed implementation of the builder are described in sections 5.2 and 5.3 respectively. A comparison with other similar products is outlined in section 5.4 and an example of the usage is shown in section 5.5.

5.1 Overview

Smart RAD has a number of important features, which are radical to the ordinary B2C and P2P e-commerce applications:

- Strong Authentication Service
- Reliable Transaction Service
- High Customization Capability
- Accountability
- Standardized Public Key Infrastructure

Strong Authentication Service

Unlike most of the e-commerce applications, the authentication service is based on what the card holder has and what the card holder knows. Nowadays, simple password based authentication systems cannot cope with the stringent security requirement of many e-commerce systems. Our authentication protocol [72] is intended to ensure the confidentiality, non-repudiation, anonymity and authentication of the services provided. Different from other smart card based authentication services [7] [9] [35] [38], it can perform mutual authentication,

and handle the failure of smart cards or the authentication server properly as mentioned in chapter 3.

Reliable Transaction Service

Transactions occur frequently between the application service provider and all card holders. Hence, we offer the transaction service that enables the developer to deploy, and the non-repudiated transaction protocol such that the conflict between the application service provider and card holder can be resolved easily. The details of the transaction mechanism can be found in chapter 4.

High Code-level Customization Capability

Considering that different application service providers have different application logics, we allow the developer to embed their own application logic within our system framework. The developer is able to build their services based on the authentication service, and therefore, controls the interaction between the card and application server.

Security

With a view to providing a fast and secure communication channel between the application provider and card holder, the communication channel is protected with a triple DES [31] session key for each new login session. Triple DES with 112-bit key length means a very high level of security. It has stronger resistance to the linear and differential cryptanalysis [59] than DES .

Accountability

All the transaction and interactions with a bank are recorded and kept for audit trace. The application service provider and card holder can keep their own logs and use it for any possible conflict resolutions.

Standardized Public Key Infrastructure

The system is built on the Public Key Infrastructure (PKI). The certificate is created and signed with accordance to the X.509 standard. With this feature, the system can inter-operate with other PKI systems.

The card holder can take advantage of the RSA key [57] in the smart card to sign the sensitive data and use the triple DES session key for encryption and decryption. On the other hand, other card holders can retrieve the certificate of a particular user with the help of the certification authority and therefore, verify a signed message.

When a smart card is issued to a card holder, a pair of RSA keys is generated on the card. The private key is stored in the card while the public key is downloaded from the card and bound to a X.509 certificate, which would be stored in the certification authority.

5.2 Design of Smart RAD

With a view to offering a comprehensive solution for B2B & P2P e-commerce software development, the design of the builder is based on the principle that it can provide sophisticated e-commerce operations reliably and be customized easily by the developer. Also, the builder architecture should be extended and enhanced easily.

5.2.1 Mechanism

Figure 5.1 shows the system architecture of *Smart RAD*. It provides an interface for the developer to access the services provided by the Host layer and Java Card layer. Some utilities are offered to allow the developer to access and maintain all servers. In our case, the servers are the authentication server, certification authority and time server. The developers should also define their

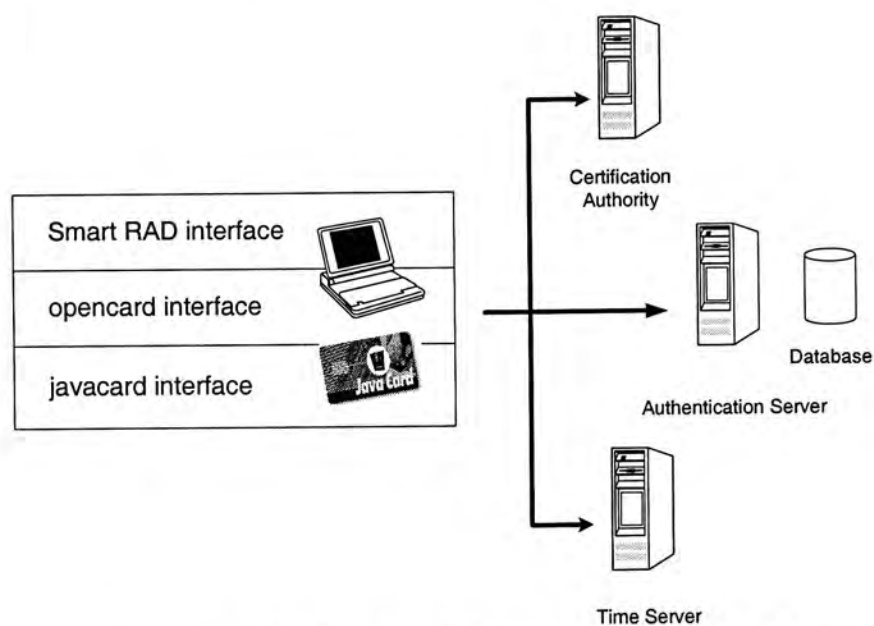


Figure 5.1: System Diagram of Smart RAD

database schema and build their own application server based on the authentication server. For the interests of brevity, some UML [28] diagrams are drawn as circles or the object attributes are not shown. The Smart RAD application builder is a software development toolbox containing the following three packages corresponding to the Java Card, Host and Server layers respectively.

- Package *cuhk.cse.demo.applet* in Java Card Layer
- Package *cuhk.cse.demo.client* in Host Layer
- Package *server* in Server Layer

The above three packages are detailed below.

5.2.2 Java Card Layer

Figure 5.2 shows the UML diagram of the objects in the Java Card layer. UML is the universal modelling language, which is used to describe the class formally with a diagram, and show the relationship with other classes. The objects in the Java Card layer manages the logic and information in the card. Package *cuhk.cse.demo.applet* is made up of a number of classes. The purposes of each class are listed as follows.

- UFOApplet

It is the central unit of the card. It initializes and manages all the information. It is also responsible for the communication between the card and card terminal.

- HashGenerator

It is used to generate a pair of hash function during the initialization of the card for the service subscriber. The generated hash functions f and F are used for the authentication.

- Hash

It is the logical representation of the mathematical hash function f . It is used for the authentication.

- SmartPurse

It is the on-card agent for the management of credit records. It can insert, delete and update records.

- Record

It is the logical representation of a bank credit record.

- TransactionManager

It is the on-card agent for the management of all the transaction-related operations such as insertion, deletion and updates of transaction records. The agent can keep the information for any conflict resolutions.

- Transaction

It is the logical representation of a transaction record. It would store the attributes of the transaction such as the status of emptiness and completeness.

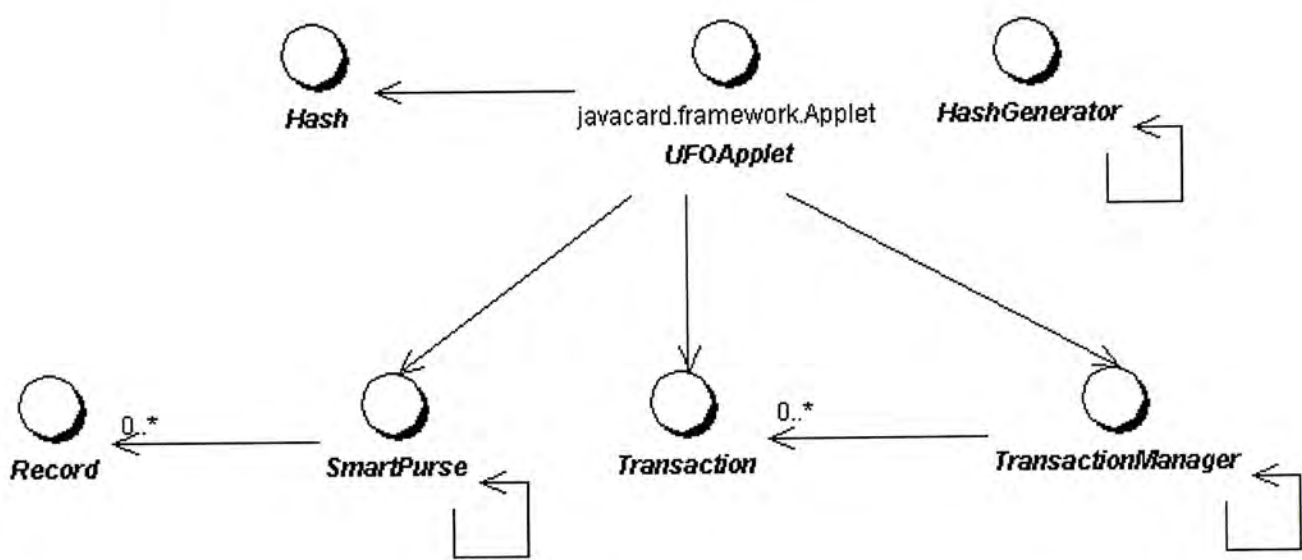


Figure 5.2: UML diagram of the objects in Java Card layer

5.2.3 Host Layer

Figure 5.3 shows the UML diagram of the objects in the Host layer. The Host layer forms an interface between the developer and card. The developer can retrieve, update, add or clear the information on the card via this layer. Here, we outline the function of each class in this layer. These classes correspond to the classes in the package *cuhk.cse.demo.client*. The function of each class in this layer is outlined as follows.

- ClientBase
It provides the basic facility to access the card applet.
- Install
It uploads the packages onto the card and initializes the card applet. It updates the information in the user database and certification authority.
- BigHash
It is the logical representation of the mathematical hash function F . It is used for the authentication.
- CustomCardAndKeyGen

It is the utility class providing the facility to create and sign a certificate for a specified user by a specified issuer.

- CardUtility

It is the utility class that provides an interface for the developer to access the function of the card. The developer must use it to perform the functions such as the generation of the hash function and a pair of RSA key, as well as the query of transactions and banking records.

- CAUtility

It is the utility class that allows the developer to retrieve the certificate in the certification authority server.

- TimeUtility

It is the utility class that allows the developer to access the time server to retrieve the current GMT time.

- Transaction

It is the logical representation of the transaction record.

- BankRecord

It is the logical representation of the bank credit record.

5.2.4 Server Layer

Figure 5.4 shows the UML diagram of the objects in the utility package. In general, the utility package provides the services related to the certification authority, authentication server (application server) and time server.

- Certification Authority Server

The certification authority has the certification repository that can add, update and distribute the certificates it stored.

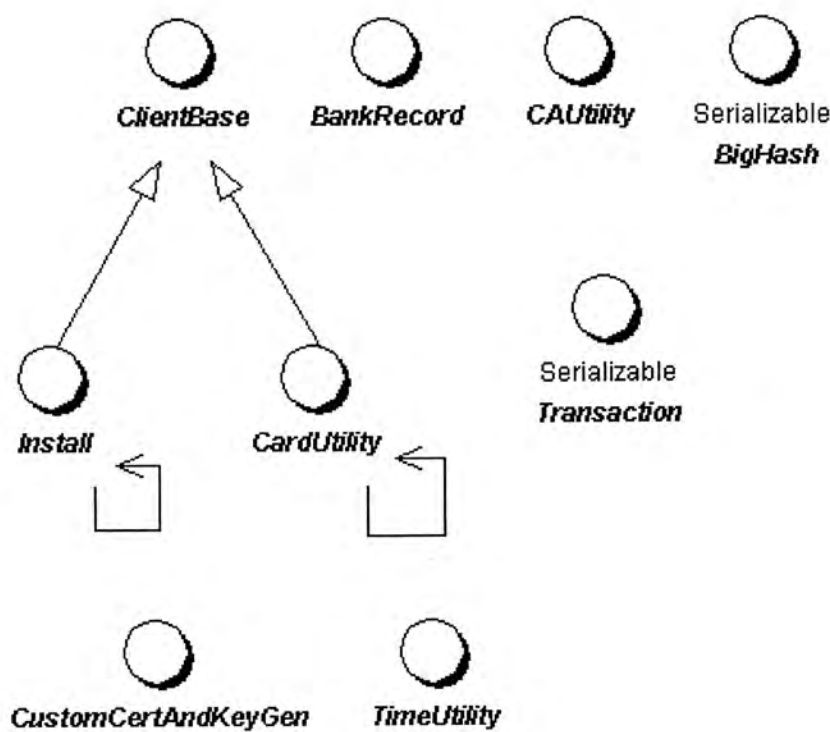


Figure 5.3: UML diagram of the objects in Host layer

- Time Server
The time server can distribute its GMT time.
- Authentication Server
The authentication server can perform the remote authentication protocol with remote clients and establish a secret triple DES key for subsequent communications with the top-level application layer in the client side.

5.3 Implementation

We describe the implementation details of the builder from the development tools to implementation issues. The proposed implementation is cross-platform.

5.3.1 Implementation Reflection

The following softwares/hardwares are deployed for building the builder:

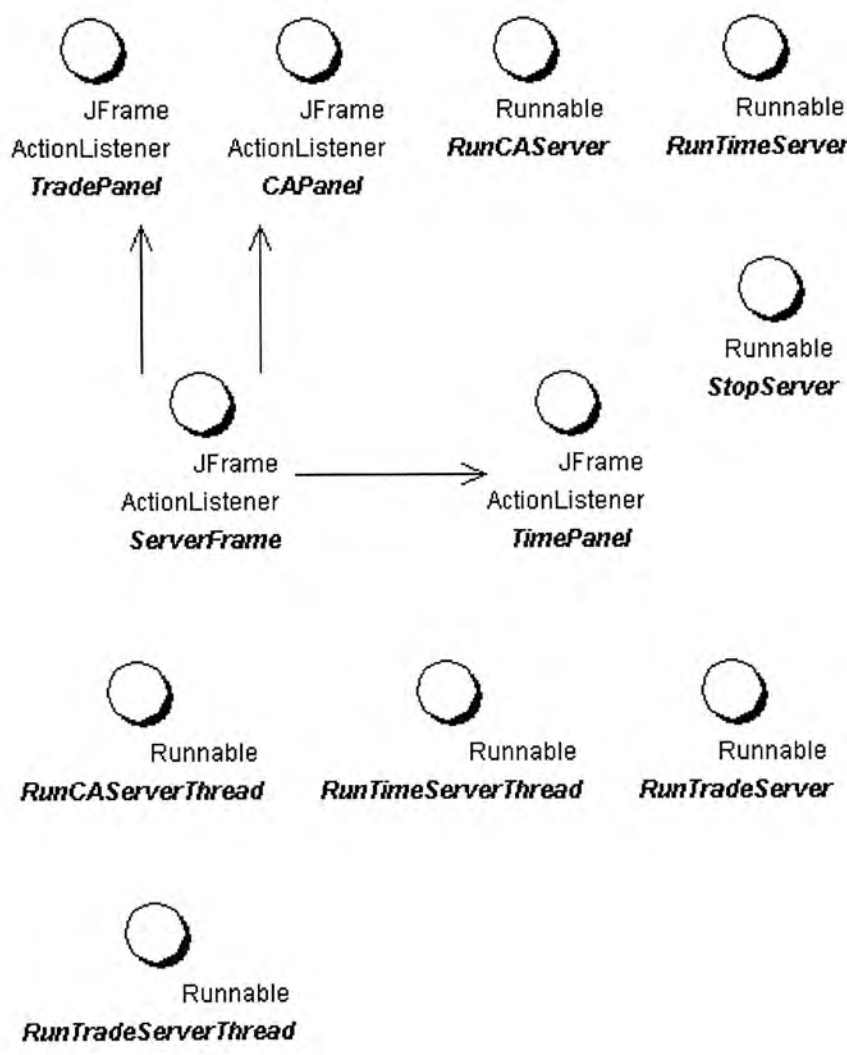


Figure 5.4: Utility Services

1. GemXpresso RAD 211 version 2.4

It is the development builder [21] for building an application on Gemplus smart cards. It provides the Gemplus-specific application-programming interface [23] [25] [24].

2. Java Development Kit 1.2.2

It provides the necessary Java Virtual Machine and Application Programming Interface [65] for building Java applications.

3. Java Secure Socket Extension 1.0.2

It is a Java package [68] that enables secure Internet communications. It implements a Java version of Secure Sockets Layer (SSL) [46] and Transport Layer Security (TLS) protocols and includes functionality for data encryption, server authentication, message integrity, and optional client authentication.

4. Java Cryptography Extension 1.2.1

It is an application-programming interface [67] that provides a framework and implementations for encryption, key generation and key agreement, and Message Authentication Code (MAC). Encryptions including symmetric, asymmetric, block and stream ciphers are also supported.

5. OpenCard Framework

It is an application programming interface for bridging the communication between the card reader and card terminal.

6. GCR 410 Smart Card Reader

It is an external reader for serial port. The reader supports ISO 7816-1/2/3/4 memory/microprocessor card and accepts T=0 and T=1 protocols.

7. GemXpresso PKis smart card

It has a RSA crypto-processor and support triple DES algorithm.

5.3.2 Implementation Issues

In the implementation of the builder, the following problems are encountered and we describe how we resolve them.

1. Limitation of Java Card VM

Due to the memory constraint of the ordinary smart card, the Java Card VM only supports a subset of features of Java languages. Therefore, numbers of double, float, or long type cannot be manipulated.

2. Limitation of the cryptographic processor of the smart card

Smart cards cannot support all cryptographic algorithms. GemXpresso 211PKis card supports both triple DES and RSA. That means only one secret key algorithm and one public key algorithm can be used. The principle is to use the public key algorithm to establish a secret session key because the cost for using the public key algorithm is higher in general. It would be cost-effective and secure to adopt a hybrid approach.

3. Memory constraint of the smart card

The smart card always suffers from two kinds of memory-related problems. Lack of memory confines the functionality of smart card application and memory leakage can lead to fault in runtime. In our case, GemXpresso 211PKis card offers 32 K of ROM, 32 K of EEPROM and 2 K of RAM. Out of these resources, about 23 K of EEPROM and 0.8 K of RAM are available to application developers. Hence, we are particularly careful to instantiate an object. Reuse the object as much as possible and confine to the singleton pattern [36] if the object to be instantiated is large or resource-intensive.

4. I/O speed and bandwidth of the card terminal

The cost of the latency for communications between the card and card terminal should be considered. Most of the smart card readers use the

serial port as the terminal connection with the client host. Also, there is a constraint on the maximum size of the data to be transferred between them.

5. Type conversion between the card and card terminal

As the underlying transmission protocol between the card and card terminal only supports the transfer of byte stream, the object to be transferred between them should be converted to byte array. Hence, for these objects, they should have the constructor with the byte array as the input argument and the method *toByteArray()* to generate the corresponding byte array.

6. Implementation of PKI with open standard API

In order to build a generic PKI, we should not use the third party API for building the PKI. This introduces one difficulty since JDK cannot generate a certificate without using *keytool*. Even the package *sun.security* can only be used to issue a self-signed certificate. We therefore implement our own class for issuing self-signed certificates and issuer-specific certificates in accordance to X.509 standard [33].

7. Performance of the implementation

Since the implementation of this builder is based on the Java language. We should be particularly careful when dealing with I/O, graphical user interface control [71] and threading [39].

5.4 Evaluation

In this section, we compare our integrated solution with other smart card products. The products being compared with include CyberFlex Access [55], GemMobile Card Issuer [18], Gemplus Wallet [19], and GemUtilities [20]. Table 5.1 describes these products briefly and table 5.2 shows the comparison

of the builder with different similar products according to their functionalities and features. Analysis and experimental results of the authentication and the transaction protocols have been described in chapters 3 and 4 respectively.

Products	Description
CyberFlex Access	Cross platform development kit for multi-function cryptographic cards
Gemplus Wallet	Development kit for Smart card-based electronic wallet
GemUtilities	E-commerce software toolkit to automate repetitive tasks so customers can easily and quickly navigate and make purchases on the Internet
GemMobile Card Issuer	Personalization software tool enabling quick personalization of SIM cards in a secure environment

Table 5.1: Brief description of smart card development products

Products	Smart RAD	CyberFlex Access	Gemplus Wallet	GemUtilities	GemMobile Card Issuer
Local Authentication	Yes	No	Yes	Yes	No
Remote Authentication	Yes	No	Yes	No	No
Reliable Transaction	Yes	No	Yes	No	No
Personalization	Yes	No	Yes	Yes	Yes
Accountability	Yes	No	Yes	No	No
Code-level Customization	Yes	Yes	No	No	No
PKI Support	Yes	Yes	Yes	No	No

Table 5.2: Comparison of Smart RAD with other products

As shown in the table, most of the smart card products are only designed for local access, which is not suitable for the ordinary e-commerce applications. The e-commerce application always involves the remote access service. On the other hand, *Smart RAD* provides both local and remote authentications such that the system can be applied either locally or remotely.

As a matter of fact, products like Gemplus Wallet and GemMobile Card Issuer are limited to apply to the transaction related in-house application developments. Their features of personalization are also very restricted and definitely not appropriate for more sophisticated software development. By providing code-level customization, the application developer is able to build smart card based applications according to their requirements.

Unlike other products, our system can inter-operate with other secure services through the common public key infrastructure.

5.5 An Application Example: Multi-MAX

After understanding the internal structure of Smart RAD, we show how to deploy this builder in the design and implementation phase of P2P and B2C e-commerce application development. We illustrate it with Multi-MAX. The builder is used in the the development of the transaction and authentication modules of Multi-MAX. **Multi-MAX** is a large-scale online trading platform that is a P2P and B2C application. It allows the service subscriber to buy and sell product with other service subscribers. At the same time, the server subscriber can use the same interface to shop on web-based retail portals.

5.5.1 System Model

Figure 5.5 describes the system model of Multi-MAX. The seller and buyer are the service subscribers. The seller can sell an electronic product or a physical product by publishing the product list in the directory repository. The potential buyer can browse the item listed in the directory repository. Whenever a buyer locates a desired item for purchase, the buyer can negotiate with the seller in real time through audio/video conferences. Alternatively, retail portals can publish their products on the web and the buyer can also buy it.

Once the seller and buyer have made the deal, the payment is made from the buyer's smart card and deposited on the application server which is built on top of the authentication server. The seller can retrieve the payment later with his smart card or through a specific bank account. The service subscriber can use the same graphical user interface to access the trading services through a browser or a stand-alone application.

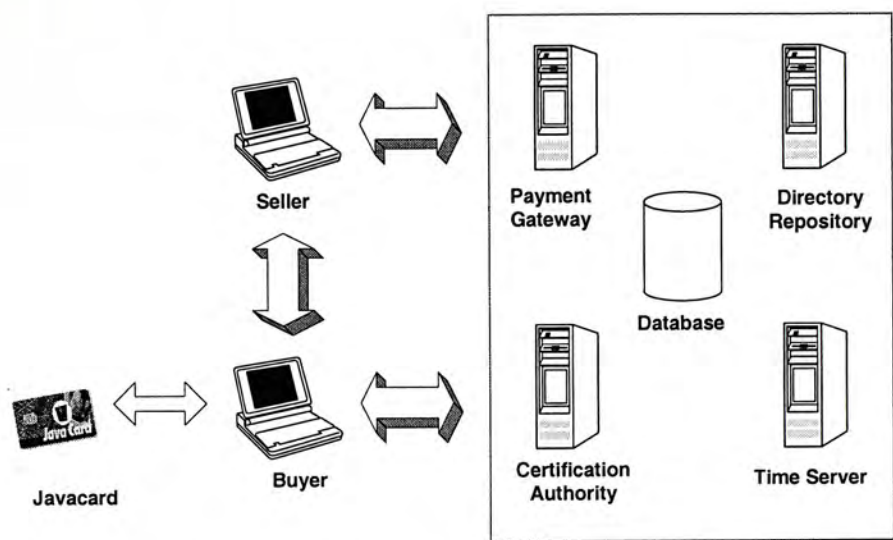


Figure 5.5: System Model of Multi-MAX

5.5.2 Design Issues

To deploy *Smart RAD* builder for building Multi-MAX, several issues need to be considered in the design phase.

- What kind of customized service would be built on top of the authentication server. For example, the developer can provide the services such as auction, gambling or trading.
- What information is needed during the initialization of the card for the service subscriber. In addition, the developer can perform other tailored initialization other than the initialization of the smart card.
- What utility servers need to be integrated with the system such as the time server, certification authority server or other developer's servers.

5.5.3 Implementation Issues

For the purpose of deploying the builder in the implementation of Multi-MAX, we demonstrate the procedures for implementing the system with four steps.

1. Extend class `cuhk.cse.demo.client.Install` and override method `initCard()`

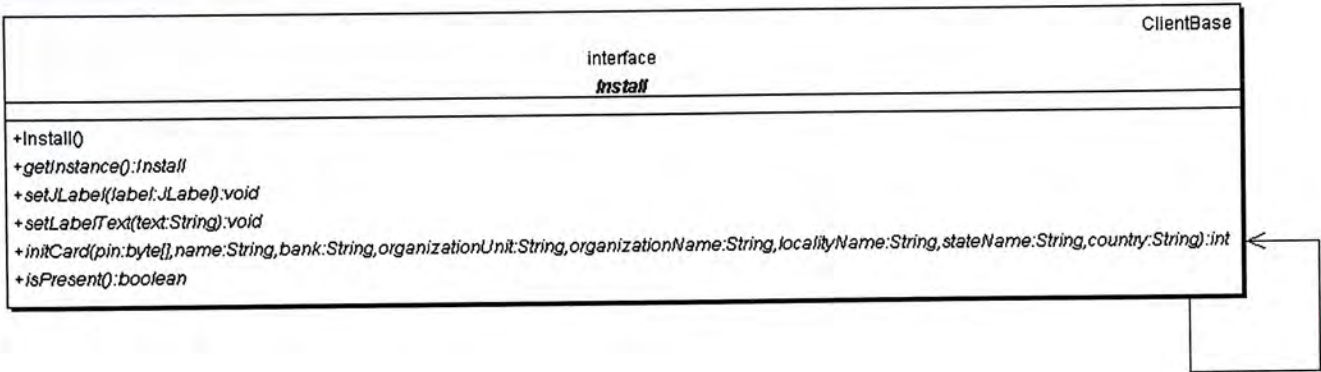


Figure 5.6: UML diagram of class Install

to perform the customized initialization. Figure 5.6 shows its UML diagram. The developer can perform the proper initialization with the card and user database.

- 2. Extend class *server.RunTradeServerThread* and override method *customized()*. The secret triple DES session key, input and output streams will be passed to the method as arguments. The developer can use the stream to communicate with the client while the triple DES session key, which is established during authentication can be used for encryption and decryption of messages sent between the client and server. Figure 5.7 shows the UML diagrams of class *RunTradeServer* and *RunTradeServerThread*. Finally, the registry of *RunTradeServerThread* in *RunTradeServer* should be updated with the classes that are extended from them.
- 3. Extend class *cuhk.cse.demo.client.CardUtility* and then the developer can add his customized methods that are corresponding to the developer's customized application server. With this, the developer can control the logic and information in the card. Figure 5.8 illustrates the UML diagram of the class *CardUtility*.
- 4. Extend the classes *server.RunTimeServerThread* and *server.RunCAServerThread*,

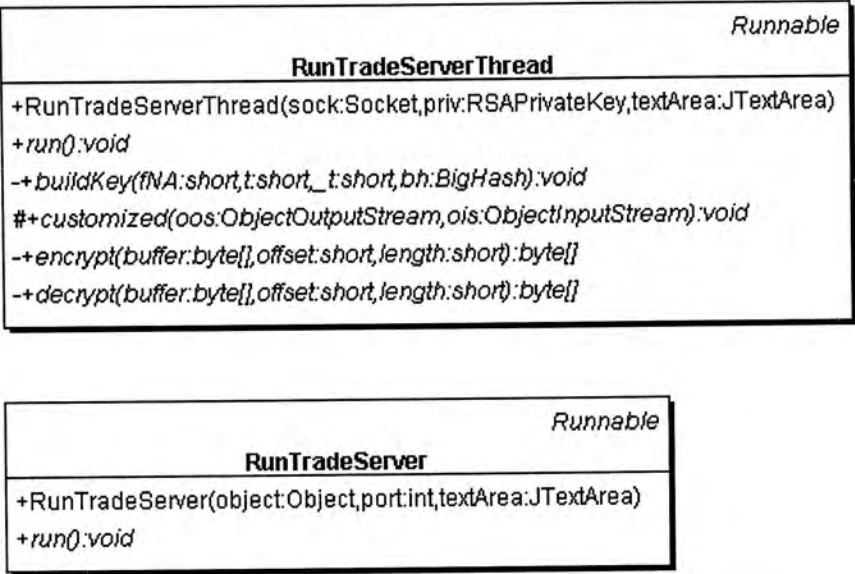


Figure 5.7: UML diagram of class RunTradeServer and RunTradeServerThread

and overrides methods *getCert()* and *getTime()* respectively if necessary so as to integrate the system with the developer’s certification authority server and time server. Figure 5.9 shows the UML diagram of these classes. The registry of classes *RunTimeServerThread* and *RunCAServerThread* in *RunTimeServer* and *RunCAServer* should be updated with the classes that are extended from them.

With the use of the Java Network Launching Protocol (JNLP) [56], the application can run alone or in a browser. Namely, it is promising that the service subscriber can access the services with the same interface in different environments. Figure 5.10 shows a sample JNLP file. To gain the advantage of JNLP, the web server should be JNLP enabled and the desktop of the service subscriber has to be installed with the Java Web Start client program [70] which is able to handle the JNLP request.

With Java Web Start, the subscriber can launch the application by simply clicking on a web page link. If the application is installed for the first time, it would be cached on the user computer. The subscriber can invoke the cached application from the desktop or browser. It should be noted that the

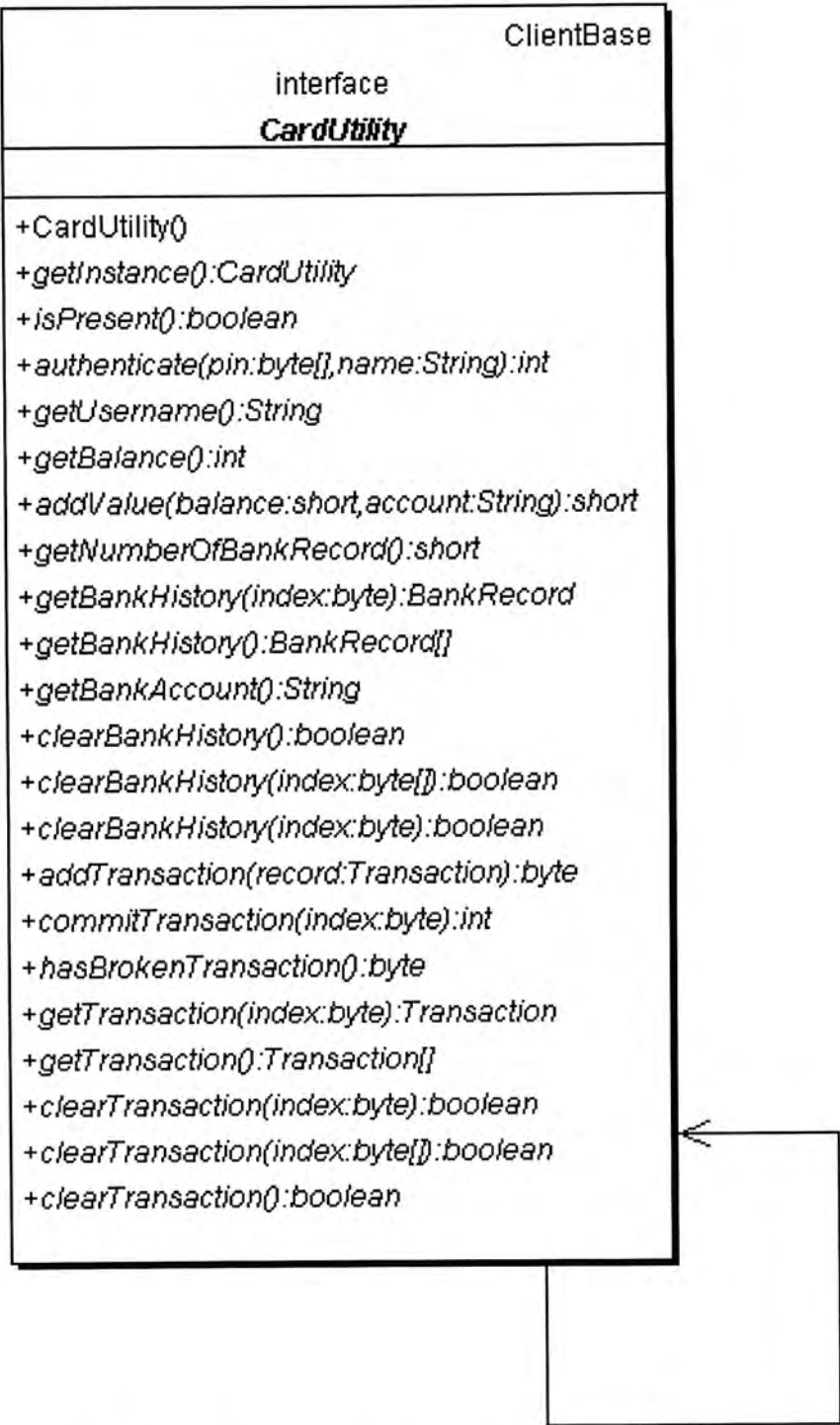


Figure 5.8: UML diagram of class CardUtility and ClientBase

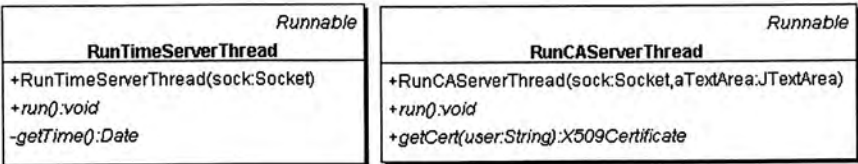


Figure 5.9: UML diagram of class RunTimeServerThread and RunCAServerThread

Java Web Start application is independent from the implementations of the browsers. It provides the flexible application version management and security management.

```
<?xml version="1.0" encoding="utf-8" ?>
<jnlp spec="0.2 1.0" codebase="http://www2.cse.cuhk.edu.hk/ hctsang" href="hctsang.jnlp">
  <information>
    <title>Multi-MAX Demo</title>
    <vendor>CSE, CUHK</vendor>
    <homepage href = "http://www.cse.cuhk.edu.hk"/>
    <description>Multi-MAX Demo</description>
    <description kind="short">Mutli-MAX Project</description>
    <icon href="logo.jpg"/>
  </information>
  <resources>
  <security>
    <all-permissions/>
  </security>
  <application-desc main-class="client.LogonFrame">
    <argument>Product Code< /argument>
    <argument>Merchant Code< /argument>
    <argument>Price< /argument>
    <argument>Product Description< /argument>
    <argument>Time< /argument>
  </application-desc>
</jnlp>
```

Figure 5.10: Sample JNLP file

5.5.4 Evaluation

We have tested the trading platform, Multi-MAX and evaluated its performance during the initialization, authentication and transaction.

multimax smart card reg.

MultiMAX
smart card registration

personal information

username

password

retype password

organization unit

locality name

state

country AD

organization

bank account information

account number

Registration

submit clear

Figure 5.11: Snapshot of initialization form

Initialization

A snapshot of the initialization to the card and server is shown in figure 5.11. During the initialization, the information of the service subscriber would be loaded onto the card. Concurrently, an X.509 certificate can be created and a signed request can be submitted to the certification authority. The certificate of subscriber can be signed and stored in the certification authority while the private key can be stored safely in the card.

Figure 5.12 shows the shopping portal, in which the user can select the desired item for shopping. Using Java Web Start, Multi-MAX can integrate with the browser to offer the same graphical user interface to the user.

Authentication

Figure 5.13 shows the authentication screen. By supplying a valid username and the password, the service subscriber can activate the PIN-protected smart card after the authentication protocol has performed a local and remote authentication with the remote application server successfully. Afterwards, the service subscriber can access the services provided by the application server.

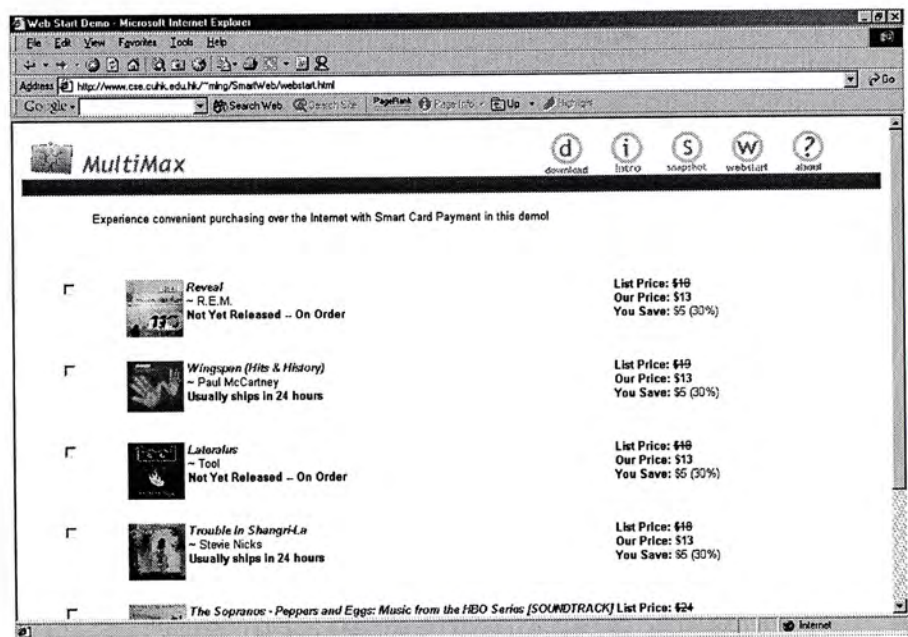


Figure 5.12: Snapshot of web-based shopping portal

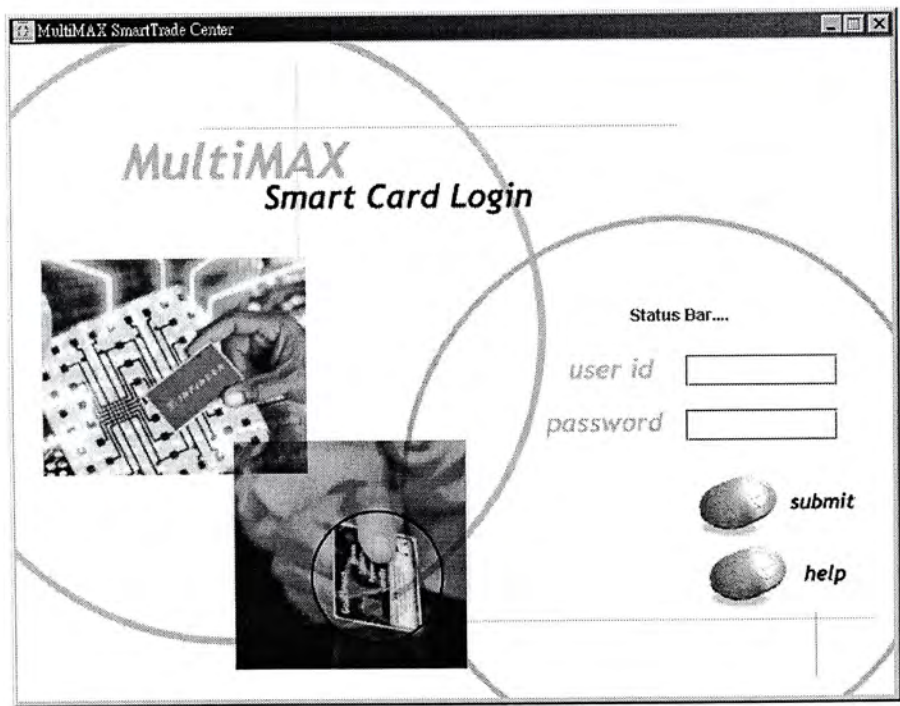
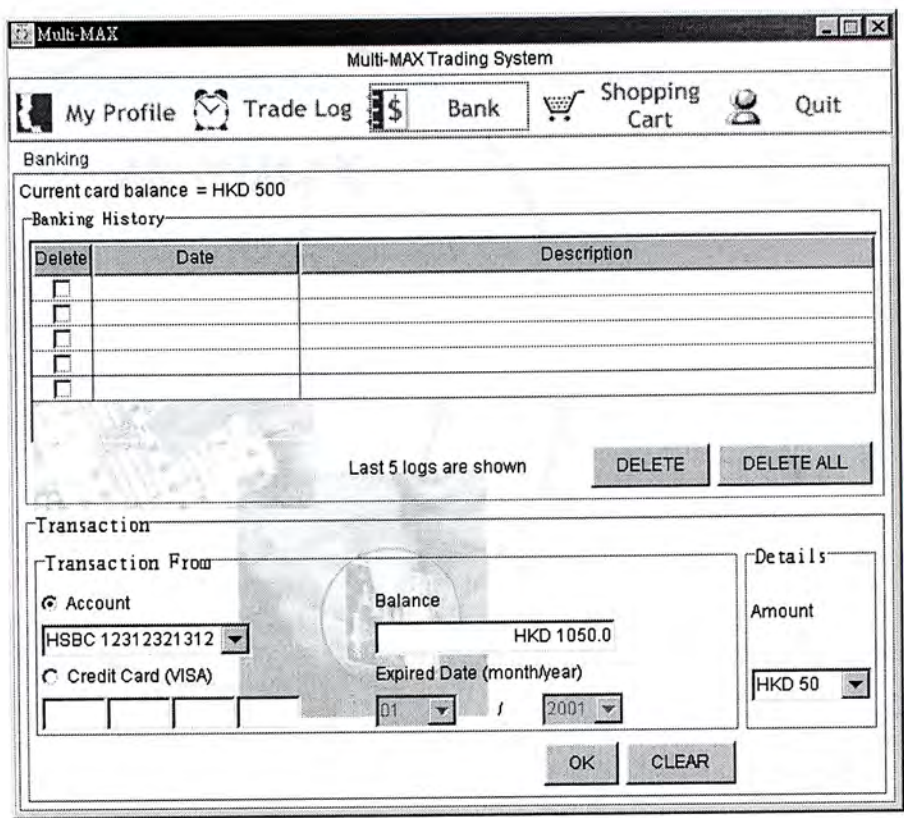


Figure 5.13: Snapshot of authentication form



The image shows a screenshot of a web application window titled "Multi-MAX Trading System". The window has a navigation bar with links: "My Profile", "Trade Log", "Bank" (highlighted), "Shopping Cart", and "Quit". Below the navigation bar, the "Banking" section displays the "Current card balance = HKD 500". A "Banking History" table shows the last five logs, with columns for "Delete", "Date", and "Description". The table is currently empty. Below the table, a message states "Last 5 logs are shown" with "DELETE" and "DELETE ALL" buttons. The "Transaction" section contains a "Transaction From" area with radio buttons for "Account" and "Credit Card (VISA)". The "Account" option is selected, showing a dropdown for "HSBC 12312321312" and a "Balance" of "HKD 1050.0". The "Credit Card (VISA)" option shows an "Expired Date (month/year)" of "01 / 2001". A "Details" section on the right shows an "Amount" of "HKD 50". "OK" and "CLEAR" buttons are at the bottom.

Delete	Date	Description
<input type="checkbox"/>		
<input type="checkbox"/>		
<input type="checkbox"/>		
<input type="checkbox"/>		
<input type="checkbox"/>		

Figure 5.14: Snapshot of banking form

For example, the card can be deposited with more money on the banking form. The payment can be charged either from the credit card account or banking account. Logs which stores the last five deposit records are shown in the same form as shown in figure 5.14.

Transaction

Figure 5.15 shows the snapshot of Multi-MAX on how it handles transaction related operations. A list of transactions is displayed in the item table. By selecting the desired item from the item table, the service subscriber is prompted for the confirmation before the purchase of the selected item(s). Only after successful transaction, the payment would be deducted from the card and the corresponding log would be recorded in the card. The log can be used for audit checks or dispute resolution. Figure 5.16 shows the log of all transactions.

The resulting system can be built rapidly with Smart RAD. The remote

The screenshot shows a web application window titled "Multi-MAX Trading System". The top navigation bar includes links for "My Profile", "Trade Log", "Bank", "Shopping Cart", and "Quit". The "Shopping Cart" link is active. Below the navigation bar, the page is titled "Shopping Cart". It features a table with the following columns: "Quantity", "Item", "Price", "File Description", and "Group". The table contains one row with the following data: "1", "W. Brown", "100", "100", and "1". Below the table, there is a "Total price: \$100.00" label. At the bottom right, there are three buttons: "Checkout", "Clear", and "Clear All".

Quantity	Item	Price	File Description	Group
1	W. Brown	100	100	1

Total price: \$100.00

Checkout Clear Clear All

Figure 5.15: Snapshot of transaction form

The screenshot shows a web application window titled "Multi-MAX Trading System". The top navigation bar includes links for "My Profile", "Trade Log", "Bank", "Shopping Cart", and "Quit". The "Trade Log" link is active. Below the navigation bar, the page is titled "Trade History". It features a table with the following columns: "Delete", "Date", "Seller", "Filename", "Price", "File Description", and "Group". The table contains 15 rows, each with a checkbox in the "Delete" column. At the bottom right, there are two buttons: "DELETE" and "DELETE ALL".

Delete	Date	Seller	Filename	Price	File Description	Group
<input type="checkbox"/>						
<input type="checkbox"/>						
<input type="checkbox"/>						
<input type="checkbox"/>						
<input type="checkbox"/>						
<input type="checkbox"/>						
<input type="checkbox"/>						
<input type="checkbox"/>						
<input type="checkbox"/>						
<input type="checkbox"/>						
<input type="checkbox"/>						
<input type="checkbox"/>						
<input type="checkbox"/>						
<input type="checkbox"/>						
<input type="checkbox"/>						

DELETE DELETE ALL

Figure 5.16: Snapshot of transaction log form

authentication and transaction services can be integrated to perform the online trading process. The performance of the authentication and transaction are efficiency enough to work properly. The remote authentication services can recognize the correct card holder correctly and the transaction is performed with recoverabilities.

As mentioned before, *Smart RAD* provides the public key infrastructure so that Multi-Max can be compatible with other PKIs. In other words, the certificate can be recognized by other parties and it is trivial to interact with other systems using the same trading protocol.

5.6 Future Work

With the thriving development of e-commerce, different applications of smart cards can be built either vertically or horizontally. In view of coping with these tremendous changes, the following features can be added to *Smart RAD* in the future.

- Develop a more generic interface for the transaction records in the card applet: Different e-commerce projects may have their own transaction interface and need different customizations. Different option attributes, which are used by the developer can be added.
- Develop the copyright agent: It acts as a security manager to ensure the usage of a product compliant with the license agreement, or the behavior of the service subscriber is compliant with the permission context, which defines the access rights of the subscriber.
- Allow the card applet to be automatically customized by the developer and therefore the developer can add their own Java Card classes to the card. It involves auto code generation.

- Integrate with the personal email system: The service subscribers can use the same smart card to protect their emails from being read or sent by unauthorized parties, or being misused by hostile parties for any illegal use.

Chapter 6

Conclusion

The objective of this thesis is to describe the unique and efficient smart card based secure services for e-commerce. We have particularly tackled the approach of using smart cards incorporated with the public key infrastructure. We have designed the unique remote authentication protocol, which improves significantly on the existing protocols.

To perform the remote authentication, the approach of using one-way hash functions f and F has been described. To ensure the authentication protocol from being suffered from replay attacks, a number of nonce have been used to ensure the freshness of messages. Experimental results have indicated that its implementation on the Java Card is feasible and its performance is good in speed. A formal notation *BAN logic* has been deployed for the analysis of the correctness of the protocol. On the other hand, we have identified different problems in existing authentication protocols and shown how to correct them with our protocol.

Based on the proposed authentication protocol, an efficient and secure transaction protocol has been built. It supports the transaction recovery and handles micro-payments. The transaction is performed atomically such that either credit and debit operations have to be completed, or the original state has to be restored.

It should be noted that the remote authentication is of paramount important to the e-commerce application with remote service access. With the highly integration of transaction and authentication protocols, the security of the e-commerce application is less susceptible to be intruded than the mandatory approach that the security measure of the transaction protocol is separated from the authentication protocol.

Towards a better utilization of smart cards in the e-commerce, we have designed and developed a generic builder, *Smart RAD*, which is an application-programming toolbox for application developers to build secure smart card based applications. The developer can develop secure e-commerce systems rapidly without knowing the sophisticated architecture of smart cards. We have developed a web-based B2C and application-based P2P trading platforms called *Multi-Max* for evaluation purpose. This trading platform has demonstrated the feasibility and flexibility of our toolbox developed.

Appendix A

Detail Experimental Result

A.1 Authentication Time Measurement

CPU Memory	333MHz 224M	500MHz 128M	667MHz 128M
	Authentication Time (ms)	Authentication Time (ms)	Authentication Time (ms)
	3960	3620	3300
	3790	3570	3350
	3680	3300	3180
	3730	3290	3250
	3850	3630	3240
	3840	3290	3240
	3840	3250	3130
	3680	3290	3180
	3680	3400	3190
	3620	3460	3240
	3680	3300	3630
	3850	3290	3240
	3680	3570	3130
	3510	3350	3240
	3630	3190	3240
	3840	3300	3290
	3740	3290	3140
	3620	3410	3300
	3520	3190	3300
	3730	3180	3350
	3520	3180	3130
	3840	3620	3190
	3680	3570	3290
	3630	3350	3300
	3570	3410	3130
	3900	3680	3130
	3680	3400	3240
	3620	3360	3190
	3570	3350	3190
	3520	3400	3180
	3840	3290	3300
	3680	3300	3300
	3630	3300	3130
	3570	3350	3190
	3900	3180	3190
	3680	3290	3190
	3620	3250	3130
	3570	3350	3180
	3520	3290	3190
Average Time (ms)	3700	3361	3229
Standard Deviation	123	136	93
Maximum Time (ms)	3960	3680	3630
Minimum Time (ms)	3510	3180	3130

Table A.1: Authentication Time Measurement

A.2 On-Card and Off-Card Computation Time in Authentication

The client host and the server have 667MHz and 128M ram running on windows 98 SE.

	On-card (ms)	Off-card (ms)	Total (ms)
	2690	610	3300
	2750	490	3240
	2630	500	3130
	2690	550	3240
	2690	550	3240
	2590	650	3240
	2630	720	3350
	2690	500	3190
	2690	490	3180
	2700	600	3300
	2690	610	3300
	2700	600	3300
	2630	550	3180
	2700	490	3190
	2690	600	3290
	2690	610	3300
	2630	500	3130
	2630	500	3130
	2740	610	3350
	2690	440	3130
	2640	600	3240
	2680	560	3240
	2690	500	3190
	2690	440	3130
	2700	600	3300
	2750	540	3290
	2640	490	3130
	2640	600	3240
	2690	610	3300
	2690	550	3240
	2640	490	3130
Average Time (ms)	2677	558	3235
Standard Deviation	38	67	69
Maximum Time (ms)	2750	720	3350
Minimum Time (ms)	2590	440	3130

Table A.2: On-card and off-card computation time in authentication

A.3 Authentication Time with Different Servers

The testing client host has 500MHz CPU and 128M ram running on windows 98 SE platform.

Server CPU Server Memory Server OS	500MHz 224M Windows 98 SE Authentication Time (ms)	270MHz 512M Solaris 2.6 Authentication Time (ms)	12400 MHz 8G Solaris 7 Authentication Time (ms)
	3620	3730	3300
	3570	3840	3350
	3300	3740	3350
	3290	3900	3300
	3630	3790	3400
	3290	3840	3350
	3250	3850	3410
	3290	3900	3350
	3400	3730	3290
	3460	3850	3350
	3300	3730	3240
	3290	3790	3300
	3570	3740	3410
	3350	3680	3290
	3190	3840	3300
	3300	3680	3400
	3290	3680	3300
	3410	3730	3290
	3190	3740	3300
	3180	3850	3300
	3180	3730	3520
	3620	3900	3350
	3570	3730	3240
	3350	3850	3290
	3410	3790	3410
	3680	3850	3350
	3400	3790	3410
	3360	3840	3240
	3350	3790	3290
	3400	3740	3460
	3290	3840	3240
	3300	3790	3300
	3300	3790	3290
	3350	3790	3300
	3180	3740	3350
	3290	3790	3460
	3250	3790	3300
	3350	3840	3350
	3290	3850	3350
Average Time (ms)	3361	3791	3335
Standard Deviation	136	61	65
Maximum Time (ms)	3680	3900	3520
Minimum Time (ms)	3180	3680	3240

Table A.3: Authentication time measurement with different servers

A.4 Transaction Time Measurement

Server CPU	500MHz	667MHz
Server Memory	128M	128M
Client CPU	500MHz	667MHz
Client Memory	128M	128M
	Transaction Time (ms)	Transaction Time (ms)
	3190	3180
	3080	3030
	3190	3080
	3080	3130
	3240	3070
	3070	3130
	3020	3130
	3130	3130
	3130	3070
	3130	3020
	3170	3130
	3190	3070
	3070	3130
	3130	3020
	3130	3080
	3190	3130
	3130	3080
	3070	3030
	3020	3130
	3130	3020
	3130	3080
	3130	3130
	3020	3130
	3190	3130
	3070	3020
	3130	3070
	3130	3020
	3190	3030
	3130	3070
Average Time (ms)	3124	3082
Standard Deviation	57	44
Maximum Time (ms)	3240	3130
Minimum Time (ms)	3020	3020

Table A.4: Transaction time measurement

A.5 On-card and Off-card Computation Time
in Transaction

The client host and the server has 667MHz and 128M ram running on windows 98 SE platform.

	On-card (ms)	Off-card (ms)	Total (ms)
	2480	820	3300
	2470	660	3130
	2410	770	3180
	2530	660	3190
	2470	720	3190
	2470	660	3130
	2420	770	3190
	2410	780	3190
	2420	710	3130
	2360	710	3070
	2360	660	3020
	2360	770	3130
	2360	720	3080
	2420	660	3080
	2360	660	3020
	2360	660	3020
	2360	770	3130
	2360	660	3020
Average Time (ms)	2410	712	3122
Standard Deviation	54	55	77
Maximum Time (ms)	2530	820	3300
Minimum Time (ms)	2360	660	3020

Table A.5: On-card and off-card computation time in transaction

Appendix B

UML Diagram

B.1 Package cuhk.cse.demo.applet

UFOApplet

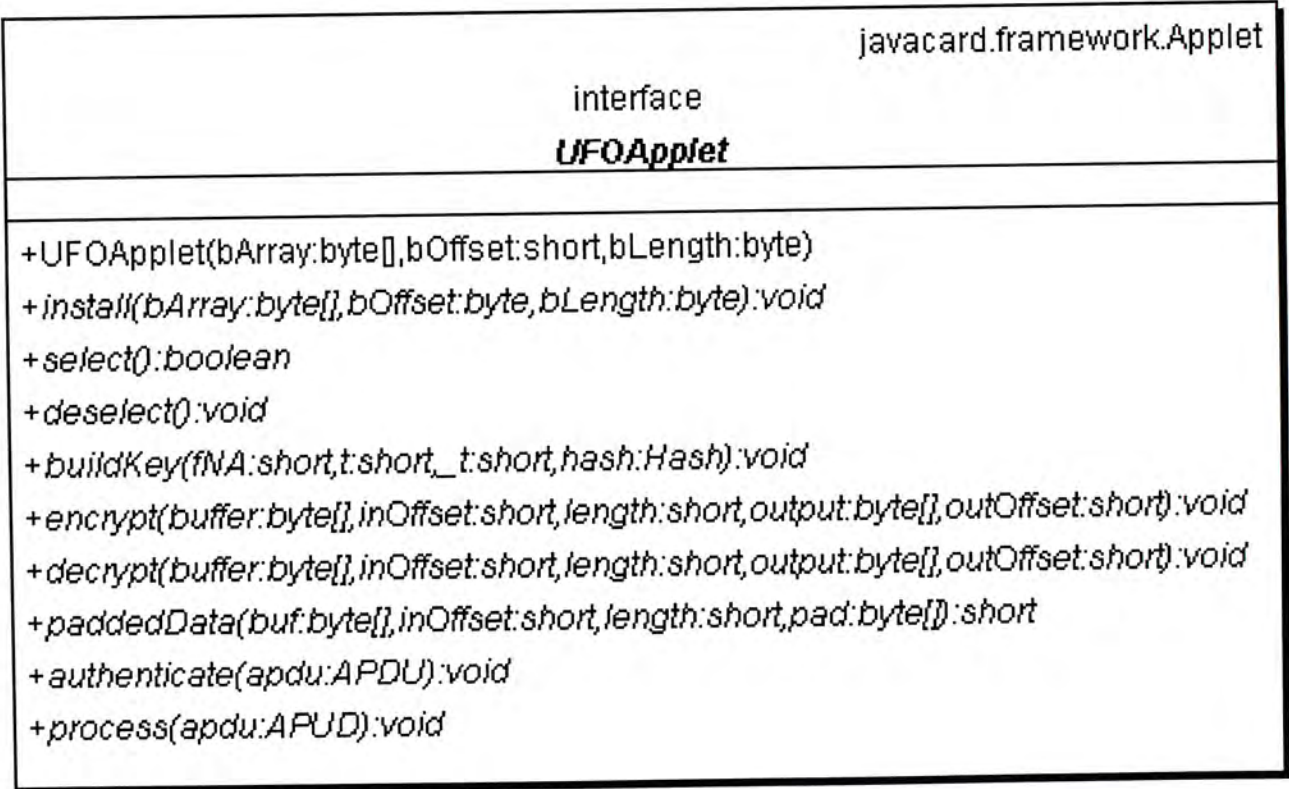


Figure B.1: UML diagram of UFOApplet

Hash

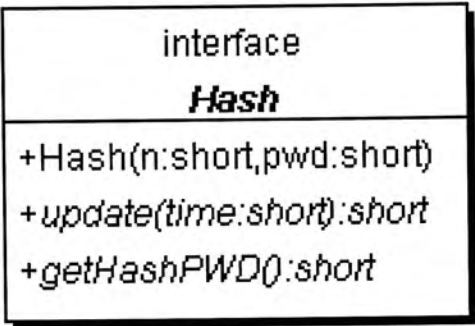


Figure B.2: UML diagram of Hash

HashGenerator

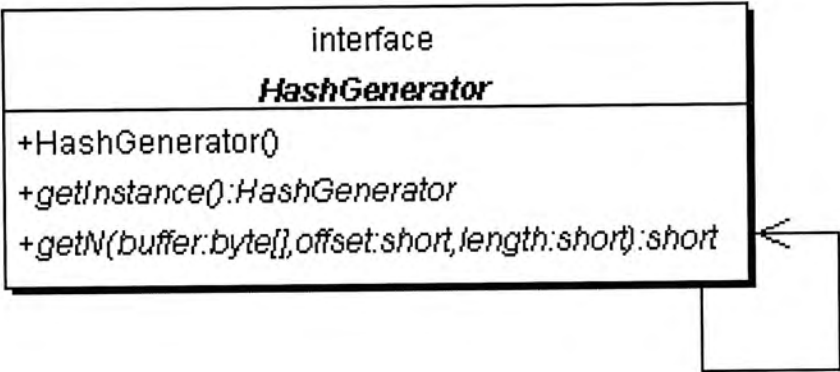


Figure B.3: UML diagram of HashGenerator

Record

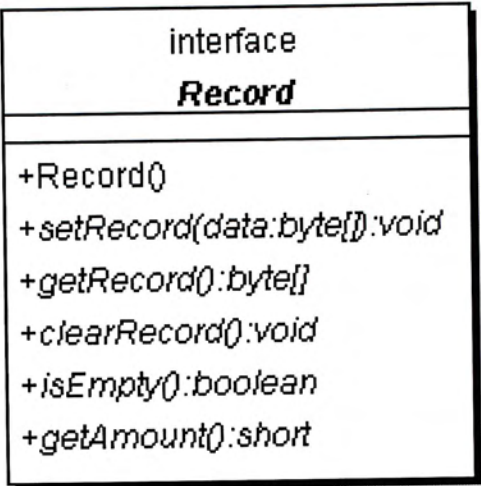


Figure B.4: UML diagram of Record

SmartPurse

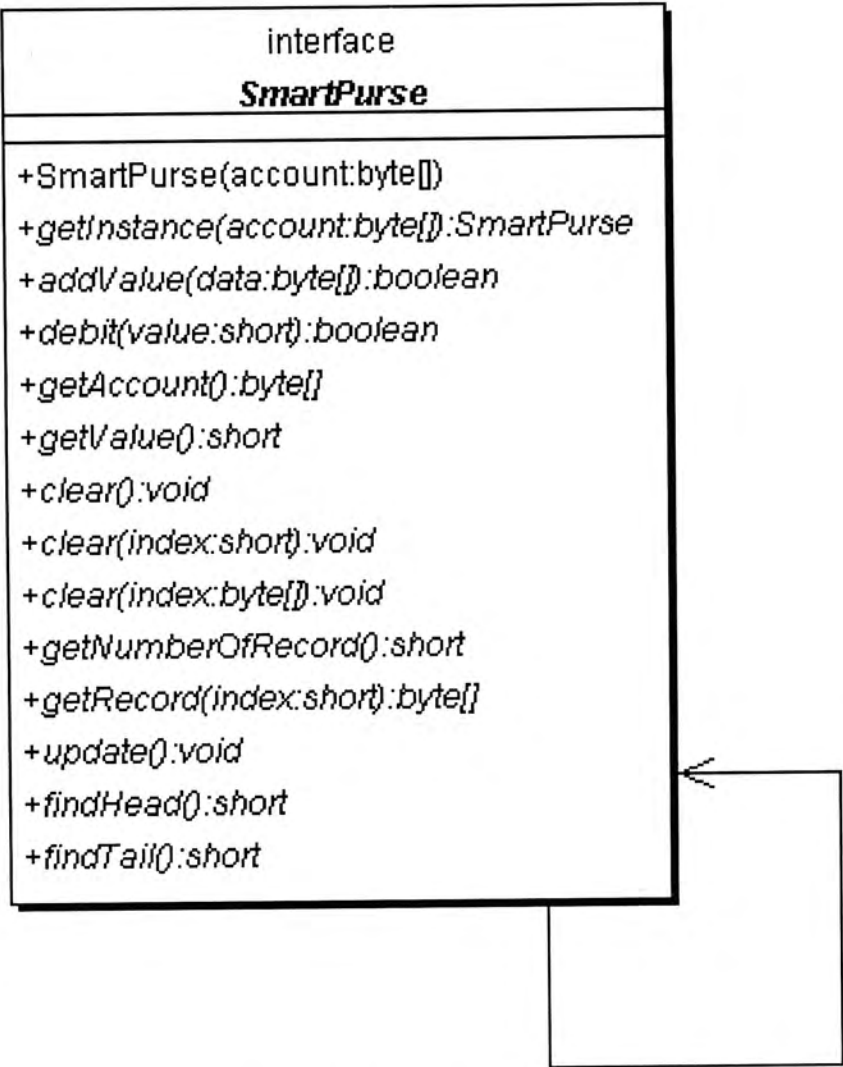


Figure B.5: UML diagram of SmartPurse

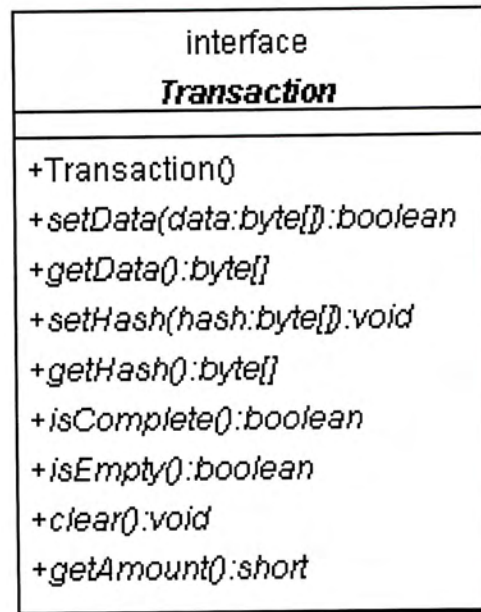
Transaction

Figure B.6: UML diagram of Transaction

TransactionManager

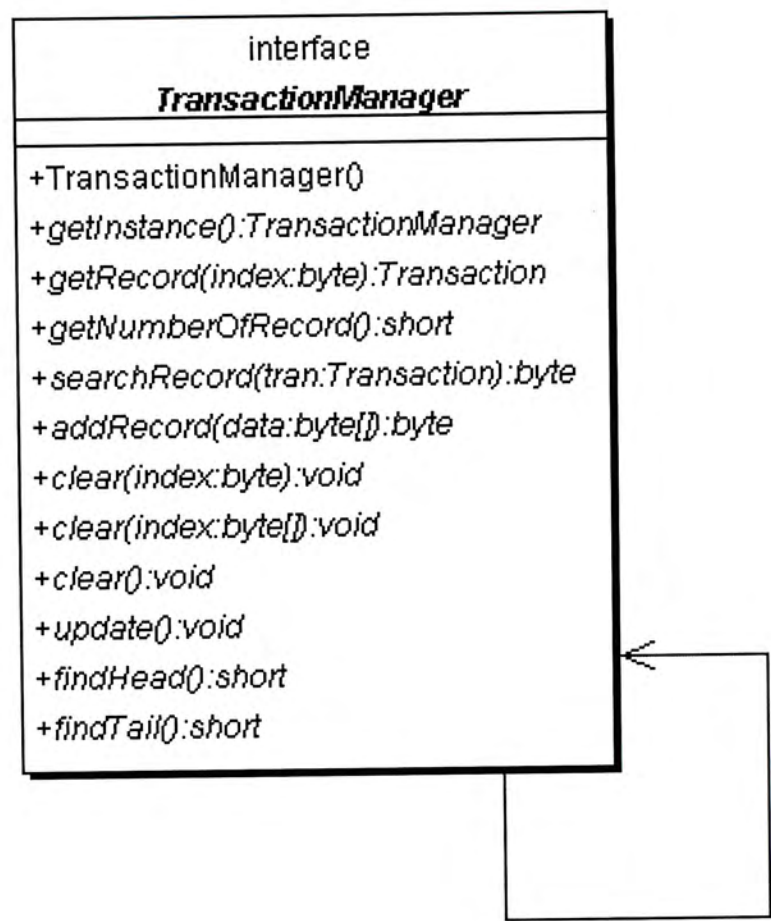


Figure B.7: UML diagram of TransactionManager

B.2 Package cuhk.cse.demo.client

ClientBase

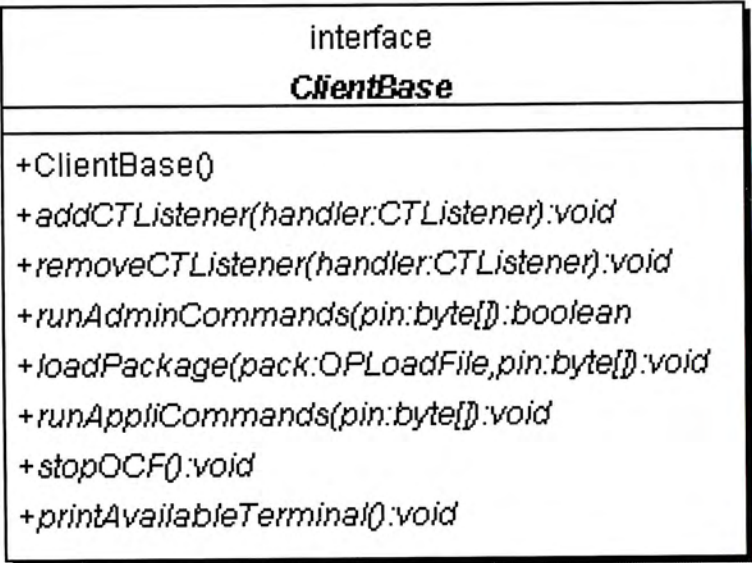


Figure B.8: UML diagram of ClientBase

CardUtility

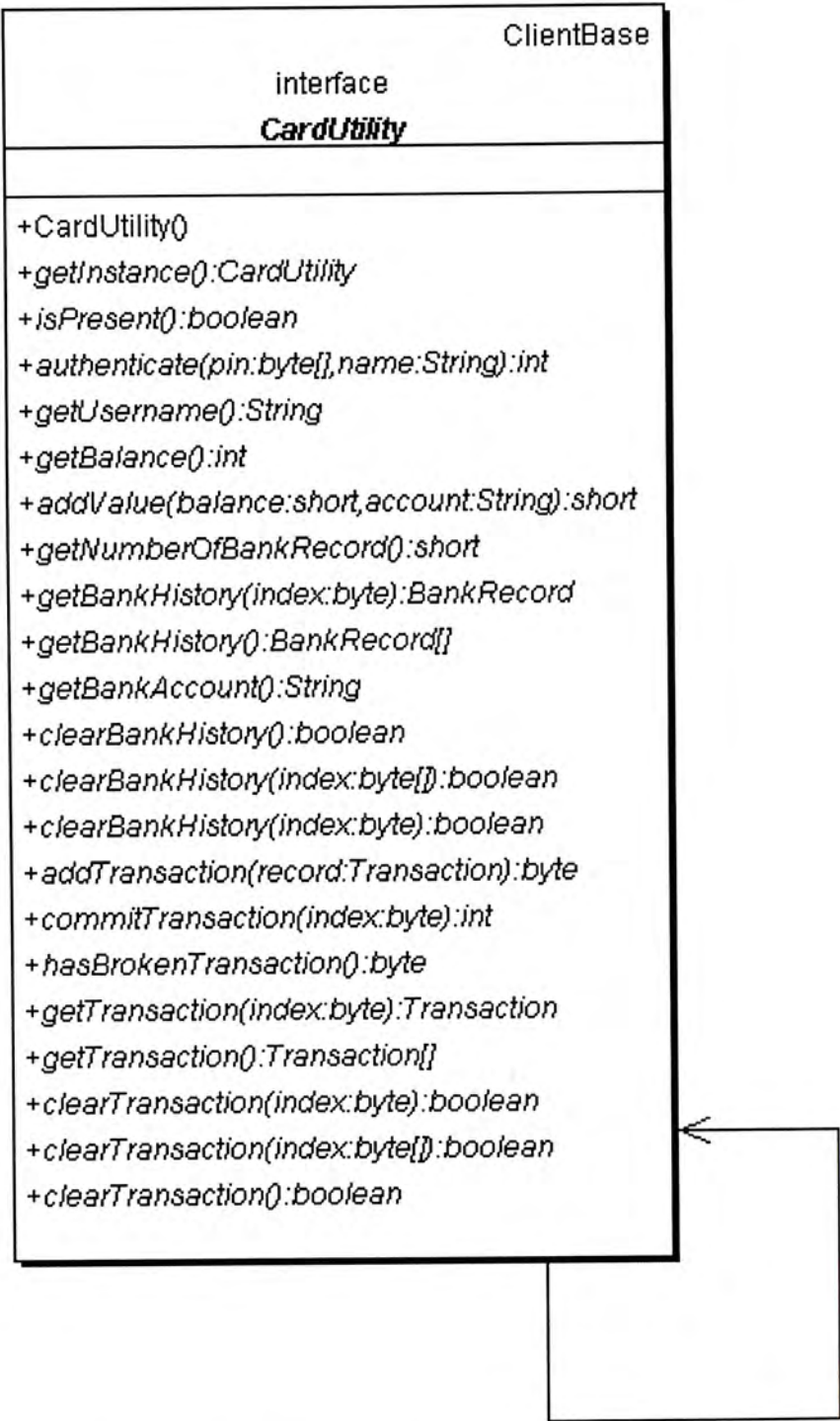


Figure B.9: UML diagram of CardUtility

Install

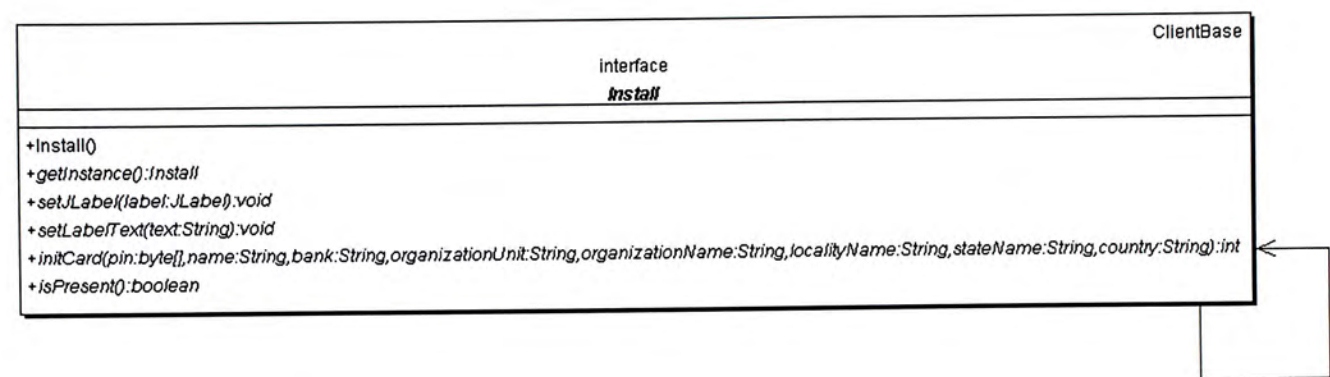


Figure B.10: UML diagram of Install

BankRecord

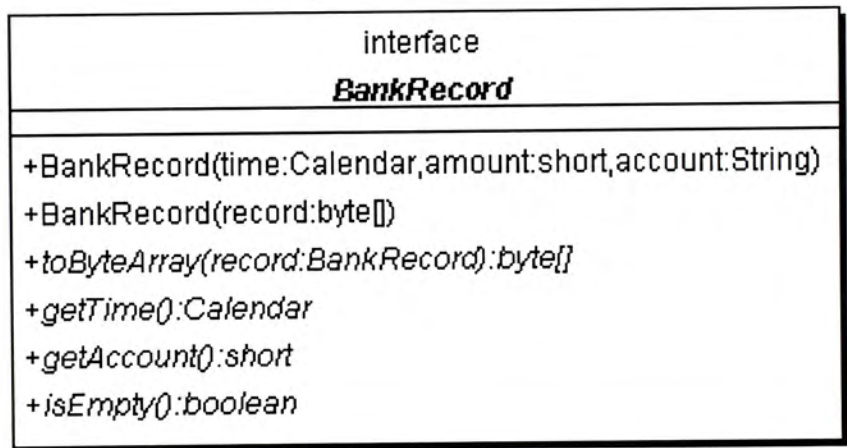


Figure B.11: UML diagram of BankRecord

Transaction

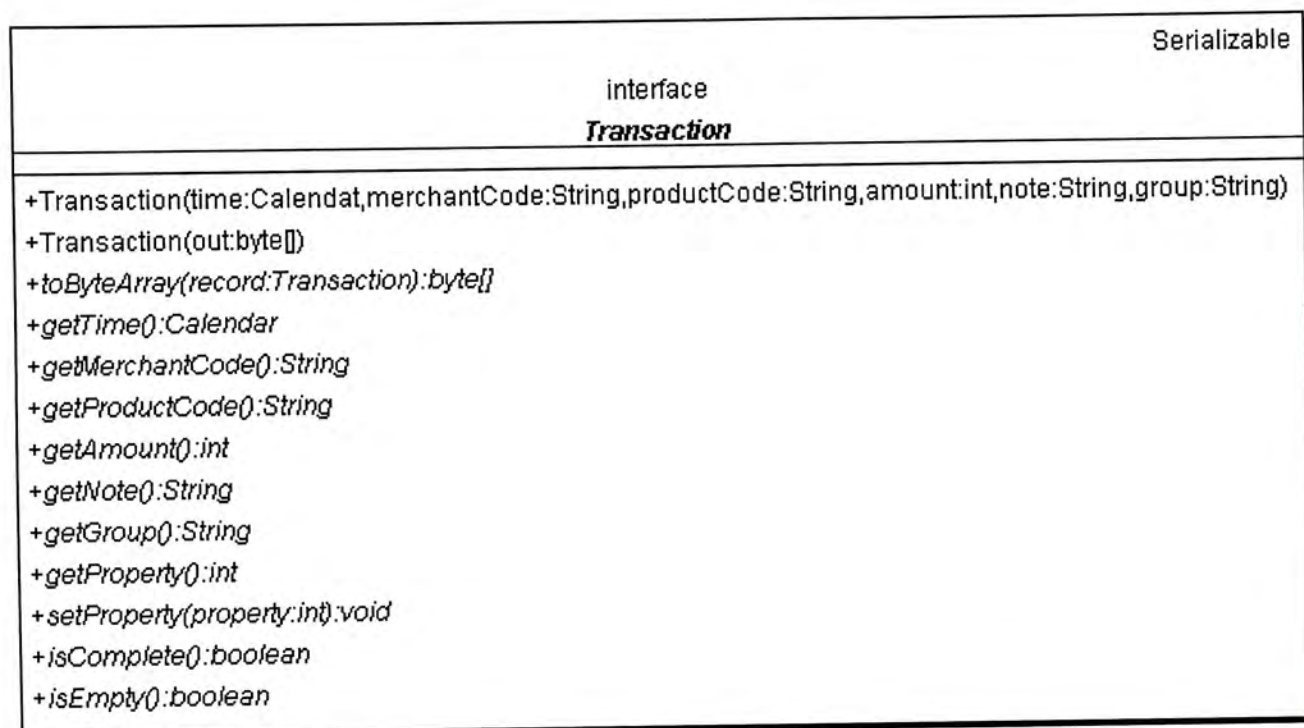


Figure B.12: UML diagram of Transaction

BigHash

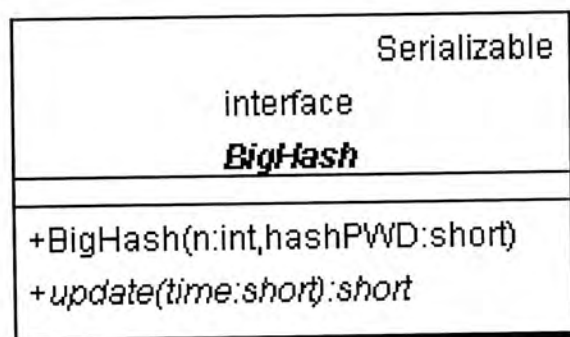


Figure B.13: UML diagram of BigHash

TimeUtility

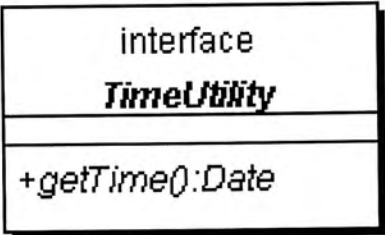


Figure B.14: UML diagram of TimeUtility

CAUtility

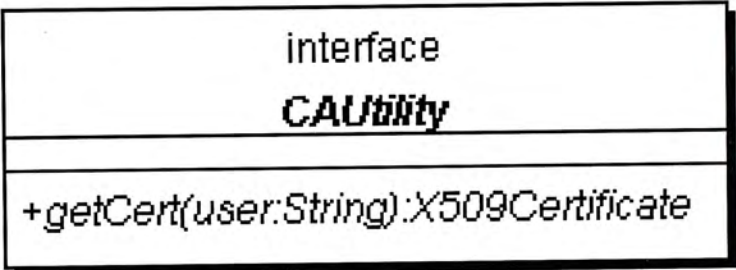


Figure B.15: UML diagram of CAUtility

CustomCertAndKeyGen

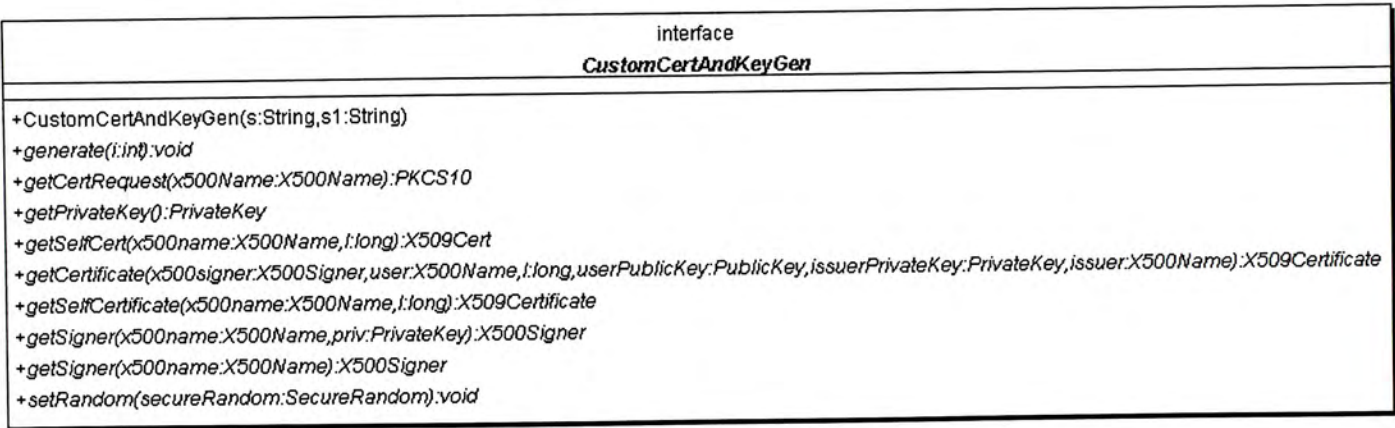


Figure B.16: UML diagram of CustomCertAndKeyGen

B.3 Package server

TradePanel

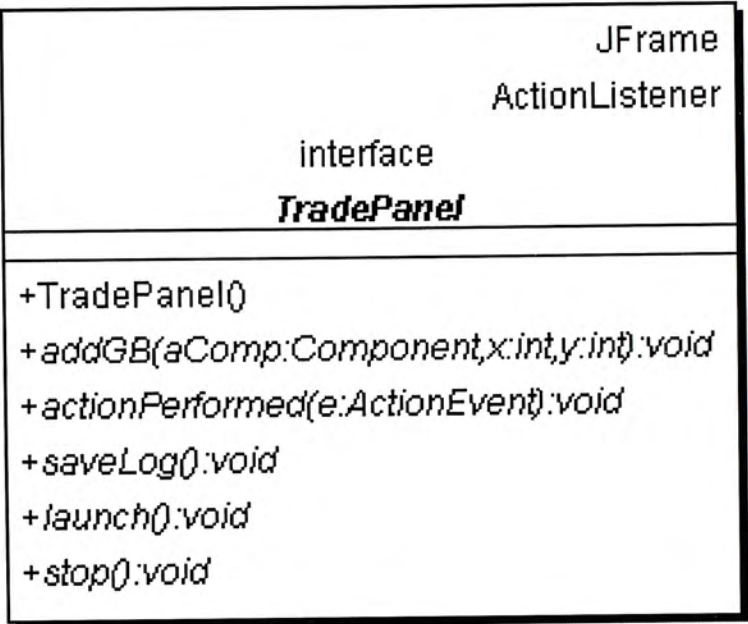


Figure B.17: UML diagram of TradePanel

CAPanel

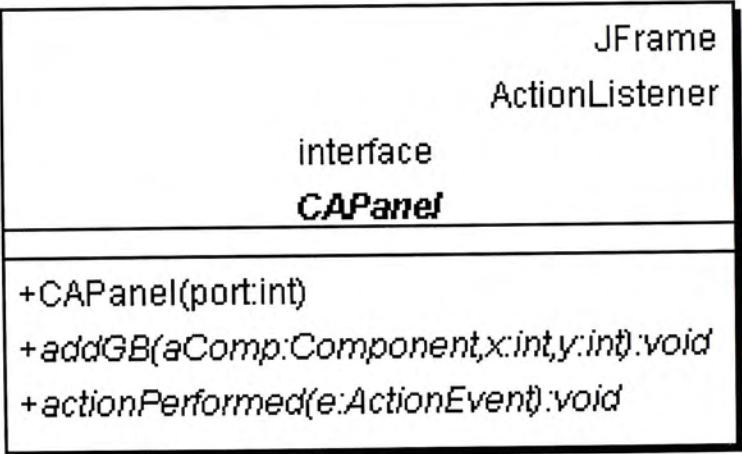


Figure B.18: UML diagram of CAPanel

TimePanel

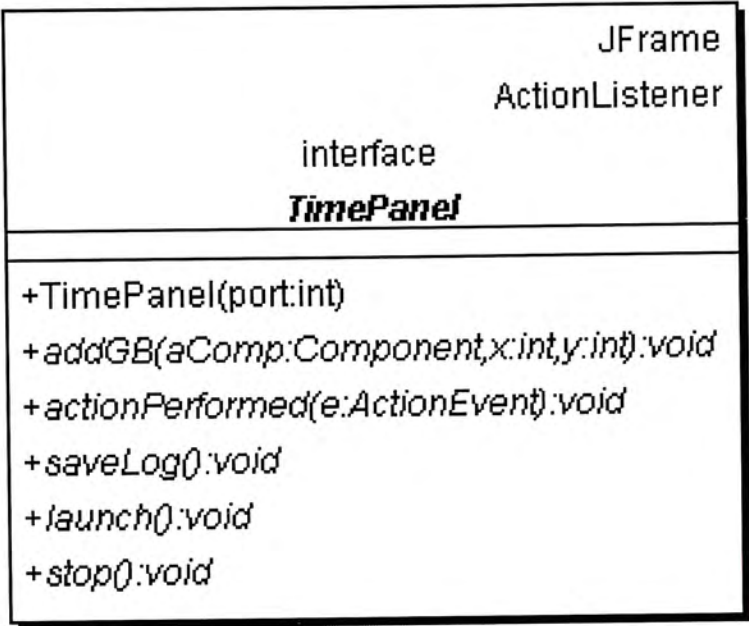


Figure B.19: UML diagram of TimePanel

ServerFrame

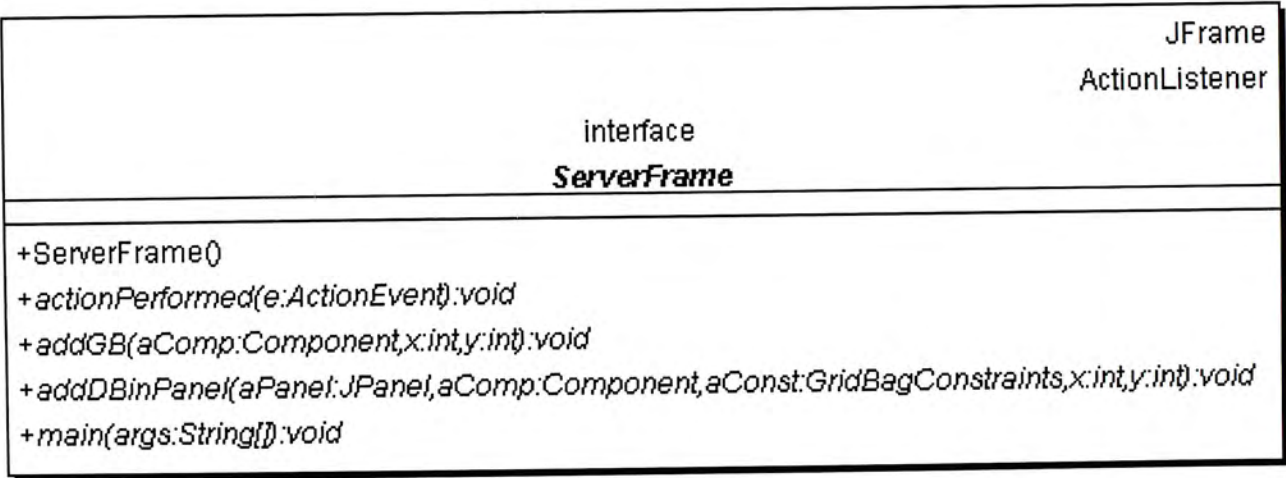


Figure B.20: UML diagram of ServerFrame

RunCASServer

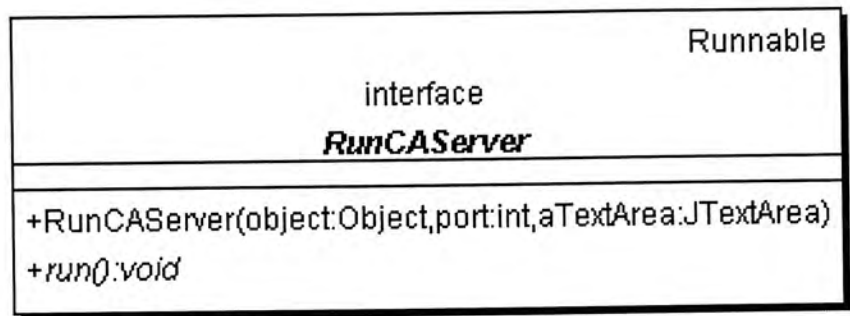


Figure B.21: UML diagram of RunCASServer

RunCASServerThread



Figure B.22: UML diagram of RunCASServerThread

RunTimeServer

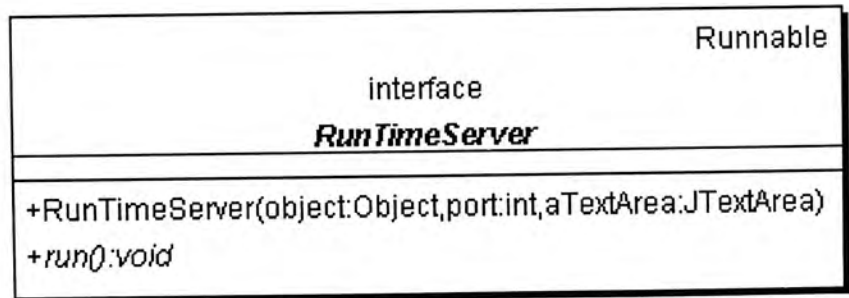


Figure B.23: UML diagram of RunTimeServer

RunTimeServerThread



Figure B.24: UML diagram of RunTimeServerThread

RunTradeServer



Figure B.25: UML diagram of RunTradeServer

RunTradeServerThread

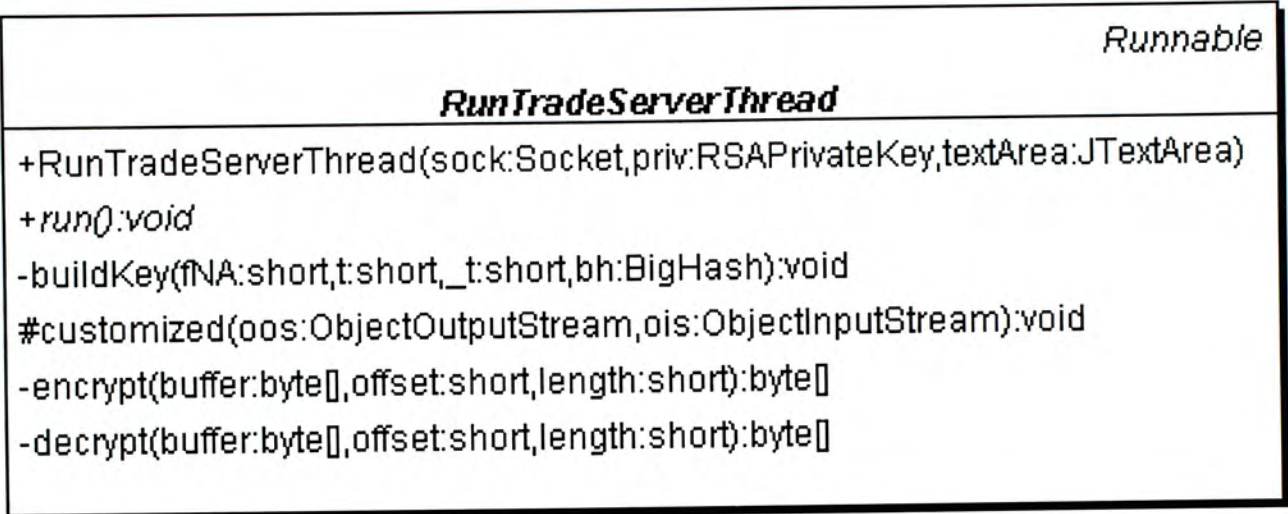


Figure B.26: UML diagram of RunTradeServerThread

StopServer

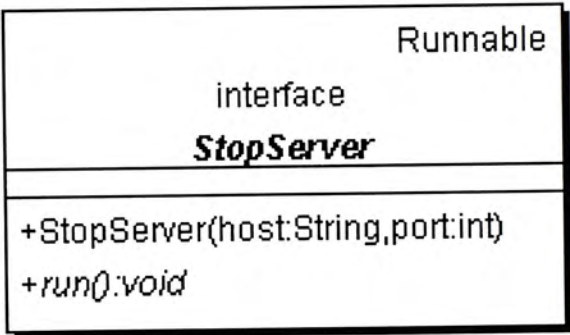


Figure B.27: UML diagram of StopServer

Appendix C

Glossary and Abbreviation

- 3DES - Triple Data Encryption Standard
- APDU - Application Protocol Data Unit
- B2C - Business to Customer, a service that allows the customer to purchase a product from the Internet
- BAN logic - Logic for analyzing the correctness of authentication protocol
- Card Applet - A Java program that runs and executes on Java Card
- Command APDU - the APDU sent to the card
- DES - Data Encryption Standard
- Digital Signature - the fingerprint of a message for integrity and authenticity
- EDE - Encrypt-decrypt-encrypt, a sequence of 3DES encryption
- EEE - Encrypt-encrypt-encrypt, a sequence of 3DES encryption
- EEPROM - Electrically Erasable Programmable Memory.
- Hash - a function to generate a fingerprint for input data
- Java Card - A smart card that can run Java program

- JCE - Java Cryptography Extension
- JDK - Java Development Toolkit
- JSSE - Java Secure Socket Extension
- JVM - Java Virtual Machine
- Local Authentication - authentication for local access
- OpenCard - Standard for smart card access on Java platform
- P2P - Peer-to-Peer
- PKI - Public Key Infrastructure
- Private Key - It is used for the generation of signature and decryption
- Public Key - It is used for the verification of signature and encryption
- Public Key Algorithm - Asymmetric key algorithm where public key is accessible by any parties, while the private key is kept by the own as a secret.
- Remote Authentication - authentication for remote access
- Response APDU - APDU sent from the card
- ROM - Read Only Memory
- RSA - Asymmetric cryptographic algorithm found by Rivest, Shamir and Adlemen
- Secret Key - The key for symmetric algorithm
- Smart Card - A credit card sized plastic card with processor chip
- Tamper-proof - A device is tamper-proof if the information in that device would be lost when someone tries to tamper with the device

- UML - Universal Modelling Language

Bibliography

- [1] W. A. Aiello, A. D. Rubin, and M. J. Strauss. Using smartcards to secure a personalized gambling device. In *Proceedings of the 6th ACM conference on Computer and communications security*, pages 128–137, 1999.
- [2] N. Asokan, P. A. Janson, and M. Waidner. The state of the art in electronic payment systems. In *Computer*, volume 30, pages 28–35, September 1997.
- [3] N. Asokan, M. Schunter, and M. Waidner. Optimistic protocols for fair exchange. In *Proceedings of the 4th ACM conference on Computer and communications security*, pages 7–17, 1997.
- [4] M. Bellare, J. A. Garay, R. Hauser, A. Herzberg, H. Krawczyk, M. Steiner, G. Tsudik, E. V. Herreweghen, and M. Waidner. Design, implementation, and deployment of the ikp secure electronic payment system. In *Selected Areas in Communication, IEEE Journal on*, volume 18, pages 611–627, April 2000.
- [5] M. Burrows, M. Abadi, and R. Needham. A logic of authentication. In *ACM Transactions on Computer System*, volume 8, 1990.
- [6] D. Chadwick. Smart cards aren’t always the smart choice. In *Computer*, number 142-143, 1999.

- [7] C. C Chang, R. J. Hwang, and D. J. Buechrer. Using smart cards to authenticate passwords. In *Security Technology Proceeding*, number 154-156, 1993.
- [8] C. C Chang and C. S. Lai. Remote password authentication with smart cards. In *Computers and Digital Techniques*, volume 139, page 372, July 1992.
- [9] C. C. Chang and T. C Wu. Remote password authentication with smart cards. In *Computers and Digital Techniques*, volume 138, pages 165–168, May 1991.
- [10] F. Chau. Smart card technology. In *U.S. Department of Commerce - National Trade Data Bank*, November 2000.
<http://www.tradeport.org/ts/countries/hongkong/isa/isar0040.html>.
- [11] Z. Chen. *Java Card Technology for Smart Cards: Architecture and Programmer's Guide*. The java series. Addison-Wesley, 2000.
- [12] D. Coppersmith. The data encryption standard (des) and its strength against attacks. In *IBM Journal of Research and Development*, May 1994.
- [13] D. W. Davies and W. L. Price. *Security for Computer Networks*. John Wiley and Sons, 1984.
- [14] D. E. Denning. *Cryptography and Data Security*. Addison-Wesley, 1982.
- [15] M. T. El-Hadidi, N. H. Hegazi, and H. K. Aslan. Performance evaluation of a new hybrid encryption protocol for authentication and key distribution. In *IEEE Symposium on Computers and Communications Proceedings*, pages 16–22, 1999.

- [16] F. J. T. Fabrega, J. C. Herzog, and J. D. Guttman. Strand spaces: Why is a security protocol correct? In *11th IEEE Computer Security Foundations Workshop*, pages 160–171, 1998.
- [17] M. K. Franklin and M. K. Reiter. Fair exchange with a semi-trusted third party. In *AT&T Laboratories*, 1997.
- [18] Gemplus. Gemmobile card issuer.
http://www.gemplus.com/products/software/gmobile_card_issuer.htm.
- [19] Gemplus. Gemplus wallet.
<http://www.gemplus.com/products/software/wallet/index.htm>.
- [20] Gemplus. Gemutilities.
http://www.gemplus.com/products/software/gemutilities_mktng.htm.
- [21] Gemplus. Gemxpresso rad 211.
http://www.gemplus.com/products/software/gemxpresso_rad_211.htm.
- [22] Gemplus. Smartxcess cashcard explorer.
<http://www.gemplus.com/products/software/smartxcess/index.htm>.
- [23] Gemplus. *GemXpresso 211 V2 Card: Reference Manual Version 2.0*. Gemplus, May 2000.
- [24] Gemplus. *GemXpresso RAD 211 v2.3/2.3 IS and v2.4 PK/2.4 PK IS: Getting Started Version 2.0*. Gemplus, July 2000.
- [25] Gemplus. *GemXpresso RAD 211 v2.3/2.3 IS and v2.4PK/2.4 PK IS: User's Guide Version 1.0*. Gemplus, July 2000.
- [26] L. Gong. Variations on the themes of message freshness. In *IEE computer Security Foundations Workshop Proceeding*, June 1993.

- [27] D. Graft, M. Pabrai, and U. Pabrai. Methodology for network security design. In *Ninth Annual International Phoenix Conference on Computers and Communications*, pages 675–682, 1990.
- [28] The Object Management Group. *Unified Modeling Language Version 1.1*. 1997.
- [29] S. Haber and W. S. Stornetta. How to time-stamp a digital document. In *Journal of Cryptology*, pages 99–111, 1991.
- [30] G. Hachez and J. J. Quisquater. Biometrics, access control, smart cards: a not so simple combination. In *Proceedings of the Fourth Working Conference on Smart Card Research and Advanced Applications (CARDIS 2000)*, pages 273–288, September 2000.
- [31] U. Hansmann, M. S. Nicklous, T. Schäck, and F. Seliger. *Smart Card Application Development Using Java*. Springer, 2000.
- [32] Hendry. *Smart Card Security and Applications*. Artech House Inc., 1997.
- [33] R. Housley, W. Ford, W. Polk, and D. Solo. Internet x.509 public key infrastructure certificate and crl profile. January 1999. <http://www.faqs.org/rfcs/rfc2459.html>.
- [34] G. Howland. Development of an open and flexible payment system. November 1996. <http://www.systemics.com/docs/sox/overview.html>.
- [35] M. S. Hwang and L. H. Li. A new remote user authentication scheme using smart cards. In *IEEE Transaction on Consumer Electronics*, volume 46, pages 28–30, February 2000.
- [36] F. Joshua. When is a singleton not a singleton. In *Javaworld - Design Pattern*, January 2001.

- [37] L. Lamport. Password authentication with insecure communications. In *Communications of the ACM*, volume 24, pages 770–772, November 1981.
- [38] H. W. Lee and T. Y. Kim. Smart card based offline micropayment framework using mutual authentication scheme. In *Global Telecommunication Conference*, volume 4, pages 2514–2519, 1998.
- [39] B. Lewis and D. J. Berg. *Multithreading Programming with Java Technology*. Java series. Prentice Hall, 2000.
- [40] S. Liu and M. Silverman. A practical guide to biometric security technology. In *IT Professional*, pages 27–32, January 2001.
- [41] X. Lorphelin. Internet and smart card application deployment. In *JSource*, August 1999.
- [42] Microsoft. Smart card for windows. http://www.scia.org/SignificantEvents/98msft_sc.htm.
- [43] Y. S. Moon, H. C. Ho, K. L. Wan, and S. T. Wong. Collaborative fingerprint authentication by smart card and a trusted host. In *Electrical and Computer Engineering*, volume 1, pages 108–112, 2000.
- [44] R. Morris and K. Thompson. Password security: A case study. In *Communications of the ACM*, volume 22, pages 594–597, November 1979.
- [45] Multos. The multi-application operating system for smart cards. <http://www.multos.com/>.
- [46] Netscape. Secure socket layer ssl. <http://www.netscape.com>.
- [47] OpenCard. Opencard framework. <http://www.opencard.org>.
- [48] L. C. Paulson. Proving security protocols correct. In *14th Symposium on Logic in Computer Science*, pages 370–381, 1999.

- [49] P.A. Pays and F. D. Comarmond. An intermediation and payment system technology. In *Fifth international world wide web conference*, May 1996.
http://www5conf.inria.fr/fich_html/papers/P27/Overview.html.
- [50] M. Pistoia, D. F. Reller, D. Gupta, M. Nagnur, and A. K. Ramani. Java 2 network security. In *Prentice Hall*, 1999.
- [51] M. K. Reiter and S. G. Stubblebine. Toward acceptable metrics of authentication. In *IEEE Symposium on Security and Privacy*, number 10-20, 1997.
- [52] R. Rivest, A. Shamir, and L. Adleman. A method for obtaining digital signatures and public key cryptosystems. In *Communications of the ACM*, February 1978.
- [53] H. E. Rose. *A course in number theory*. Oxford University Press, second edition, 1996.
- [54] RSA. Rsa secureid.
<http://www.rsasecurity.com/products/securid/index.html>.
- [55] Schlumberger. Cyberflex access sdk 3.0.
<http://www.1.slb.com/smartcards/products/network/sdk.html>.
- [56] R. W. Schmidt. *Java Network Launching Protocol & API specification version 1.0*. Sun, December 2000.
- [57] B. Schneier. *Applied Cryptography second edition: Protocols, Algorithm, and Source Code in C*. John Wiley & Son, second edition, 1996.
- [58] B. Schneier. *Secrets and Lies: Digital Security in Networked World*. Wiley Computer Publishing, 2000.
- [59] B. Schneier. Self-study course in block cipher cryptanalysis. In *Cryptologia*, pages 18–34, January 2000.

- [60] B. Schneier and A. Shostack. Breaking up is hard to do: Modeling security threats for smart cards. In *USENIX Workshop on Smart Card Technology*, pages 175–185, October 1999.
- [61] M. R. Schroeder. *Number theory in science and communication*. Springer Verlag, second edition, February 1985.
- [62] H. Schuldt, A. Popvici, and H. J. Schek. Automatic generation of reliable e-commerce payment processes. In *Web Information Systems Engineering 2000*, volume 1, pages 434–441, 2000.
- [63] V. Y. Shen. ecyberpay - a micropayment solution for electronic commerce on the web. September 2000. <http://www.ecyberpay.com/zh-tw/body/press/hkcc.htm>.
- [64] W. Stallings. *Cryptography and Network Security*. Prentice Hall, second edition, 1998.
- [65] Sun. Java 2 sdk. <http://java.sun.com/products/jdk/1.2/>.
- [66] Sun. Java card technology.
<http://java.sun.com/products/javacard/index.html>.
- [67] Sun. Java cryptography extension. <http://java.sun.com/products/jce/>.
- [68] Sun. Java secure socket extension. <http://java.sun.com/products/jsse/>.
- [69] Sun. Java security. <http://java.sun.com/security>.
- [70] Sun. Java web start. <http://java.sun.com/products/javawebstart/index.html>.
- [71] K. Topley. *Core Swing Advanced Programming*. Prentice Hall, 2000.
- [72] H. C. Tsang, K. S. Leung, and K. H. Lee. Design and analysis of smart card based remote authentication protocol for internet-based system. Technical report, The Chinese University of Hong Kong, 2001.

CUHK Libraries



003871899