A NEW ARCHITECTURE FOR SECURE TWO-PARTY MOBILE PAYMENT TRANSACTIONS

YUNPU ZHU
Bachelor of Science, Beijing Jiaotong University, 2002

A Thesis
Submitted to the School of Graduate Studies
of the University of Lethbridge
in Partial Fulfillment of the
Requirements for the Degree

MASTER OF SCIENCE

Department of Mathematics and Computer Science
University of Lethbridge
LETHBRIDGE, ALBERTA, CANADA

A NEW ARCHITECTURE FOR SECURE TWO-PARTY MOBILE PAYMENT TRANSACTIONS

YUNPU ZHU

Approved:

Signature *Date*

Supervisor: Dr. Jacqueline E. Rice
Assistant Professor
Department of Mathematics and Computer Science
Faculty of Arts and Science

Committee Member: Dr. Brian Dobing
Associate Professor
Faculty of Management

Committee Member: Dr. Gongbing Shan
Associate Professor
Department of Kinesiology
Faculty of Arts and Science

Chair, Thesis Examination Committee:
Dr. Hadi Kharaghani
Professor
Department of Mathematics and Computer Science
Faculty of Arts and Science

I wish to dedicate this thesis to my beloved parents, Daping Zhu and Zehua Feng, who have raised me to be the person I am today. Thank you for all the unconditional love, guidance, and support that you have always given me.

Abstract

The evolution of wireless networks and mobile device technologies has increased concerns about performance and security of mobile systems. We propose a new secured application-level architecture for a two-party mobile payment transaction that is carried out between a resource-limited mobile device and a resource-rich computer server over wireless networks. As an example of such transactions, the mobile banking transaction is focused on throughout this thesis. The proposed architecture, namely *SA2pMP*, employs a lightweight cryptography scheme (combining both a Public-key cryptography algorithm (ECDSA) and a Symmetric-key cryptography algorithm (AES)), a multi-factor authentication mechanism, and a transaction log strategy. The proposed architecture is designed to satisfy the four properties of confidentiality, authentication, integrity and non-repudiation that are required by any secure system. The architecture can be implemented on a Java ME enabled mobile device. The security API library can be reused in implementing other two-party mobile applications. The present study shows that *SA2pMP* is a unique lightweight security architecture providing comprehensive security for two-party mobile payment transactions. In addition, simulations demonstrate that *SA2pMP* can be installed in resource-limited mobile devices as a downloadable software application. The main contribution of the thesis is to suggest a design for a security architecture for two-party mobile payment transactions, for example, mobile banking. It suggests a four-layer model of mobile payment participants, based on Karnouskos (2004). This model clarifies how participants are involved in a mobile payment transaction. In addition, an improved model is suggested to guide security aspects of system design, which is based on an Onion Layer Framework (Wei, C.Liu, & Koong, 2006).

List of Appended Papers

Parts of the thesis have been published in the following papers.

Paper A) Rice, J. E., & Zhu, Y. (2009, August). A proposed architecture for secure two-party mobile payment. *2009 IEEE Pacific Rim Conference on Communications, Computers and Signal Processing.*

Paper B) Zhu, Y. and Rice, J. E. (2009, August), A Lightweight Architecture for Secure Two-Party Mobile Payment. *7th IEEE/IFIP International Conference on Embedded and Ubiquitous Computing.*

Acknowledgments

I would like to acknowledge many people for helping me during my master work.

I would especially like to thank my supervisor, Dr. Jacqueline E. Rice, for her generous time and commitment. Throughout my master work, she encouraged me to develop independent thinking and research skills. She patiently followed and sustained me in every step of this work.

I am also grateful for having an exceptional master committee and wish to thank Dr. Brian Dobing and Dr. Gongbing Shan for their support and greatly assisting me with scientific writing.

Many people on the faculty and staff in University of Lethbridge assisted and encouraged me in various ways during my course of studies. I am grateful to Dr. Wendy Osborn, Dr. Howard Cheng and Dr. Hua Li for all that they have taught and helped me. I was also inspired pedagogically by Mrs. Nicole Wilson for whom I was a Teaching Assistant. I thank Dr. Jo-Anne Fiske, Mrs. Kathy Schrage, and Mrs. Barbara Williams for their help and encouragement.

I also give a special thanks to Dr. Hiroshi Shimazaki. It was a great honor for me to serve as his last research assistant before the retirement. He is now an Aspiring Landscape Painter.

Thanks to Uncle Zhong, Auntie Danhua and Sister Yunyan for their continuous encouragement and support. Thanks to Wen. Thanks to my friends for their friendship and encouragement.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

With the development of computer technologies and wireless communications, more people are carrying mobile devices. Such devices are rapidly becoming both cheaper and more powerful. A single mobile device can now be employed not only as a cell phone, but also as a GPS, a camera, a music and video player, a text messenger and an Internet connection device (Wilcox, 2005). However, security concerns have so far hindered mobile devices from playing larger roles in financial transactions. The goal of this research is to design and demonstrate a new and secure approach to support payment transactions over a mobile device.

1.1 Motivation

Mobile networking applications allow customers to gain network access anytime and almost anywhere. "The number of people worldwide who possess mobile devices is hurtling toward the two billion mark, and the amount continues to grow, and grow rapidly. It will exceed the number of people who hold bank accounts. It will also exceed the number of people who carry credit cards." (Global Information Inc., 2005) Service providers will compete in this marketplace by offering lower prices but, more importantly, more and better services, including promising opportunities in financial services.

Within financial services, the mobile payment is the fundamental transaction and can be defined as any payment transaction which involves a mobile device (Deans, 2004). With the growing prevalence of electronic commerce, mobile commerce, and the widespread use of mobile devices, mobile payments have already been predicted to have a bright future

in becoming a successful mobile service (Ondrus & Pigneur, 2005). "Eventually, mobile device-based payment transactions will surpass card-based payment transaction." (Bruene, 2007). Ultimately, mobile payments will allow users to pay anyone, anywhere at anytime for any purpose.

1.2 Current Limitations

Security is always a challenge. According to the Unisys Security Index 2008 [1], seventy one percent of the 13,296 consumers surveyed in fourteen countries would not consider online banking or shopping via mobile devices due to security concerns. Less than ten percent of respondents currently employ mobile devices to perform money transfers, credit-card transactions or deposits (Sacco, 2008). Secure strategies are essential to convince mobile users and financial service providers to make use of mobile payment transactions.

Designing a security architecture to protect mobile payment transactions requires trade-offs between security and practicality. A monetary transaction needs comprehensive protection for confidentiality, authentication, integrity and non-repudiation (Park & Song, 2001), which normally requires considerable computational resources. Unfortunately, mobile devices have relatively limited computational resources (less memory, lower CPU speed and an inconvenient I/O interface) compared to PCs (Lai, P.Lin, & Huang, 2006). Therefore, designing a security architecture suitable for resource-limited mobile devices and wireless networks is an important challenge to the success of mobile payment.

Currently several research projects, as well as products on the global level, aim to support emerging two-party mobile payment solutions (e.g., VISA and MasterCard, 1997; Janson, 2007; Itani & Kayssi, 2004; Lam et al., 2003). Hardware and software providers

[1] The Unisys Security Index provides a recurring, statistical measure of consumer concern about four areas of security: national, financial, Internet and personal safety. (http://www.unisyssecurityindex.com/)

for the mobile market, as well as mobile network operators (MNO) and financial service providers, have attempted to specify guidelines for such systems (Karnouskos & Vilmos, 2004). However, no technique has yet been widely accepted for practical implementation.

One may argue that with improving technology it will soon be possible to transplant heavyweight security architectures simply from PC applications to mobile devices. However, current top-end devices cost around $300 (MobileMentalism.com, 2009) which is too expensive for many people, especially those from economically underdeveloped areas. Some software providers and cryptography scientists are taking a different approach, looking for lightweight security strategies capable of running with limited computational resources. These strategies are able to contribute security protection for payment transactions; meanwhile, due to savings on computational resources, the strategies can be implemented in most middle-end, or even low-end, mobile devices. These lightweight architectures can be adapted to top-end mobile devices as well. The key problem of this approach is how to balance security with practicality on mobile devices. Once this problem is solved, the popularity of the technology will lead to a lower cost in building the architecture, and may also contribute to forming a large global consumer market.

Another problem facing current two-party mobile payment systems lies in the mobile service technologies employed in payment transactions. The existing mobile payment systems are mostly based on the Short Messaging Service (SMS) or the Wireless Application Protocol (WAP) (Kuwayama, 2008). However, SMS and WAP have technical limitations that make security architectures very difficult (refer to Subsections 2.2.1, 2.2.2). Although SMS and WAP are likely to dominate mobile payments, the downloadable applications, with their own verifiable security, offer the greatest promise for the future (Bhise, 2009).

1.3 Problem Statement

The objective of this thesis is to design and demonstrate a new security architecture for two-party mobile payment transactions, carried out over resource-limited mobile devices and wireless networks. In existing research focusing on security for mobile payment transactions, there is no solution with an appropriate balance between security and practicality. Some approaches are transplanted from PC-based security strategies, offering good security protection, but are too computationally complex and too time-consuming to be applied to resource-limited mobile devices. Other research offers more computationally lightweight approaches; these solutions generally perform well on resource-limited mobile devices but fail to provide comprehensive security for payment transactions. This thesis contributes a solution offering a good balance between security issues and practical implementations.

A new security architecture is proposed to provide comprehensive security by satisfying the four key requirements of confidentiality, authentication, integrity and non-repudiation for two-party mobile payment transactions; meanwhile, it offers acceptable performance on resource-limited mobile devices. The proposed architecture is computationally lightweight and compatible with a wide range of mobile devices. The proposed architecture can also be implemented as a downloadable application, which avoids limitations existing in SMS- or WAP-based approaches.

1.4 Research Scope

The research aims to provide an application layer architecture for ensuring security during a mobile payment transaction. A mobile payment transaction that has only two participants is defined as a two-party mobile payment transaction. In this work a two-party mobile

payment transaction model is employed, in which payment transactions are carried out between a resource-limited mobile device and a resource-rich business computer server via a wide-ranging wireless network (Itani & Kayssi, 2004). Two-party mobile payment transactions, as defined here, have a wide range of applications, such as mobile banking or mobile stock trading. The *mobile device* is recognized as a resource-limited handheld device, with wireless network connectivity, Internet function, lower-speed CPU and the ability to run Java applications. The business server processing usually has none of these limitations. *Mobile Phone Networks* are employed in this research to provide wide-range wireless networks, such as the General Packet Radio Service (GPRS). In this thesis a mobile banking scenario is used to demonstrate the architecture. The architecture is provided in the form of an application package based on the Java ME platform, as Java ME is the predominant mobile service technology and has been supported by a wide range of mobile device operating systems for a number of years (Riggs & Vandenbrink, 2001).

1.5 Research Contributions

The thesis attempts to offer guidance in matters of designing an application architecture for two-party mobile payment transactions over resource-limited mobile devices. The main contribution is designing a practical security architecture on the application layer. The proposed security architecture is for two-party mobile payment transactions, referred to as *SA2pMP*. It employs a lightweight cryptography scheme combining a public-key cryptography algorithm (ECDSA) and a Symmetric-key cryptography algorithm (AES), a multi-factor authentication mechanism and a distributed transaction log strategy. *SA2pMP* satisfies comprehensive security requirements (confidentiality, authentication, integrity and non-repudiation) for a two-party payment transaction, and is suitable for the current resource-

limited environment of mobile devices and wireless networks.

The simulation demonstrates that the proposed architecture can be practically implemented in a resource-limited mobile device. The proposed architecture employs a Java ME based downloadable software application that can be easily installed in a wide range of mobile devices without any hardware upgrading.

The thesis also contributes to the literature on security models. A Four-layer model of mobile payment participants based on Karnouskos (Karnouskos, 2004) clarifies how participants are involved in mobile payment transactions. An improved research model based on Onion Layer Framework (Wei et al., 2006) is proposed.

1.6 System Development

To measure the practicality of the proposed architecture (*SA2pMP*), a two-party mobile payment system (mobile banking) was simulated on an IBM IntelliStation M Pro PC, with Pentium 4 CPU 2.80 GHz and 2 GB RAM. The operating system was Windows XP Professional SP3.

Three Java ME enabled mobile device emulators, Nokia S60 Emulator (Nokia, 2009), Sony Ericsson Emulator (Sony Ericsson, n.d.), and Sun WTK 2.5.2 CLDC simulator (Sun Microsystems, 2009), were employed for simulation. A Java EE application based on the Apache HTTP Server 2.2, cooperating with Tomcat 5.0, was developed as a banking business service. MySQL Server 5.1 was used as the database server.

The program was developed using NetBeans IDE 6.0[2] and Eclipse SDK 3.4.1[3]. The implementation of cryptography algorithms was aided by the third party cryptography API

[2] http://www.netbeans.org/
[3] http://www.eclipse.org/

provider referred to as Bouncy Castle[4].

The evaluation considered both the time delay and the code size. The evaluation on the time delay shows whether the architecture performs efficiently enough to be accepted by the public, while the evaluation on the code size determines if the architecture can be implemented in limited storage in mobile devices. The detailed evaluation shows that SA2pMP can be feasibly implemented on a Java ME enabled mobile device for secure two-party mobile transactions such as mobile payments.

1.7 Thesis Outline

The remainder of this thesis is organized as follows.

Chapter 2 In this chapter, background knowledge is introduced. Since the development of wireless network technologies leads the appearance and development of mobile payment transactions, wireless network technologies are introduced first, with a focus on the mobile phone network. Several mobile service technologies are then introduced, including SMS, WAP, and Java ME. Mobile service technologies provide the customer with options in building mobile payment systems. In the next an overview of mobile payments and mobile payment models is introduced. Since a mobile payment requires a mobile device, a description of the current status of mobile devices is provided. Several operating systems for mobile devices are introduced as well. A section is provided especially for Java ME since the proposed architecture is based on Java ME. This section provides a relatively comprehensive introduction to Java ME, including the Connected Limited Device Configuration (CLDC), the Mobile Information Device Profile (MIDP), the MIDlet, and

[4]http://www.bouncycastle.org

the Record Management System (RMS). This is followed by the topic of security. As a strong authentication strategy is necessary for a payment system, related background knowledge of authentication is introduced. Both Single-Factor Authentication (SFA) and Multi-Factor Authentication (MFA) are described. Finally, background knowledge of cryptography is introduced, including basic cryptography concepts, Symmetric-key cryptography, and Public-key cryptography. Some cryptography algorithms that can be employed in the proposed security architecture are also described, as is a comparison of these two categories. Since the proposed architecture is implemented with the software, the advantages and limitations of software encryption are explained.

Chapter 3 This chapter analyzes security requirements and describes how the proposed architecture is designed to provide comprehensive security for two-party mobile transactions, with a focus on mobile payment transactions. A description of the scope and limitations of this research is included. In the context of the system design, the mobile devices, the two-party mobile payments, the wireless networks, and the application layer are clearly defined. To clarify how participants are related to a mobile payment, a Four-layer model of mobile payment participants is proposed. The security requirements which the architecture needs to fulfill are then outlined. Based on an Onion Layer Framework (Wei et al., 2006) and the security objectives (Merz, 2002), a security map is proposed to guide the present research. The results of the requirement analysis are explained in this chapter. The technical solutions for system design are recommended from four perspectives: cryptography, protection of authentication, protection of non-repudiation, and the proposed implementation.

Chapter 4 In this chapter a new Security Architecture for Two-party Mobile Payment (*SA2pMP*) is presented. This begins with a description of a detailed network module.

There are three main security strategies involved in the *SA2pMP* security architecture: a lightweight cryptography scheme, a multi-factor authentication strategy, and a distributed transaction log strategy. Corresponding to these strategies, a key management strategy is proposed to protect two pairs of keys for the cryptography algorithms. As the mobile banking transaction is a typical two-party mobile payment, *SA2pMP* is explained in the context of a mobile bank. For the readers' convenience, the beginning of this chapter describes the notations for this thesis.

Chapter 5 This chapter presents how the proposed architecture is simulated on PCs. The business scenario for implementing *SA2pMP* is a mobile banking transaction consisting of money transfer. The hardware and software environment for system simulation is described, followed by an introduction to the specific implementation. The specific evaluations on the time delay and the code size are also explained. Based on evaluations on both the time delay and the code size, *SA2pMP* is demonstrated to have practical applications in Java ME enabled CLDC-1.1 mobile devices, although some visual or multi-modal feedback needs to be considered for some applications.

Chapter 6 This chapter provides comparisons between *SA2pMP* and related work in terms of security and practicality. A J2ME application-layer security architecture (JASA) proposed by Itani and Kayssi (2004), a lightweight security mechanism (LSM) proposed by Lam et al. (2003), the *i*KP protocols proposed by Janson (2007) and Bellare et al. (2000), and SET (VISA and MasterCard, 1997) are first introduced as the comparable candidates, followed by their advantages and limitations. Comparisons are made based on the architecture design, time delay and code size for measuring the security and practicality of the architecture. Based on the comparison, the security and practicality advantages of *SA2pMP* in protecting two-party mobile payment transactions over resource-limited mobile devices

are presented.

Chapter 7 This chapter is a conclusion with suggestions for areas of future work.

Chapter 2

Background

In this chapter background information is provided for wireless network technologies in Section 2.1, mobile service technologies in Section 2.2, mobile payment in Section 2.3 and the current state of mobile devices in Section 2.4. A description of Java ME is provided in Section 2.5, followed by an introduction of authentication in Section 2.6. Finally, cryptography is introduced in Section 2.7.

2.1 Wireless Network Technologies

The focus of this work is on transactions over wireless networks, which are normally provided by mobile network operators. Wireless networks are categorized as low-power, local-area systems or high-power, wide-area systems (Pahlavan & Levesque, 2005). Wireless local area networks (WLANs) are low-power, local-area systems. In contrast, high-power, wide-areas systems are intended to serve large numbers of users who require portability and mobility over wide areas. The mobile phone network (MPN) is the best example of the high-power, wide-areas systems.

The wireless local area network (WLAN) is a wireless alternative to a computer local area network (LAN) designed to provide coverage in a small area, such as a building, home, or office. The main attraction is the flexibility and mobility supported by a WLAN. Compared to MPN where a frequency (channel) is allocated, users in a WLAN have to share frequencies (Ferro & Potorti, 2005).

Not constrained to voice data, data communication over MPN has visibly developed. Since a MPN covers a wider area than a WLAN, mobile payment transactions with a MPN

can provide more convenience, mobility and portability than over a WLAN; therefore, MPN is viewed as more suitable to support mobile transactions than WLAN.

Today MPN are based on either the Global System for Mobile Communication (GSM) or its derivative standards (such as GPRS), or on the Code Division Multiple Access (CDMA). A more specific introduction to MPNs is provided in the following.

2.1.1 Mobile Phone Network

Mobile applications need a wireless communication network. As one of two major types of wireless networks, the mobile phone network (MPN) is believed to have a much higher penetration than WLAN (Parson & Schaeffler, 2001).

MPN is a radio network consisting of a number of cells, each of which is served by one or more fixed transmitters (Parson & Schaeffler, 2001). A mobile telephone or any other mobile device that connects to an MPN will be recognized as a mobile station. A MPN covers a wider area than a WLAN.

MPNs have evolved through several generations of mobile technologies, from first-generation (1G) to second-generation (2G / 2.5G), and third-generation (3G). For example, a large majority of Rogers'[1] customers use the services on second-generation (2G / 2.5G) MPNs. Some of the mainstream standards for second-generation technology are the Global System for Mobile Communication (GSM), the Code Division Multiple Access (CDMA), and the General Packet Radio System (GPRS). These commonly used standards support mobile payment transactions in different levels.

[1] http://www.rogers.com

Global System for Mobile Communication

The Global System for Mobile Communication (GSM) is a second generation standard for mobile communication developed by the European Telecommunications Standards Institute (ETSI) with final ownership belonging to the Third Generation Partnership Project (3GPP) (Vyas & O'Grady, 2001). In GSM, the user is provided not only with a frequency band over which to transmit but also a time interval during the communication. GSM is the most widespread mobile standard currently utilized in Europe and the Asia-Pacific region. Several companies have adopted GSM in the United States. In Canada, two main carriers, Microcell (Fido)[2] and Rogers Wireless (around 2001) operate with GSM (CanadianContent, n.d.).

GSM is able to provide users several services as follows (Scourias, 2003):

- *Telecommunication services*: Currently the most basic telecommunication service supported by GSM is the mobile telephone.
- *Data Services*:
  - *Internet Services*: GSM users can send and receive data at rates up to 9.6 Kbps;
  - *Short Message Service*: SMS is a bidirectional service for short alphanumeric (up to 160 bytes) messages;
  - *Fax*: Sending and receiving fax messages using a GSM phone and a laptop computer;
  - *Secure LAN Access*: Securing access to emails, faxes and file transfer to a corporate LAN.
- *Supplementary Services*: Such services include call forwarding, call barring, caller identification and multiparty conversation (Vyas & O'Grady, 2001).

[2] http://www.microcell.ca

GSM is limited in its low data transmission speed and low bandwidth of data services. Other limitations include the fact that the charge for using GSM is based on the on-line duration and reconnection is required for each session. These limitations make it difficult to use GSM as the main stream network environment for real-time mobile payment transactions.

General Packet Radio Service

In contrast to GSM, the General Packet Radio Service (GPRS) charges only for data received and has much more stable connectivity (McKitterick & Dowling, 2003). GPRS is classified as a 2.5G technology, which means a technology between 2G and 3G. It is an extended service of the GSM network that offers the ability of surfing Internet using a phone at a slightly higher speed than GSM. GPRS Internet surfing speeds range from 9.6 Kbps to 171.2 Kbps (McKitterick & Dowling, 2003). GPRS can be considered as an overlay network on the GSM networks, adding the extra network elements on the GSM infrastructure. A phone with support capacities and a subscription from a supporting network operator is required to operate with GPRS.

The benefit of higher speed and more stable connectivity makes GPRS more suitable for operating mobile applications in real time. GPRS applications include Internet access, email, fax, unified messaging, and Short Messaging Service (SMS). The design of the proposed architecture can be considered to run on a GPRS network environment. For more details about GPRS, refer to Sanders, Thorens, Reisky, Rulik, and Deylitz (2003).

Code Division Multiple Access

As a competitor to GSM, Code Division Multiple Access (CDMA) is a proprietary standard for mobile communication[3]. CDMA was originally proposed by Qualcomm[4], and subsequently developed by Ericsson[5]. CDMA is a wide-band, spread spectrum technology, which means a signal is transmitted on a bandwidth considerably larger than the frequency content of the original information (Vyas & O'Grady, 2001). CDMA allows each user in each cell to transmit on the same frequency channel and at the same time (Glisic & Leppanen, 1997). A unique code is assigned to all conversation. CDMA has been the dominant network standard for North America and parts of Asia. For example, some of largest mobile carriers in Canada provide CDMA, such as Bell Mobility[6], TELUS Mobility[7], and SaskTel[8].

A CDMA call starts with a standard rate of 9.6 Kbps, which is then spread to a transmitted rate of about 1.23 Mbps (McKitterick & Dowling, 2003). That means CDMA has a higher speed than GSM in 2G technology. The benefit of higher speed makes CDMA an option for running the proposed architecture.

Third Generation and Fourth Generation

Companies are introducing the third generation technology as 3G, a generic term for a significant step in mobile technology development. The formal standard for 3G is the International Mobile Telecommunications 2000 (IMT-2000) in the International Telecommunication Union (ITU) family. The three optional modes for the 3G standard are Wideband

[3] GSM is an open standard
[4] http://www.qualcomm.com/
[5] http://www.sonyericsson.com
[6] http://www.bell.ca
[7] http://www.telus.ca
[8] http://www.sasktel.com/

Code Division Multiple Access (W-CDMA)[9] (Europe and the Asian GSM countries), Code Division Multiple Access (CDMA) (North America), and Time Division Duplex/Code Division Multiple Access (TDD/CDMA) (China) (McKitterick & Dowling, 2003). It is expected that IMT-2000 will provide higher transmission rates for 3G: a minimum speed of 2 Mbps for stationary or walking users, and 348 Kbps in a moving vehicle[10] (ITU, 2005). 3G means not only an optimized voice service, but also well suited data communication. With 3G, users get a better support on data communication, such as Internet, electronic commerce, and multimedia communications.

1G, 2G, and 3G systems are likely to co-exist for some time. Following the paradigm of generational changes, the fourth generation (4G) was foreseen to emerge between 2010 and 2015 as an ultra-high-speed broadband wireless network (Bohlin et al., 2004). A very high-speed wireless access of 100 Mbps to 1 Gbps is expected to be provided by 4G mobile communications systems (Adachi, Garg, Takaoka, & Takeda, 2005).

Higher transaction speed provides users better support for data communication. No doubt, more mobile applications such as mobile commerce or multimedia communications will move toward business success with the development of wireless network technologies. We can assume that a mobile architecture designed for the previous generation can also applied to present or future generations. The proposed architecture in this thesis is able to run with good performance in 3G or 4G network environments, although it is designed primarily for the current GPRS.

Different wireless network technologies support different mobile service technologies. The different mobile service technologies are able to fulfill the requirement for mobile payment applications at different levels. The following provides an introduction to mobile

[9] W-CDMA is also referred to as Universal Mobile Telephone System (UMTS).

[10] Second-generation systems only provide speeds ranging from 9.6 Kbps to 28.8 Kbps.

service technologies.

2.2 Mobile Service Technologies

Generational updates for wireless networks enable users to experience various mobile services or applications. Some mobile services or applications offer the opportunity to carry out various types of mobile transactions. This section introduces some popular mobile service technologies, including SMS, WAP, and Java ME.

2.2.1 Short Messaging Service

Short Messaging Service (SMS) is a part of the GSM standard and responsible for sending and receiving short text messages with a limited length to and from mobile phones. It is also presented on most other digital cellular networks such as CDMA and tends to operate in a similar fashion on each network. SMS enables two-way short messages to be sent between GSM subscribers. Using gateways, it is also possible to interchange messages with other systems such as Internet email and the Web.

Currently, SMS is one of the most common and affordable messaging tools for customers. It is not necessary to install any additional software. The banks and other financial sectors can easily send real-time messages to customers. SMS has already been employed in banks to provide mobile services (GoMoNews, 2009a, 2009b; Kuwayama, 2008).

However, SMS has some limitations. SMS limits alphanumeric messages to 160 characters and does not offer a secured environment (Mobile Marketing Association, 2009). These limitations mean that SMS can only serve part of the banking business such as advertising or client alerts, but for business requiring high security SMS has poor performance

as it does not provide a secured environment.

2.2.2 *Wireless Application Protocol*

The Wireless Application Protocol (WAP) was developed by the WAP Forum[11], which has over 500 members. In June 1997, Unwired Planet[12], along with Ericsson[13], Nokia[14], and Motorola[15] announced the formation of the WAP Forum (Mann, Sbihli, & NetLibrary, Inc, 2002). WAP-2.0, as a re-engineering of WAP, was released in 2002. (More detailed information about WAP can be found on each of the founder companies' websites.)

WAP is an open standard for applications over wireless networks. It specifies a series of protocols covering all the protocol layers from the transport to the presentation levels. WAP provides a mechanism for displaying Internet information on a mobile phone or any wireless device by translating Internet information into a format which can be displayed in a resource-limited mobile device (Vyas & O'Grady, 2001). To access Internet content the mobile device needs to be WAP-enabled, and Internet content should be described in Wireless Markup Language (WML) format. A WAP gateway is also essential between the client mobile device and the WML host server, with the goal to translate the WAP request. WML is a mobile equivalent to HTML for web pages.

WAP enabled devices can be handheld digital wireless devices such as mobile phones, pagers, two-way radios, smart phones, and communicators. WAP can work with most wireless networks such as GSM, CDMA, and GPRS. It can also be built on any operating system including PalmOS, EPOC, Windows CE, and JavaOS (WAP Forum, 2002).

[11] http://www.wapforum.org/what/technical.htm
[12] http://www.phone.com
[13] http://www.ericsson.com/se/
[14] http://www.nokia.com
[15] http://www.motorola.com

Although WAP is designed for handheld wireless devices ranging from low-end to high-end, it has some technical limitations. Limited to simple layout of text, images, and input controls, WAP has not provided the expected support for multimedia. Second, WAP is an application which transmits a request to the server. When network connections are dropped, developers have no control over the alert message. Thirdly, a WAP application works only when the phone is connected to WML server, since the WAP browser relies on a WML server to provide the WAP content. These limitations make it inappropriate for implementing a security architecture on mobile devices.

2.2.3 Java ME

The Java Platform, Micro Edition (Java ME) was developed specifically for small mobile devices ranging from smart cards to personal digital assistants (PDAs). Categorized as a mobile service technology, Java ME has the benefit of being cross-platform, and is currently the standard for all mobile devices. For these reasons, Java ME is employed to construct the proposed architecture (see section 2.5 for details).

2.3 Mobile Payment

This thesis aims to provide an architecture for securing two-party mobile payment transactions. However, the proposed architecture is suggested to extend to other two-party mobile transaction applications. In this section an introduction focusing primarily on mobile payment transactions is provided with an overview of mobile payment, followed by an introduction to some popular mobile payment models. Finally, some existing mobile payment protocols are presented.

2.3.1 *Mobile Payment Overview*

A mobile payment can be defined as any payment transaction which involves a mobile device (Deans, 2004). With the growing prevalence of electronic commerce, mobile commerce and the widespread use of mobile devices, mobile payments have already been predicted to have a bright future in becoming a successful mobile service (Ondrus & Pigneur, 2005).

In the financial industry, the number of people worldwide who possess hand-held devices is beyond two billion, and the number continues to grow rapidly, exceeding the number of people with bank accounts and credit cards (Global Information Inc., 2005). Based on a survey provided by Aite Group *LLC.* in March 2007, among twenty-two of the top 100 U.S. deposit institutions, over ninety percent of banks, consider client convenience to be a top driver behind the adoption of mobile banking capabilities (Aite Group, 2007). Mobile devices are playing an increasing role in consumers' lives, and banks already recognize the benefits of deploying mobile banking solutions, which increase their own competitive capabilities. According to TowerGroup Inc.[16], the number of people using their phones for banking activities will be driven up to more than 53 million by 2013 (American Banker, 2009).

Although many observers are projecting that cell phones and PDAs will soon replace wallets and pocketbooks (FSTC Press Releases, 2007), the mobile payment application has not been as successful as anticipated. This can be partially explained by the youth of the market and a lack of standards (Ondrus & Pigneur, 2005). Mobile payment system designers are confronting a large challenge related to security.

In a general mobile payment service, both practicality and security are important factors. Therefore, an acceptable mobile payment platform should balance the requirements

[16]TowerGroup Inc. is an independent research firm owned by MasterCard Inc.

coming from both sides. Unfortunately there is currently no widely accepted protocol or architecture for mobile payments.

2.3.2 Mobile Payment Models

Gao et al. *(2005) Model*

Existing mobile payment systems can be classified into two types (Gao, Cai, Patel, & Shim, 2005). The first type of mobile payment system is mobile point of sale (POS) system that enables customers to purchase products on vending machines with their mobile devices. The second type is account-based payment systems which can be mobile phone-based, smart card, or credit card mobile payment systems, which is the focus of this research.

Karnouskos Model

Figure 2.1 depicts the parties involved in mobile payments (Karnouskos, 2004).

According to Karnouskos Model, the client receives the service of mobile devices from the merchant. In most cases, the client pays the merchant for this service. The merchant acts as an intermediary between the client and the service providers.

The transaction between the service provider and the client involves the other players. The mobile network operators are in charge of the wireless network. They have a large client base and influence all parties involved in a mobile payment. However, they cannot be fully responsible for the mobile payment system, as they have limited experience in payment services. The financial sector has the required experience with an understanding of the complexities of financial transactions (Laukkanen & Lauronen, 2005). The main

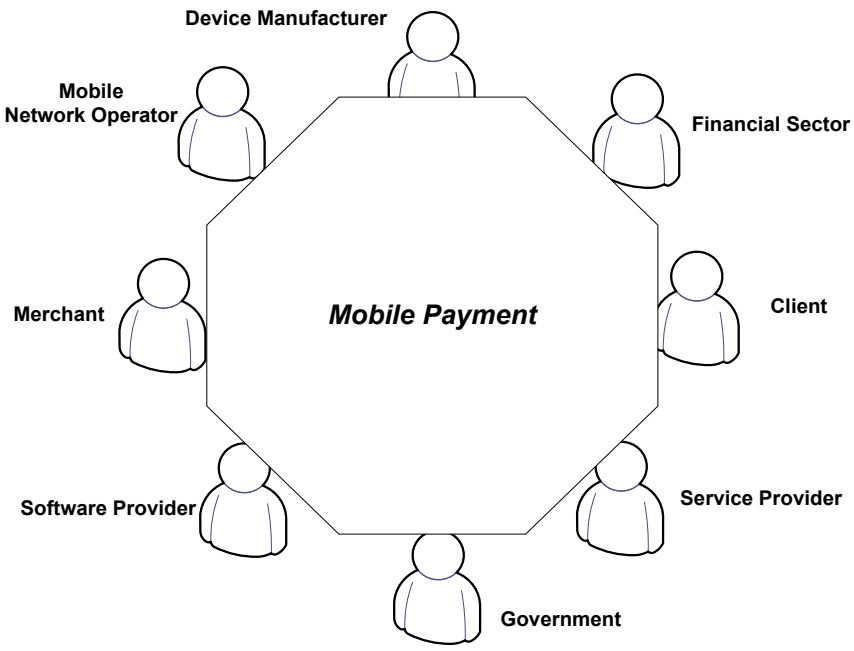


Figure 2.1: The major mobile payment players (Karnouskos, 2004).

players in the financial sector include banks, credit card companies, and other financial institutions (For example, PayPal[17]) (Herzberg, 2003).

The device manufacturers produce the mobile phones that are used by the customers. They control the technology and capabilities of the end-device, which affects the implementation and deployment of the mobile payment services (Karnouskos, 2004). Device manufacturers collaborate in defining mobile device capabilities, leading to the development of devices such as mobile devices and PDAs (Herzberg, 2003).

Another player involved in mobile payment is the software provider. They contribute to the implementation of a mobile payment infrastructure by producing standard compliant software that will connect the different parts of the mobile payment process (Karnouskos, 2004). The government, the last player, is not directly involved with mobile payment but sets standards and regulations for the other players in mobile payments (Karnouskos, 2004).

[17] http://www.paypal.com

According to (Karnouskos, 2004), to ensure that mobile payments are successful and to maintain the efficiency of the services, all players must cooperate and stay open-minded to the development of new technologies and models.

This work aims to design a security architecture for two-party mobile payments which can be extended to ensure security for other two-party mobile transactions, rather than to provide a transaction model for mobile payment; however, it is indispensable for the work to clarify the transaction model between several parties involved in a mobile payment. Although Karnouskos summarized the related parties in a mobile payment, they do not clarify how these different participants are related to a mobile payment, and on which level each participant is involved. This limitation leads this work to extend the model suggested by Karnouskos (2004) to a Four-layer model of mobile payment participants. The Four-layer model can be used to analyze the transaction relationship between the participants involved in a payment transaction; meanwhile, it can also be employed to demonstrate that this work on two-party mobile transactions can be extended to research on multi-party mobile transactions, especially on mobile payment transactions.

The detail of the Four-layer model can be found in 3.1.2.

2.4 Mobile Devices

In a mobile application via wireless networks, a mobile device plays a vital role on the client's side of the transaction. In this section, an overview of mobile devices is provided, followed by a introduction of popular mobile operating systems running in mobile devices.

2.4.1 *Mobile Devices' Overview*

A mobile device is a wireless communication tool that offers users advanced computing capabilities which are often same as a PC functionality (Nambiar, Lu, & Liang, 2004). Normally a mobile device is the size of pocket, making it easy to carry.

Electronic commerce is used to share business information, maintain business relationships, and conduct business transactions via telecommunications networks (Zwass, 1996). A major factor that defines an electronic commerce system as being mobile is the support of mobile devices as a transaction platform for end users (Lam et al., 2003). In order to attain high penetration into mobile commerce, end user convenience is a major concern. Mobile devices such as pocket PCs and smart phones are attractive options for the following reasons (Ginevan, 2002):

- low-cost,
- low battery consumption,
- highly portable,
- wireless capability,
- reasonable computing and display capability for simple transactions, and
- instantaneous power up (no lengthy boot up latency).

Today a large number of different mobile devices exist, but most differences are addressed by the integration of Java Virtual Machines (JVM). The Java Platform, Micro Edition (Java ME) allows a fast deployment of applications that "compile once and run anywhere" (Tillich & Großschadl, 2004). Hence, most of today's devices conform to Java ME.

In this research, it is assumed that the mobile device is a Java-ME enabled phone. The specific definition of mobile device in this research is given in subsection 3.1.1.

2.4.2 Mobile Operating Systems

A mobile operating system is an operating system for a mobile device, which provides a software platform to run application programs. Although most mobile operating systems have been in the alliance supporting Java, they have some differences in performance in running Java applications. The proposed architecture in this research is built on mobile devices with Java ME. The typical mobile operating systems include Symbian, Windows Mobile, Palm, Linux, Android, iPhone OS and BlackBerry OS.

Symbian OS

Symbian OS is a proprietary operating system for mobile devices. It is associated with libraries, user interface frameworks, and reference implementations of common tools for programming. Originally it was produced by Symbian *Ltd.* owned by Nokia, Sony Ericsson[18], Panasonic[19], and Samsung[20]. In 2009, the Symbian Foundation was established to create an open and complete mobile software platform and to make it available at no charge to users (Symbian Foundation, 2009).

There are multiple platforms based upon Symbian OS that provide a software development kit (SDK) for application developers wishing to work on a Symbian OS device. The main ones are UIQ (User Interface Quartz) and Nokia S60. Java ME applications for Sym-

[18] http://www.sonyericsson.com/
[19] http://www.panasonic.com/
[20] http://www.samsung.com/

bian OS are developed using standard techniques and tools such as the Sun Java Wireless Tool kits (formerly the J2ME Wireless Tool kits). They are packaged as a Java Archive (JAR) (and possibly with a Java Application Descriptor (JAD)) files. Since Symbian OS offers good support for Java ME, a Nokia S60 emulator running with Symbian OS was employed in the simulation to demonstrate the practicality of the proposed architecture in this research.

See Symbian Foundation (2009) for further details regarding Symbian OS.

Windows Mobile OS

Windows Mobile is a lighter version operating system combined with a suite of basic applications for mobile devices based on the Microsoft Win32 API. Devices that run Windows Mobile include Pocket PCs, Smartphone, and on-board computers for certain automobiles. It is designed to be similar to Windows desktop versions. Additionally, third-party software development is available for Windows Mobile. Windows Mobile 6.5 is the current version with version 7 planned for release in the Spring 2010 (Chapman, 2009).

Windows Mobile operating system supports third party software development. Developers can choose several options to deploy a mobile application on Windows Mobile. The development of the application normally uses Visual C++ and works with the .NET Compact Framework. Microsoft[21] usually releases Windows Mobile Software development kits (SDKs) which work in collaboration with their Visual Studio development environment. Windows Mobile also includes a Java Virtual Machine to support Java applications.

Microsoft licenses Windows Mobile to a large number of mobile computer or PDA

[21] http://www.microsoft.com

manufacturers that are mostly from the PC industry, such as HP[22], Cisco[23],Philips[24], and LG[25]. However, Windows Mobile has faced problems of simplicity, robustness, synchronization and memory requirements (McKitterick & Dowling, 2003). As Windows Mobile supports Java applications, the proposed architecture is able to be implemented in Windows Mobile.

For more details see Microsoft Corporation (2009).

Palm

Palm OS is a proprietary, embedded operating system initially developed for PDAs by Palm Computing, Inc. in 1996. It has been implemented on a wide range of mobile devices, such as Smartphone, handheld gaming facilities, and GPS devices. Since 2007 Palm OS has also been referred to as Garnet OS (ACCESS Press Release, 2007).

Palm OS is designed for ease of use with a touch screen-based graphical user interface. It is provided with a suite of basic applications for personal information management. Palm OS Garnet applications are primarily coded in C/C++. A Java Virtual Machine (JVM) was previously available for the Palm OS platform; however in 2008 Palm Inc. announced that JVM would no longer be supported by Palm OS. Many successful applications can be installed on a Palm OS device. Especially in the US, the Palm OS has a particularly wide acceptance (McKitterick & Dowling, 2003).

Refer to Palm official website[26] for details.

[22] http://www.hp.com
[23] http://www.cisco.com
[24] http://www.philips.com
[25] http://www.lg.com
[26] http://www.palm.com

Linux

Linux is a Unix-like computer operating system. Linux is one of the most popular examples of free software and open source development, available for free to modify, use, and redistribute. Linux is used as an operating system for a wide variety of computer hardware, including desktop computers, supercomputers, video game systems and embedded devices such as mobile devices.

ABI Research[27] indicates that more than 127 million devices will be enabled with a commercial Linux OS by 2012, up from 8.1 million in 2007 (ABI Research, 2007). Furthermore, device shipments which incorporate Linux as a real-time operating system (RTOS) replacement are set to grow to more than 76 million units in 2012, up from nearly zero in 2007. Linux is currently rated as a low-cost for money, license-free solution for commercial smart mobile phones, and for real-time operating system replacement in middle-tier devices (ABI Research, 2008). On mobile phones, Linux distributions support many programming languages, including C, C++, Java, and FORTRAN. The proposed architecture in this research can be applied to Linux-based mobile devices.

Android

Android is a set of software including an operating system, a middleware and key mobile applications for mobile devices (Open Handset Alliance, n.d.). It was initiated by Google[28] and later developed by Open Handset Alliance[29]. Open Handset Alliance is a group of hardware and software developers, including Google, NTT DoCoMo[30], Sprint Nextel[31],

[27] ABI Research is a market intelligence company specializing in global connectivity and emerging technology.

[28] http://www.google.com

[29] For more details about the Open Handset Alliance, refer to http://www.openhandsetalliance.com/

[30] http://www.nttdocomo.com/

[31] Sprint Nextel is a telecommunications company, based in Kansas, USA

and HTC[32], whose goal is to create a more open cell phone environment.

As an operating system, Android is lightweight with full features. Android allows developers to program in the Java language, to control the device via Google-developed Java libraries, and to run on the Linux 2.6 Kernel. One of the more exciting and compelling features of Android is that third-party applications are executed with the same system priority as those that are bundled with the core system in Android (DiMarzio, 2008). Android provides developers not only software development toolkits and a well-formed library, but also a right to access to anything that the operating system can access. An adequate Java support has the ability to enable the implementation of the proposed architecture in Android mobile devices.

Refer to the Android Official website[33] for details .

iPhone OS

The iPhone OS, also referred to as OS X iPhone, is the operating system developed by Apple Inc. for the iPhone and iPod Touch. On March 17, 2009, Apple presented the blueprint for iPhone OS 3.0 (Apple Inc., 2009).

The iPhone SDK was released on March 6th, 2008. It allows developers to develop applications for the iPhone and iPod Touch, as well as test them in an "iPhone simulator"; however, an iPhone Developer Program fee is charged for loading an application. Apple has not announced that they will enable Java to run on iPhone. Although Sun Microsystems announced plans to release a Java Virtual Machine (JVM) for the iPhone OS, based on the Java Platform, Micro Edition (Java ME), which would enable Java applications to run on the iPhone and iPod Touch (Krill, 2008), it is clear that running Java on the iPhone is not

[32] www.htc.com/
[33] http://www.android.com/about/

included in the iPhone SDK agreement. However, future support for Java ME on the iPhone looks promising. This will enable the implementation of the proposed architecture on the iPhone.

Refer to Apple inc. official website[34] for details.

BlackBerry OS

BlackBerry OS, for BlackBerry smartphones, is the proprietary software platform of the Canadian wireless device company, Research In Motion (RIM), for their BlackBerry Smartphones. The BlackBerry is primarily known for its ability to send and receive e-mail, although it has other functions such as a phone, camera, and media player, and includes maps, organizer, applications, games, and Internet (Research In Motion Limited, n.d.). The BlackBerry OS supports MIDP-2.0 and WAP 1.2.

Since the BlackBerry OS supports a Java ME based application, it provides its Java API for third-party developers, and allows them to build software applications. It supports the proposed architecture. However, any application using certain restricted functions requires a digital signature in order to guarantee an authorship of the application.

For more details, refer to BlackBerry[35] and RIM[36] official websites.

2.5 Java ME

Java is a programming language or a platform which was originally developed by Sun Microsystems. Java applications are compiled into byte-code and are executed on a Java

[34] http://www.apple.com/
[35] http://www.blackberry.com
[36] http://www.rim.com

virtual machine (JVM). One of the advantages of a virtual machine is that the same code can be executed in all environments where one is located. This advantage is referred to as cross-platform. Java ME is the abbreviation of Java Platform, Micro Edition. Since the security architecture implemented in Java ME is proposed in this research, it is necessary to give a more detailed introduction to Java ME. Additionally, as the Record Management System (RMS) will be used, an overview of RMS is also provided.

2.5.1 Java ME: An Overview

Java Platform, Micro Edition (Java ME), is a collection of technologies and specifications that create a platform fitting the requirements for mobile devices such as consumer products, embedded devices, and advanced mobile devices (Sun Microsystems, n.d.-a).

Java ME technology was originally created in order to deal with the constraints associated with building applications for small devices. For this purpose Sun Microsystems[37] defined the basics for Java ME technology to fit such a limited environment and make it possible to create Java applications to run on small devices with limited memory, display and power capacity.

Java ME platform is a collection of technologies and specifications that can be combined to construct a complete Java runtime environment specifically to fit the requirements of a particular device or market. This offers a flexibility and co-existence for all players in the system to seamlessly cooperate to offer the most appealing experience for the end-user.

The Java ME technology is based on three elements:

1. a *configuration* which provides the most basic set of libraries and virtual machine capabilities for a broad range of devices,

[37] http://www.sun.com/

2. a *profile* which is a set of APIs that support a narrower range of devices, and

3. an *optional package* which is a set of technology-specific APIs.

A Java ME application environment includes both a configuration such as the Connected Limited Device Configuration (CLDC) and a profile such as the Mobile Information Device Profile (MIDP). In addition, optional packages provide capability in specific areas of functionality, such as wireless messaging and multimedia capture and playback. CLDC and MIDP is introduced in the following two subsections.

2.5.2 Connected Limited Device Configuration

The Connected Limited Device Configuration (CLDC), for small devices, specifies a framework for Java ME applications targeted at devices with very limited resources, such as mobile phones. A more capable configuration is called the Connected Device Profile (CDC), which was developed under Java Community Process (JCP)[38].

The JCP is an open organization inclusive of active members and non-member public participants. It was established in 1998, and is responsible for the development of Java technology and the approval of Java technical specifications.

Based on the CLDC definition, mobile devices are normally equipped with the following capabilities (Java Community Process, n.d.-b):

- a 16-bit/32-bit processor,
- processor speeds starting from 8-32 MHz,
- at least 160 KB of non-volatile memory allocated for the CLDC libraries and virtual machine,

[38] http://www.jcp.org

- low power consumption, usually operating on battery power, and
- connectivity to some kind of network, often with a wireless, limited (9600 bps or less) bandwidth.

CLDC provides the basic set of libraries and virtual-machine features which are in every implementation of a Java ME environment. Combined with some profiles, such as MIDP, CLDC gives developers a reliable Java platform for developing applications for mobile devices.

2.5.3 Mobile Information Device Profile

The Mobile Information Device Profile (MIDP) is another key element of Java ME. It specifies a profile for the use of Java ME on mobile devices. It defines a platform for dynamically and securely deploying optimized, graphical, networked applications.

MIDP-2.0 is backwardly compatible with MIDP 1.0, with the same target of small, high-volume wireless devices and with the same objective to maintain a tight control on growth in core APIs (Java Community Process, n.d.-a). MIDP focuses on mobile commerce and service-based applications. MIDP-2.0 has more advanced benefits of (Sun Microsystems, n.d.-c):

- rich user interface capabilities,
- multimedia and game functionality,
- extensive connectivity,
- over-the-air provisioning, and
- end-to-end security.

MIDP occupies the layer above CLDC, and define a standard Java runtime environment for most mobile devices.

2.5.4 MIDlets

A MIDlet is a Java application, running on MIDP compliant devices (Debbabi, Talhi, & Zhioua, 2007). Programmers writing a MIDlet are restricted to CLDC APIs, MIDP APIs and optional packages. A MIDlet consists of at least one Java class which is derived from the abstract class *javax.microedition.midlet.MIDlet*.

A group of MIDlets can be collected into one *MIDlet suite*. The MIDlet suite is packaged and implemented into a mobile device as a single entry that can only be removed as a group. MIDlets in the same MIDlet suite share static and runtime resources. Normally a MIDlet suite is packaged in a Java Archive (JAR) file. The package information is provided in a Java Application Descriptor (JAD) file. A MIDlet is designed for simulating a two-party mobile payment transaction to evaluate the proposed architecture.

2.5.5 Record Management System

MIDP specification requires that a platform provides persistent storage via nonvolatile memory. The storage can be viewed as a simple record-oriented database, referred to as the record management system (RMS) (S. Ghosh, 2002). RMS in MIDP manages several Record Stores, which are simply flat files containing binary data. Every piece of data in a record has an associated numeric record ID that is characteristic to each record store. Each record store has a name that must be unique within MIDlet suite that created it. MIDlets can access only record stores created by themselves or others in the same suite. When a

MIDlet suite is removed from a device, all its associated record stores are deleted.

The *javax.microedition.rms* package contains a *RecordStore* class that provides rudimentary access to data in a record store. Record store implementation ensures no corruption of data will occur with multiple accesses. The record store is time-stamped to mark the last modification time, and maintains a version which is an integer that is incremented for each operation that modifies the contents of the record store. Each record in a record store is an array of bytes and has a unique integer identifier. The RMS APIs (Mahmoud, 2000; Giguere, 2004):

1. allow MIDlets to manipulate (add and remove) records within a record store,
2. allow MIDlets in the same application to share records, and
3. support to prohibit sharing records between different applications (in MIDP-2.0).

RMS is a good choice of designers wanting to store persistent data in mobile devices. In the proposed architecture, RMS is an option for storing the encryption key in the key management strategy. Refer to Subsection 4.1.6 for design details.

2.6 Authentication

A mobile transaction system, particularly for banking payments, requires an advanced security architecture which implements a strong strategy to protect authentication. In this section, an introduction to authentication is provided. *Authentication* is defined, followed by an introduction to single-factor authentication (SFA) and multi-factor authentication (MFA).

2.6.1 *Authentication: Definition*

In computer security, authentication is used to ensure that the communicating entity is who they are claiming to be (Stallings, 2006). According to the Federal Information Processing Standards (FIPS) 200, authentication verifies "the identity of a user, process, or device, often as a prerequisite to allowing access to resources in an information system" (NIST, 2006).

Authentication is regularly confused with the related term of authorization. The difference between authentication and authorization is that authentication is the process of verifying one's identity while authorization is the process of verifying that a known entity has the authority to perform certain operations. Based on authentication, systems or users are offered different levels of authority (Krawetz, 2006).

Authentication can be accomplished by using any combination of three kinds of factors (Maiwarld, 2004; Schneider, n.d.):

- *something you have*: ID card, security token, software token, phone, or cell phone,
- *something you know*: a password, pass phrase, or personal identification number (PIN), or
- *something you are*: fingerprint or retinal pattern, DNA sequence (there are assorted definitions for what is sufficient), signature or voice recognition, unique bio-electric signals, or another biometric identifier.

A broad range of authentication methods and properties is shown in Table 2.1.

Categorization based on how many authentication factor(s) be employed, authentication strategies can be divided into single-factor authentication and multi-factor authentication.

Table 2.1: Authentication methods and their properties.

Method	Examples	Properties
What you know	ID,PIN, Password	Shared, Forgotten Easy to guess
What you have	Cards, Badges, Keys	Lost Shared,Duplicated,
What you know and have	PIN for ATM	Shared, Weak PIN
Something unique	Fingerprint, face, voiceprint	Not possible to share Repudiation unlikely; Forging difficult; Cannot be lost or stolen

2.6.2 Single-factor Authentication

Single-factor authentication (SFA) is a traditional security process that requires one of the elements (something you have, something you know, or something you are) (Maiwarld, 2004) before granting access rights to the user.

Offering a password is the most basic method for single-factor authentication. Like the Password Authentication Protocol (PAP), it offers a basic access strategy for logging onto a network (Hassell, 2002). A table of usernames and passwords is stored on a server. When users log on, their usernames and passwords are sent to the server for verification. SFA security relies on the diligence of users, who should take additional precautions, such as creating a strong password and protecting it from access by others. However, SFA security has its limitations. "Something you have" can be stolen, while "Something you know" can be guessed, shared or lost to other methods. "Something you are" is commonly the strongest method; however, the implementation is costly and there is still the possibility of access being compromised (EDS.com, n.d.).

2.6.3 *Multi-factor Authentication*

For applications that require greater security, it is advisable to implement multi-factor authentication (MFA), which is sometimes referred to as a strong authentication.

In U.S. Government National Information Assurance Glossary (The United States Federal Government, 2006), strong authentication is defined as a "layered authentication approach relying on two or more authenticators to establish the identity of an originator or receiver of information." This means that a system with a strong authentication property must require multiple factors for user authentication and use advanced technology to verify a user's identity. Financial sectors such as banks hold high risk systems.

Given the limitations of single-factor authentication, the logical alternative is two-factor authentication, in which two of the elements are applied. An example is a system that is employed to authenticate the automated teller machine (ATM) users (Firesmith, 2003). The system combines a card (what you have) with a personal identification number (what you know).

Requiring more than one factor significantly enhances security because single-factor authentication by itself may not be sufficient to perform authentication that can be relied on. Accordingly, properly designed and implemented multi-factor authentication methods are more reliable. To offer secured and reliable services to their customers, financial systems, such as payment systems, should employ multi-factor strategy for authentication.

2.7 Cryptography

Cryptography is the art and science of securing messages so unintended audiences cannot read, understand, or alter the message (Tipton & Krause, 2003). A security architecture

must have a cryptography implementation, as cryptography helps provide confidentiality, integrity and even non-repudiation. The goal for proposing the architecture is to design a mobile payment security architecture which is able to be implemented practically on resource-limited mobile devices and is compatible on wireless networks. The employment of cryptography strategy contributes to sufficient security required by a monetary architecture but often imposes computational requirements that cannot be met by resource-limited environments. In this section basic cryptography concepts are introduced, followed by Symmetric-key cryptography and Public-key cryptography. The representative algorithms are introduced in these subsections. Symmetric-key cryptography and Public-key cryptography are compared and finally the advantages in software implementation of cryptography is analyzed.

2.7.1 Basic Cryptography Concepts

The fundamental objective of cryptography is to enable two parties to communicate over an insecure channel while ensuring that a third party cannot understand what is being said (Stinson, 2002). Modern cryptography concerns the construction of information systems that are robust against malicious attempts to make these systems divert from their prescribed functionality (Goldreich, 2005).

A pair of transformations exists in cryptography: encryption and decryption. Encryption converts a meaningful chunk of data (plaintext) to a meaningless chunk of data (ciphertext), which is unreadable to anybody except those possessing knowledge of the data or its key, while decryption transforms the ciphertext back into the plaintext. Plaintext has a wide range of meanings, such as a stream of bits, a text file, or a digitized voice or image (Schneier, 1994). In this research, Plaintext is a text message.

A cryptography algorithm is a mathematical function which is used to perform encryption and decryption. It can be implemented in software, or hardware, or both. The cryptography algorithm used between communication parties should be the same. A communication party is someone who sends, receives, or manipulates data. In a two-party communication, a sender is the transmitter of data, who encrypts the plaintext and sends out the ciphertext. A receiver is the intended recipient of information, who decrypts the ciphertext and recovers the plaintext. The intruder, a third party who tries to recover the plaintext, is one of the two most publicized threats to security (the other is viruses) (Stallings, 2006).

Most cryptographic systems incorporate random number algorithms. A random number is seeded with a key, which is used by cryptography algorithms to vary the ciphertext. This means that plaintext encrypted with different keys generates different ciphertexts (Krawetz, 2006). Without the correct key, the intruder cannot recover the plaintext.

Commonly, two forms of cryptography are in use: Symmetric-key cryptography and Public-key cryptography. Subsection 2.7.2 provides an introduction to Symmetric-key cryptography and subsection 2.7.3 gives a survey of Public-key cryptography.

2.7.2 Symmetric-key Cryptography

Symmetric-key cryptography works on the basis that the same key is shared between the sender and the receiver. Alice (a sender) and Bob (a receiver) are used as an example to describe a two-party communication with the Symmetric-key encryption:

1. Alice and Bob share a secret key.
2. Alice encrypts the plaintext with the key, and sends the ciphertext to Bob.
3. Bob decrypts the ciphertext using the same key.

The two primary types of Symmetric-key cryptography are stream ciphers and block ciphers. A block cipher transforms a block of plaintext with a fixed size into a block of ciphertext with equal length. A block size is typically of 64 or 128 bits (Stallings, 2006). The Advanced Encryption Standard (AES) is an example of block ciphers. A stream cipher operates on data stream. It encrypts a digital data stream one bit or one byte at a time. The Rivest Cipher #4 (RC4) is a stream cipher (Krawetz, 2006). The byte-by-byte encoding method of stream ciphers is not suitable for software implementation. It also affects significantly the throughput of hardware implementations. As a result, current stream ciphers have been designed to support a certain length of plaintext to be encrypted at a time.

Schneier (1994) outlines the advantages and disadvantages of block ciphers and stream ciphers. The primary benefit of block ciphers is that the security is stronger in principle, whereas stream ciphers are more easily analyzed mathematically and therefore less secure. The main limitation of block ciphers is that the software and hardware implementations are more complicated than stream ciphers. Because of the security concerns, block ciphers are considered in the present study.

The main problem with Symmetric-key cryptography is that the sender and receiver have to share the same secret key. If they are in separate physical locations, they must trust a courier, or a phone system, or some other transmission medium to protect the secret key. Anyone who overhears or intercepts the secret key in transition can read, modify, or falsify messages encrypted with that key. *Key management* is in charge of the generation, transmission and storage of keys. Because all keys in a Symmetric-key cryptography algorithm must be kept secret, it is essential to also provide secure key management for a Symmetric-key cryptography approach. Refer to (Stallings, 2006) for further information.

AES is a well-known Symmetric-key cryptography algorithm, and is described in detail in the following subsection.

Advanced Encryption Standard

The Advanced Encryption Standard (AES), a typical symmetric-key cryptography algorithm, was published by the National Institute of Standards and Technology (NIST) in 2001. It was intended to replace the Data Encryption Standard (DES), including Triple DES as the approved standard for a wide range of applications and platforms (Stallings, 2006). AES was originally published as Rijndael algorithm (Daemen & Rijmen, 1999, 2006), and is named after its developers, Belgian researchers Rijmen and Daemen.

AES is a symmetric block cipher algorithm, with a fixed block size of 128 bits and a cipher key size of 128, 192, or 256 bits. AES performs four operations on a 128-bit block of data for a certain number of repetitions. The four operations are SubBytes, ShiftRows, MixColumns, and AddRoundKey. The number of these repetitions depends on key size. For example, an AES algorithm is implemented to encrypt plaintext block size of 128 bits, with cipher key size of 256 bits, then 14 rounds of these four operations are required. Every state of AES operation can be pictured as a 4 x 4 matrix of bytes. All the intermediate results of the 128-bit block, as well as the input and the output block, are called states. At each stage of the transformation, the block of data is transformed from its current state to the new state according to the different operations.

SubBytes is a simple "table lookup" operation. It uses a substitution table (*S-box*) to perform a byte-by-byte substitution of the block. *S-box* is a 16 x 16 matrix of bytes, which contains a permutation of all possible 256 8-bit values. *InvSubBytes* is the inverse operation of SubBytes.

ShiftRows is the forward shift row transformation. Each row of the state is shifted left over a different number. Figure 2.2 depicts that the first row of state performed a 0-byte circular left shift. The second row performed a 1-byte circular left shift. For the third row, a 2-byte circular left shift is performed, and a 3-byte circular left shift is performed for the

fourth row of state. The inverse operation of ShiftRows, *InvShiftRows* shifts to the right on decryption.

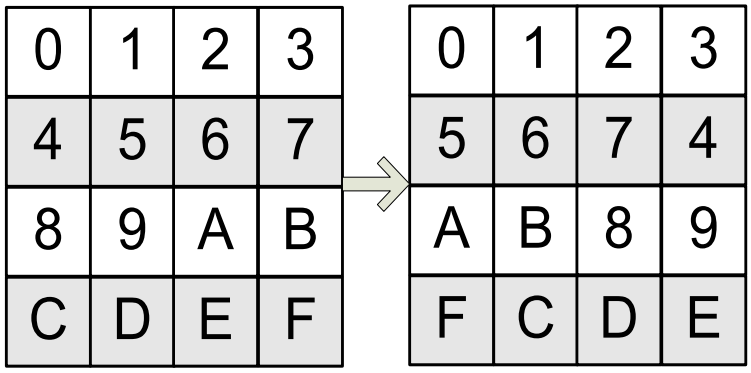


Figure 2.2: AES Shiftrows Transformation (Stallings, 2006).

MixColumns operates on each column individually. Each byte of a column is mapped into a new value. Each column is treated as a polynomial over $GF(2^8)$, and is multiplied modulo (x^4+1) by the constant polynomial

$$a(x) = \{03\}x^3 + \{01\}x^2 + \{01\}x + \{02\} \tag{2.1}$$

The inverse operation of MixColumns, *InvMixColumns* operation is a multiplication of each column with

$$b(x) = a^{-1}(x) \quad mod \quad (x^4+1) = \{0B\}x^3 + \{0D\}x^2 + \{09\}x + \{OE\} \tag{2.2}$$

AddRoundKey performs a logical *XOR* of the 128 bits of state with the 128 bits of the round key. The inverse operation of AddRoundKey is to forward another AddRoundKey transformation, as the XOR operation is its own inverse.

The AES algorithm is required to be royalty-free for use worldwide and offers security of a sufficient level to protect data for the next 20 to 30 years. AES is simple for the design purpose, and supports a cipher key size varying from 128 bits to 256 bits. Although the number of rounds is fixed in the specifications, this can be modified as a parameter in case of security problems (Daemen & Rijmen, 1999). AES can be implemented on a Smart Card using a short count of code and cycles (Sanchez-Avila & Sanchez-Reillol, 2001). When comparing speed and reliability in implementations, it has better performance in both software and hardware than the other symmetric algorithms such as MARS (submitted by IBM Research), RC6 (submitted by RSA Security), and Twofish (Schneier). (Dray, 2000; IBM MARS Team, n.d.; Rivest, Robshaw, Sidney, & Yin, n.d.; Schneier et al., 1998).

AES also has other advantages mentioned below. The proposed architecture employs AES as part of its cryptography solutions. (Refer to Subsection 3.4.1 for details.)

2.7.3 Public-key Cryptography

The concept of Public-key cryptography was proposed in 1976 by Diffie and Hellman (Diffie & Hellman, 1976) in order to solve the key management problems. In their technique, each party gets a pair of keys, one of them is referred to as the public key and the other as the private key. Each party's public key is published while the private key remains secret. It is not necessary to share the secret key between the sender and receiver. All communications involve only public keys and no private key is ever transmitted. It is no longer necessary to trust some communication channels to be secure against eavesdropping or betrayal. The only requirement is that public keys are associated with their users in a trusted manner (for instance, in a trusted database).

In Public-key cryptography the two-party communication encryption works as follows:

1. Alice gets Bob's public key from the database.

2. Alice encrypts her message using Bob's public key and sends it to Bob.

3. Bob then decrypts Alice's message using his private key.

The public key is not always used for encryption. The private key can also be employed to create ciphertext. In this scenario, anybody holding the public key can decode the ciphertext. This approach is commonly referred to as a digital signature. The sender cannot falsely claim that they did not encode a plaintext, and all receivers know that the plaintext is authentic (Krawetz, 2006).

A comprehensive approach therefore involves a sender who may use private and public keys for encoding a message. The receiver's public key ensures message confidentiality, and the sender's private key signs the encryption, which ensures non-repudiation. (Refer to section 3.2 for information about confidentiality and non-repudiation.)

The digital signature is one application of Public-key cryptography, employing the private key to encrypt a message or message digest, while making use of the public key to decrypt the ciphertext. The following provides an introduction to digital signatures.

Digital Signature

A *digital signature* is not only used to protect data integrity but also used to achieve authentication and non-repudiation (Stallings, 2006). A digital signature mechanism can be employed to authenticate the identity of the sender of a message, and sometimes to ensure that the original content of the message that has been sent is unchanged. Digital signatures can protect the two parties against each other, because there is no complete trust between sender and receiver. The digital signature approach is analogous to a handwriting signature

on a paper document (Stallings, 2006).

A digital signature uses a private key to encrypt the messages and a public key to decrypt messages. By comparing the signature with the original message it is possible to see that the message has not been changed and that it has been signed using a particular key-pair.

In digital signature schemes, three main properties are required:

- verification of the sender at the time of the signature,
- authentication of the content at the time of the signature, and
- verifiability by third parties to resolve disputes between sender and receiver.

Bob and Alice can again be used to illustrate this concept. Alice is supposed to send a message to Bob. Alice has to provide Bob the assurance that the message is unchanged from what she sent and that it is really from Alice. To do so, Alice uses her private-key of a key pair to encrypt the message to produce a digital signature. Alice then sends the message along with this digital signature. As discussed in 2.7.3, most digital signature implementation in practice only signs the message digest, not the entire message.

Since Alice's public-key is the only key that can decrypt that message, Bob constitutes a successful decryption with Alice's Public-key making a *digital signature verification*, which means that Alice's private key did encrypt the message. In practice, the message digest which Bob makes the *hash value A* from the received message is compared with *the hash value B* decrypted from the digital signature with Alice's public-key. If these two hash values match each other, the received message is successfully verified. In other words, the message Bob received is unchanged from what Alice sent and the message Bob received is really from Alice. Figure 2.7.3 illustrates the digital signature signing and verification process.

A digital signature includes three process steps: a key generation process, a signature signing process, and a signature verifying process.

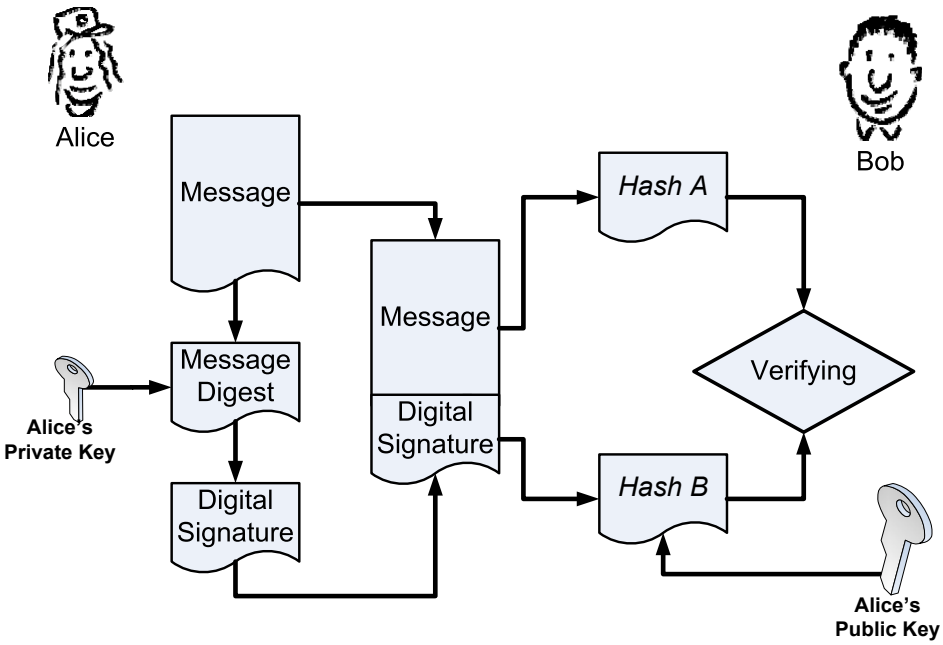


Figure 2.3: Digital Signature (Stallings, 2006).

- A key generation algorithm selects a private key uniformly at random from a set of possible private keys. The algorithm outputs the private key and a corresponding public key.
- A signing algorithm produces a digital signature for the given message (or a message digest) by a private key.
- A signature verifying algorithm outputs a Boolean value representing accept or reject, with the input values of a given message (or a message digest), a public key, and a digital signature.

Some public-key cryptography algorithms are suitable for digital signature and/or encryption/decryption. Table 2.2 indicates the applications supported by the Digital Signature Algorithm (DSA), the Rivest-Shamir-Adleman algorithm (RSA) and the Elliptic Curve Digital Signature Algorithm (ECDSA) discussed in this chapter.

Table 2.2: Applications for DSA, RSA, and ECDSA.

Algorithm	Encrypt/Decrypt	Digital Signature
DSA	X	√
RSA	√	√
ECDSA	X	√

Although the digital signature application is the only application for the public-key cryptography algorithms considered in the present research, there are options for various digital signature algorithms that can be implemented in the architecture. In the following paragraphs, typical digital signature algorithms (DSA, RSA, and ECDSA) are introduced. Because of the benefit of shorter key size and higher security level, ECDSA is employed in the cryptography solutions of the proposed architecture. (The cryptography solutions are described in detail in subsection 3.4.1.)

Digital Signature Algorithm In 1991, the National Institute of Standards and Technology (NIST) published the Digital Signature Standard (DSS) in Federal Information Processing Standard FIPS 186 (Stallings, 2006). An expanded version of DSS was published as FIPS 186-2 in 2000 (see (NIST, 2000)). The DSS makes use of SHA to present a digital signature technique, which is the Digital Signature Algorithm (DSA). Unlike RSA, DSA only provide digital signature function used for encryption or key exchange.

DSA is based on the difficulty of computing discrete logarithms. The schemes on which DSA is based are originally presented by ElGamal (1985)) and Schnorr (1991). Stallings (2006) and NIST (2000) provides further details of DSA.

The *Rivest-Shamir-Adleman* Algorithm The Rivest-Shamir-Adleman algorithm (RSA) is for Public-key cryptography. It was the first algorithm for digital signature and encryption/decryption. RSA has been widely employed in electronic commerce protocols. The transaction in an electronic commerce is considered secure if given sufficiently long keys

and up-to-date implementations.

RSA was publicly developed in 1977 by Rivest, Shamir, and Adleman at MIT and then published in 1978 (Rivest, Shamir, & Adleman, 1978). The security of RSA relies on the fact that it is hard to factorize a product of two large prime numbers. RSA can be used to implement digital signatures. The Security Division of EMC[39] has published two different signature schemes, RSASSA-PKCS1-v1_5 and RSASSA-PSS, in the PKCS#1 specification (RSA Laboratories, 2002). RSASSA-PSS is the recommended algorithm as its encoding uses a random value, therefore, it is more robust against attacks (RSA Laboratories, 2002).

RSA is a block cipher in which the plaintext and the ciphertext are integers between 0 and $n-1$ for some number n (Stallings, 2006). A typical size for the number n is 1024 bits. The security of RSA with a key size of 1024 bits is equivalent to a symmetric setting of 80 bits (Barker, Barker, Burr, Polk, & Smid, 2007).

Refer to (Stallings, 2006; RSA Laboratories, 2002; Rivest et al., 1978) for further information of RSA.

ECDSA Elliptic curve cryptography (ECC) is an approach to Public-key cryptography based on the algebraic structure of elliptic curves over finite fields (Koblitz, 1987). Compared to RSA, ECC appears to provide equal security using much smaller key size (approximately one-eighth the key size), thus reducing processing overhead (Stallings, 2006). For example, a 160-bit ECC key provides the same level of security as a 1024-bit RSA key, and 224-bit ECC is equivalent to 2048-bit RSA (Chang, Eberle, Gupta, & Gura, n.d.). This means ECC offers faster computations, lower power consumption and memory, and bandwidth savings. These properties are useful for mobile devices which are typically limited in the CPU resources, power and network connectivity. Thus, ECC is theoretically more

[39] http://www.emc.com

suitable for securing mobile banking than RSA.

Elliptic curves used in cryptography are defined over two kinds of fields:

- *prime curves*: $GF(p)$, p is a large prime number
- *binary curves*: $GF(2^m)$, 2^m element are binary polynomials

Elliptic Curve DSA (ECDSA) is one approach to realize the Digital Signature Algorithm (DSA). ECDSA was first proposed by Vanstone (1992), in response to NIST's request for public comments on the proposal for DSS. It was accepted in 1998 as an ISO (International Standards Organization) standard (ISO 14888-3), accepted in 1999 as an ANSI (American National Standards Institute) standard (ANSI X9.62), and accepted in 2000 as an IEEE (Institute of Electrical and Electronics Engineers) standard (IEEE 1363-2000) and a FIPS standard (FIPS 186-2) (Johnson, Menezes, & Vanstone, 2001).

Good overviews of elliptic curve cryptography can be further found in Koblitz, Menezes, and Vanstone (2000) and Lopez and Dahab (2000a).

Hash Function

Hash functions are employed in conjunction with Public-key cryptography algorithms to produce digital signatures. In the following an introduction to hash functions is provided.

When implementing a digital signature, it is unusual to encrypt a whole message for security and performance reasons. The proposed architecture employs a hash function to create a *message digest*, which is used to generate the digital signature for the message.

A *hash function* works on a message with an arbitrary length, and returns a fixed-size hash value (Delfs & Knebl, 2002). This hash value is sometimes called *message digest* or *digital fingerprint*. The ideal cryptography hash function has four characteristics:

- it should be simple to calculate the *message digest* for any given message,
- it should be computationally impractical to find a message with a given *message digest*,
- it should be computationally impractical to alter a message without modifying its *message digest*, and
- it should be computationally impractical to find two different messages with the same *message digest*.

Hash functions are widely used currently. The message digest can be used in creating digital signature schemes. For security and performance reasons, most digital signature algorithms specify only to sign the digest of the message, not the entire message. In addition, a hash function can be used to control the integrity of a message. Determining whether any changes have been made to a message (or a file), for example, can be accomplished by comparing message digests calculated before, and after, transmission or any other event.

Secure Hash Algorithm The Secure Hash Algorithm (SHA) is the most widely used hash function. It was developed by NIST and published as FIPS 180 in 1993. In 1995, a revised version was published as FIPS 180-1. This revised version is generally called SHA-1 or Secure Hash Standard in the standards document. Currently the updated Secure Hash Standard is FIPS 180-3, which was published in October 2008 (NIST, 2008).

SHA-1 is the most established of the SHA hash functions, and has been employed in widely used security applications and protocols. SHA-1 calculates a condensed representation of a message. When a message of any length < 264 bits is input, the SHA-1 produces a 160-bit message digest (NIST, 1995).

2.7.4 A Comparison of Symmetric-key and Public-key

The two commonly used categories are Symmetric-key cryptography and Public-key cryptography. Each of them has its own advantages and limitations. In this subsection, a brief comparison of these two cryptography categories is provided.

The main advantage of Public-key cryptography is increased security: only the private key needs to remain secret (Mollin, 2003). In contrast, a Symmetric-key cryptography system has to transmit the secret keys (either manually or via a communication channel); therefore, it is possible for an intruder to discover the secret keys during their transmission.

A disadvantage of using Public-key cryptography for encryption is efficiency. There are many popular Symmetric-key encryption algorithms significantly faster than any existing Public-key encryption algorithms such as DES. The trade-off process between operation speed and security level must be evaluated, particularly for resource-limited mobile devices. The good news is that Public-key cryptography computation capabilities and operation speed are increasing significantly with the development of mobile device manufacturing technology.

It is not necessary to choose between Public-key and Symmetric-key cryptography. A hybrid of Public-key and Symmetric-key cryptography combines the benefits of both. One type of hybrid cryptography involves a symmetric key algorithm being used to hide the object while a public key mechanism is used to manage the keys of this symmetric algorithm (Tipton & Krause, 2007). It is a good combination for implementing Public-key cryptography along with Symmetric-key cryptography on the systems that make use of mobile devices. In the proposed architecture, both Public-key and Symmetric-key cryptography algorithms are employed.

As the present research is proposed on a software application level architecture, it is

necessary to explain the rationale for software implementation for cryptography.

2.7.5 Software Encryption

In this subsection an argument for the benefits of software encryption is explained.

It is necessary to consider hardware or software when an encryption is implemented or integrated in an electronic commerce. Each cryptography algorithm, which can be implemented in either hardware or software, has its related costs and benefits.

Generally, hardware encryption has better performance than software encryption. Unfortunately, there are several problems with implementing cryptography algorithms in hardware. One problem is that cryptography hardware is not ubiquitous, cheap, or readily exportable (Nahum, O'Malley, Orman, & Schroeppel, 1995). It is impractical to add more than one piece of cryptography hardware to a given network facility, such as a mobile device or a network server.

The software encryption is currently becoming prevalent in business applications. Any encryption algorithm can be implemented in software. The advantages of software implementations relate to cost and portability. The software implementations can be inexpensively implemented and copied on many computational facilities. Furthermore, regarding mobile devices, almost all manufacturers support the standard of Java ME technology running on their products. The popularity of Java ME, the adoption of the security and Trust Services API (Sun Microsystems, n.d.-b), and some third party security software development kits (SDKs) contribute a good basis for software encryption.

This chapter provided background knowledge related to mobile transaction security. The next chapter analyzes security requirements and explains the model for designing ar-

chitecture to provide comprehensive security for two-party mobile payment transactions.

Chapter 3

System Analysis

This chapter analyzes security requirements and describes how the proposed architecture is designed to provide a comprehensive security for two-party mobile payment transactions. Constraints and limitations of the present research are described in section 3.1, basic security objectives which the proposed architecture is to protect are introduced in section 3.2, and the security map model based on an Onion Layer framework is explained in section 3.3. Finally the strategy and solutions employed in the proposed architecture are suggested in section 3.4.

3.1 Research Constraints

In this section, mobile devices, the payment type and the wireless network environment are defined, followed by an explanation of the reason for the proposed architecture focusing on the application layer security.

3.1.1 Mobile Devices

In this research, mobile devices are recognized as handheld devices with capabilities such as General Packet Radio Service (GPRS) connectivity, Internet browsing, and basic computation. Smart phones or Personal Data Assistants (PDAs) are examples of mobile devices in the present research. Another characteristic of mobile devices in this research is that they are Java enabled. (Refer to section 2.4 for background knowledge of mobile devices.) Based on MIDP-2.0 and CLDC-1.1, a suitable mobile device is assumed to possess a mini-

mum base memory of 192 KB and a 16/32 bit processor with a speed ranging from 8 to 32 MHz.

A mobile device is often viewed as a part of the identity of an individual (Roussos, Peterson, & Patel, 2003). In this research we treat a mobile device in a similar way as, for example, a credit card; that is, each person has their own credit card and would not normally share it with anyone. It is assumed that the same situation will hold true for a mobile device. In such a way a mobile device can be considered in a sense to be a suitable personal identifier.

3.1.2 Two-party mobile payments

A mobile payment can be defined as any payment transaction which involves a mobile device (Deans, 2004). Although much research on mobile payments places emphasis on three parties or more (Peiro et al., 1998; Ham et al., 2002), the transaction between two parties is the basic payment transaction. Therefore, the model on which the present research focuses is the two-party mobile payment transaction.

Mobile Payment Participant Layers

Karnouskos (2004) outline a mobile payment model in which a group of payment participants are involved. (Refer to Section 2.3 for details.) Although this model summarizes a mobile payment in relation to several different participants, it does not clarify how the various participants are related to a mobile payment, and on what level of involvement a participant is related to a mobile payment.

To address this limitation these participants are categorized into four layers based on

how they are related to a mobile payment. These four layers include the direct payment layer, the network layer, the technical component layer and the supervision layer. Figure 3.1 depicts the four layers of participants involved in the mobile payment process.

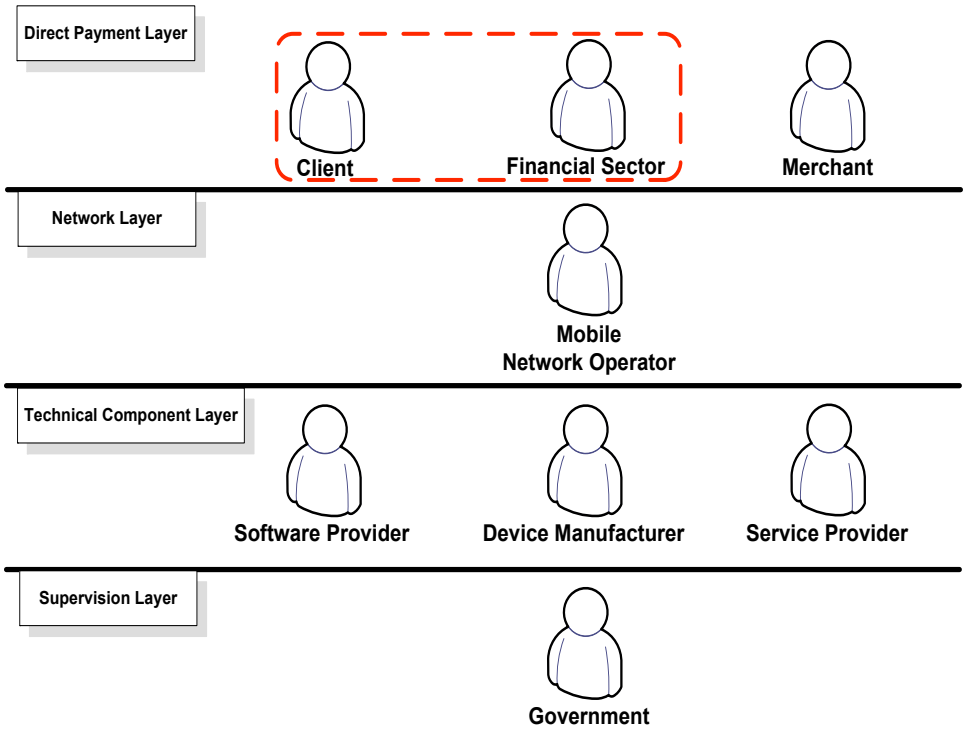


Figure 3.1: Four-layer structure of mobile payment participants.

- *Direct Payment Layer*: The client, the financial sector and the merchant are the direct participants who are involved in a mobile payment. Therefore, there are three participants in the direct payment layer.

- *Network Layer*: A mobile payment is based on a wireless network, which is maintained by the mobile network operator. The mobile network operator is located in the network layer.

- *Technical Component Layer*: The software provider, the device manufacturer and the service provider are located in the technical component layer. The software provider

produces software components that connect different participants in the direct payment layer in a mobile payment transaction, while the device manufacturer provides the mobile devices as the client facility component. The service provider designs the specific payment service component for the direct payment players. The technical components assist the execution of a payment transaction. These component providers offer a technical environment to run mobile payment transactions.

- *Supervision Layer*: The government is not directly involved in mobile payments but outlines standards and ordinances for all other participants. For example, the government may prevent the service providers from holding a monopoly, and can regulate the device manufacturer and the software provider to ensure compatibility between devices (Karnouskos, 2004). Therefore, the government is located in the supervision layer.

Based on the Four-layer mobile payment participant model, the mobile payment can be categorized according to how many participants are involved in the direct payment layer.

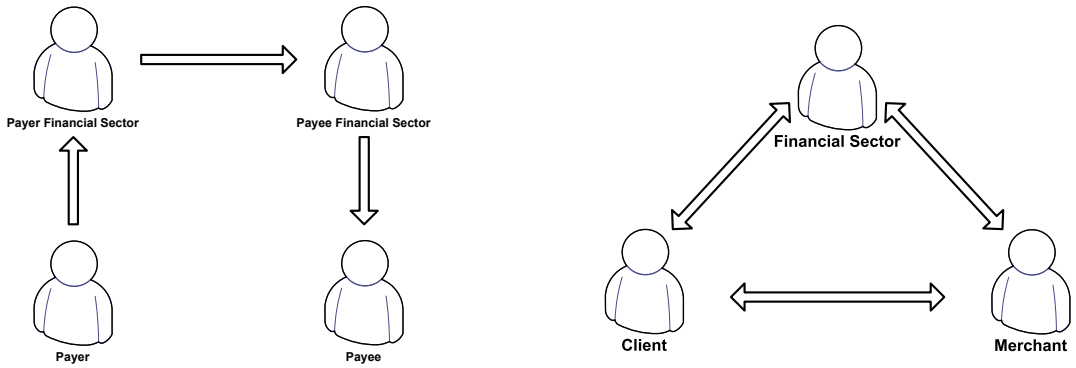
X-Party Mobile Payment

The intent of a payment is to transfer monetary value from the payer to the payee (Peiro et al., 1998). The immediate participants in a payment are located in the direct payment layer in the Four-layer structure of mobile payment participants (see Figure 3.1). The type of payment varies based on how many participants are involved in a specific payment transaction. Defining a mobile payment model is generally based on the combination of the participants in the direct payment layer.

In the literature there are several models for mobile payment transactions. This research is not aim to make a review on every mobile payment model, so only three of which are

depicted in the thesis.

Four-party mobile payment model Peiro et al. (1998) propose a payment model which is composed of four directly involved parties: the payer, the payee, the issuer (the payer's financial sector), and the acquirer (the payee's financial sector). Figure 3.2 (A) illustrates this four-party mobile payment model. The financial sectors may be banks, and the goal of the payment is to transfer money from the payer to the payee.



(A) (B)

Figure 3.2: (A) Four-party mobile payment model (Peiro et al., 1998). (B) Three-party mobile payment model (Ham et al., 2002).

Three-party mobile payment model A mobile payment model involving three parties is specified by Ham et al. (2002). Figure 3.2 (B) illustrates this three-party mobile payment model. The client, the merchant, and the financial sector communicate with each other, exchanging information to conduct a payment transaction from the client to the merchant.

Two-party mobile payment model This thesis discusses a simple but special payment model that involves only the client and the financial sector in the direct payment layer. The model is referred to in this thesis as the two-party mobile payment model. Mobile banking can be viewed as typical two-party mobile payments. Illustrated in Figure 3.1,

the financial sector and the client are bonded in a broken line boundary. It shows, in a special context (such as mobile banking), only the financial sector and the client carrying out payment transactions with each other. Therefore, three or four participants in the direct payment layer are combined by several two-party models. Figure 3.3 illustrates the two-party mobile payment model.

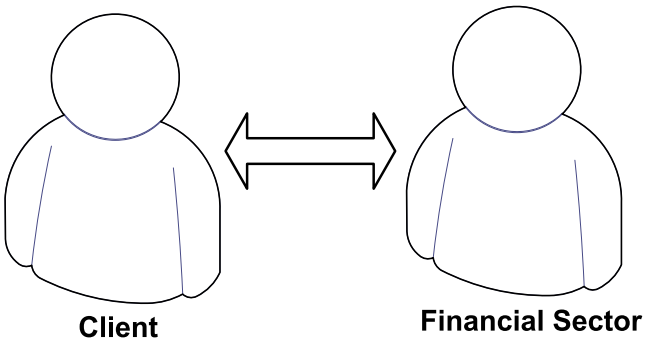


Figure 3.3: Two-party mobile payment model.

The detailed transaction process is varied among different X-party mobile payments; however, the similar basic characteristic among different payment models is that the information (monetary value, order information, or real goods) exchange takes place between two parties. Thus, the discussion of the security architecture for two-party mobile payments will contribute to realizing the security for other mobile payment models. Meanwhile, the study of a two-party mobile payment can be extended to any two-party mobile transactions.

Although the present research outlined in this thesis is based on the direct payment layer in the Four-layer structure depicted in Figure 3.1, participants of the other layers need to be considered for a full-fledged payment system. From the perspective of the network layer, the mobile network operator is a participant in the process of mobile payments. The proposed security architecture is implemented for a mobile bank scenario in this research.

A mobile payment is one of the most characteristic mobile transactions focusing on service in the financial field. The transaction model in mobile payments can be extended and employed in other mobile transaction application areas, such as mobile stocks and mobile games. Although this research targets a mobile payment transaction, the proposed architecture is expected to serve in other mobile transaction applications.

3.1.3 Wireless Networks

In mobile payment transactions, mobile devices communicate with the payment service of the financial section via wireless networks. Presently there are two common channels that can be recognized as wireless networks, which include the wireless local area network and the mobile phone network (Varshney & Vetter, 2000). Background on this area is given in section 2.1.

The mobile phone network (MPN) is a radio network which consists of a number of cells. Each cell is served by one or more fixed transmitters (Parson & Schaeffler, 2001). A mobile telephone or any other mobile device that connects to a mobile phone network is recognized as a mobile station. The mobile phone network is designed to cover a wider area. At present, the HTTP(s) applications requiring Internet access can be operated on the mobile phone network.

Considering the typical mobile payment user, the mobile phone network is believed to have a much higher penetration compared to the WLAN (Parson & Schaeffler, 2001). Therefore, the current research targets the more popularly used mobile phone network.

3.1.4 *Application Layer*

The current mobile payment network environment is depicted in a three-layer network model (see in Figure 3.4). The bottom layer is a physical infrastructure layer, which supports mobile payment transactions with hardware facilities. The middle layer is a protocol layer, which covers current network protocols such as HTTP or WAP. A business application layer occupies the top layer, which contains the software application solutions.

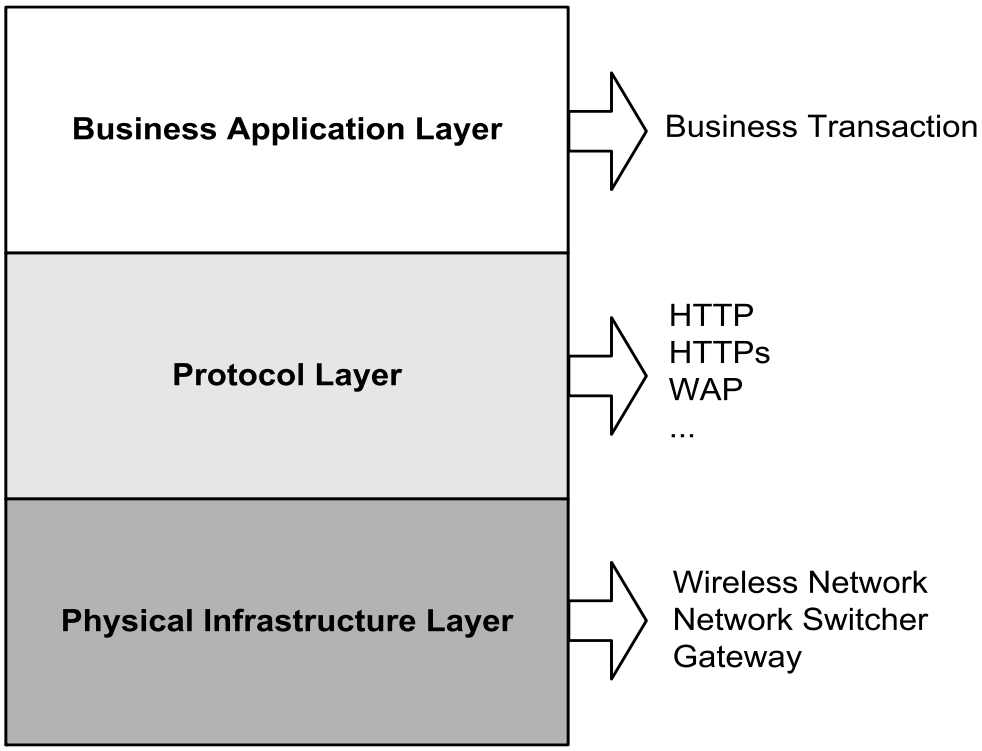


Figure 3.4: The Three-layer network model for mobile payments.

This research aims to protect security for two-party mobile payments on the application layer. Generally, to provide security for two-party transactions, the solution implementation is able to reside on the application layer (Itani & Kayssi, 2004), where the architecture in this thesis is proposed. The security architecture of the application layer is independent of the security protocols of the other lower layers, and the application handles all the security-

related functions.

Another benefit in proposing the security architecture on the application layer is that the proposed architecture enables mobile payment systems to make use of current network infrastructures (such as network gateways and hardware facilities) and protocols (such as HTTP and WAP) rather than modifying them.

3.2 Security Objectives

In the scope of this research as defined above, the security objectives which an advanced security system must fulfill are discussed in this section.

In order for payment transactions over mobile networks to gain public trust, a secure commercial information exchange is necessary. Four properties are essential for a secured transaction: confidentiality, authentication, integrity, and non-repudiation (Park & Song, 2001). Adopted from Merz (2002), Table 3.1 describes security objectives and some enabling technologies.

Table 3.1: The security properties (Merz, 2002).

security objectives	Definition	Technology
Confidentiality	Property ensuring that transaction information can not be viewed by unauthorized persons	Encryption
Authentication	Property ensuring that the transaction information actually originates from the presumed transaction partner	Possession Knowledge Property
Integrity	Property ensuring that the transaction information remains intact during transmission and can not be altered	Digital Signatures
Non-repudiation	Property ensuring that nobody is able to claim that the transaction on his/her behalf was made without their knowledge	Digital Signature

The information in the table is explained as follows:

- *Confidentiality*: Confidential information must be protected from viewing by an unauthorized party. A system providing confidentiality removes the risks from an eavesdropper or attacker. For example, if a short message is sent in plain text, anybody who can intercept the message may read it, whereas an encrypted short message ensures that the message cannot be read. Confidentiality ensures that the operation remain private (Krawetz, 2006).

- *Authentication*: Authentication ensures two parties gain the right to access a system to take part in a transaction. The purpose is to prevent anyone impersonating anyone else. For a network communication system it is important to ensure authentication as the two parties are not directly connected.

- *Integrity*: The information and systems must be guaranteed against corruption by outside parties. With integrity the transaction information persists intact and non-altered during transmission. A system with a high level of integrity should be difficult to tamper with or to alter.

- *Non-repudiation*: The user must not be able to deny the performed transaction and must provide proof in case that this situation occurs. Non-repudiation ensures that the originator cannot falsely refuse or deny a transaction. Non-repudiation is an indispensable security requirement for systems in electronic business and mobile commerce where disputes of transactions can occur.

The goal of this research is to enable an architecture that fulfills these four security requirements. The next section explains how to choose technologies and solutions to fulfill these requirements.

3.3 System Analysis based on Security Map

This section explains how the choice of technical solutions is made. Since a secured payment system needs to fulfill at least four security requirements, it is important to find a series of components furnishing the system with that security protection. Many researchers make use of an "Onion Layer" model to analyze a security system. One of the Onion Layer models is the Onion Ring framework proposed by Wei et al. (2006). Adopting the Onion Ring framework, this thesis proposes a security map to guide the research activity and system design. The Onion Layer Framework is described in Subsection 3.3.1; the proposed security map and how the security map works are described in Subsection 3.3.2.

3.3.1 Onion Layer Framework

The Onion Ring Framework, an Onion Ring m-commerce security framework, is based on VAX/OS architecture which is a popular operating system with a multi layered architecture (Wei et al., 2006). This five-layer framework is proposed for analyzing and improving mobile commerce security requirements and performance. It is employed as a perspective in building the security map in the research, and provides several starting points in order to reach proper solutions for security. The Onion Ring offers a good security performance by coordinating and pairing access authority to increasing levels of accountability (Wei et al., 2006). Figure 3.5 depicts an Onion Layer m-commerce security framework adopted from Wei et al. (2006). The five layers include the security of mobile devices, language, wireless communication access control, access management, and transactions.

According to Wei et al. (2006):

- The mobile device security layer is used to implement a security strategy for all mobile communication devices. A mobile device has limitations such as being suscepti-

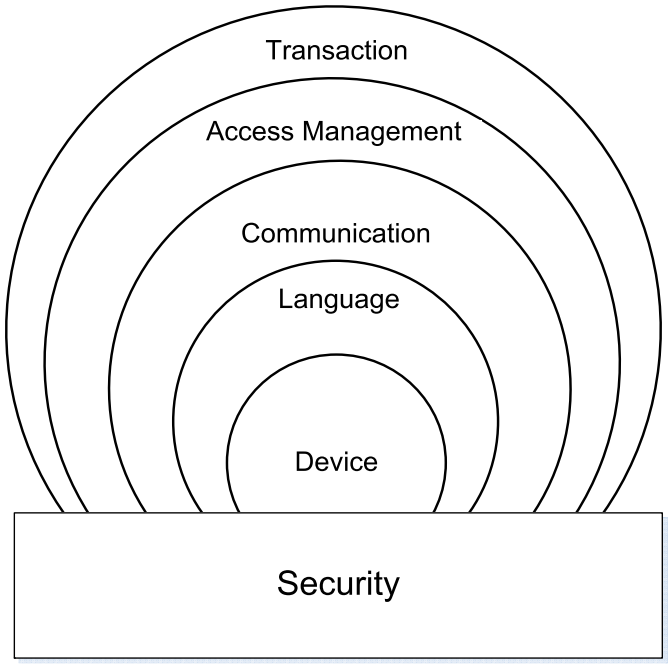


Figure 3.5: An Onion Layer Framework for m-commerce security (Wei et al., 2006).

ble to power outages, and having narrow bandwidth and low computational capacity. This research aims to find solutions that can be used to heighten security enforcement in this layer. As a mobile device is always utilized as a front-end facility involved in securing payment transactions, it is vital to design authentication strategies, secured channels, user-friendly payment schemes and receipt delivery in development of mobile devices (Thanh, 2000).

- The language security layer concerns the programming language used. A secure programming language in m-commerce requires and ensures that all programming codes have restricted access to operations that can affect the environment (Wei et al., 2006). Java is classified as a secure language as it is an object-oriented programming language which permits libraries to offer secure interfaces to incoming code. In addition, Java has a byte-code verifier which can be used in checking a program at load time.

- The wireless communication access control security layer is intended to restrain mobile devices from accessing wireless communication channels. Currently, some technologies have been employed to enhance security in this layer. Secure Sockets Layer (SSL) over Global System for Mobile (GSM) mentioned by Vihinen (2004) is a good example of security protection in this layer. The HTTP combined with SSL as Hypertext Transfer Protocol Secure (HTTPs) also provides a security strategy in this layer.

- The access management security layer can restrict resource access, audit mobile payment actions, and provide non-repudiation of transactions. A variety of technologies that can support security functions can be utilized in this layer. Multiple authentication strategy system is one candidate choice.

- Transaction security concerns application level transaction security. At the application level much progress can be made toward achieving transaction security by offering users authentication, logging the transaction information, and generating digital confirmation.

The Onion Layer Framework provides programmers or system architects a starting point to design a secured system or architecture. Some security technologies and solutions are also suggested in the Onion Layer framework.

3.3.2 Security Map

In this subsection, a security map is proposed on the basis of the Onion Layer framework in order to find proper technologies and solutions that are employed in designing the architecture in the present research.

Illustrated in Figure 3.6, the X-axis parameterizes the Onion Layer framework discussed in Subsection 3.3.1, the Y-axis represents the four security objectives explained in Section 3.2, and each black spot on the grid represents a security component.

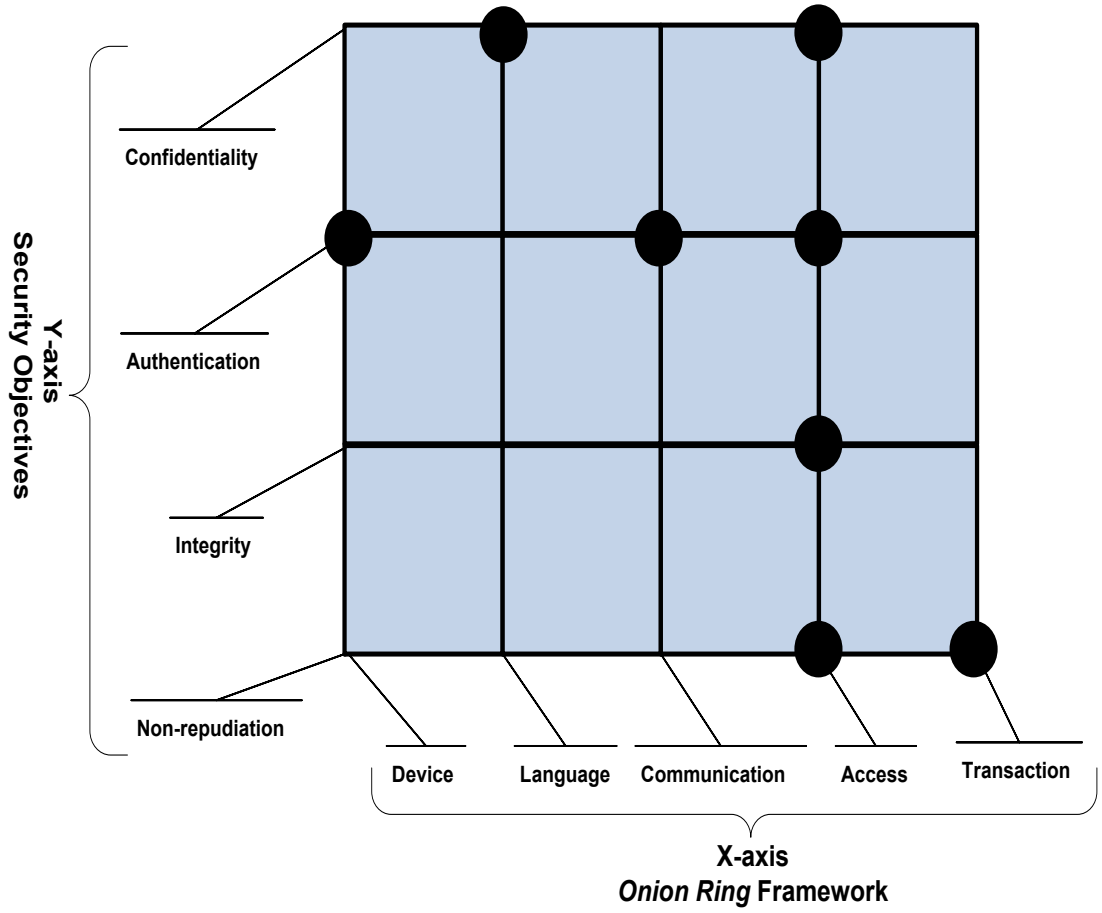


Figure 3.6: Security Map.

- *Onion Layer Framework*: Presently there are various security technologies. To choose these security technologies, the five layers in the Onion Layer is used as the starting-point. In Figure 3.6, the X-axis represents the five layers outlined in the Onion Layer framework.

- *Security Objectives*: The present research goal is to enable the proposed architecture

protect the four objective security properties described in section 3.2. The Y-axis in Figure 3.6 represents four objective security requirements. A secured architecture should be able to meet these security requirements. The Y-axis represents the four security properties.

- *Security component*: A potential secured technology employed in a secured system is denoted as a black spot. Each spot is placed at a cross point in the security map. The process of placing a spot at the cross point identifies a process looking for the suitable security technology for that requirement in a system. For example, researchers are looking for technologies to fulfill the security requirement on non-repudiation. The process on the security map is to place big back spots on the line signed by non-repudiation in the Y-axis. In the X-axis, there are five fields which are considered to be candidate solutions. In the access management security layer, as a public-key cryptography solution (digital signature) ensures non-repudiation, a spot is placed on the cross point of the non-repudiation line and the access line. In the same way, writing a log in the transaction security layer can be employed to meet the requirement of non-repudiation, and so a dot can be placed on the cross point of the non-repudiation line and the transaction line.

When the proper solutions for building a system are chosen, it is necessary to consider the actual running environment of the system. A payment transaction system runs in a computational environment, which includes several computational resources, such as mobile device CPU, memory, and the bandwidth of wireless networks. A proper design needs to consider these environmental factors. Since the proposed architecture is implemented for financial transactions over mobile devices, which are resource-limited facilities, designing a lightweight architecture that will run efficiently and feasibly is the goal of the present research.

As illustrated in Figure 3.6, the research activity can be viewed as a process of mapping secure solutions on the security map to meet the objective security requirements. Secure technologies are chosen from the collection mentioned in the Onion Layer framework (X-axis). The potential secured technologies can be utilized to meet one or more security objectives (Y-axis). The chosen technologies or solutions (security components) are restricted by the particular resource environment. The most appropriate technologies are combined to build the proposed architecture.

3.4 Analysis Result

This section describes the result of system analysis based on the security map.

Linck, Pousttchi, and Wiedemann (2006) details a number of concerns from clients regarding security in mobile payments. Table 3.2 is a summary of security objectives coming from these concerns, and technologies recommended for addressing them. The third column describes the specific solutions proposed in the architecture for addressing each of the concerns. The fourth column describes which layer the solution belongs to.

As depicted in Table 3.2, the proposed architecture employs a combination of particular security solutions to provide comprehensive security, which are explained from the perspectives of cryptography solutions, authentication solutions and non-repudiation solutions in the following.

3.4.1 Cryptography Solutions

The cryptography algorithm implementations need to be lightweight in function in the resource-limited environment.

Table 3.2: Security, Technology and Solution.

Security Objectives	Technology	Solution	*Onion Layer* Number
Authentication	Possession	mobile device	①
	Knowledge	PIN	③
	Property	userid/password	④
	Digital Signature	ECDSA	④
Integrity	Digital Signature		④
Non-repudiation	Digital Signature		④
	Log	Business Transaction Log	⑤
Confidentiality	Encryto/Decrypto	AES	④

[1] Column *Onion Layer* Number represents the layers in Onion Layer Framework: ① mobile device security, ② language security, ③ wireless communication access control security, ④ access management security, ⑤ transaction security;
[2] PIN: Personal Identification Number;
[3] ECDSA: Elliptic Curve Digital Signature Algorithm;
[4] AES: Advanced Encryption Standard

Digital Signature Solution

First of all, the implementation of a digital signature aims to ensure the property of integrity and non-repudiation. A digital signature also assists in providing authentication. The experiments mentioned in Stallings (2006), Jurisic and Menezes (1997) and Chang et al. (n.d.) demonstrate that there is a computational benefit in using Elliptic Curve Cryptography (ECC) with a shorter key length than RSA. (For background knowledge of ECC, refer to 2.7.3). According to Lopez and Dahab (2000b), Boneh and Daswani (1999) and Z. Li, Higgins, and Clement (2001), the performance of RSA digital signature and ECDSA with software implementation on PCs and mobile devices, indicated that ECDSA is more suitable than other public-key cryptography algorithms for the resource-limited environment. In addition, the result mentioned by Boneh and Daswani (1999); Lopez and Dahab (2000b) demonstrates that ECDSA (with 163 bits key) is faster than RSA (with a 1024 bit key) in overall timing. As *SA2pMP* aims to secure mobile payment transactions over

mobile devices, the employment of ECDSA is an advantage.

Table 3.3: Key size (bits): the comparison between (DSA or RSA) vs ECDSA (Boneh & Daswani, 1999; Lopez & Dahab, 2000b).

Security Bits	DSA or RSA	ECDSA
80	1024	160 - 223
112	2048	224 - 255
128	3072	256 - 383

Table 3.3 compares key sizes of DSA or RSA, and ECDSA. The first line in Table 3.3 is an example. To arrive at the security level which DSA or RSA implements with a key size of 1024 bits, ECDSA only needs a 160-to-233-bit long key for implementation. As ECDSA has the advantage in small key size offering an equal security level when compared to RSA and DSA, it has been chosen as the digital signature algorithm in the proposed architecture.

Data Encryption Solution

Data encryption is employed to maintain confidentiality. A symmetric-key cryptography algorithm is employed because it is much faster than a pubic key cryptography algorithm (refer to 2.7 for background information).

The primary advantage of public-key cryptography is its increased security and convenience: private keys never need to be transmitted or revealed to anyone. In a symmetric-key cryptography system, in contrast, the secret keys must be transmitted (either manually or through a communication channel). A limitation of using the existing public-key cryptography algorithms for encryption is the speed. However, there are many popular symmetric-key encryption algorithms (such as DES and AES), which are significantly faster than any existing public-key encryption algorithms. Choosing an encryption algorithm for resource-limited mobile devices is a trade-off process that depends on memory size, operation effi-

ciency, running speed, and security level. Since the digital signature is a choice made for this research to ensure the properties of authentication, integrity, and non-repudiation in a payment transaction, size and efficiency are given high priority. AES is then employed in *SA2pMP* to realize the encryption and decryption functions. The benefit of AES in speed allows adequate performance of the symmetric-key cryptography algorithm even in mobile devices. The particular solution chosen for *SA2pMP* is AES (Refer to 2.7.2).

3.4.2 Authentication Solutions

The proposed architecture provides a strong authentication strategy by employing multiple factors. Based on the analysis by Schneider (n.d.), human authentication factors can be categorized as follows (Layer number is referred to Table 3.2 and Figure 3.6).

- *Something you have*: Mobile devices, as physical objects, are possessed by users. The factor occupies the layer of ① mobile device security in the Onion Layer Framework.
- *Something you know*:
    1. A password for transactions is offered by banks. Bank transactions need the client account. Layer ④ access management security covers the security factor of username/password.
    2. A Personal Identification Number (PIN) is offered by mobile network operators or mobile device providers. For example, the Subscriber Identifier Module number of a given mobile device can be used as a PIN. PIN authentication is part of the layer ③ wireless communication access control security.

- *Something you are or do*: This makes use of behavioral and physiological characteristics of the principal (Schneider, n.d.). A digital signature can be categorized as a behavior. The employment of digital signature occupies the layer of ④ access management security.

As described above, the proposed security strategy entails a multi-factor process which provides strong authentication security. Subsection 4.1.4 describes details of designing multi-factor authentication strategy.

3.4.3 Non-repudiation Solutions

A complete payment system should provide appropriate dispute management services (Peiro et al., 1998). Mobile payment transactions need to avoid false repudiation, which can be achieved by recording a transaction log. A distributed transaction log strategy is employed in the proposed architecture. Further details in design is offered in subsection 4.1.5. In practice, the digital signature's implementation also contributes to ensuring non-repudiation.

3.4.4 Implementation Proposal

As described in section 3.5, Java is a secured programming language suggested by layer ② in the Onion Layer Framework. The proposed architecture, which employs a typical client/server (mobile device/mobile payment server) prototype, makes use of Java. The most widely supported mechanism for Java deployment on mobile devices is Java Platform, Micro Edition (Java ME) (Sun Microsystems, n.d.-a). The basic set of application programmer interfaces (APIs) is defined in CLDC 1.0 (Sun Microsystems, 2000a) and 1.1 (Sun Microsystems, 2003). MIDP is available in version 1.0 (Sun Microsystems, 2000b)

and version 2.0 (Java Community Process, 2002). CLDC and MIDP compose a Java runtime environment for Java-enabled mobile devices. The proposed architecture is implemented in Java ME (CLDC and MIDP) on a mobile client, with a server support, probably implemented in Java Platform, Enterprise Edition (Java EE).

Java ME has been selected for implementing the proposed architecture for the following additional characteristics:

1. The client software can run on mobile devices while the client is off-line. In other words, messages need to occupy network bandwidth only after they are processed. This method reduces the cost and the time required for network connection.

2. Although mobile devices are still recognized as having limited power, computational ability and resources as compared with desktop computers, they are developing rapidly in all these areas. It is reasonable to believe that mobile devices will have enough computational ability to carry out off-line transactions and handle more complex security algorithms.

3. Java ME is a standard technology, which has the advantage of running cross-platform. A popular group of smart mobile devices support Java ME, so an architecture based on Java will have broad market compatibility.

This chapter offers an analysis of a combination of technologies to be employed for building the security architecture. In the next chapter, details for the system design are discussed.

Chapter 4

System Design

This chapter proposes a new security architecture for Two-party Mobile Payment transactions (*SA2pMP*). There are three main security strategies involved in *SA2pMP* security architecture: a lightweight cryptography scheme, a multi-factor authentication strategy, and a distributed transaction log strategy. Corresponding to these strategies, a key management solution is proposed to protect two pairs of keys for cryptography algorithms. As the mobile banking transaction typically involves two parties, *SA2pMP* is depicted in the context of a mobile bank. Section 4.1 introduces the security architecture of *SA2pMP* and Section 4.2 introduces the application architecture for a mobile bank. As the proposed architecture is designed for a specific scenario of mobile banking, the term *bank* is used to represent the financial sector for convenience.

4.1 Security Architecture

This section describes the security architecture of *SA2pMP*. In the following, the notations for this research is defined first, and then the network module of *SA2pMP* is described. The design of the lightweight cryptography scheme, the multi-factor authentication strategy, and the distributed transaction log strategy are introduced, followed by a description of the key management solution.

4.1.1 Security Architecture Notations

In this subsection, general notations employed in the rest of this thesis are defined.

- *Bank*: Denotes a server computer held by the financial service participant in the transaction.
- *Client*: Denotes the person using a mobile device on the client side of the transaction.
- *ClientDevice*: Denotes the mobile device used by the client to carry out their mobile banking transaction.
- ID_C: Denotes the identity information of the client. $ID_C = (SIM + PHID + ACCID)$.
- $added\ identifier$: Denotes the identity information of the client, formulated by ID_C. The $added\ identifier$ is a combination of the *SIM*, *PHID*, and *ACCID*.
- $TimeStamp$: Denotes a time stamp.
- *Gateway*: Denotes the wireless gateway offered by a mobile network operator.
- *Session*: Denotes the HTTP session set up between *Bank* and *ClientDevice*.
- X: Denotes any participant involved in a mobile payment transaction. The X may be *ClientDevice* or *Bank* in the context of the research.
- RK_S: Denotes X's private key which is used to generate the digital signature and to sign the data transferred in transaction.
- PK_S: Denotes X's public key which is used to verify the digital signature in the transaction.
- $[RK_S,PK_S]$: Denotes a digital signature key pair including a private key and a public key.
- *DS*: Denotes the digital signature.
- *DSign*: Denotes the signing process for the digital signature.

- $DVerify$: Denotes the verification process for the digital signature.
- $Verify$: Denotes the process for verifying the transaction's authentication.
- PWD: Denotes the password of the mobile device client $Client$, which is known only to $Client$ and is verifiable by $Bank$. For example, the debit card has its password.
- $PHID$: Denotes PHone IDentifier, such as the mobile phone serial number.
- $ACCID$: Denotes the $Client$'s bank account number, or ACCount IDentifier.
- PIN: Denotes the $Client$'s Personal Identifier Number in the mobile device.
- SIM: Denotes the Subscriber Identifier Module number of $ClientDevice$. Normally SIM is offered by the related mobile network operator. For authentication purposes this is considered to be a physical characteristic of $ClientDevice$.
- TD: Denotes the transaction data transferred in the transaction. TD is plain text.
- K_E: Denotes the secret key for the symmetric-key cryptography algorithm.
- $Encrypt$: Denotes the encryption process in the symmetric-key cryptography algorithm.
- $Decrypt$: Denotes a decryption process in the symmetric-key cryptography algorithm.
- Yes/No: Denotes the transaction's status: approved or rejected.
- msg: Denotes a message.
- $h(msg)$: Denotes a one-way hash function for the message msg, such as SHA-1.
- $TLog$: Denotes the transaction log recording the transaction history in detail.

4.1.2 *Network Module*

In this subsection, the details of the network module of *SA2pMP* are described in detail.

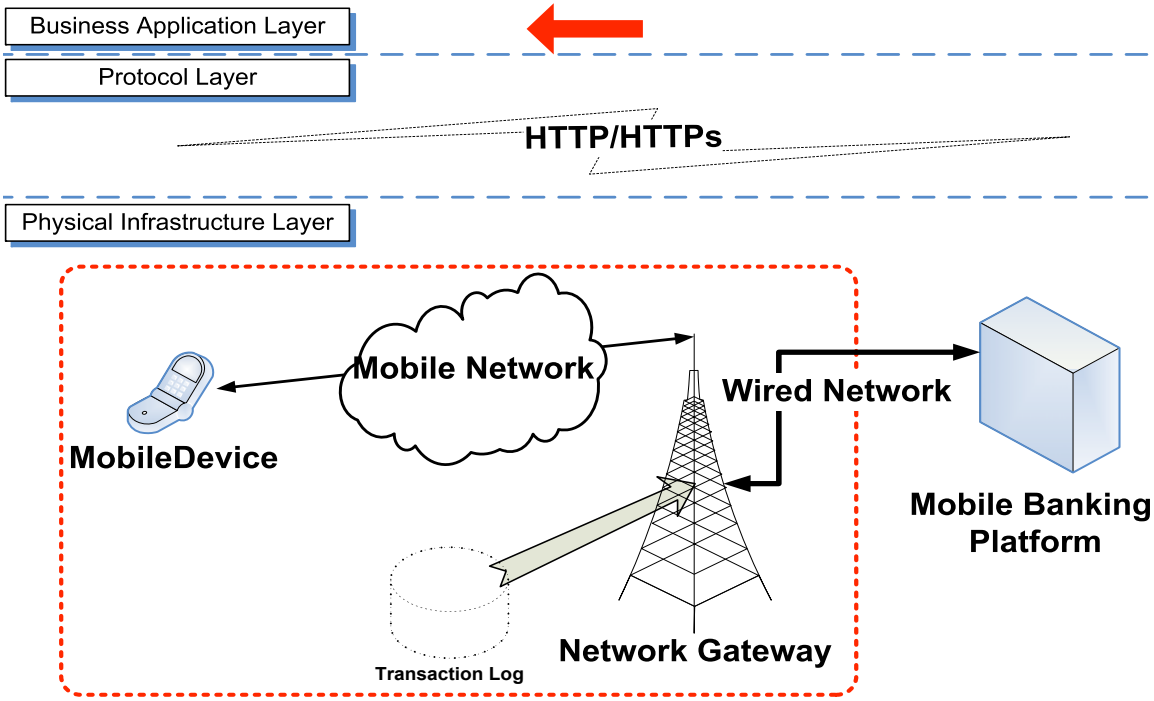


Figure 4.1: The network module of *SA2pMP*.

There are two parties involved in a normal banking transaction, the client and the bank. A client in a mobile banking transaction is also the mobile device holder. The mobile device communicates with the bank's mobile server via HTTP(s). As illustrated in Figure 4.1, a mobile device connects to a network gateway through a wireless network, which is served by a mobile network operator. For example, Rogers provides its users with data services, which enables the mobile devices to access HTTP applications through their wireless networks. Wired networks connect banking systems with wireless network gateways. Except for constraints in network bandwidth and mobile devices, the physical network infrastructure is transparent to the mobile banking platform. Since *SA2pMP* is designed for

the application layer, it focuses on the security of the business application layer in the three layer network model for mobile payment which is illustrated in Figure 3.4. In other words, *SA2pMP* does not make any modification to the other two layers, the protocol layer and the physical infrastructure layer .

Mobile payments are assumed to be based on wireless networks; however, communication is achieved both by wireless and wired networks. The wireless network gateway is a "bridge" linking wireless and wired networks. *SA2pMP* focuses on the area in the broken line boundary in Figure 4.1. The transaction log is physically located on the wireless network gateway, which is explained in subsection 4.1.5.

4.1.3 Lightweight Cryptography Scheme

SA2pMP employs a lightweight cryptography scheme. The symmetric-key cryptography algorithm (AES) contributes to the encryption, while the digital signature algorithm (ECDSA) is used to sign the digital signature for the transaction information. As illustrated in Figure 4.2, the plaintext message is signed before being encrypted, which can be referred to as "Sign-and-Encrypt". Signing and encryption are not independent of one another. When signing and encryption are combined, the signature layer should somehow depend on the encryption layer, so as to reveal any tampering with the encryption layer (Davis, 2002).

Conceptually, the independent operation makes it easy for users and programmers to layer cryptography operations and to avoid constraining application designs of the developers. However, such independent operations make it hard to fulfill the recipient's security expectations. For a suitable combination, the signature layer and the encryption layer actually need to refer to each other. The recipient needs proof that the signer and the en-

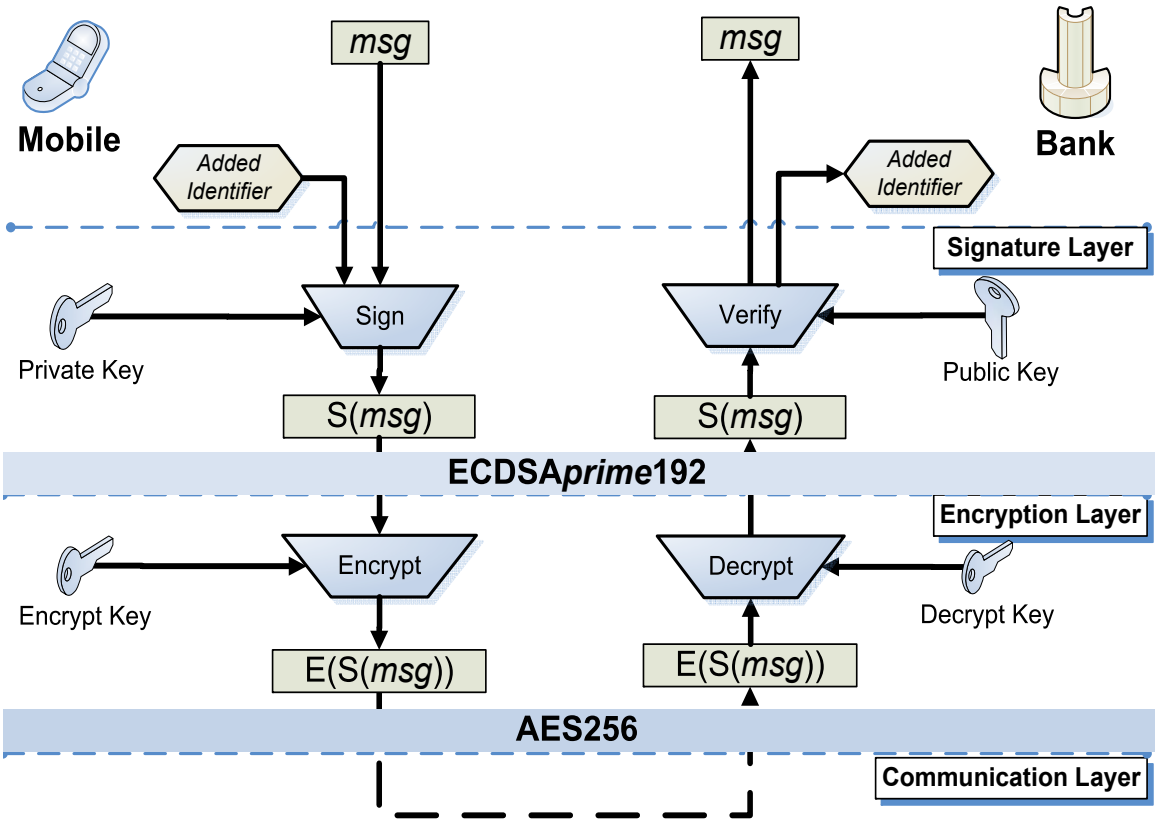


Figure 4.2: The lightweight cryptography scheme for *SA2pMP*.

crypter were the same sender (Davis, 2002). In *SA2pMP* the digital signature layer and the encryption layer are not independent, as these two layers are connected to each other by employing an *added identifier*. The other reason to cross between the signature layer and the encryption layer is to ensure the message is sent from the appointed client to the appointed server.

As illustrated in Figure 4.2, the communication layer refers to an open, public wireless network environment over which data are transferred. The public wireless network environment is a network environment which is not secure enough for payment transactions. During a transaction, the transaction message is transferred in this unsecured network. The public key employed in digital signature does not need to be encrypted; the public key can be transferred over an open wireless network. The transaction information, denoted as *msg*, needs to be kept away from third party eavesdropping, so both the signature layer and the encryption layer are employed to process *msg*. For these reasons, *SIM*, *PHID*, and *ACCID* are combined as the *added identifier*, which is then signed along with the message.

After the signing process, the message is encrypted by the encryption secret key K_E. The process is described by Equation 4.1.

$$ClientDevice : Encrypt(DSign(msg, ID_C), K_E) \tag{4.1}$$

In Figure 4.2, the added identifier is represented by ID_C. *SIM* ensures that the holder of the mobile device possesses the right to access wireless network resources; *SIM*, together with *PHID*, assures that the sender of the message is the appointed *ClientDevice*. *ACCID* identifies the unique character of the client for the bank and each client has a bank account. Meanwhile, *ACCID* is employed to identify the banking server, as banks have the information included in their account numbers. Equation 4.2 describes ID_C. It should be noted that the + operator in Equation 4.2 denotes simple concatenation of the strings.

$$ID_C = (SIM + PHID + ACCID) \tag{4.2}$$

Digital Signature Layer

SA2pMP employs a digital signature to protect authentication, integrity, and non-repudiation for a mobile banking system. A digital signature is a public-key cryptography algorithm, which makes use of a private key to encrypt messages and a public key to decrypt them. Normally an encryption process is referred to as a signing process for the digital signature algorithm, while a decryption process is referred to as a verification process. (Refer to 2.7.3 for background knowledge of digital signature.)

Compared to desktop computers, mobile devices are viewed as resource-limited in computing capability, CPU processor speed, memory and battery life. Factors such as the cryptography's key size and the computing efficiency need to be considered when choosing a proper digital signature algorithm to implement in the proposed architecture. The advantages of ECDSA in short key size and high-security enable it to be employed naturally in resource-limited environments. Refer to Subsection 3.4.1 for details.

ECDSAprime192 is employed in *SA2pMP*. The implementation of ECDSA uses elliptic curves over the prime field Z_p with the key size of 192-bit. Fernandes (1999) suggests that prime curves are best for software applications. The key size has been chosen for this research based on recommendations by Lenstra and Verheul (1999) and JSR177 (Sun Microsystems, n.d.-b). The SHA-1 algorithm is employed for generating the message digest. ECDSAprime192 has an equal security level to DSA with key size of 1024-bit.

Encryption Layer

SA2pMP utilizes a symmetric-key cryptography algorithm to ensure a secured channel and to protect confidentiality. To encrypt a plain-text message, the symmetric-key cryptography algorithms paralleled with the public-cryptography algorithms are chosen for the proposed architecture. *SA2pMP* employs AES (with the key size of 256 bits) as the encryption algorithm to protect confidentiality.

Compared to the public-key cryptography algorithms, AES has a smaller size and a better operational speed. In the group of symmetric-key cryptography algorithms, AES possesses obvious advantages (see 2.7.2), such as simplicity in design, variant cipher key size, better performance in speed, and reliability in both software and hardware (Dray, 2000). For ensuring the security of the secret key, AES is a suitable implementation for *SA2pMP*. (Subsection 3.4.1 explain the rationale for the choice.)

To protect the security of secret key, the strategy of Java package security or separated hardware is suggested. The secret key security is introduced in subsection 4.1.6.

4.1.4 Multi-factor Authentication Strategy

Authentication is concerned with assuring that the communicating entity is the one that it claims to be (Stallings, 2006). *SA2pMP* provides a strong authentication by a multi-factor authentication strategy which offers four factors for protection. The numbers ① ② ③ ④ refer to the security layers in the security map. For details of the security layers, refer to Table 3.2 and Figure 3.6.

- *Something you have*: Mobile devices, as physical objects, are possessed by users. Currently a mobile device can be viewed as an identifier for a particular individual, in that each individual is generally the owner of the mobile device which is not usually

shared with others (Roussos et al., 2003). This factor lies in Layer ① of mobile device security.

- *Something you know*: The PIN is offered by mobile network operators or mobile device providers. A PIN is used to identify a particular individual, such as the International Mobile Equipment Identity (IMEI), which is used to identify valid devices connected to mobile phone networks; a Subscriber Identity Module (SIM) on a removable SIM Card is used to identify a network subscriber in mobile devices. In particular, the IMEI identifies a mobile device, while the SIM identifies an available network. PIN authentication lies in Layer ③ of wireless communication access control security.

- *Something you know*: Banks require a password in order for clients to participate in banking transactions. Bank transactions are accountable. Clients have their own username/password to access the banking service. Layer ④ of access management security covers the security factor of username and password.

- *Something you are or do*: Digital signature. Implementing a digital signature contributes to authentication protection. The strategy of digital signature lies in Layer ④ of access management security.

As described above, these four factors are independent of each other. A mobile device acts as a physical characteristic of the client. The username/password is the property in the banking business field, and the mobile phone number is provided by the wireless network providers. A digital signature acts as an identifying characteristics of the client. All these factors in different fields contribute to a strong authentication strategy. Meanwhile, the solution of username and password is employed to determine the client identity. The relationship between client and business operations provides the functions to ensure access

control.

4.1.5 *Distributed Transaction Log Strategy*

Since a mobile device is considered as an access facility to Internet, the structure on the Internet side is similar between mobile payments and Internet payments. The main addition is the mobile network operator. Therefore, all participating entities can be synthesized into a service chain, in which each entity contributes services (such as network or finance) to a mobile payment transaction. Figure 4.3 depicts the service chain for the mobile banking transaction.

As depicted in Figure 4.3, a business transaction between the client and the bank is actually served by a service chain including the mobile network operator and the bank. The role of the mobile network operator is essential for a mobile commerce transaction. Based on where it stands in the service chain, the role of mobile network operator can vary from a simple mobile network provider to an intermediary, portal or trusted third party (Tsalgatidou & Veijalainen, 2000). In this service chain the mobile network operator technically provides a communication channel. Each communication massage is transferred through a wireless network gateway.

Since each entity in the service chain contributes services to the business transaction, it correspondingly profits from the service it provides. The relationship between the mobile network operator and the bank can be referred to a relationship between "cooperator and monitor". The bank needs to cooperate with the mobile network operator in order to realize a mobile banking business. Since the mobile network operator obtains other revenue (such as network service fees) than the bank in the service chain, it can serve as a third auditor party (TAP) auditing every business transaction. In other words, the mobile network

operator can function as a monitor.

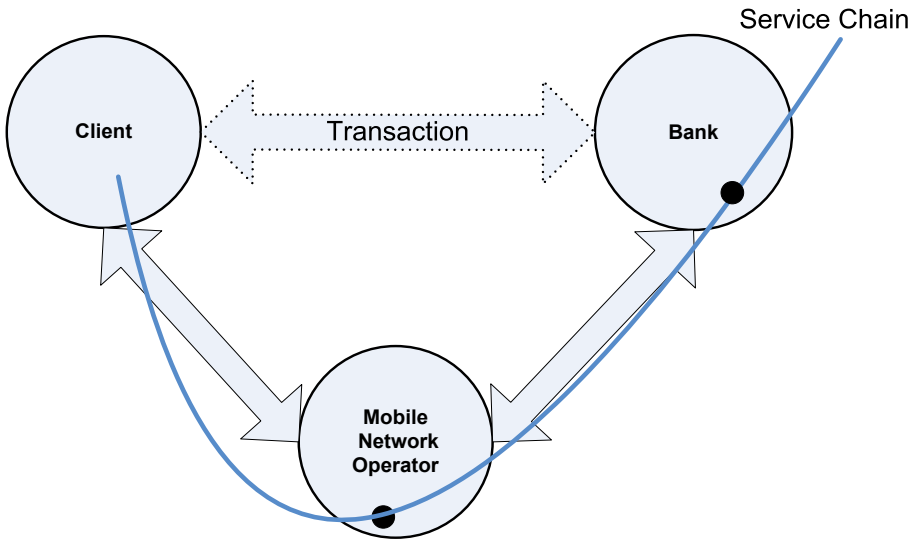


Figure 4.3: The service chain in a mobile banking transaction.

Based on the business model of "cooperator and monitor", this thesis suggests a distributed transaction log strategy to additionally contribute to ensuring non-repudiation for a mobile payment transaction (In the lightweight cryptography scheme of *SA2pMP*, the digital signature is employed partly to provide non-repudiation. Refer to subsection 4.1.3 for more details).

The distributed transaction log strategy is a defensive security mechanism to protect clients and banks from a false repudiation. If the client or bank refuses to admit participating in a mobile payment transaction, the transaction log can be used to judge whether, when and how the transaction happened. The transaction log is recorded by a business transaction log server. Because it is part of the business layer, the business transaction log server is a part of *SA2pMP* server architecture. In other words, the business transaction log server is a logical component of the bank server architecture. Physically the business transaction

log server is maintained by the mobile network operator. Since this physically-distributed-but-logically-integrated transaction log strategy utilizes the mobile network operator as a monitor, the transaction log must be trusted not only by the bank but also by the client.

The distributed transaction log strategy focuses on the way that the transaction log is integrated logically and distributed physically. It can be argued that logical integration benefits bank management, while physical distribution ensures that there is more trust in the monitor. As shown in Figure 4.1, the transaction log can be viewed as a part of mobile banking platform, while it is physically located on the network gateway.

4.1.6 Key Management

Key management concerns secure generation, distribution, and storage of keys (RSA Laboratories, 2000). Secure methods of key management are important to a secured mobile payment system. When the key is randomly generated, impersonation of the key must be prevented. In practice, most attacks on public key systems are aimed at key management, rather than at the cryptography algorithm (RSA Laboratories, 2000). This means that a secured mobile payment structure should pay particular attention to a trusted key management strategy.

In *SA2pMP*, the two key pairs required are used both for the digital signature and in encryption. Figure 4.4 illustrates the key management strategy for digital signature.

Key Generation

Digital Signature Key Generation *SA2pMP* employs a digital signature to ensure non-repudiation during the transaction. Because it uses public-key cryptography, a digital sig-

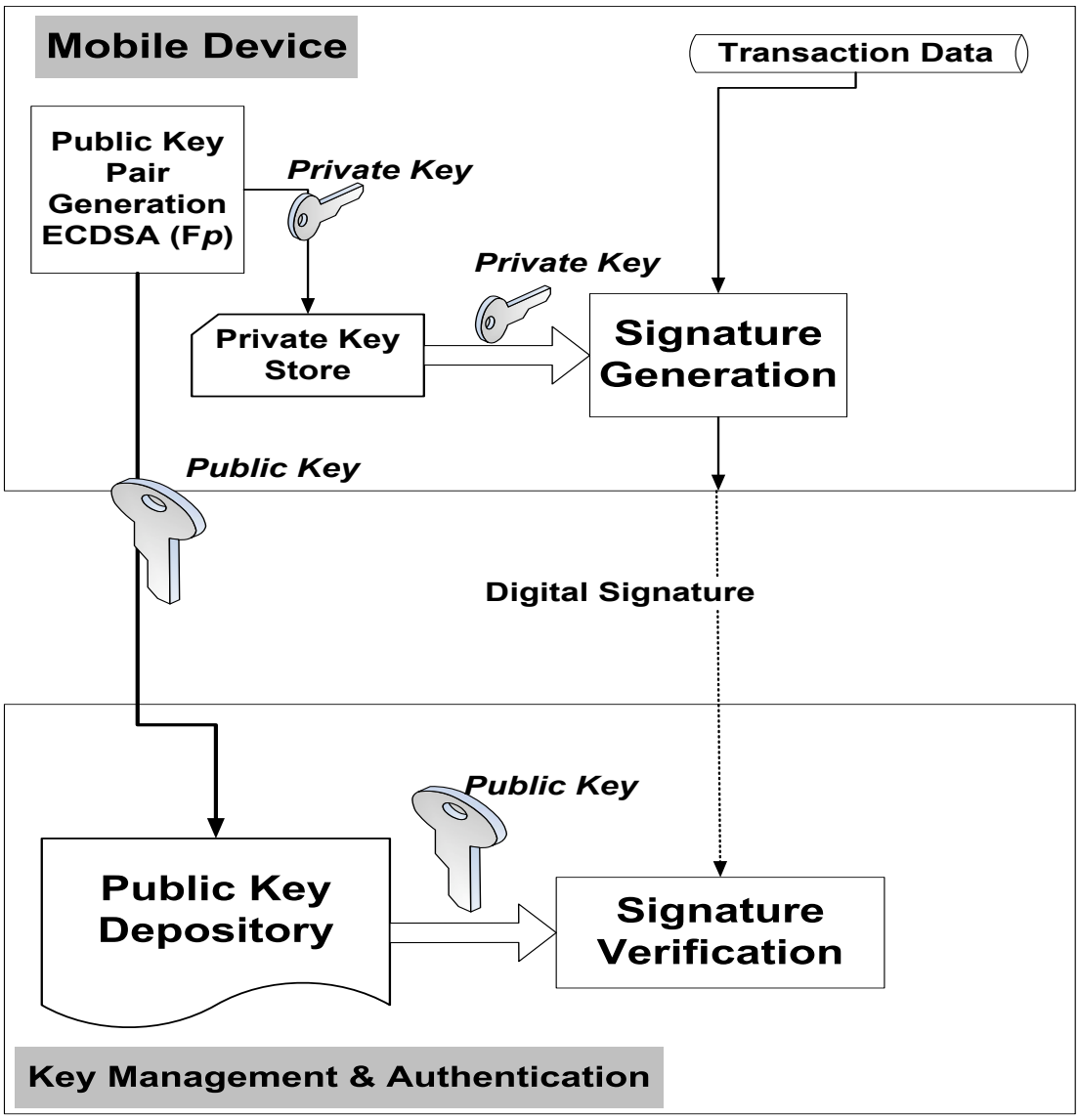


Figure 4.4: The key management strategy for the digital signature.

nature does not share anything secret between the two partners. The digital signature algorithm needs a key pair ([RK_S,PK_S]). The private key (RK_S) is used for signing and the public key (PK_S) is used for verification. To keep the private key secret, the key pair is generated by the mobile device on the client side of the transaction. To enable a transaction model that is more compact and in which only two parties are involved, *SA2pMP* does not make use of a trusted third party (TTP) for key generation.

When a client starts to use the mobile payment application, there is no key existing in the system. Before processing any transactions, the key generation is initialized. A Key Management Module (KMM) (illustrated in Figure 4.8 A) in the mobile payment application is in charge of initializing key management functions. At this moment, the KMM asks for the key generation, which is used to generate the key pair. This calls the key distribution and the key store function. After the first transaction, except for when the system asks for a renewed key pair, the KMM will not need to operate.

The KMM functions separately from the mobile payment transaction, which means the key management can run off-line. Without communication requirements, the process of key management does not occupy any network resources. By the time any transaction is operating, the key management's function should be completed. Therefore, the key management function will not compete for computational resources with the mobile payment transaction function.

Encryption Key Generation For the symmetric key algorithm, the operations for both encryption and decryption need two parties sharing the same secret key (K_E). The encryption key is generated by the banking server. After the generation of a key pair, the structure needs to distribute the private and the public keys separately to different key storages.

Key Distribution

Digital Signature Key Generation As ECDSA is a public-key cryptography algorithm which is implemented as the digital signature in the proposed architecture, it is acceptable that the public key is transferred over the wireless network (Menezes, Oorschot, & Vanstone, 1996). Once the key pair is generated, the public key is transferred to the Authentication Server, which is a part of the bank server. The transfer operation is via HTTP(s) through a wireless network. This distribution is initiated by a client application.

The private key is stored in the mobile device. As the private key is needed in generating a digital signature, impersonation and attack on this key must be prevented. The strategy should ensure that the private key is and only can be accessed by the application program itself. Due to this requirement, considering the prerequisite of mobile devices and Java ME, two strategies (File-Stored-in-JAR and Record-Stored-in-RMS) are proposed, which are introduced in the following.

Encryption Key Generation For encryption the secret key is not allowed to be transferred over the open wireless network, because it is easily intercepted by attackers. The secret key is stored in the program application JAR package, which is secured by the Java file security standard. After a client registers a payment service with their bank they will download the application package along with the encryption secret key from the mobile banking platform server.

Key Storage

Digital Signature Key Storage After the public key and the private key are distributed to both the server and the client, they are stored separately and securely. The public key

is transferred to the authentication server in the mobile banking platform server, and then it is stored in the Public Key Depository. The public key depository is a structured data storage file. For example, a database can be used as a secured public key depository, which is suggested in the proposed structure. The public key is stored, along with the client information (such as the account number and the mobile device tracking number) and key expiration information. This information is used to manage the key, assisting the authentication server to verify digital signatures in the mobile payment transactions. The Public Key Depository is part of the authentication server in the bank server. The local server security infrastructure is not within the scope of the present research.

On the client side, the private key is stored in the mobile device. The principle of private key storage is to keep it from being revealed to an attacker. The File-Stored-in-JAR (Figure 4.5 A) is used to store the private key in the same JAR package with the application program. *SA2pMP* is designed for a Java ME enabled mobile device, and the client application is a Java ME program. The Java ME application is packaged inside a Java archive (JAR) which contains the application's class and the private key. The JAR package file is downloaded to mobile devices, along with the Java application descriptor file. The private key, as the output of the key generation function, is exported and stored in a dedicated folder in the JAR package. Usually a Java ME application downloaded to mobile devices only includes class files, as well as the source files obfuscated. It is difficult to decompile and reverse the application program. Putting the private key in JAR package ensures the privacy of this Key and prevents direct exposure to an attacker.

Record-Stored-in-RMS (Figure 4.5 B) is another way to store the private key in the mobile device. RMS is the Record Management System, which is a key subsystem of MIDP in the Java ME standard (Giguere, 2004). With RMS, an MIDP application can use on-device data persistence. RMS provides the structure for on-device persistent data storage. Furthermore, RMS cannot be accessed by other applications besides the one that created it. The

strategy of Record-Stored-in-RMS only permits the mobile payment client application to access the private key, and thus prevents illegal attack from any other application program. Therefore, the security of the private key is ensured.

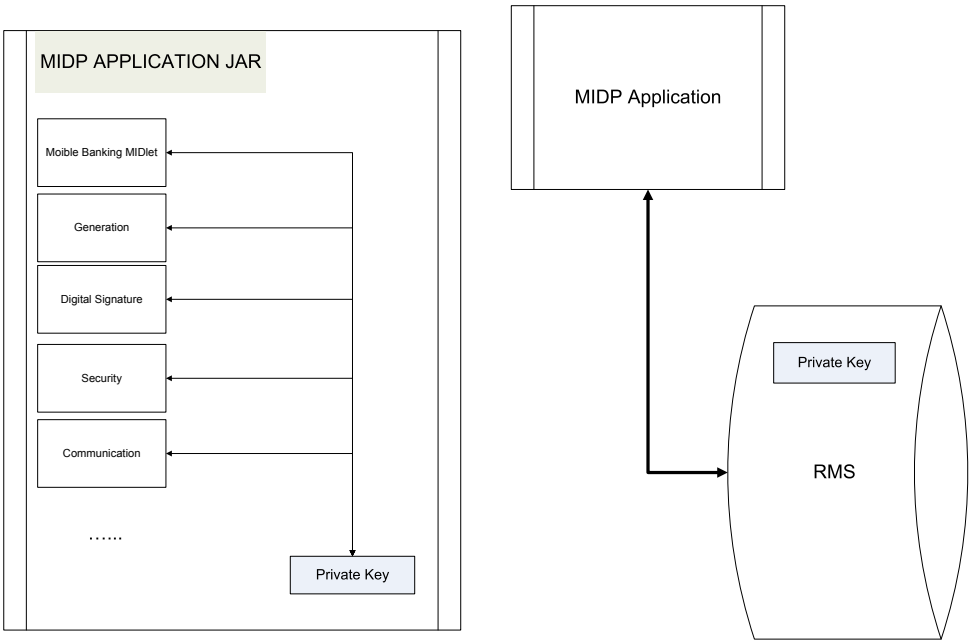


Figure 4.5: (A) The private key stored in JAR. (B) The private key stored in RMS.

Encryption Key Storage Although the encryption/decryption processes require that the mobile device and the banking server share the same secret key, the two copies are stored separately. On the banking server, the shared secret key is stored in the database, which is assumed to be secured by other computer security strategies such as in the database management system. In the mobile device, the secret key is stored in a JAR file or in RMS, which is secured by the Java file security standard. Additionally, the obfuscation renames classes, methods, and fields (Ortiz, 2009) to contribute to file security as well. Alternatively, a tamper-resistant hardware (RSA Laboratories, 2000) might be another choice for storing the Key. In fact, tamper-resistant hardware devices have been used extensively in financial

and government systems (Lam et al., 2003). (RSA Laboratories (2000) offers an review of tamper-resistant hardware devices.)

Key Renewal

Each key pair has an expiration time. The expiration time defines a period over which the key is valid. The expiration time limits the use of keys to fixed periods, after which the keys must be replaced. An expiration time must be chosen properly and distributed in an authenticated network channel. The key pair renewal process is initiated in the mobile banking platform server. The Public Key Depository records expiration information for the public key, which is used to determine whether the key pair needs to be replaced. Once Key Pair Renewal is decided, an alert (such as a short message) is sent to the mobile device to signal the client to generate a new key pair, in order to continue ensuring secured mobile payment transactions. If there is another request, such as the client requesting a renewal of the key pair, or there is a system update requirement, these situations have higher priority than expiration time in the key pair renewal process. The key renewal process can then be initiated, before the expiration time.

The security architecture has been described. To simulate the performance of security, the security architecture is implemented as part of a mobile banking application system.

4.2 Application Architecture

In this section, *SA2pMP* is discussed in the context of a mobile bank, and the application architecture of mobile banking platform is described. Two parties in a mobile banking

transaction communicates with each other via HTTP(s). The mobile banking platform (MBP) server on the bank side is a principal entity providing the web service and information exchange channel for the client. MBP is a subsystem of the complete banking infrastructure, communicating with the core bank. Figure 4.6 illustrates the general mobile banking process.

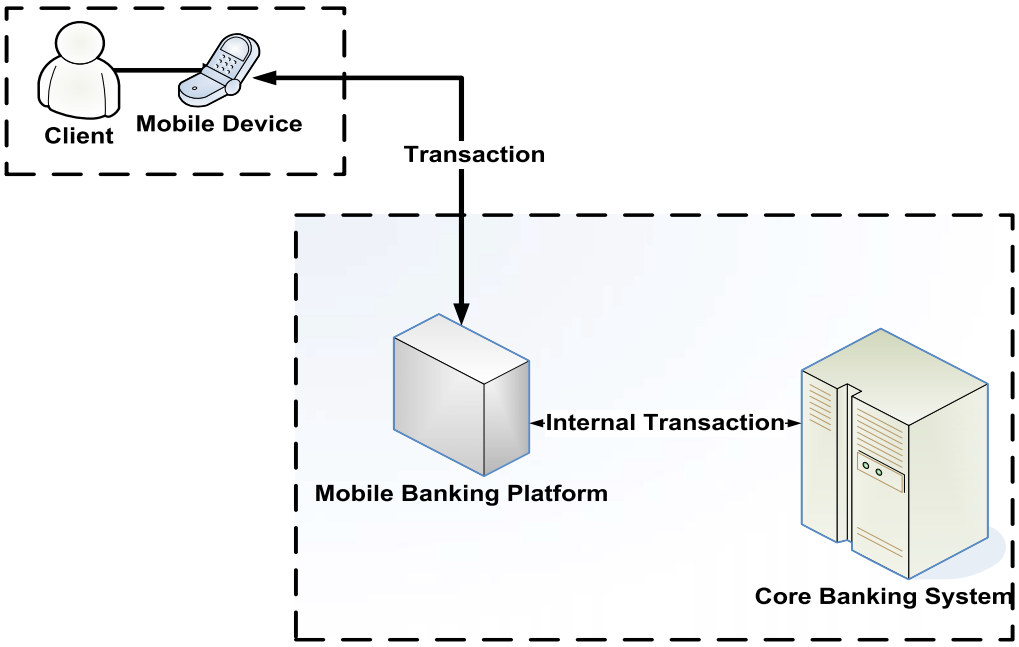


Figure 4.6: The mobile banking process.

Both MBP and the core banking system belong to the overall banking infrastructure. They are linked to each other via an internal network. Therefore, the transaction between MBP and the core banking system is internal. As in Figure 4.7, the mobile banking system is divided into a mobile client and a banking server.

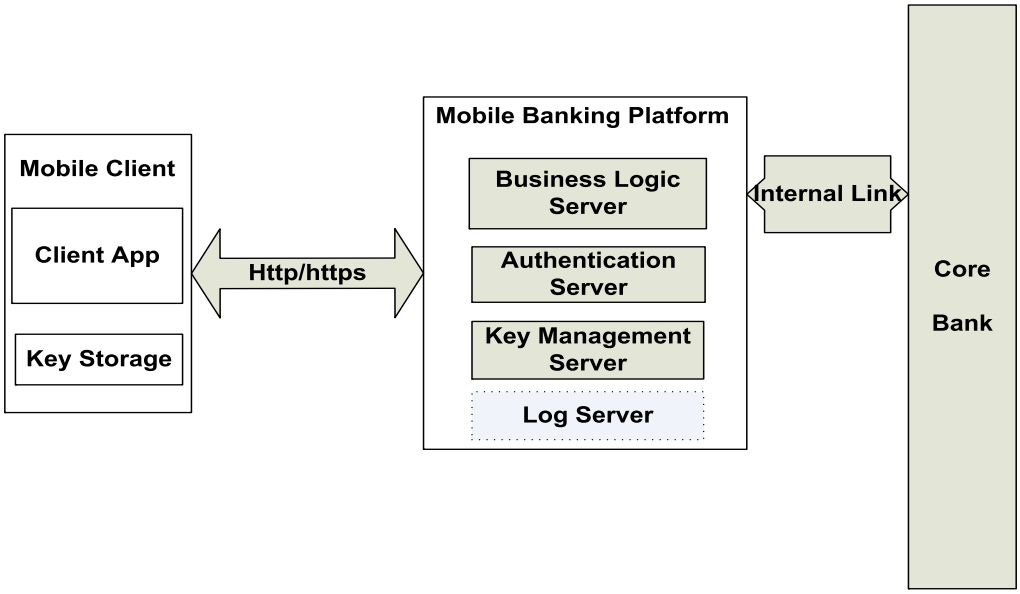


Figure 4.7: The mobile banking module.

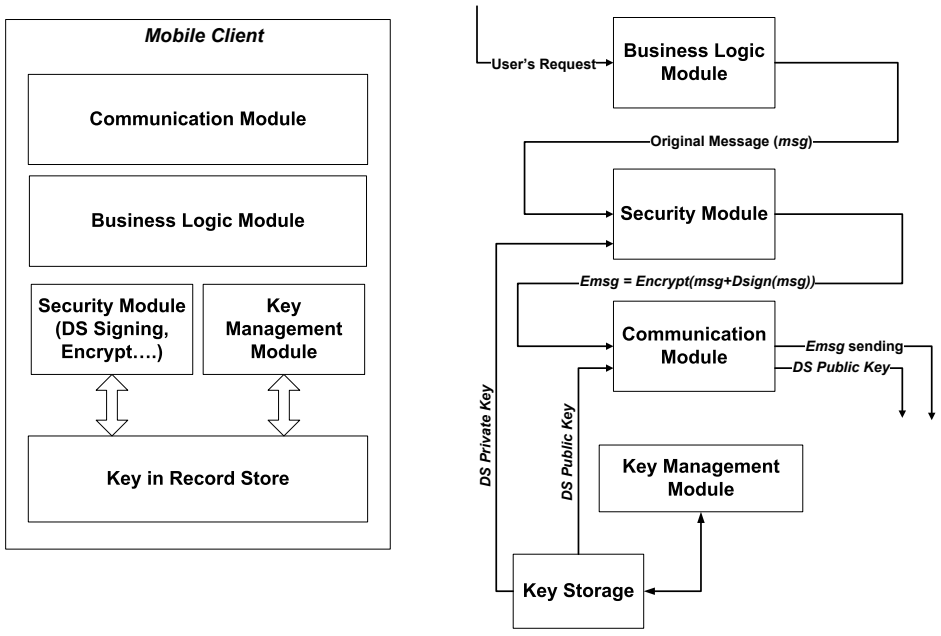
4.2.1 *Mobile Client Architecture*

The mobile client application is built on Java ME enabled mobile devices. Figure 4.8 (A) illustrates the architecture of the mobile client application. Figure 4.8 (B) illustrates the process by which the four modules cooperate with each other and work corresponding to clients' requests.

The mobile client system consists of four modules: the Business Logic Module (BLM), the Security Module (SM), the Communication Module (CM) and the Key Management Module (KMM).

- *Business Logic Module*: BLM is in charge of all business functions between banks and clients. Clients can make their own business item list according to their individual requirements.

- *Security Module*: SM is responsible for security issues. The main security architec-



(A) (B)

Figure 4.8: (A) The mobile client architecture. (B) The mobile client work process.

ture of *SA2pMP* is realized in SM. After the client's request is processed in BLM, an information message is generated. This message is processed with digital signature and encryption in SM. Following the rules of ECDSA, the private key can be read from KMM, and be used to sign a digital signature on the original message. After signing, the encryption is executed. The encrypted message is then sent by the CM to the mobile banking platform server.

- *Communication Module*: CM is the module in charge of network communication. HTTP(s) over wireless networks is used. CM handles information exchange between a mobile client application and a mobile banking platform server.

- *Key Management Module*: KMM is in charge of key management. Since a digital signature is employed in *SA2pMP*, the private key must be kept secured and confidential. KMM generates the digital signature key pair which is comprised of a private

key and a public key. It then distributes the public key to the mobile banking platform server. KMM stores the private key in the record store or in the JAR package. The private key is used for signing a digital signature. The encryption/decryption key pair remains the same for both in the mobile client application and the mobile banking platform server. KMM stores the encryption key in the record store or in the JAR package. Tamper-resistant hardware is also an option. KMM is also in charge of renewing the key pair when the mobile device receives a renewal request for the key pair.

4.2.2 Server Architecture

The Mobile Banking Platform Server consists of the Business Logic Server (BLS), the Authentication Server (AS), the Key Management Server (KMS), and the Business Transaction Log Server (BTLS). Figure 4.9 illustrates the architecture of the mobile banking platform server.

- *Key Management Server:* KMS in the mobile banking platform server is in charge of the storage of the public key, and initializing renewal process of a key pair. If the clients decide to use a mobile banking service, the first step is to invoke KMS module in the mobile client to generate the key pair in addition to installing the client application software. Distributing the key is a separate procedure from business operations. For the key pair of the digital signature, the public key is sent from the mobile client to the mobile banking platform server while the private key is stored in the mobile client. After receiving the public key, KMS stores it, as well as the client personal information. During the payment transaction, when key verification procedures ask for a public key, it is provided by KMS. KMS is responsible for generating

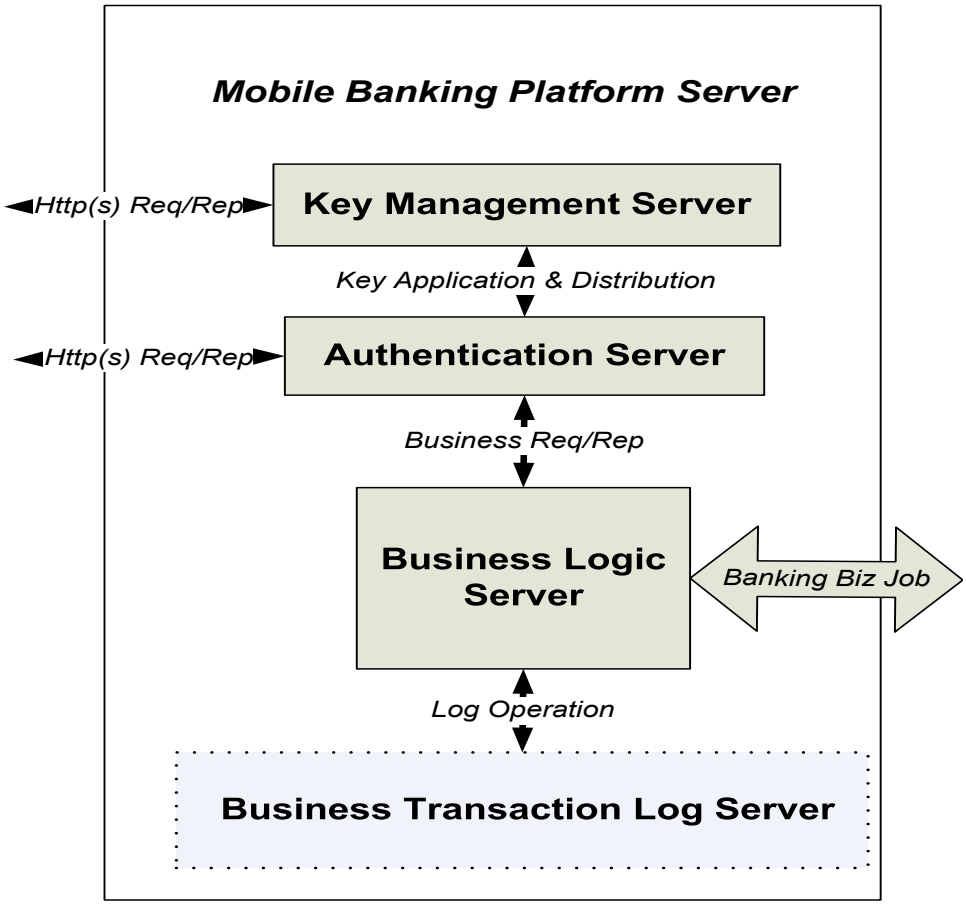


Figure 4.9: The mobile banking platform server's architecture.

the encryption and decryption public key, saving one copy in the server and sending the other one to the mobile device. The key pair has a life cycle. When the key pair needs renewing, KMS is responsible for sending a command to initiate a renewal procedure in the mobile client application.

- *Authentication Server:* AS provides the authentication service for the mobile banking platform server. The following security duties are operated in AS:

  1. Checking for username/password legality during log on,
  2. Verification of the digital signature, and
  3. Decryption of message(s).

  Any message passing through the AS is a legal business message, so it can be sent to BLS for processing.

- *Business Logic Server:* BLS handles all legal business requests. BLS interacts with the regular bank to process business logic and to exchange business information. When BLS has finished processing a business job, it responds by sending the resulting information to the client's mobile device.

- *Business Transaction Log Server:* BTLS is the component realizing the distributed transaction log strategy, which contributes to maintain non-repudiation. It creates and maintains the log files for transactions. BTLS is indicated by a broken line boundary in Figure 4.9, as BTLS is located in the mobile banking platform server logically rather than physically. The transaction log is physically positioned on the wireless network gateway. As shown in Figure 4.1, the network gateway is a physical facility maintained by the mobile network operator.

SA2pMP is an architecture proposed in this thesis for protecting the security for two-party mobile payment transactions. To evaluate *SA2pMP*'s performance in security and practicality, a series of simulations are conducted. In the next chapter, details for system simulation are introduced.

Chapter 5

System Simulation

In this chapter the system simulation is explained. A mobile bank is employed as a scenario for system simulation. The proposed architecture, *SA2pMP* is implemented to facilitate a "money transfer" banking transaction. The hardware and the software environments for system simulation are described in Section 5.1, followed by an introduction of the specific implementation in Section 5.2. Finally, the evaluation of the system simulation is described in section 5.3.

5.1 Simulation Environment

In this section, the computer environment for simulating *SA2pMP* is described.

As illustrated in Table 5.1, the present simulation is operated on an IBM IntelliStation M Pro PC, with Pentium 4 CPU 2.80 GHz and 2 GB RAM. The operating system is Windows XP Professional SP3.

Since the goal of the implementation is to enable *SA2pMP* to run on a typical Java ME enabled mobile device, three mobile device emulators were chosen for the simulation. All three support Java ME. These mobile device emulators are the Nokia S60 Emulator (Nokia, 2009), the Sony Ericsson Emulator (Sony Ericsson, n.d.), and the Sun WTK 2.5.2 CLDC simulator (Sun Microsystems, 2009). The simulators support the Device Configuration of CLDC-1.1 and the Device Profile of MIDP-2.0. For the simulated banking server, a Java Platform, Enterprise Edition (Java EE) application is implemented on an Apache HTTP Server 2.2, cooperating with Tomcat 5.0. A MySQL Server 5.1 serves as the database server.

Table 5.1: The simulation environment.

Item	Requirements
Hardware	IBM IntelliStation M Pro: Pentium 4 CPU 2.80GHz RAM 2.00GB
Operating System	Windows XP Professional SP3
Banking Server	Apache HTTP Server 2.2 Tomcat 5.0 MySQL Server 5.1
Mobile Client	Nokia S60 Emulator *and* Sony Ericsson Emulator *and* Sun Java Wireless Toolkit 2.5 for CLDC Simulator: Device Configuration as CLDC-1.1 Device Profile as MIDP-2.0
Development Tools	NetBeans 6.0 Eclipse 3.4.1 BouncyCastle 1.41

The simulation system was developed on NetBeans IDE 6.0[1] and Eclipse SDK 3.4.1[2]. NetBeans IDE 6.0 was used as it integrates well with the emulators. In addition, the visual MIDlet designer in NetBeans IDE 6.0 enables developers to design a business logical process easily. The implementation of cryptography algorithms (AES, SHA-1 and ECDSA) was done with help from the third party cryptography API provider Bouncy Castle[3]. Subsection 5.1.1 gives a brief introduction to Bouncy Castle.

5.1.1 Bouncy Castle

Sun Microsystems provides cryptography support through the Java Cryptography Architecture (JCA) and the Java Cryptography Extension (JCE). However, JCA and JCE are sometimes too heavyweight for an MIDP application (S. Li & Knudsen, 2005). The Bouncy Castle cryptography API provider is an appropriate choice for providing lightweight cryptography for Java ME.

Bouncy Castle is an open-source collection of lightweight cryptography APIs. It provides APIs for both the Java and the C# programming languages. One of the original design considerations for Bouncy Castle came from one of the developers who are active in Java ME development, and as a result there are two distinct library sets. At this time, the newest versions are Java version 1.43 and C# version 1.4. As the security architecture outlined in this thesis is designed for Java ME enabled mobile device, the discussion is focused on Java version of Bounce Castle API.

The Bouncy Castle Java ME supported package contains the implementation of the Bouncy Castle lightweight API, as well as two core Java classes which are not supported in

[1] http://www.netbeans.org/
[2] http://www.eclipse.org/
[3] http://www.bouncycastle.org

Java ME/CLDC: *java.math.BigInteger* and *java.security.SecureRandom* (Rittinghouse & Ransome, 2004). These two classes have the same names as the core classes in Java SE but function differently to the core Java classes in Java SE. A collision will occur when JDK is used to compile the code importing these two classes. To avoid this collision, one method is to employ some software such as Proguard[4] to make an obfuscation on the code. The compiler is able to find the target classes without confusing the different classes with the same name, then avoids the conflict that occurred.

In developing *SA2pMP*, the Bouncy Castle Cryptography with version of 1.42 was employed. The implementation of cryptography algorithms makes use of the Java ME lightweight API package in Bouncy Castle. Proguard is employed in the development as indicated above. Proguard 4.0 has already been bundled in NetBeans IDE, so this was an easy choice.

5.2 Simulation Implementation

A mobile bank is employed as a scenario for system simulation. The banking transaction of money transfer is assumed to describe how the system works. Readers should notice that in the following:

- "$->$" means a data transfer process.
- "$+$" means a data combination operation. In this implementation, most $+$ operations denotes simple concatenation of the strings.

[4] http://proguard.sourceforge.net/

5.2.1 Business Work Flow

The work flow in a money transfer transaction is explained in this subsection. Figure 5.1 illustrates the work flow. The notation can be referred to in Subsection 4.1.1.

As both encryption key and public key pairs are needed for the cryptography calculation, they must be generated and distributed before initiating a specific transaction.

1. *Bank* generates a secret key for AES algorithm encrypting or decrypting a message, then stores one copy locally and sends one copy to *ClientDevice*, where the encryption key is stored. The process distributing the encryption key must be kept secure from eavesdropping. (The distribution method of the encryption key is suggested in Subsubsection 4.1.6.) The encryption key's distribution process is explained in 4.1.6.

$$Bank : Generate(K_E), Store_{Server}(K_E) \tag{5.1}$$

$$Bank -> ClientDevice : K_E \tag{5.2}$$

$$ClientDevice : Store_{Client}(K_E) \tag{5.3}$$

2. *ClientDevice* generates a public key pair for the ECDSA algorithm. The private key is stored locally by *ClientDevice* for signing a digital signature, while the public key is distributed to and stored by the *Bank* for verifying the digital signature. The public key can be distributed in a public wireless network environment.

$$ClientDevice : Generate([RK_S, PK_S]), Store_{Client}(RK_S) \tag{5.4}$$

$$ClientDevice -> Bank : PK_S \tag{5.5}$$

$$Bank : Store_{Server}(PK_S) \tag{5.6}$$

Figure 5.1 illustrates the steps for carrying out a transfer transaction. These steps are

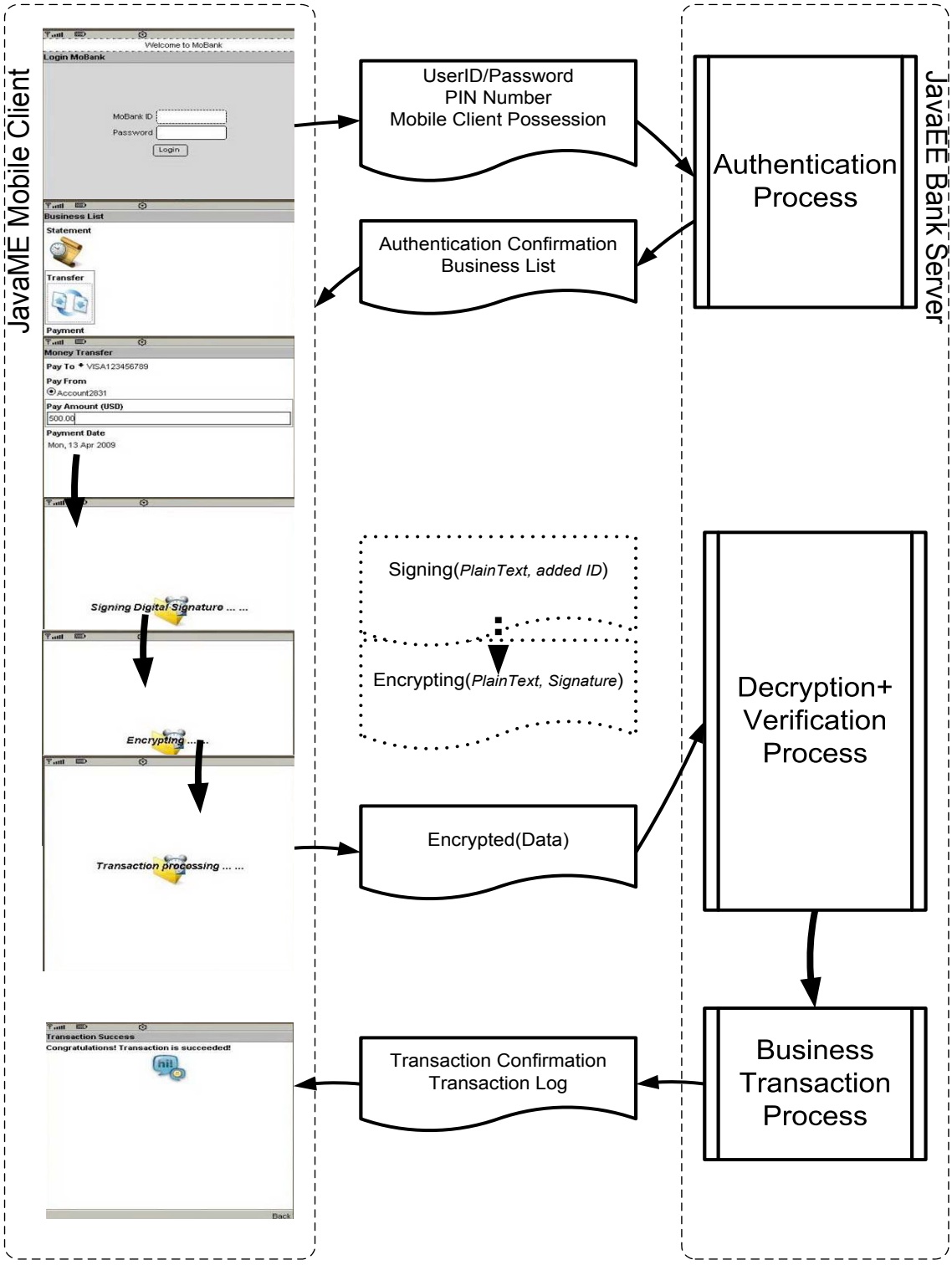


Figure 5.1: The secured work flow between the mobile client and the bank server.

described as follows:

1. To initiate a banking transaction, *Client* uses *ClientDevice* to send the authentication information to *Bank*. All information must be encrypted.[5]

$$ClientDevice : msg_{Auth} = Encrypt(ACCID, PWD, ID_C) \tag{5.7}$$

$$ClientDevice -> Bank : msg_{Auth} \tag{5.8}$$

2. *Bank* performs an authentication verification to decide whether it will allow *ClientDevice* to perform the transaction or not.

$$Bank : (ACCID, PWD, ID_C) = Decrypt(msg_{Auth}) \tag{5.9}$$

$$Bank : Verify(ACCID, PWD, ID_C) \tag{5.10}$$

3. *Bank* responds with an authentication confirmation to *ClientDevice*. The authorized transaction list (*Bizlist*) is downloaded by *ClientDevice* if the access is authorized. Meanwhile, an HTTP communication session is set up between *Bank* and *ClientDevice*. The ID_C along with a time stamp, $TimeStamp$, is inserted into *Session* as a session key. This session key is used to identify that a transaction is processed between *Bank* and *ClientDevice* at a specific time.

$$Bank -> ClientDevice : (AuthConfirm, Bizlist) \tag{5.11}$$

$$Session : (ID_C, TimeStamp) \tag{5.12}$$

4. *Client* provides the money transfer transaction information and input it into *ClientDevice*.

[5]Readers should notice that *ACCID* is also referred in ID_C (refer to Equation 4.2). To access to *Bank*, *ACCID* and *PWD* are necessary for authentication. ID_C is an added identifier to keep the information of *ACCID* and *PWD* is sent from the registerred *ClientDevice* through the pre-defined network environment.

ClientDevice generates a string-based message containing the transaction information.

$$ClientDevice : msg < -TD \tag{5.13}$$

5. The transaction message is hashed with SHA-1 to generate a message digest *h(msg)*, which is then signed with a digital signature using an ECDSA private key, along with ID_C.

$$ClientDevice : DS = DSign(ID_C, h(msg), RK_S) \tag{5.14}$$

6. The transaction message and the digital signature are encrypted with an AES secret key.

$$ClientDevice : crypto\_msg = Encrypt(DS + msg, K_E) \tag{5.15}$$

7. The encrypted message is sent from *ClientDevice* to *Bank*.

$$ClientDevice -> Bank : crypto\_msg \tag{5.16}$$

8. *Bank* decrypts the encrypted message sent from *ClientDevice*. As a symmetric-key cryptography algorithm is utilized, *Bank* decrypts the message with the same secret key.

$$Bank : DS + msg = Decrypt(crypto\_msg, K_E) \tag{5.17}$$

9. *Bank* processes the decrypted message, separating the digital signature and the transaction message. The public key for the digital signature algorithm is employed in verification. The transaction message is hashed in *Bank* to get the message digest,

which is used to compare with the digital signature verification's product.

$$Bank : Yes/No = DVerify(DS, h(msg), PK_S) \tag{5.18}$$

10. If the verification process in formula 5.18 responds with "TRUE", it means the message is valid, as it is sent from the identified *ClientDevice*. Afterwards, *Bank* processes the money transfer transaction as requested by *ClientDevice*.

$$Bank : BizTransaction \tag{5.19}$$

11. Once the bank transaction processing is finished, *Bank* sends back a confirmation of transaction completion. At the same time, the transaction log is recorded physically on the wireless *Gateway*:

$$Bank -> ClientDevice : BizConfirm \tag{5.20}$$

$$Gateway : TLog \tag{5.21}$$

After *ClientDevice* finishes all the transactions, *Client* must log out from *Bank* application. *ClientDevice* disconnects the communication with *Bank*, and the *Session* is released.

5.2.2 Data Transformation

Corresponding to the money transfer transaction work flow, several business parameters, as the examples shown in Table 5.2, are defined for the use of security and business logic. Figure 5.2 illustrates the variable transformation process crossing the signature layer and the encryption layer in the lightweight cryptography scheme (refer to subsection 4.1.3 for

details of the lightweight cryptography scheme for *SA2pMP*). In this subsection, the process of the data transformation is described along with the process that *ClientDevice* uses send to the transaction information to *Bank*.

Table 5.2 describes a business record in a money transfer transaction. In the money transfer transaction, the following information needs to be recorded.

- F_Acc_ID denotes the ID number of the bank account from which the money will be transferred.
- T_Acc_ID denotes the ID number of bank account to which the money is transferred.
- *Amount* denotes the amount of money transferred.
- *Time* denotes the time when money transfer transaction is initiated.

Table 5.2: Business Record in Transaction.

Business Record	
F_Acc_ID = 123456789	Digital Signature
T_Acc_ID = 987654321	
Amount = 500	
Time = 200906061220	

After *ClientDevice* submits a business record containing the transaction information, the variable of *RawText* is employed to connect all business record parts to a string. Each variable is linked with the symbol of "&".

$$RawText = F\_Acc\_ID + T\_Acc\_ID + Amount + Time \tag{5.22}$$

Next, *RawText* is added with ID_C, which is used to identify the given mobile device and a specific bank account.

$$PlainText = RawText + ID_C \tag{5.23}$$

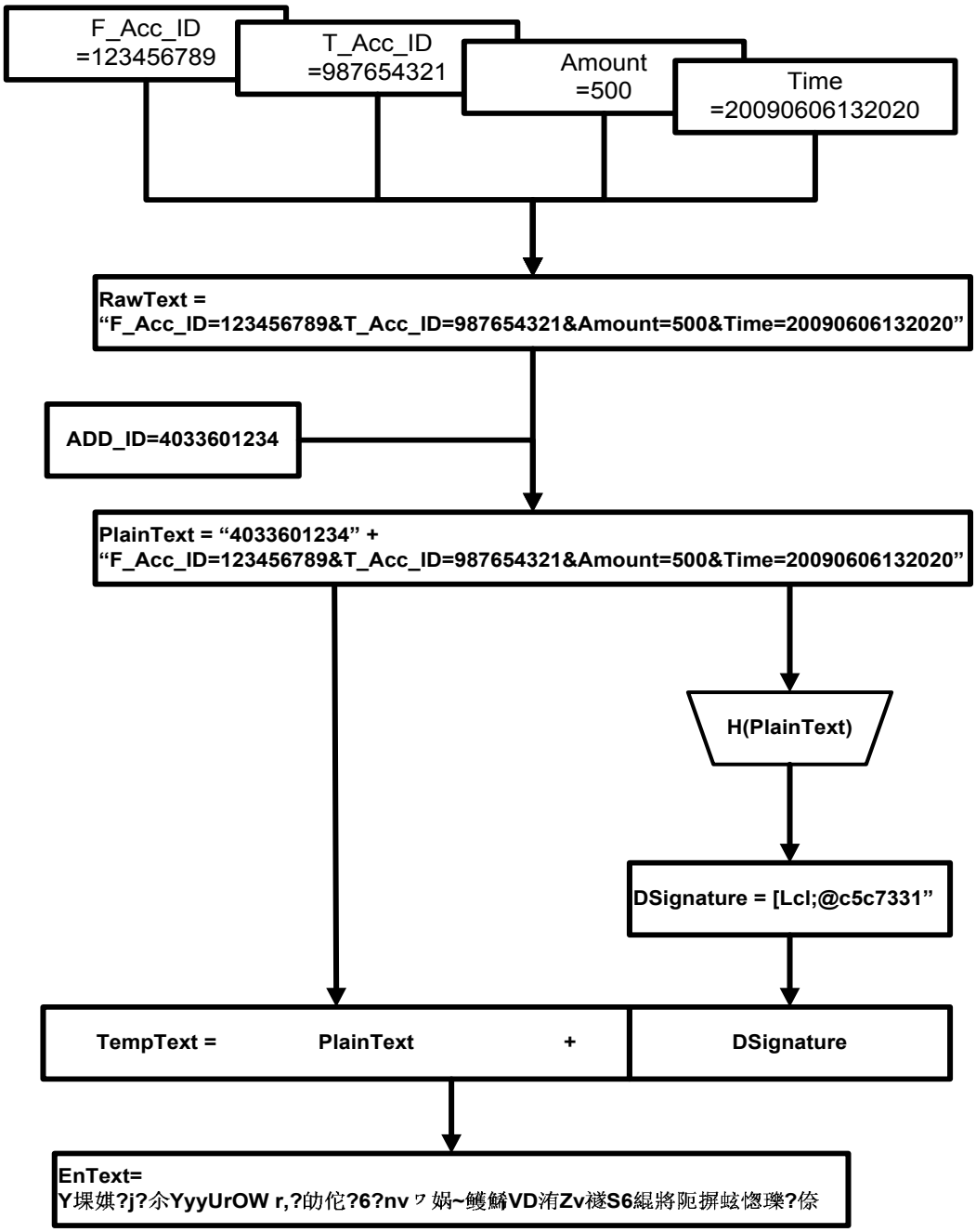


Figure 5.2: The variables’ transformation process.

The digital signature is produced by *PlainText* and the private key for digital signature. *PlainText* needs to be hashed to a message digest, which is used to create a digital signature with the private key. In the present simulation, SHA-1 is employed to hash *PlainText* to generate a message digest. Then, ECDSA on prime integer with the key size of 192 bits (ECDSAprime192), is used to calculate the digital signature.

$$TempText = PlainText + DSign(h(PlainText), RK_S) \tag{5.24}$$

The signed message *TempText* is next encrypted using a symmetric encryption operation. AES256 is employed as the symmetric-key cryptography algorithm in the implementation.

$$EnText = Encrypt(TempText, K_E) \tag{5.25}$$

The *EnText* is finally sent to *Bank* via HTTP(s) through the wireless network.

5.2.3 Cryptography Simulation

SA2pMP employs a lightweight cryptography scheme, in which a lightweight digital signature algorithm is combined with a symmetric key encryption algorithm in accordance with the rule of "Sign-and-Encrypt". Table 5.3 describes the specific parameters for cryptography algorithms implemented in *SA2pMP*. AES256 is the AES algorithm with the key size of 256 bits. It is used to provide encryption and decryption functions. SHA-1 is employed to generate a message digest for the plain text message. The message digest is used to generate a digital signature. Signing and verifying the digital signature makes use of ECDSA algorithm over prime curves with the key size of 192 bits, namely ECDSAprime192. For

background on the cryptography algorithms, refer Section 2.7.

Table 5.3: Cryptography Implementation.

Algorithm	Type	Key Length
AES256	Encryption	256 bits
SHA-1	Message Digest	
ECDSAprime192	Digital Signature	192 bits

The performance of the cryptography implementation is evaluated on three different emulator platforms: the Sun Java Wireless Toolkit 2.5.2 for CLDC, the Sony Ericsson SDK 2.5.0.3 for the Java ME Platform, and the Nokia S60 3rd Edition FP1 SDK for MIDP. Table 5.4 summarizes the information for the emulator devices, the configurations, and the profile.

Table 5.4: The mobile device emulators.

Emulator Platform	Device	Configuration	Profile
Sun WTK 2.5.2	QwertyDevice	CLDC-1.1	MIDP-2.0
Nokia S60 3rd	S60Emulator	CLDC-1.1	MIDP-2.0
Sony Ericsson SDK 2.5.0.3	Sony Ericsson Z800	CLDC-1.1	MIDP-2.0

As defined in MIDP-2.0 and CLDC-1.1, these mobile devices must possess a minimum base memory of 192 KB and a 16/32 bit processor with a speed of 8-32 MHz. In the real world, Sony Ericsson Z800 has 6 MB phone memory and 1 GB Memory Stick support (Sony Ericsson, n.d.). There are currently several models of mobile devices using the Nokia S60 3rd platform. Nokia N80/N91 are S60 series mobile devices.

The same MIDlet was run in three emulator platforms in order to measure the time delay caused by cryptography operation. In developing a money transfer transaction, *PlainText* with 312-bit length was used.

5.3 Simulation Evaluation

The practicality of *SA2pMP* was evaluated based on the processing time delay caused by the cryptography operations and the code size of the MIDlet suite in a mobile device. The evaluation of the time delay shows whether the architecture performs efficiently enough to receive wide acceptance, while the evaluation on the code size determines if the architecture can be implemented in limited storage in mobile devices.

5.3.1 Time Delay Evaluation

In this subsection, the results from the simulation in three emulator platforms are analyzed. The time delay criteria is discussed, the method employed in evaluation is described, and finally, the evaluation process and results are explained.

Time Delay Criteria

If the extra time delay caused by protecting security takes too long, frustration with the system results. What "too long" means depends on the participants' experience of a given situation and on the type of application.

Miller (1968) proposed a 2-second rule based on the theory of limitations in human short-term memory. According to the rule, short-term memory plays a critical role in human information processing. An individual becomes aware of the wait period after approximately 2 seconds. Thus, to maintain uninterrupted information processing, the 2-second rule is recommended (Nah, 2004). Specifically, there are several rules regarding the time limitations in computer and mobile facility response.

The Criteria of Nielsen (1995) Based on Miller's argument, Nielsen (1995) suggested a 10-second rule for computer response. This 10-second rule is widely used in industry when a performance or a user interface is required. Nielsen's criteria can be categorized as follows:

- 100 milliseconds is the time limit for people to feel that the system is reacting instantaneously;
- 1,000 milliseconds is the time limit for people to think that the system stayed uninterrupted. Normally, no special feedback is necessary during delays of longer than 100 but shorter than 1000 milliseconds;
- 10,000 milliseconds is the time limit for keeping the client's attention focused. If there are longer delays than 10,000 milliseconds, people want to perform other tasks while waiting for the computer to finish.

The Criteria of Roto and Oulasvirta (2005) According to Nielsen (1995), people usually wait 10 seconds for a response in a laboratory environment, after which they turn to other tasks. However, there was still an open question how soon the attention typically shifted in a mobile environment.

Roto and Oulasvirta (2005) conducted a user study with 27 participants to discover the point at which visual feedback stops reaching the user in a mobile context. The researchers examined the participants' attention during page loading to the phone and the environment in several different everyday mobility contexts, and made a comparison between these practical contexts to the laboratory context. Summarizing from the experiment, Roto and Oulasvirta (2005) argued that:

- People rarely move their focus of attention away from the dialogue within the first 2,000 milliseconds, so non-visual feedback is not required for a short time delay.

- Visual attention switched away from the mobile browser normally between 4,000 milliseconds and 8,000 milliseconds in the mobile context. Multi-modal feedback is recommended for delays of more than 4,000 milliseconds in mobile applications.

In the following, the practicality of the system and the reaction of users are evaluated based on the 10-second rule (Nielsen, 1995) and the study (Roto & Oulasvirta, 2005) on mobile Human Computer Interaction (HCI).

Time Delay Evaluation Method

Regarding with a mobile payment system implementing *SA2pMP* architecture, the total time delay T can be calculated as the following equation.

$$T = T_{CBiz} + T_{CSec} + T_{Com} + T_{SSec} + T_{SBiz} \tag{5.26}$$

T_{CBiz} denotes the time delay caused by the business computation on $ClientDevice$. T_{CSec} denotes the time delay caused by the security computation on $ClientDevice$. Most of this time delay is caused by cryptography computations, including signing a digital signature and performing the encryption on $ClientDevice$. T_{Com} denotes the time delay caused by HTTP(s) communication via wireless networks. T_{SSec} denotes the time delay caused by the secure verification process and the decryption process on *Bank*. T_{SBiz} denotes the time delay caused by the specific business processing computation on *Bank*. Since *Bank* is built on a resource-rich computer server platform, there is a high computational speed. Furthermore, the distributed transaction log is written simultaneously. Considering these issues, the operation on *Bank* is assumed to be instantaneous. As the time delays of T_{CBiz} and T_{Com} are not caused by the security modules, they can be excluded from extra time delay caused by providing security. For these reasons, the extra time delay due to security is

focused on the delay caused by protecting security on *ClientDevice*, namely T_{CSec}. Coupled with equation 5.24 and equation 5.25, T_{CSec} can be calculated as follows:

$$T_{CSec} = Time_{encrypt(DSign(msg)+msg)} \tag{5.27}$$

$$= T_{encrypt} + T_{DSign} + T_{+} \tag{5.28}$$

In equation 5.27, $T_{encrypt}$ is the time delay for the encryption process or the decryption process, while T_{DSign} represents the time delay caused by signing or verifying a digital signature, and T_{+} denotes a set of operations for formatting the string-based message. For example, when a mobile device begins a transaction, which is sending the message from a mobile device to a server, the operations related to security include formatting a new message string, signing a digital signature, and encrypting a plain text message. The entire time delay for security (T_{CSec}) is described by these three parameters.

Time Delay Evaluation

In *SA2pMP*, AES256 is employed for encryption and decryption, while ECDSAprime192 is used for signing a digital signature. In ten practical system simulations (refer to 5.2.3) on three different emulators, the time delays of both the AES256 encryption and decryption are less than 1 millisecond. Based on this experimental result, the encryption is considered to be instantaneous ($T_{encrypt} \approx 0$). The time delay of T_{+} is caused by a string formatting operation. In the simulation, this part of time delay was less than 1 ms. As $T_{+} \approx 0$, the formatting operation was considered to be instantaneous as well.

Therefore, the operation for digital signature consumes almost all time delay for security on the mobile client device. Thus, T_{DSign} is considered to be the significant factor of

the time delay for security. Another reason for using T_{DSign} to represent the entire security time delay is for the convenience in comparing the implementation of the digital signature to other work. The time delay comparison can be found in section 6.3.

Based on the evaluation method explained above, the simulation and evaluation results can be described in the following.

Figures 5.3 - 5.5 illustrate the time delay measured on the Nokia S60 emulator platform, the Sony Ericsson Z800 emulator platform, and the Sun WTK 2.5.2 QwertyDevice emulator platform.

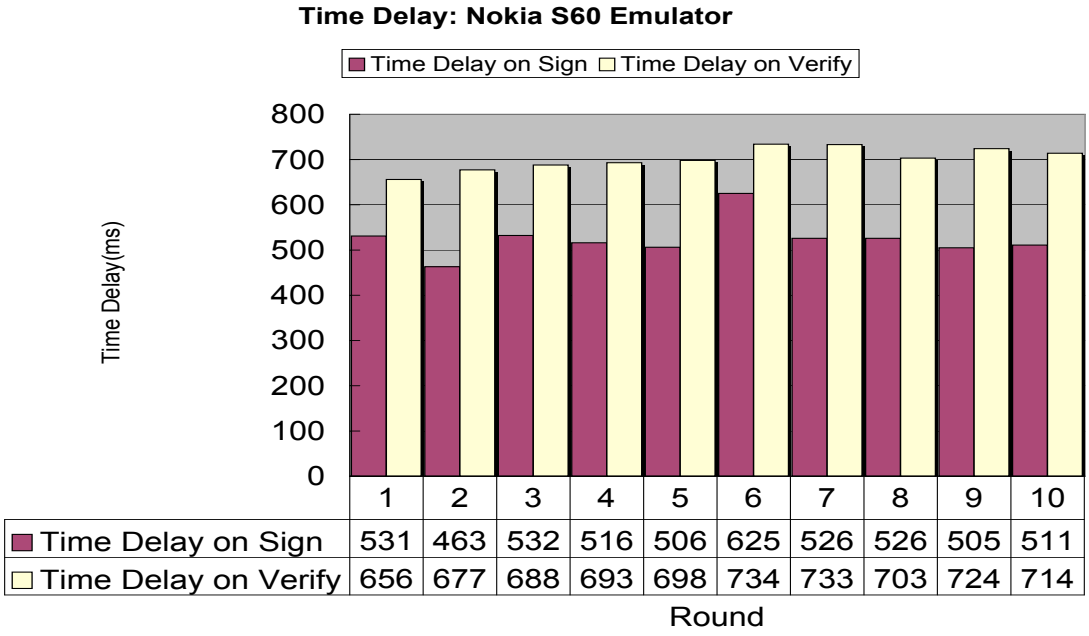
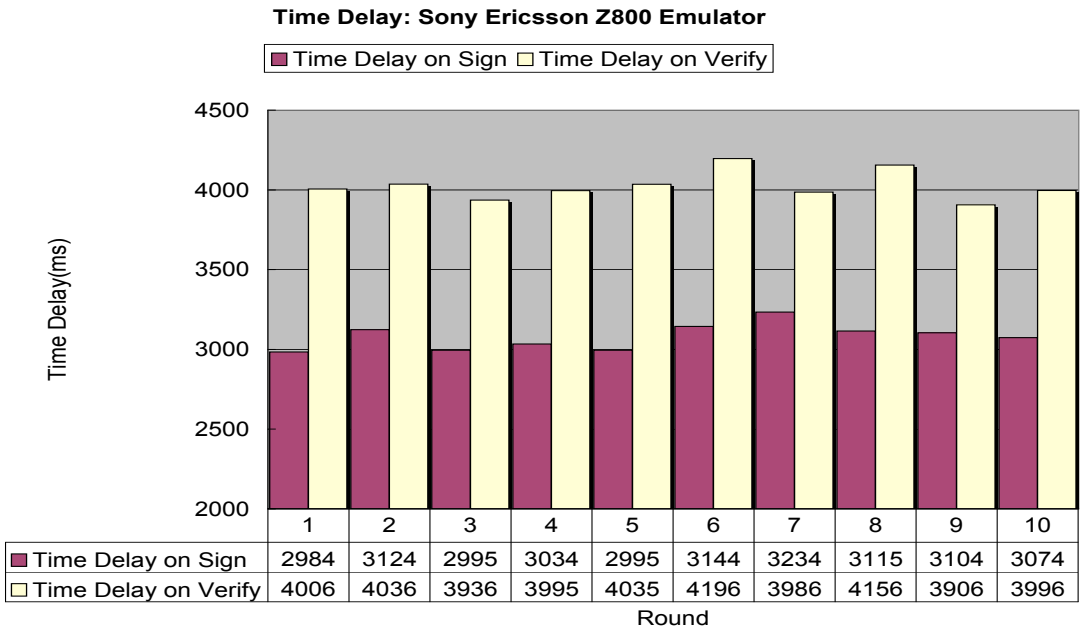


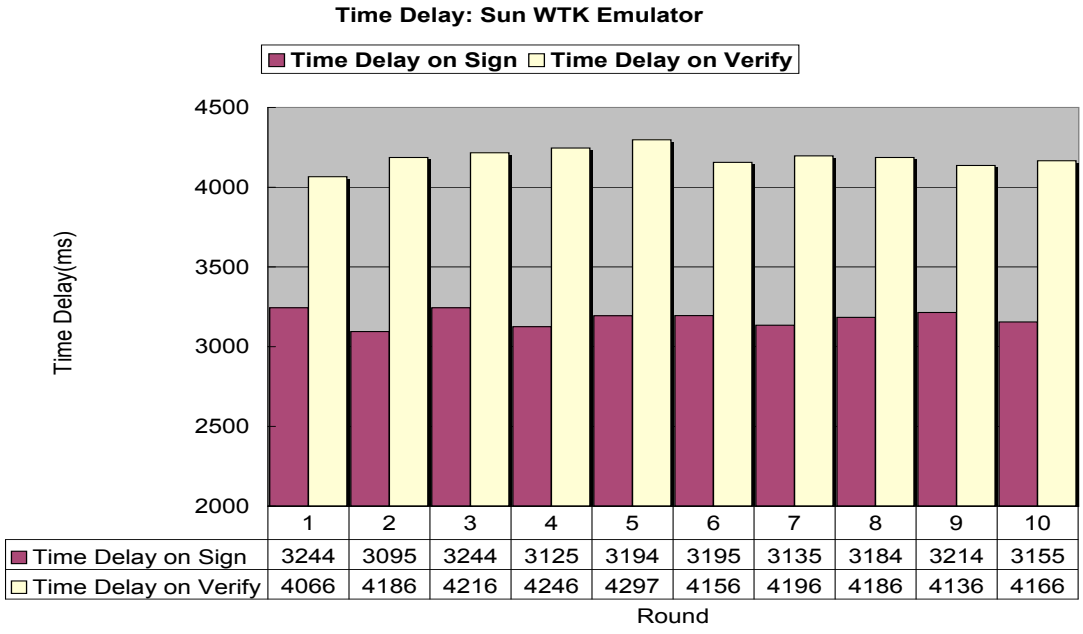
Figure 5.3: The time delay on Nokia S60 Emulator Platform (ms).

Table 5.5 shows the calculation of the average time delay on these three different emulator platforms. The simulation on Nokia S60 is the most positive. The average time delay of signing on the Nokia S60 is 524.1 ms, which is much better than the data on the Sony Ericsson Z800 (3080.3 ms) or the Sun WTK QwertyDevice (3178.5 ms). The same situation occurs with the verification process: the average time delay for verification on the



	1	2	3	4	5	6	7	8	9	10
Time Delay on Sign	2984	3124	2995	3034	2995	3144	3234	3115	3104	3074
Time Delay on Verify	4006	4036	3936	3995	4035	4196	3986	4156	3906	3996

Figure 5.4: The time delay on Sony Ericsson Z800 Emulator Platform (ms).



	1	2	3	4	5	6	7	8	9	10
Time Delay on Sign	3244	3095	3244	3125	3194	3195	3135	3184	3214	3155
Time Delay on Verify	4066	4186	4216	4246	4297	4156	4196	4186	4136	4166

Figure 5.5: The time delay on Sun WTK 2.5.2 QwertyDevice Emulator Platform (ms).

Nokia S60 is 702 ms, which is much better than on either the Sony Ericsson Z800 (4024.8 ms) or the Sun WTK QwertyDevice (4185.1 ms).

Table 5.5: The average time delay (ms).

	Nokia S60	Sony Ericsson Z800	Sun WTK QwertyDevice
Sign	524.1	3080.3	3178.5
Verify	702	4024.8	4185.1

The criteria of Nielsen (1995) and the criteria of Roto and Oulasvirta (2005) are used to evaluate the average time delay in the simulation. Figure 5.6 illustrates the evaluation results based on the criteria of Nielsen (1995). In Figure 5.6, the lowest area shows the time delay is less than 100 milliseconds, while the middle area indicates the time delay is between 100 milliseconds and 1,000 milliseconds, and the top area indicates the time delay is between 1,000 milliseconds and 10,000 milliseconds. The time delay on the Nokia S60 is between 100 milliseconds and 1,000 milliseconds, which means the system seems uninterrupted to clients and no special feedback is necessary based on the criteria of Nielsen (1995). The time delay on the Sony Ericsson Z800 or the WTK QwertyDevice is between 1,000 milliseconds and 10,000 milliseconds, which means the system still keeps clients' attention focused.

Figure 5.7 illustrates the evaluation result based on the criteria of Roto and Oulasvirta (2005). The lowest area indicates the time delay is less than 2,000 milliseconds, while the middle area indicates the time delay is between 2,000 milliseconds and 4,000 milliseconds, and the top area indicates the time delay is more than 4,000 milliseconds. Based on the criteria of Roto and Oulasvirta (2005), the time delay for the Nokia S60 is less than 2,000 milliseconds, which means the system maintain focus by client and non-visual feedback is unnecessary. The time delay caused by the signing operation on the Sony Ericsson Z800 or the WTK QwertyDevice is between 2,000 milliseconds and 4,000 milliseconds, which means the system needs some visual feedback to maintain client attention. The time delay

caused by verifying operation on the Sony Ericsson Z800 or the WTK QwertyDevice is above 4,000 milliseconds, which means that the clients' visual attention wanders from the mobile screen.

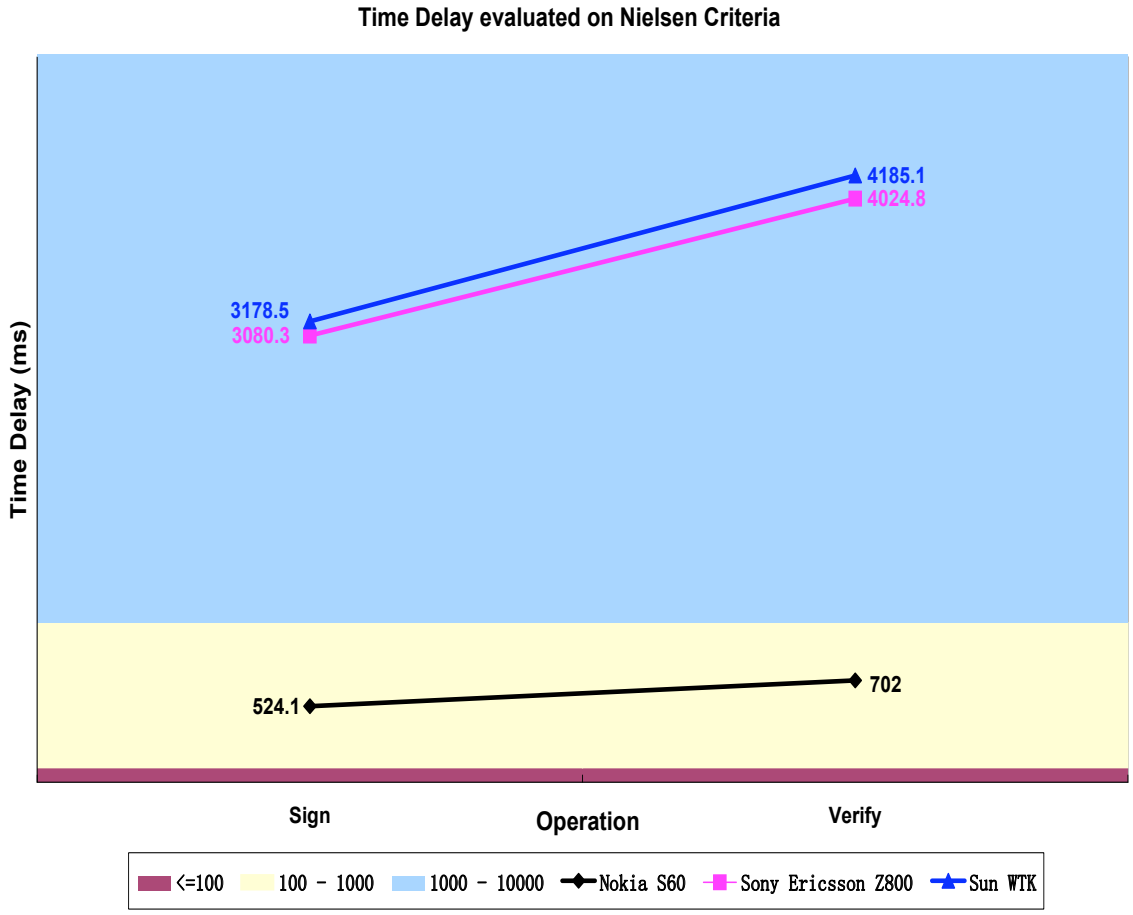


Figure 5.6: The time delay evaluated based on Nielsen Criteria.

Although the software application architecture is designed for a standard JVM, different mobile devices have different implementations for efficiency and optimization. This is shown in the simulation. The Symbian-based Nokia S60 has a more efficient implementation than the Sony Ericsson Z800.

In summary, evaluating the extra time delay based on the measurements respectively offered by Nielsen and Roto, *SA2pMP* is practical for implementing on mobile devices,

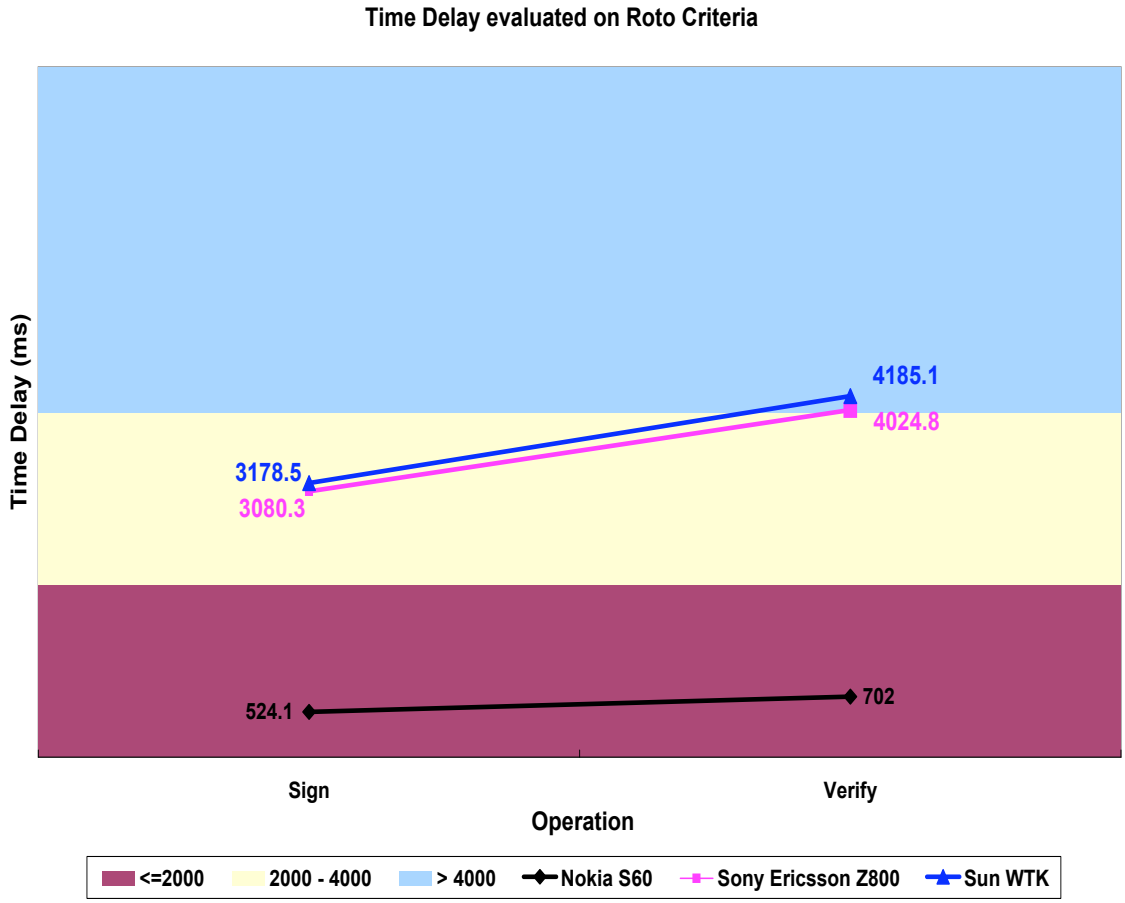


Figure 5.7: The time delay evaluated based on Roto and Oulasvirta Criteria.

although some visual or multi-modal feedback needs to be considered.

5.3.2 Code Size Evaluation

Another perspective evaluation for practicality is based on the code size. Currently the Java ME MIDP (2.0, 2.1) standard does not allow sharing of libraries among different MIDlet suites. Each MIDlet suite should have its own library. A MIDlet suite with an API library implementation normally has a large JAR-file. Mobile devices usually have their own size limitation for implementing MIDlet suites. For example, CLDC-1.0 specifies that the minimum base memory available for the Java platform is 160 KB, while CLDC-1.1 defines that the minimum base memory available for the Java platform is 192 KB (Java Community Process, n.d.-b). When the code size or the file stored in the mobile device is larger than the memory size, there will be problems. Furthermore, because the code needs to be loaded into memory to start a program, the size of code in the MIDlet may cause the program to have slower initiation. Thirdly, a large JAR-file also increases the download time if the application is installed over-the-air (OTA).

The Bouncy Castle API library, which is employed in developing the proposed architecture, contains different code used for different cryptography algorithms. The size of the completed API library is too large to be stored in some mobile devices. For example, *cldc_classes.zip* in Bouncy Castle 1.4.2 Java API package is 939 KB in size. As implementing *SA2pMP* only relates to the digital signature and the symmetric-key cryptography algorithm, the JAR-file size will shrink if the unused code of the Bouncy Castle API library is not contained in MIDlet suites.

Code obfuscation is the process of "obscuring" Java classes with the purpose of making reverse-engineering more difficult. Obfuscation typically entails renaming classes, meth-

ods, and fields to use shorter names, effectively reducing the size of the Java class (Ortiz, 2009). Proguard was utilized as an obfuscator in the present simulation.

In the simulation, the mobile client application, *MoBankClient.jar* is 130 KB in size, which is quite feasible for a CLDC-1.1 platform. Only a small part of the security package is added in this JAR-file. That means the API library designed in this research meets the requirement for code size, and the implementation on CLDC-1.1 mobile devices can be in practice.

In this chapter, the system simulation, and the evaluation for both time delay and code size are provide. In the following chapter, *SA2pMP* is compared to related work from the aspects of the architecture and time delay.

Chapter 6

System Comparison

In this chapter, *SA2pMP* is compared to some existing mobile payment architectures in terms of security and practicality. To make this comparison, other works are first introduced in section 6.1, followed by an architecture comparison in section 6.2. Finally, the time delay comparison is made in section 6.3.

6.1 Other Works

Mobile payments can be viewed as an extension of electronic payments. Existing secured architectures have been used to support electronic payments over Internet, such as E-cash for electronic cash (A. Ghosh, 1998), eCheck for electronic cheque (Anderson, 1998), and Secure Electronic Transaction (SET) for credit card payments (VISA and MasterCard, 1997). SET-based protocols are secure but not convenient because SET was originally designed for electronic payments over a wired network, and intermediary agents are required. E-cash, eCheck, and SET are standards for Internet payment protocols; they cannot be directly adopted for mobile payments as they do not address the limited resources of the mobile devices such as lower power, lower transmission rate, and less memory.

This section provides an investigation of other architectures proposed for ensuring security for mobile payment applications. These architectures are JASA, LSM, SET, and *i*KP.

6.1.1 *J2ME application-layer end-to-end security architecture*

A J2ME application-layer security architecture called JASA, a security architecture based on the Java 2 Platform, Micro Edition (J2ME) was developed by Itani and Kayssi (2004). to ensure end-to-end security for m-commerce. JASA uses pure Java components to provide end-to-end security between a wireless J2ME-based client and J2EE-based servers. The architecture can be implemented with the available limited resources of a Java MIDP device. As *SA2pMP*, JASA was designed for the application layer, so it does not require any modification of the underlying protocols or infrastructure.

JASA consists of a client application and a server. The client application complies with the MIDP 1.0 specification. The architecture can be employed in the current wireless network environment via HTTP because MIDP on top of CLDC provides an *HttpConnection* interface (for more details about MIDP and CLDC, refer to section 2.5).

Encryption and decryption operations and services in JASA are based on the AES algorithm. The SHA-1 algorithm performs the hashing operations to ensure integrity of data during transportation over the network. At the client end, the MIDP application is packaged inside a JAR file which contains the application class and resource files. The JAR file can be downloaded to the mobile device, along with the JAD file. The server application, specified with J2EE, is packaged in a web archive (WAR) file and deployed on the J2EE application server. As JASA employs AES, a symmetric ciphering algorithm, the server and the client application share the same key for encryption and decryption.

The performance of JASA was measured on the Sony Ericsson P800 Java phone, while the server application was tested using the J2EE server version 1.3.1 running with Windows 2000. A GPRS connection was utilized to perform the HTTP interactions between the client and the server. Itani and Kayssi (2004) claimed that the AES encryption operation

runs at over 165 Kbits/s on the Java phone, and the AES decryption operation runs at over 105 Kbits/s on the Java phone.

One limitation of JASA is that AES is a symmetric ciphering algorithm, meaning that the server and the client share the same key. Thus JASA can provide good end-to-end client authentication, data confidentiality, and integrity. However, the employment of AES is not able to guarantee non-repudiation in the transaction between the two parties, as AES is a symmetric-key cryptography algorithm. For background knowledge of AES, refer to Subsubsection 2.7.2.

6.1.2 Lightweight security for mobile commerce transactions

Lam et al. (2003) proposed a lightweight security mechanism (LSM) for protecting electronic transactions over handheld devices. In their research, the wireless network over which the mobile commerce transactions take place is the mobile phone network. The concept of a wireless protocol gateway was introduced in their proposal. A wireless protocol gateway serves as a wired agent for the handheld devices. As illustrated in Figure 6.1, handheld devices are connected to the application sever though the wireless protocol gateway. In other words, they are connected to the gateway via the mobile phone network and the gateway is connected to the application server via a fixed line network.

To overcome intensive computation on resource-limited mobile devices, LSM utilized a wireless protocol gateway. LSM is located on the layer above the Transport Layer Security protocol (TLS) and it presumes that the mobile handheld device supports plug-in or applet implementations in an Internet browser environment. Catering to the limited resources of mobile devices, LSM employs the Wireless Transport Layer Security protocol (WTLS) as defined by WAP Forum (The Open Mobile Alliance Ltd, n.d.). As illustrated in Figure

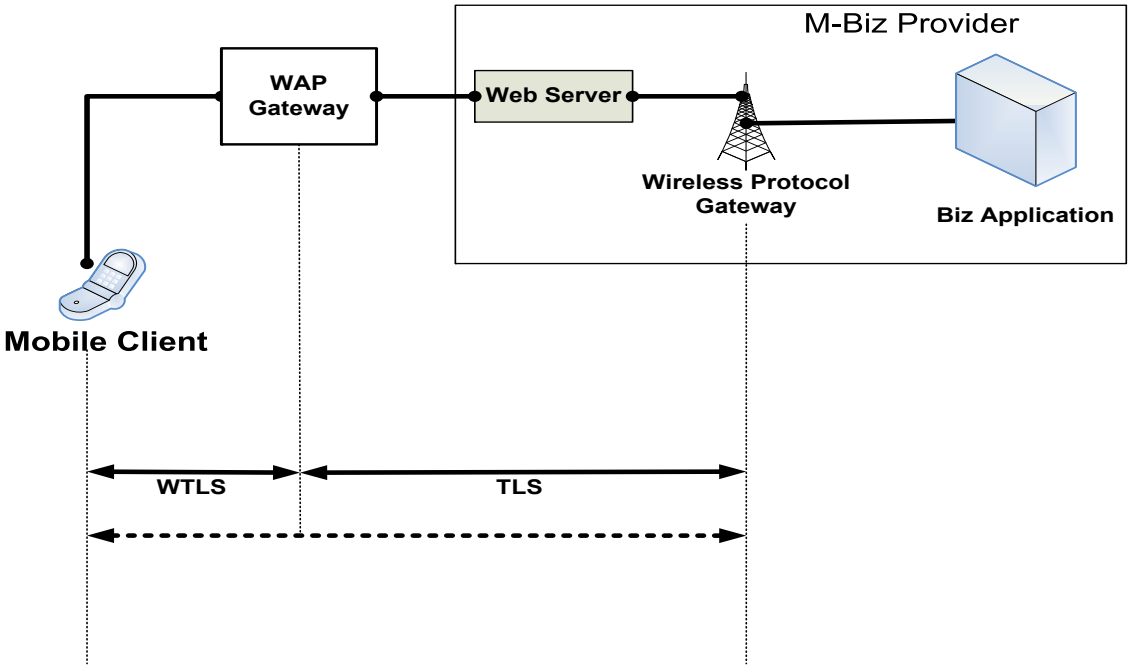


Figure 6.1: The system architecture adopted from the LSM (Lam et al., 2003).

6.1, the WAP Gateway plays an agent role to "translate WTLS-protected traffic to TLS-protected traffic"(Lam et al., 2003).

According to Lam et al. (2003), LSM is divided into two parts. For communications between the handheld device and the wireless protocol gateway, LSM uses an authentication protocol that requires sharing a symmetric secret key. This protocol intends to guarantee a secure connection between the handheld device and the wireless protocol gateway. The wired network communication between the wireless protocol gateway and the application server requires more complex transaction protocols. LSM employs a combination of a public-key cryptography mechanism and a simple password authentication solution. Lam et al. (2003) suggested the use of some tamper-resistant hardware device in order to ensure non-repudiation.

LSM meets the security requirements of mobile commerce in authentication, confidentiality, integrity and non-repudiation. The combination of the wireless protocol gateway

and the end-to-end security mechanism plays a role as an effective balance between high computational requirements and good security protection. However, there are still some limitations in this proposal.

One requirement of LSM is that the wireless protocol gateway shares many cryptography calculation duties with mobile devices. The wireless protocol gateway is responsible for maintaining a network communication channel, and an agent between the wired and wireless networks. More importantly, the wireless protocol gateway can be viewed as a part of mobile devices, with extensive computational capability. Much cryptography computation is performed in the wireless protocol gateway. This approach leads to a limitation for LSM that a security gap can exist in the wireless protocol gateway. The wireless gateway receives the traffic from handheld devices, decrypts the traffic with the symmetric key and encrypts them again using public-key cryptography, then sends the data to the application server. This can result in the exposure of the data at the wireless protocol gateway.

Another limitation is caused by the approach in which the mobile transaction is implemented on the mobile phone network. The mobile phone network is provided by mobile network operators, while mobile application services are offered by application providers. Financial service providers may not want mobile network operators to be excessively involved in their security strategy. However, LSM employs the wireless protocol gateway as a rich resource for performing intensive cryptography computations. This approach cannot avoid a high involvement of a third party, which undoubtedly increases the difficulties of a suitable collaboration among enterprise systems.

6.1.3 *Internet Keyed Payment Protocols*

Internet Keyed Payment Protocols (*i*KP) is a group of secure payment protocols developed by IBM Research Division (Janson, 2007; Bellare et al., 2000). All *i*KP protocols are based on RSA Public-key cryptography. The number of public keys, which is referred to as (*i*), varies according to the participants in a specific business requirement. The *i* is mirrored in the name of the individual protocols: 1KP, 2KP, and 3KP. The simplest protocol, 1KP, only asks for the acquirer to hold one public key (Janson, 2007). The main reason for designing these three variants was to enable gradual deployment.

Originally *i*KP was designed to suit any browser, server and platform. Now it is compatible with the existing card-based business models and payment system infrastructures. There are several participants involved in *i*KP:

- *Buyer*, who makes the actual payment,
- *Merchant*, who will receive the payment, and
- *Acquirer Gateway*, which plays a role as an intermediary between the electronic payment world and the existing payment infrastructure, and authorizes transactions using the latter.

Initially, *i*KP was designed for implementation in software. Now all *i*KP protocols can be implemented in either software or hardware. In fact, in 1KP and 2KP the buyer does not even need a personalized payment device: only credit card data and a PIN (if present) must be entered to complete a payment. However, to fulfill the increased requirement for security, it is obviously desirable to use a tamper-resistant device to protect the PIN and the secret key of the buyer (Bellare et al., 2000).

6.1.4 *Secure Electronic Transaction protocol*

The Secure Electronic Transaction protocol (SET) specification is an open encryption and security specification designed to protect credit card transactions on Internet. SET's development was led by VISA[1] and MasterCard[2]. Various well-known companies were involved in the development of SET, including IBM[3], Microsoft[4], Netscape[5], RSA[6], and VeriSign[7]. Currently the major supporters are VISA and MasterCard. As a standard protocol SET has been primarily defined to ensure the security of credit card payments on Internet, although the transaction flow and implementation of security can also be applied to wireless networks.

SET is very similar to *i*KP, especially the 3KP variant. The buyer, the merchant and the acquirer are all involved in the payment transactions. SET is not actually a payment system itself, but a set of security protocols enabling users to employ the existing credit card payment infrastructure on an open network, like Internet, in a secure environment. In general, SET includes three services:

1. providing a secure communications channel among all parties involved in a transaction;
2. ensuring trust through the use of the digital certificates; and
3. ensuring privacy because the information is only available to parties in a transaction when and where necessary.

There are several key components of the SET protocol (VISA and MasterCard, 1997).

[1] http://www.visa.com/
[2] http://www.mastercard.com/
[3] http://www.ibm.com/
[4] http://www.microsoft.com
[5] Netscape became a holding company of AOL in 1998
[6] RSA is the security division of EMC, http://www.rsa.com/
[7] http://www.verisign.com/

- The Cardholder Application, also referred to as a digital wallet, is held by an online consumer and packages a digital signature and credit card information that ensures the identity and safeguards the financial information through an encryption system.

- The Merchant Server component is the verification product held by the merchant to process the online card payment.

- The Payment Gateway component is supported by an acquiring bank or other trusted third party who can process the merchant's verification and the client's payment information and filters them to their appropriate financial institutions.

- The Certificate Authority component, run by a financial institution, is the trusted agent that issues the digital certificates and is responsible for ensuring that all users of digital certificates are in fact secure and trustworthy clients.

Details of the SET specification can be referred to (VISA and MasterCard, 1997).

Both SET (VISA and MasterCard, 1997) and *i*KP (Janson, 2007) are credit-card payment protocols that were designed originally for electronic commerce. Although they have been successfully implemented for e-commerce on wired networks, they are too heavy-loaded to operate on resource limited mobile devices and network environments such as mobile devices and wireless networks. SET and *i*KP are comprehensive architectures, not specifically designed for two-party payment transactions; this makes SET and *i*KP too complex for efficient and realistic implementations of two-party mobile payment transactions.

6.2 Architecture Comparison

A comparison of *SA2pMP* with the other existing security architectures for mobile payment is provided in this section. JASA proposed by Itani and Kayssi (2004) provides

end-to-end client authentication, and data confidentiality and integrity; however, it cannot guarantee non-repudiation during transactions. SET (VISA and MasterCard, 1997) and *i*KP (Janson, 2007) are two practical credit-card payment protocols, which were designed originally for electronic commerce. Although they have been successfully implemented for e-commerce on a wired network, their computations are too heavyweight to operate in a resource-limited environment. Lam et al. (2003) proposed LSM, which made use of a wireless protocol gateway to maintain a security for mobile commerce. However, the transaction data are possibly exposed to intruders at the wireless protocol gateway, as the data are decrypted and then re-encrypted there. Additionally, the intensive computation at the wireless protocol gateway, which is maintained by the mobile network operator, leads to an excessive involvement of the third party for a two-party (such as client and bank) payment transaction.

The proposed architecture, *SA2pMP*, is compared to JASA (Itani & Kayssi, 2004), LSM (Lam et al., 2003), SET and *i*KP (VISA and MasterCard, 1997; Janson, 2007) in eight dimensions in Table 6.1. "Targeted to M-Payment" evaluates if the architecture is designed for payment transactions over a mobile environment; "Targeted to 2-Party" shows if the architecture is proposed for a transaction in which only two parties are involved; "Cryptography Algorithm" compares which cryptography algorithm(s) are employed in the architecture; "Authentication Strategy" shows whether the architecture realizes a single-factor authentication or a multi-factor authentication; "Non-repudiation" indicates if the architecture addresses the problem of false denial or repudiation; "Java ME Enabled" describes the implementation approach; "Computational Requirements" indicates whether the architecture's computational requirement can be handled in a resource-limited mobile device; and "3rd-Party involvement" evaluates the level to which the architecture requires the third party to be involved.

As illustrated in Table 6.1, the comparison can be explained as follows:

Table 6.1: Architecture Comparison.

Fields \ Architecture	JASA	SET and iKP	LSM	*SA2pMP*
Targeted to M-Payment	√	No	√	√
Targeted to 2-Party	√	(1)KP	√	√
Cryptography Algorithm	AES	RSA	RSA	ECDSA AES
Authentication Strategy	SFA	SFA	SFA	MFA
Non-Repudiation	No	√	√	√
Java ME Enabled	√	-	-	√
Computational Requirements	Light-weight	Heavy-weight	Light-weight	Light-weight
The 3rd-Party Involvement	Low	High	High	Medium

[1] SFA: Single-factor Authentication;
[2] MFA: Multi-factor Authentication;

- In comparing the employment of cryptography algorithms, JASA utilizes AES, while LSM, SET and *i*KP employ RSA as the cryptography algorithm. AES is a Symmetric-key cryptography algorithm. Although AES has a better computational speed than Public-key cryptography algorithms (such as RSA), it is difficult to ensure non-repudiation for systems which employ it. RSA, as a Public-key cryptography algorithm, is able to fulfill the security requirements for authentication, integrity, confidentiality, and non-repudiation. However, RSA is too computationally heavyweight to be implemented in a resource-limited mobile device. *SA2pMP* introduces an integrated scheme combining both AES and ECDSA.

- Comparing authentication strategy, JASA, LSM, SET and *i*KP all utilize single-factor authentication, which is not enough to secure a financial application, but *SA2pMP* implements a multi-factor authentication strategy. The multiple factors are constituted by the technical properties in possession of a mobile device, in the knowledge of the personal identifier of the mobile device, and in the business transaction's account information. The combination of these multiple factors provides a strong authentication strategy. Additionally, the digital signature contributes to maintaining authentication in *SA2pMP*.

- From the perspective of computational requirements, JASA has the advantage in computational speed and resource requirement over AES; therefore, JASA can be viewed as lightweight in computational requirement. LSM utilizes the wireless protocol gateway as the agent of the mobile device. Therefore, it changes the wireless network to the wired network and changes the resource-limited mobile devices to resource-rich ones. The intensive computing is translocated from mobile devices to the wireless protocol gateway. The real computational requirement for mobile devices is not high. Therefore LSM is also lightweight in computational requirements.

SET and *i*KP were not originally designed for mobile commerce; the computational requirement is high. For this reason, SET and *i*KP are too heavyweight for the mobile payment activities. *SA2pMP* balances the security and the computational complexity and requirements, AES is chosen to realize a symmetric-key cryptography algorithm, which is faster and computationally lighter compared to the public-key cryptography algorithm. ECDSA is a digital signature algorithm based on the elliptic curve. ECDSA is believed to be more suitable for resource-limited mobile devices, as it provides equal security level with the shorter key size. Hence, *SA2pMP* is lightweight in computational requirements, without losing assurance in security.

- The design of each architecture has its target. JASA and LSM were designed for transactions in which only two participants are involved. SET is not designed for two-party payment transactions.The buyer, the merchant, and the financial services are all involved. *i*KP includes a series of protocols, in which 1KP is able to be applied for a two-party transaction. *SA2pMP* aims to maintain comprehensive security for two-party mobile payments. Although the network gateway is physically involved in *SA2pMP*, only two parties (client and financial service) participate in business transactions. Moreover, JASA, LSM, and *SA2pMP* are designed for payment transaction on mobile devices and over wireless networks, while SET and *i*KP were originally developed for electronic commerce over PCs and wired networks.

- Non-repudiation is a key requirement for comprehensive security. In all architectures listed in Table 6.1, only JASA did not suggest approaches to solve the false repudiation problem. LSM, SET and *i*KP made use of cryptography algorithms to maintain non-repudiation. Besides the employment of digital signature, *SA2pMP* utilizes a distributed transaction log strategy to provide a defensive approach to ensure non-repudiation.

- A mobile payment transaction have to cross different enterprises. An inter-enterprise transaction should not only support efficient collaboration, but also respect each enterprise protecting self-determination and privacy (Biennier & Favrel, 2005). In the mobile payment transaction, the financial service cooperates with the other supporting services, such as the mobile network operator. The models of the inter-enterprise business collaboration employed in these architectures are not same. In the case in this research, the collaboration between the financial service and the third party (such as the mobile network operator or TTP) is evaluated based on the third party's involvement. LSM stores its cryptography key pair in the network protocol gateway. One limitation is that this approach leads to an excessive involvement of the third party (the mobile network operator). The key pairs are essential to a cryptography strategy. Positioning key pairs with a third party rather than with one of the two participants of a transaction makes this third party important in the system. This excessive involvement of the third party will cause an excessive dependence on the third party. SET and *i*KP require high third-party involvement, as they rely on TTP to offer the key pairs. As JASA did not employ any third party in its architecture, it has low third-party involvement. *SA2pMP* utilizes the network gateway in its distributed transaction log strategy. The point is that *SA2pMP* treats the network gateway as the third trusted party for monitoring or auditing payment transactions. This approach avoids the third party's excessive involvement, but it still contributes to ensuring non-repudiation.

- Regarding the implementation, both JASA and *SA2pMP* mainly make use of Java ME to develop a software architecture. LSM, SET and *i*KP were designed to be suitable to any browser, server and platform. The development of LSM, SET and *i*KP can be with software, hardware, or a combination of software and hardware.

6.3 Time Delay Comparison

In this section a comparison from the implementation perspective is provided. The time delay during the cryptography computation is compared, and it is shown that *SA2pMP* has the advantage in operating feasibly on resource-limited mobile devices.

As *SA2pMP* employs the lightweight cryptography scheme combining ECDSA and AES, the computational time delay of AES is compared with JASA (Itani & Kayssi, 2004), then the computational time delay of ECDSA is compared with the result provided by Kilas (2009) (we use *Kilas* to represent the Kilas work). *Kilas* evaluated the practicality of digital signatures on Near Field Communication (NFC) tags in a Java-powered mobile phone. Although *SA2pMP* has different goal than does *Kilas*, it is possible to compare the time delay as both employed an ECDSA implementation.

Comparison the related works in terms of time delay is difficult. First, the simulation in this thesis is on PCs, while JASA and *Kilas* are based on actual devices. Second different brands or series of mobile devices were employed in the experiment. Table 6.2 depicts the models and technical specifications of the various mobile devices employed. Although mobile device manufacturers support unified standards such as MIDP and CLDC, they add unique features and functions that support their own branded mobile devices more efficiently. Therefore, different models may function differently, and cause further difficulty in the comparison.

Table 6.2: The mobile device models employed in JASA, *Kilas*, and *SA2pMP*.

	Model	Technical Specifications
JASA	Sony Ericsson P800	MIDP-1.0, CLDC-1.0
Kilas	Nokia 6131 NFC Sony Ericsson Z750 trial	MIDP-2.0, CLDC-1.0 MIDP-2.0, CLDC-1.1
SA2pMP	Nokia S60 Emulator Sony Ericsson Z800 Emulator	MIDP-2.0 MIDP-2.0, CLDC-1.1

Although it is impossible to make use of current data comparing *SA2pMP* with the other two related works, a rough evaluation of time delay is made.

If the AES implementation in JASA is examined, the AES encryption operation with key size of 128 bits ran at over 165 Kbits/s on the Java phone, and the decryption operation with key size of 128 bits ran at over 105 Kbits/s (Itani & Kayssi, 2004). In *SA2pMP*, AES encryption and decryption, both with the key size of 256 bits, cost less than 1 millisecond. As the plaintext is of 312-bit length (refer to subsection 5.2.3), the encryption speed can be roughly evaluated as:

$$Speed(Encrypt) = Length(plaintext)/T_{encrypt} > 312Kbits/s \tag{6.1}$$

Based on Formula 6.1, *SA2pMP* encryption speed ($> 312Kbits/s$) is clearly higher than JASA ($165Kbits/s$). With the same evaluation method, the decryption speed ($> 312Kbits/s$) of *SA2pMP* is also higher than JASA ($105Kbits/s$).

There are several different Java ME implementations for ECDSA designed by e.g. Tillich and Großschadl (2004); Zheng, Shao, Huang, and Yu (2008); Kilas (2009). As the platforms in which ECDSA is implemented are not the same, a strict comparison is difficult to make. Tillich and Großschadl (2004) and Zheng et al. (2008) suggested that ECDSA could be performed faster, while the emulation data provided by Kilas (2009) are less efficient than the data in the simulation of *SA2pMP* on the Nokia S60 emulator. A comparison of ECDSA's implementation is made between *Kilas* and *SA2pMP*. Both *Kilas* and *SA2pMP* employ a 192-bit key for signing and verification. As both *Kilas* and *SA2pMP* used SHA-1 to generate the message digest before the operation of signing and verification, they sign or verify a digital signature for the same sized message. The average time delay on different emulators is compared in Table 6.3.

As depicted in Table 6.3, the simulation on the Nokia S60 Emulator had the best per-

Table 6.3: The time delay caused by ECDSA implementation: *Kilas* and *SA2pMP*.

	Kilas	*SA2pMP*		
	Emulator	Nokia S60 Emulator	Sony Ericsson Z800 Emulator	Sun WTK Emulator
Signing	1088	524.1	3080.3	3178.5
Verification	2603	702	4024.8	4185.1

formance in all experimental implementations, when examining with both signing and verification calculations. The data from *Kilas* are better than *SA2pMP* according to the experiments on Sony Ericsson Z800 Emulator and Sun WTK 2.5.2 Emulator.

Although a strict comparison of time delay cannot be made due to the restrictions of experimental facility and environment, the result still shows some interesting information. Java ME allows developers to create an application compatible over numerous devices with the same piece of code. However, it varies in performance in that the application performs particular tasks differently on different platforms. The reason lies in the wide deviation of processor platforms, virtual machine (VM) implementations, and device memory capabilities (Yi, Reddy, & Ang, 2002). Table 6.3 reflects different mobile device manufacturers add characteristic functions and contribute different optimization to their own branded mobile devices. Clearly, Nokia S60 has the most positive performance in supporting the cryptography computation based on Table 6.3.

In this chapter, a comprehensive comparison of *SA2pMP* to the related approaches is provided. In combination with the result from the evaluation in Chapter 5, the thesis supports the belief that *SA2pMP* has it own advantages in security and practicality as an implementation for a two-party mobile payment. The thesis is concluded in the next chapter.

Chapter 7

Conclusions and Future Work

With the evolution of technologies in wireless networks and mobile devices, an increasing number of people are becoming users of mobile applications, on account of the advantages in convenience and portability. Mobile payment, as an important mobile application in the financial field, is attracting wide attention from researchers, developers, bankers, merchandisers, and clients. However, it has not yet become a mainstream approach for making payments. Non-secured mobile payments are simply not acceptable.

The goal of mobile payment research to enable payment transactions to operate on mobile devices and wireless networks. Although the technologies in these two fields have improved and are experiencing a significant development, mobile devices and wireless networks are still "resource-limited" compared to PCs and fixed-line networks. The difficulty in building mobile payment systems lies in how to provide payment transactions with security and practicality.

In this respect, the focus of this research is mobile transaction security. The goal is to design a security architecture for two-party mobile payment transactions. A wide investigation of security issues on mobile transactions is provided, particularly focusing on mobile payments. The Four-layer mobile payment participant model on the basis of the model suggested by Karnouskos (2004) clarifies how participants are involved in mobile payments. Based on the Onion Layer Framework, a security map is proposed to guide the present research analysis activities. Finally a new security architecture for two-party mobile payment referred to as *SA2pMP* is proposed.

SA2pMP operates on Java-enabled mobile devices with Internet browser capability. The participants in a logical payment transaction only include the financial sector and the client. As the mobile payment runs over a mobile phone network, the mobile network operator is

physically employed to provide a network communication channel. *SA2pMP* is built on the application layer, which means that it does not make any modification to current network protocols and wireless network infrastructures. Since the mobile bank is a typical two-party mobile payment, it is used as a scenario for describing system design and implementation.

SA2pMP employs a combination of technologies to provide comprehensive security for mobile payments. It fulfills the four basic security requirements of authentication, integrity, confidentiality, and non-repudiation. Some of the highlights of *SA2pMP* are as follows:

- A lightweight cryptography scheme is implemented, which combines a symmetric-key cryptography algorithm (AES) along with a digital signature algorithm (ECDSA). ECDSA is a lightweight digital signature algorithm for better operating on resource-limited mobile devices. It contributes to ensuring the properties of authentication, integrity, and non-repudiation. AES is a popular symmetric key algorithm which has the benefit of running faster on mobile devices, as compared to a public key algorithm. It is employed to ensure confidentiality during a payment transaction. The ease of using a simple "Sign-and-Encrypt" approach avoids high computational requirements on resource-limited mobile devices.

- A multi-factor authentication strategy is proposed, which employs the inherent factors of mobile devices, bank account, and networks to provide a strong authentication for a mobile transaction.

- A distributed transaction log strategy is suggested to partly maintain non-repudiation. The distributed transaction log strategy makes use of the "cooperator and monitor" business model between financial sectors and mobile network operators. The mobile network operator is viewed as a third party auditor for monitoring the business transaction. It is responsible for maintaining a transaction log.

Additionally, the security API package created in implementing *SA2pMP* can be reused as the security library for developing other mobile security applications. Designed originally for two-party mobile payments, *SA2pMP* can potentially be extended to ensure security for other two-party mobile applications.

The simulation is performed on an IBM IntelliStation M Pro PC, with Pentium 4 CPU 2.80 GHz and 2 GB RAM. The operation system is Windows XP Professional SP3. Three emulators (the Nokia S60 Emulator, the Sony Ericsson Emulator, and the Sun WTK CLDC simulator) were employed in the system simulation. The evaluation of both time delay and code size shows that *SA2pMP* is practical and efficient for implementation on mobile devices.

Compared to some related works such as JASA (Itani & Kayssi, 2004), SET and *i*KP (VISA and MasterCard, 1997; Janson, 2007), and LSM (Lam et al., 2003), *SA2pMP* has the advantage of more lightweight computation, more comprehensive security, and less third party involvement.

Future work could focus on the following areas:

- Currently, due to limitations of hardware environments, the present experiment in this thesis were operated on a PC. Three emulators were used to simulate mobile devices. In future work, the actual device will be employed in experiment and evaluation.

- A good cryptography approach needs a good key management strategy. *SA2pMP* employs paralleled key management for the key pair of digital signature, and the symmetric key pair of encryption and decryption. Although some options are suggested for key storage and distribution, especially for the symmetric-key cryptography algorithm, these candidates need to be evaluated in the context of a practical mobile payment system.

- The distributed transaction log strategy employed in *SA2pMP* makes use of the mo-

bile network operators' role as a monitor to audit business transactions that take place between the client and the financial sector. Although it successfully contributes a technical strategy to solve the problem that lies in business repudiation, it still needs analysis on a case-by-case basis in real applications. The role that the mobile network operator eventually plays will be dependent on what is allowed by the specific business contract or the relevant legislation. For example, current legislation in Finland does not allow Mobile Network Operators to charge for services exceeding a certain amount of money. This example means that the operator may need to found subsidiary companies in order to act as a trusted third party and/or to acquire licenses for a bank (Tsalgatidou & Veijalainen, 2000). Another issue that must be concerned is the client privacy. A business transaction log contains some sensitive information of the clients. Allowing mobile network operators to maintain the business log does possible business-sensitive information among these three parties (clients, financial sectors, and the mobile network operators) to be leak. How to ensure client privacy in this approach is worth consideration. Future work in the distributed transaction log strategy needs further analysis than simply basing on technological practicality.

- Adequate security architecture is the most important measure to protect mobile payment systems. The security architecture can be viewed as the front line for prevention of attacks. However, it is almost impossible to implement a completely secure system. It is inevitable to have bugs and mistakes during the implementation process. A number of security vulnerabilities and incident reports have been issued by, for example, the CERT Coordination Center (Carnegie Mellon Software Engineering Institute, 2004). The fraud detection system (FDS) is in place to detect attempted and completed frauds; therefore, it can act as the second line of defense for protection of systems (Barse, 2004). The fraud detection is not in the research scope in

this thesis. However, the transaction log data could be employed in fraud detection research in the future.

Clearly wireless networks and mobile device technologies are still in rapid development. The growth of 3G/4G network technology and the Smartphone brings more and more opportunities to mobile applications. *SA2pMP*, as a lightweight architecture, can be feasibly implemented on mobile devices to provide a comprehensive security for two party mobile payments. The financial sector in two-party mobile payments might be represented not only by the traditional commercial banks but by stock traders, or even by Internet payment agents, for example, PayPal. Furthermore, the implementation of *SA2pMP* is low cost and can also be easily integrated into the other end-to-end mobile applications. We expect *SA2pMP* to provide a lightweight security architecture for ensuring comprehensive security; meanwhile, it can be practically implemented in a resource-limited environment.

Glossary

ACCID	Denotes the *Client*'s bank account number, or ACCount IDentifier
Amount	Denotes the amount of money transferred
Bank	Denotes a server computer held by the financial service participant in the transaction
Client	Denotes the person using a mobile device on the client side of the transaction
ClientDevice	Denotes the mobile device used by the client to carry out their mobile banking transaction
DS	Denotes the digital signature
DSign	Denotes the signing process for the digital signature
$DVerify$	Denotes the verification process for the digital signature
Decrypt	Denotes the decryption process for the symmetric-key cryptography algorithm
Encrypt	Denotes the encryption process for the symmetric-key cryptography algorithm
F_Acc_ID	Denotes the ID number of the bank account from which the money is transferred
Gateway	Denotes the wireless gateway offered by a mobile network operator
ID_C	Denotes the identity information of the client. $ID_C = (SIM + PHID + ACCID)$

K_E	Denotes the secret key for the symmetric-key cryptography algorithm
$PHID$	Denotes PHone IDentifier, such as the mobile phone serial number
PIN	Denotes the $Client$'s Personal Identifier Number in the mobile device
PK_S	Denotes X's public key which is used to verify the digital signature in the transaction
PWD	Denotes the password of $Client$, which is known only to $Client$ and is verifiable by $Bank$. For example, the debit card has its password.
RK_S	Denotes X's private key which is used to generate the digital signature and to sign the data transferred in the transaction
SIM	Denotes the Subscriber Identifier Module number of $ClientDevice$
$Session$	Denotes the HTTP communication session set up between $Bank$ and $ClientDevice$
TD	Denotes the transaction data transferred in a transaction
$TLog$	Denotes the transaction log
T_Acc_ID	Denotes the ID number of bank account to which the money is transferred
T_+	Denotes a set of operations for formatting the string-based message

T_{CBiz}	Denotes the time delay caused by the business computation on $ClientDevice$
T_{CSec}	Denotes the time delay caused by the security computation on $ClientDevice$
T_{Com}	Denotes the time delay caused by the HTTP(s) communication
T_{DSign}	Denotes the time delay caused by signing or verifying a digital signature
T_{SBiz}	Denotes the time delay caused by the specific business processing computation on *Bank*
T_{SSec}	Denotes the time delay caused by the secure verification process and the decryption process on *Bank*
$T_{encrypt}$	Denotes the time delay caused by the encryption process or the decryption process
$Time$	Denotes the time when money transfer transaction is initiated
$TimeStamp$	Denotes a time stamp
$Verify$	Denotes the process for verifying the transaction's authentication
X	Denotes any participant involved in a mobile payment transaction
Yes/No	Denotes the transaction's status: approved or rejected

$[RK_S, PK_S]$	Denotes a digital signature key pair including a private key and a public key
$added\ identifier$	Denotes the identity information of the client, formulated by ID_C. The $added\ identifier$ is a combination of the *SIM*, *PHID*, and *ACCID*
$h(msg)$	Denotes a one-way hash function for the message
msg	Denotes a message
3GPP	The Third Generation Partnership Project
AES	Advanced Encryption Standard
AES256	The AES algorithm with the key size of 256 bits
Android	A set of software for mobile devices: an operating system, middleware and key mobile applications (Open Handset Alliance, n.d.)
APIs	Application Programming Interfaces
AS	Authentication Server
ATM	Automated Teller Machine
Authentication	Ensuing that the communicating entity is who they are claiming to be (Stallings, 2006)
BlackBerry OS	The proprietary operating system made by a Canadian wireless device company, Research In Motion, for BlackBerry Smartphones
BLM	Business Logic Module
BLS	Business Logic Server

Bouncy Castle	An open-source collection of lightweight cryptography APIs
BTLS	Business Transaction Log Server
CDC	Connected Device Profile
CDMA	Code Division Multiple Access
CLDC	Connected Limited Device Configuration
CM	Communication Module
Confidentiality	Ensuring the operation in private
Cryptography	The art and science of securing messages so unintended audiences cannot read, understand, or alter that message (Tipton & Krause, 2003)
DES	Data Encryption Standard
DSA	Digital Signature Algorithm
DSS	Digital Signature Standard
ECC	The Elliptic Curve Cryptography
ECDSA	The Elliptic Curve Digital Signature Algorithm
ECDSAprime192	The ECDSA algorithm on prime integer with the key size of 192 bits
ETSI	European Telecommunications Standards Institute
FDS	Fraud Detection System

FIPS	Federal Information Processing Standards
FORTRAN	A programming language that is especially suited to the numeric computation and the scientific computing
GPRS	General Packet Radio Service
GSM	Global System for Mobile Communication
HCI	Human Computer Interaction
HTTPs	Hypertext Transfer Protocol Secure
iKP	Internet Keyed Payment Protocols
IMT-2000	The International Mobile Telecommunications 2000
Integrity	The information and systems must be guaranteed against corruption by outside parties
iPhone OS	The operating system developed by Apple Inc. for the iPhone and iPod Touch
ITU	International Telecommunication Union
J2EE	Java 2 Platform, Enterprise Edition
J2ME	Java 2 Platform, Micro Edition
JAD	Java Application Descriptor
JAR	Java ARchive file

JASA	A J2ME application-layer security architecture proposed in (Itani & Kayssi, 2004)
Java EE	Java Platform, Enterprise Edition, which was formerly known as Java 2 Platform, Enterprise Edition (J2ME)
Java EE	Java Platform, Enterprise Edition
Java ME	Java Platform, Micro Edition, which was formerly known as Java 2 Platform, Micro Edition (J2ME)
JCA	Java Cryptography Architecture
JCE	Java Cryptography Extension
JCP	Java Community Process
JSR	Java Specification Request
JVM	Java Virtual Machine
KMM	Key Management Module
KMS	Key Management Server
LAN	Local Area Network
Linux	A Unix-like computer operating system, which can be implemented in the mobile device
LSM	The lightweight security mechanism proposed by Lam et al. (2003)
MBP	Mobile Banking Platform

MFA	Multi-Factor Authentication
MIDlet	The MIDP application
MIDP	Mobile Information Device Profile
Mobile Device	a handheld devices with functions such as GPRS connectivity, Internet browsing, and basic computational capabilities
MPN	Mobile Phone Network
NFC	Near Field Communication
NIST	National Institute of Standards and Technology
Nokia S60	A Symbian OS for Nokia mobile phones
Non-repudiation	Ensuring that the originator cannot falsely repute or deny a transaction
Onion Layer Framework	A security framework adopted from Onion Ring Framework (Wei et al., 2006)
Onion Ring Framework	An m-commerce security framework proposed by Wei et al. (2006)
Palm	An embedded operating system for PDAs
PAP	Password Authentication Protocol
PDAs	Personal Digital Assistants

Pocket PCs	The handheld devices that enable users to store and retrieve e-mail, contacts, appointments, tasks, play multimedia files, games, exchange text messages with Windows Live Messenger (formerly known as MSN Messenger), browse the Web, and more
POS	Point of Sale
Proguard	An obfuscation software
Public-key Cryptography	The cryptography based on that the sender and the receiver hold the different keys
RIM	Research In Motion, a Canadian wireless device company.
RMS	Record Management System in MIDP
RSA	The *Rivest-Shamir-Adleman* algorithm
RTOS	real-time operating system
SA2pMP	The security architecture for two-party mobile payment
SDK	Software Development Kit
Security Component	A potential secured technology employed in a secured system
Security Map	A security solution analysis mode based on the Onion Layer Framework
SET	Secure Electronic Transaction

SFA	Single-Factor Authentication
SHA	The Secure Hash Algorithm
SHA-1	A Secure Hash Standard
SM	Security Module
Smartphone	The powerful, multi-function cell phones that incorporate a number of PDA functionality (Yang, Zheng, & Ni, 2007)
SMS	Short Messaging Service
SSL	Secure Sockets Layer
Symbian	A proprietary operating system for mobile devices
Symmetric-key Cryptography	The cryptography based on that the same key is shared among the sender and the receiver
TAP	Third Auditor Party
TDD/CDMA	Time Division Duplex/Code Division Multiple Access
The strong authentication	The layered authentication approach relying on two or more authenticators to establish the identity of an originator or receiver of information (The United States Federal Government, 2006)
TTP	Trusted Third Party
UIQ	User Interface Quartz
UMTS	Universal Mobile Telephone System

Unisys Security Index	The Unisys Security Index presents a social indicator regarding how safe consumers feel on key areas of security. Conducted twice a year the Unisys Security Index provides a regular, statistical measure of concerns about four areas of security: national, financial, Internet and the personal safety.(http://www.unisyssecurityindex.com/)
VM	Virtual Machine
W-CDMA	Wideband Code Division Multiple Access
WAP	Wireless Application Protocol
WAR	Web ARchive file
Windows Mobile	A light version operating system combined with a suite of basic applications for mobile devices based on the Microsoft Win32 API
WLAN	Wireless Local Area Network
WML	Wireless Markup Language

References

ABI Research. (2007). *203 Million Mobile Phones Will Use Linux Operating Systems by 2012, with 76 Million as RTOS Replacements.* www.abiresearch.com.

ABI Research. (2008). *Mobile Linux Bringing License-Free Operating Systems to Smartphones and Middle-Tier Devices* (Tech. Rep.). Oyster Bay, NY, USA: Author.

ACCESS Press Release. (2007). *ACCESS Debuts New ACCESS Powered Mark.*

Adachi, F., Garg, D., Takaoka, S., & Takeda, K. (2005). Broadband CDMA techniques. *IEEE Wireless Communications*, *12*(2), 8-18.

Aite Group. (2007). *Mobile Banking Security: The Black Cloud Attached to the Silver Lining.* Boston, MA, USA.

American Banker. (2009). *Mobile Growth Forecast by TowerGroup.* American Banker.

Anderson, M. (1998). The Electronic Check Architecture. *Financial Services Technology Consortium*.

Apple Inc. (2009). *iPhone OS 3.0 Software, Get an Advance Preview.* http://www.apple.com/iphone/preview-iphone-os/.

Barker, E., Barker, W., Burr, W., Polk, W., & Smid, M. (2007). *NIST SP800-57: Recommendation for Key Management Part 1: General (revised)* (Tech. Rep.). Gaithersburg, Maryland, USA: National Institute of Standards and Technology.

Barse, E. L. (2004). *Logging for intrusion and Fraud Detection*. Unpublished doctoral dissertation, Department of Computer Engineering, Chalmers University of Technology, Sweden.

Bellare, M., Garay, J., Hauser, R., Herzberg, A., Krawczyk, H., Steiner, M., et al. (2000). Design, Implementation, and Deployment of the iKP Secure Electronic Payment System. *Selected Areas in Communications, IEEE Journal*, *18*, 611-627.

Bhise, L. (2009). *Future of Mobile Internet: Downloadable Mobile Applications.* www.alootechie.com.

Biennier, F., & Favrel, J. (2005). Collaborative Business and Data Privacy: Toward a Cyber-Control? *Computers in Industry*, *56*, 361 C 370.

Bohlin, E., Lindmark, S., Bjorkdahl, J., Weber, A., Wingert, B., & Ballon, P. (2004). *The Future of Mobile Communications in the EU: Assessing the Potential of 4G* (Tech. Rep.). Seville, Spain: Institute for Prospective Technological Studies (IPTS).

Boneh, D., & Daswani, N. (1999). Experimenting with Electronic Commerce on the PalmPilot. In *Proceedings of Financial Cryptography '99, Lecture Notes in Computer Science* (Vol. 1648, p. 1-16). New York, NY, USA: Springer.

Bruene, J. (2007). *Mobile Money & Payments, Why Credit & Debit Card Issuers Should Embrace Mobile Delivery Now* (Tech. Rep.). Seattle, WA, USA: Online Financial Innovations.

CanadianContent. (n.d.). *GSM Technology.* http://www.canadiancontent.net/mobile/.

Carnegie Mellon Software Engineering Institute. (2004). *CERT Coordination Center.* http://www.cert.org/.

Chang, S., Eberle, H., Gupta, V., & Gura, N. (n.d.). *Elliptic Curve Cryptography: How it Works.* http://research.sun.com/projects/crypto.

Chapman, S. (2009, October). *Updated Windows Mobile 7 RTM Time Frame and Office Mobile 7 Teaser.* http://msftkitchen.com.

Daemen, J., & Rijmen, V. (1999). *AES Proposal: Rijndael.* Available from http://csrc.nist.gov/CryptoToolkit/aes/rijndael/

Daemen, J., & Rijmen, V. (2006). The Block Cipher Rijndael. *Smart Card. Research and Applications*, *LNCS1820*, 277-284.

Davis, D. (2002). Defective Sign & Encrypt in S/MIME, PKCS#7,MOSS, PEM, PGP, and XML. In *Proceedings of the General Track: 2002 USENIX Annual Technical Conference* (p. 65-78). Monterey, CA, USA: USENIX Association.

Deans, P. C. (2004). *E-Commerce and M-Commerce Technologies*. Hershey, PA, USA: IRM Press.

Debbabi, M., Talhi, C., & Zhioua, S. (2007). *Embedded Java Security: Security for Mobile Devices*. New York, NY, USA: Springer.

Delfs, H., & Knebl, H. (2002). *Introduction to Cryptography: Principles and Applications*. New York, NY, USA: Springer.

Diffie, W., & Hellman, M. (1976). New Directions in Cryptography. *IEEE Transactions on Information Theory*, *IT*(22), 644-654.

DiMarzio, J. (2008). *Android: A Programmer's Guide*. New York, NY, USA: McGraw-Hill Professional.

Dray, J. (2000). *NIST Performance Analysis of the Final Round Java AES Candidates* (Tech. Rep.). Gaithersburg, Maryland, USA: Computer Security Division, The National Institute of Standards and Technology.

EDS.com. (n.d.). *EDS Authentication.* EDS Slides on EDS.com.

ElGamal, T. (1985). A Public-Key Cryptosystem and a Signature Scheme Based on Discrete Logarithms. *IEEE Transactions on Information Theory*.

Fernandes, A. (1999). Elliptic Curve Cryptography. *Dr. Dobb's Journal*.

Ferro, E., & Potorti, F. (2005). Bluetooth and Wi-Fi Wireless Protocols: A Survey and A Comparison. *IEEE Wireless Communications*, *12*, 12-26.

Firesmith, D. (2003). Security Use Cases. *Journal of Object Technology*, *2*(3), 53 - 64.

FSTC Press Releases. (2007). *M-Commerce: Why Rollout, Patch Later Is a Dangerous Prescription.*

Gao, J., Cai, J., Patel, K., & Shim, S. (2005). A Wireless Payment System. In *Second International Conference on Embedded Software and Systems (ICESS 2005)* (Vol. 16-18, p. 8). Xi'an, China: Springer.

Ghosh, A. (1998). *E-Commerce Security Weak Links, Best Defenses* (First Edition ed.). New York, NY, USA: Robert Ipsen.

Ghosh, S. (2002). J2ME Record Management Store. *IBM DeveloperWorks*. Available from http://www.ibm.com/developerworks/library/wi-rms/

Giguere, E. (2004). *Databases and MIDP, Part 1: Understanding the Record Management System.*

Ginevan, S. (2002). Networking in the Palm of Your Hand. *Network Computing*, *13*, 67-68.

Glisic, S. G., & Leppanen, P. A. (1997). *Wireless Communications: TDMA versus CDMA*.

Norwell, MA, USA: Kluwer Academic Publishers.

Global Information Inc. (2005). *Wireless Payments: The New Payments Paradigm - 2005 to 2010.*

Goldreich, O. (2005). *Foundations of Cryptography - A Primer*. Rehovot, Israel: Dpeartment of Computer Science, Weizmann Institute of Science.

GoMoNews. (2009a). *Mobile banking Continues to Rise with Canadas Telus and ATB Financial.* www.gomonews.com.

GoMoNews. (2009b). *SMS Mobile Banking for State Bank & Trust Powered by ClairMail.* http://www.gomonews.com.

Ham, W., Choi, H., Xie, Y., Lee, M., & Kim, K. (2002). Secure One-way Mobile Payment System Keeping Low Computation in Mobile Devices. In *Proceedings of The 3rd International Workshop on Information Security Applications (WISA 2002)* (p. 287-301). Jeju Island, Korea (South): Springer.

Hassell, J. (2002). *RADIUS: Securing Public Access to Private Resources*. Sebastopol, CA, USA: OŔeilly and Associates, *Inc.*

Herzberg, A. (2003). Payments and Banking with Mobile Personal Devices. *Communications of the ACM*, *46*, 53-58.

IBM MARS Team. (n.d.). *The MARS Cipher - IBM Submission to AES.* IBM.

Itani, W., & Kayssi, A. (2004). J2ME Application-Layer End-to-End Security for M-commerce. *Journal of Network and Computer Applications*, *27*, 13-32.

ITU. (2005). *Cellular Standards for 3G.* http://www.itu.int/osg/spu/imt-2000/technology.html.

Janson, P. (2007). *Internet Keyed Payment Protocols (iKP)* (Tech. Rep.). Zurich, Switzerland: IBM Zurich Information Technology Solutions.

Java Community Process. (n.d.-a). *JSR 118: Mobile Information Device Profile 2.0.* http://www.jcp.org/en/jsr/detail?id=118.

Java Community Process. (n.d.-b). *JSR 139: Connected Limited Device Configuration 1.1.* http://www.jcp.org/en/jsr/detail?id=139.

Java Community Process. (2002). *Mobile Information Device Profile, Version 2.0.* Sun Developer Network.

Johnson, D., Menezes, A., & Vanstone, S. (2001). The Elliptic Curve Digital Signature Algorithm (ECDSA). *International Journal of Information Security*, *1*(1), 36-63.

Jurisic, A., & Menezes, A. (1997). Elliptic Curves and Cryptography. *Dr. Dobb's Journal*.

Karnouskos, S. (2004). Mobile Payment: A Journey Through Existing Procedures and Standardization Initiatives. *IEEE Communications Surveys and Tutorials*, *6*(4), 44-66.

Karnouskos, S., & Vilmos, A. (2004). The European Perspective on Mobile Payments. In *IEEE Symposium on Trends in Communications (SympoTIC '04).* Bratislava, Slovakia: IEEE.

Kilas, M. (2009). *Digital Signatures on NFC Tags*. Unpublished master's thesis, KTH Royal Institute of Technology, Sweden.

Koblitz, N. (1987). Elliptic Curve Cryptosystems. *Mathematics of Computation*, *48*, 203-209.

Koblitz, N., Menezes, A., & Vanstone, S. (2000). The State of Elliptic Curve Cryptography. *Designs, Codes and Cryptography*, *19*, 173 - 193.

Krawetz, N. (2006). *Introduction to Network Security*. Boston, MA, USA: Charles River Media.

Krill, P. (2008). Sun: We'll Put Java on the iPhone (Sun Readies Virtual Machine to Make Java Apps Run on Apple's Mobile Platform). *InfoWorld*.

Kuwayama, J. (2008). *Mobile Banking is Just Around the Corner.* BizTimes.com.

Lai, Y., P.Lin, & Huang, Y.-T. (2006). Design and Implementation of a Wireless Internet Remote Access Platform. *Wireless Communications and Mobile Computing*, *6*(4), 413-429.

Lam, K.-Y., Chung, S.-L., Gu, M., & Sun, J.-G. (2003). Lightweight Security for Mobile Commerce Transactions. *Computer Communications*, *26*, 2052-2060.

Laukkanen, T., & Lauronen, J. (2005). Consumer Value Creation in Mobile Banking Services. *International Journal of Mobile Communications*, *3*(4), 325-338.

Lenstra, A. K., & Verheul, E. R. (1999). Selecting Cryptographic Key Sizes. *Lecture Notes in Computer Science*, *1751*, 446-465.

Li, S., & Knudsen, J. (2005). *Beginning J2ME: From Novice to Professional* (Third Edition ed.). Berkeley, CA, USA: Apress.

Li, Z., Higgins, J., & Clement, M. (2001). Performance of Finite Field Arithmetic in an Elliptic Curve Cryptosystem. In *Ninth Symposium on Modeling, Analysis and Simulation of Computer and Telecommunication Systems (MASCOTS'01)* (p. 249 - 258). Cincinatti, Ohio, USA: IEEE Computer Society.

Linck, K., Pousttchi, K., & Wiedemann, D. (2006). Security Issues in Mobile Payment from the Customer Viewpoint. In J. Ljungberg & M. Andersson (Eds.), *Proceedings of the 14th European Conference on Information Systems (ECIS 2006).* Goteborg, Sweden: MPRA.

Lopez, J., & Dahab, R. (2000a). *An Overview of Elliptic Curve Cryptography* (Tech. Rep.). State of Sao Paulo, Brazil: Institute of Computing, State University of Campinas, Brazil.

Lopez, J., & Dahab, R. (2000b). *Performance of Elliptic Curve Cryptosystems* (Tech. Rep.). State of Sao Paulo, Brazil: State University of Campinas.

Mahmoud, Q. (2000). *MIDP Database Programming Using RMS: a Persistent Storage for MIDlets.* Sun Developer Network.

Maiwarld, E. (2004). *Fundamentals of Network Security*. Burr Ridge, IL, USA: McGraw-Hill Professional.

Mann, S., Sbihli, S., & NetLibrary, Inc. (2002). *The Wireless Application Protocol (WAP): A Wiley Tech Brief.* Hoboken, New Jersey, USA: John Wiley and Sons *Inc.*, Publication.

McKitterick, D., & Dowling, J. (2003). *State of the Art Review of Mobile Payment Technology* (Tech. Rep.). Dublin, Ireland: Depart of Computer Science, Trinity College Dublin.

Menezes, A., Oorschot, P. V., & Vanstone, S. (1996). *Handbook of Applied Cryptography*. Boca Raton, Florida, USA: CRC Press.

Merz, M. (2002). *E-Commerce und E-Business: Marktmodelle,Anwendungen und Technologien* (Second Edition ed.). Deutsch: Dpunkt Verlag.

Microsoft Corporation. (2009). *Windows Mobile.* http://developer.windowsmobile.com/.

Miller, R. B. (1968). Response Time in Man-Computer Conversational Transactions. In *Proceedings of the AFIPS Joint Computer Conferences (AFIPS '68), part I* (p. 267 - 277). San Francisco, California, USA: ACM.

Mobile Marketing Association. (2009). *Mobile Banking Overview (NA)* (Tech. Rep.). New York, NY, USA: MMA Mobile Marketing Association. Available from www.mmaglobal.com

MobileMentalism.com. (2009). *Toshiba TG01 Offers Largest Screen and Fastest Processor of Any Smartphone.*

Mollin, R. A. (2003). *RSA and Public-Key Cryptography*. Boca Raton, Florida, USA: CRC Press.

Nah, F. F.-H. (2004). A Study on Tolerable Waiting Time: How Long Are Web Users Willing to Wait? *Behaviour & Information Technology*, *23*(3), 153 - 163.

Nahum, E., O'Malley, S., Orman, H., & Schroeppel, R. (1995). Towards High Performance Cryptographic Software. In *Third IEEE Workshop on the Architecture and Implementation of High Performance Communication Subsystems (HPCS '95)* (p. 69 - 72). Mystic, Connecticut, USA: IEEE.

Nambiar, S., Lu, C.-T., & Liang, L. (2004). Analysis of Payment Transaction Security in Mobile Commerce. In *Proceedings of the 2004 IEEE International Conference on Information Reuse and Integration.* Las Vegas, Nevada, USA: IEEE.

Nielsen, J. (1995). *Usability Engineering*. San Francisco, CA, USA: Morgan Kaufmann Publishers *Inc.*

NIST. (1995). *FIPS PUB 180-1: Federal Information Processing Standards Publication, Secure Hash Standard (SHS)* (Tech. Rep.). Gaithersburg, Maryland, USA: National Institute of Standards and Technology.

NIST. (2000). *FIPS PUB 1862: Federal Information Processing Standards Publication, Digital Signature Standard (DSS)* (Tech. Rep.). Gaithersburg, Maryland, USA: National Institute of Standards and Technology.

NIST. (2006). *FIPS 200: Minimum Security Requirements for Federal Information and Information Systems* (Tech. Rep.). Gaithersburg, Maryland, USA: National Institute of Standards and Technology.

NIST. (2008). *FIPS PUB 180-3: Federal Information Processing Standards Publication, Secure Hash Standard (SHS)* (Tech. Rep.). Gaithersburg, Maryland, USA: National Institute of Standards and Technology.

Nokia. (2009). *S60 Platform and Device SDKs.* forum.nokia.com.

Ondrus, J., & Pigneur, Y. (2005). A Disruption Analysis in the Mobile Payment Market. In *Proceedings of the 38th Annual Hawaii International Conference on System Sciences (HICSS '05).* Big Island, Hawaii, USA: IEEE Computer Society.

Open Handset Alliance. (n.d.). *What is Android?* http://www.android.com/about/.

Ortiz, C. E. (2009). *Obfuscating Your MIDlet Suite.* http://developers.sun.com/mobility/midp/ttips/midletsize/.

Pahlavan, K., & Levesque, A. H. (2005). *Wireless Information Networks* (Second Edition ed.). Hoboken, New Jersey, USA: John Wiley and Sons *Inc.*, Publication.

Park, N., & Song, Y. j. (2001). M-Commerce Security Platform based on WTLS and J2ME. In *Proceedings of IEEE International Symposium on Industrial Electronics (ISIE 2001).* Pusan,Korea (South): IEEE.

Parson, K., & Schaeffler, J. (2001). US Wireless Phone Penetration Climbs. *Wireless Insider*.

Peiro, J. L. A., Asokan, N., Steiner, M., & Waidner, M. (1998). Designing a Generic Payment Service. *IBM Systems Journal*, *37(1)*, 72-88.

Research In Motion Limited. (n.d.). *BlackBerry Overview.* BlackBerry.com.

Riggs, R., & Vandenbrink, M. (2001). *Programming for Wireless Devices with the Java 2 Platform, Micro Edition*. Santa Clara, California, USA: Addison-Wesley Longman Publishing Co., *Inc.*

Rittinghouse, J. W., & Ransome, J. F. (2004). *Wireless Operational Security*. Burlington, MA, USA: Digital Press.

Rivest, R., Robshaw, M., Sidney, R., & Yin, Y. L. (n.d.). *The RC6 Block Cipher.* RSA Laboratories.

Rivest, R., Shamir, A., & Adleman, L. (1978). A Method for Obtaining Digital Signatures and Public Key Cryptosystems. *Communications of the ACM*.

Roto, V., & Oulasvirta, A. (2005). Need for Non-Visual Feedback with Long Response Times in Mobile HCI. In *Special interest tracks and posters of the 14th international conference on World Wide Web* (p. 775 - 781). Chiba, Japan: ACM.

Roussos, G., Peterson, D., & Patel, U. (2003). Mobile Identity Management: An Enacted View. *International Journal of Electronic Commerce*, *8*(1), 81 - 100.

RSA Laboratories. (2000). *RSA Laboratories' Frequently Asked Questions About Today's Cryptography, Version 4.1*. Bedford, MA, USA: RSA Security *Inc.*

RSA Laboratories. (2002). *Public-Key Cryptography Standards (PKCS) # 1: RSA Cryptography Specifications Version 2.1*. Bedford, MA, USA: RSA Security *Inc.*

Sacco, A. (2008). *Mobile Payments: 71 Percent of Consumers Say "No Way" to Online Shopping, Banking via Mobile Devices.* CIO.com.

Sanchez-Avila, C., & Sanchez-Reillol, R. (2001). The Rijndael block cipher (AES proposal) : A Comparison with DES. In *IEEE 35th International Carnahan Conference on Security Technology* (p. 229-234). London, England: IEEE.

Sanders, G., Thorens, L., Reisky, M., Rulik, O., & Deylitz, S. (2003). *GPRS networks*. New York, NY, USA: John Wiley and Sons *Inc.*, Publication.

Schneider, F. B. (n.d.). *Something You Know, Have, or Are.* http://www.cs.cornell.edu/Courses/CS513/2005FA/NNLauthPeople.html.

Schneier, B. (1994). *Applied Cryptography*. New York, NY, USA: John Wiley and Sons *Inc.*, Publication.

Schneier, B., Kelsey, J., Whiting, D., Wagner, D., Hall, C., & Ferguson, N. (1998). *Twofish: A 128-Bit Block Cipher.* http://www.schneier.com/paper-twofish-paper.html.

Schnorr, C. (1991). Efficient Signatures for Smart Card. *Journal of Cryptology*.

Scourias, J. (2003). *Overview of the Global System for Mobile Communication.* http://ccnga.uwaterloo.ca/ jscouri/GSM/gsmreport.html.

Sony Ericsson. (n.d.). *Z800i Specifications.* http://www.sonyericsson.com.

Stallings, W. (2006). *Cryptography and Network Security: Principles and Practice* (Third Edition ed.). Upper Saddle River, New Jersey, USA: Pearson Prentice Hall.

Stinson, D. R. (2002). *Cryptography: Theory and Practice* (Second Edition ed.). Boca Raton, FL, USA: CRC Press.

Sun Microsystems. (n.d.-a). *Introduction to the Java ME Platform.* Sun Developer Network. http://java.sun.com/javame/technology/index.jsp.

Sun Microsystems. (n.d.-b). *JSR 177: Security and Trust Services API for J2ME.* Sun Developer Network.

Sun Microsystems. (n.d.-c). *Mobile Information Device Profile (MIDP): JSR 37, JSR 118 Overview.* Sun Developer Network.

Sun Microsystems. (2000a). *Connected Limited Device Configuration, Version 1.0a.* Sun Developer Network.

Sun Microsystems. (2000b). *Mobile Information Device Profile, Version 1.0a.* Sun Developer Network.

Sun Microsystems. (2003). *Connected Limited Device Configuration, Version 1.1.* Sun Developer Network.

Sun Microsystems. (2009). *Sun Java Wireless Toolkit 2.5.2_01 for CLDC Download.* Sun Developer Network.

Symbian Foundation. (2009). *About the Symbian Foundation.* http://www.symbian.org/index.php.

Thanh, D. (2000). Security Issues in Mobile Commerce. In *Proceedings of the 1st International Conference on Electronic Conference and Web Technologies (EC-Web 2000)* (p. 412 - 425). London, UK: Springer.

The Open Mobile Alliance Ltd. (n.d.). *Wireless Application Protocol.* http://www.openmobilealliance.org/tech/affiliates/wap/wapindex.html.

The United States Federal Government. (2006). *National Information Assurance (IA) Glossary* (Tech. Rep.). USA: Author.

Tillich, S., & Großschadl, J. (2004). A Survey of Public-Key Cryptography on J2ME-Enabled Mobile Devices. *Lecture Notes in Computer Science*, *3280/2004*, 935-944.

Tipton, H. F., & Krause, M. (2003). *Information Security Management Handbook*. Boca Raton, FL, USA: CRC Press.

Tipton, H. F., & Krause, M. (2007). *Information Security Management Handbook* (6 ed.). Boca Raton, FL, USA: CRC Press.

Tsalgatidou, A., & Veijalainen, J. (2000). Mobile Electronic Commerce: Emerging Issues. In *Proceedings of 1st International Conference on E-Commerce and Web Technologies (EC-WEB 2000).* London, Greenwich, UK: Springer.

Vanstone, S. (1992). Responses to NISTs Proposal. *Communications of the ACM*, *35*, 50-52.

Varshney, U., & Vetter, R. (2000). Emerging Mobile and Wireless Networks. *Communications of the ACM*, *43*, 73 - 81.

Vihinen, J. (2004). Identifying the Limitations and Capabilities of M-commerce Services in GSM Networks. *International Journal of Mobile Communications*, *2*(4), 329 - 342.

VISA and MasterCard. (1997). *SET Secure Electronic Transaction Specification.* www.setco.org.

Vyas, A., & O'Grady, P. (2001). *A Review of Mobile Commerce Technologies* (Tech. Rep.). Iowa City, IA, USA: Department of Industrial Engineering, University of Iowa.

WAP Forum. (2002). *What is WAP?* http://www.wapforum.org/what/index.htm.

Wei, J., C.Liu, L., & Koong, K. S. (2006). An Onion Ring Framework for Developing and Assessing Mobile Commerce Security. *International Journal of Mobile Communications*, *4*(2), 128 - 142.

Wilcox, N. (2005). Whats Next in Mobile Telephony and Will It Succeed? *Telektronikk*, *3/4*, 85-95.

Yang, B., Zheng, P., & Ni, L. M. (2007). *Professional Microsoft Smartphone Programming*. New York, NY, USA: John Wiley and Sons *Inc.*, Publication.

Yi, W., Reddy, C., & Ang, G. (2002). J2ME Devices: Real-World Performance. *JavaWorld.com*.

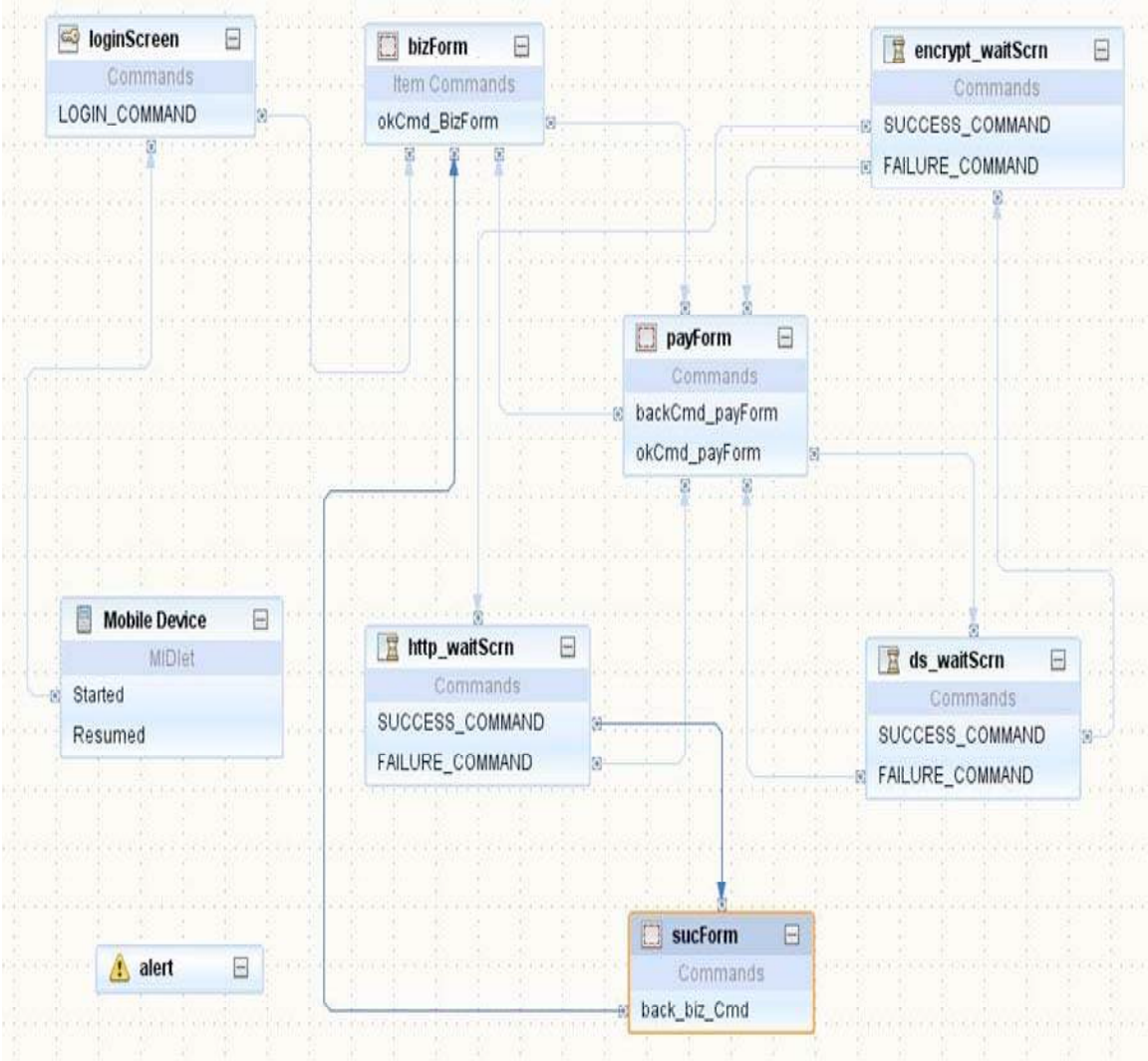
Zheng, J., Shao, Z., Huang, S., & Yu, T. (2008). Security of Two Signature Schemes Based on Two Hard Problems. In *11th IEEE International Conference on Communication Technology (ICCT 2008)* (p. 745 - 748). Hangzhou, China: IEEE.

Zwass, V. (1996). Electronic Commerce: Structures and Issues. *International Journal of Electronic Commerce*, *1*(1), 3 - 23.

Appendix

Part of Client Simulation Code

MoBankMIDlet Operating Flow



MoBankMIDlet.java

```
1  /*
2   * MobSurveyMIDlet.java
3   *
4   * Created on 15-Nov-2007, 10:37:54 AM
5   *
6   * To change this template, choose Tools | Templates
7   * and open the template in the editor.
8   */
9  package com.uleth.mobank;
10
11 import com.uleth.mobank.connection.NetConnection;
12 import java.io.IOException;
13
14 import javax.microedition.midlet.*;
15 import javax.microedition.lcdui.*;
16 import javax.microedition.lcdui.Display;
17 import org.netbeans.microedition.lcdui.LoginScreen;
18 import org.netbeans.microedition.lcdui.WaitScreen;
19 import org.netbeans.microedition.util.SimpleCancellableTask;
20
21 /**
22  * @author zhuyp
23  */
```

```
24 public class MoBankMIDlet extends MIDlet implements
       CommandListener , ItemCommandListener {
25
26     private boolean midletPaused = false;
27     //<editor-fold defaultstate="collapsed" desc=" Generated
           Fields ">//GEN-BEGIN:|fields|0|
28     private Alert alert ;
29     private LoginScreen loginScreen ;
30     private WaitScreen ds_waitScrn ;
31     private WaitScreen http_waitScrn ;
32     private WaitScreen encrypt_waitScrn ;
33     private Form loginForm ;
34     private TextField textAccound ;
35     private TextField textPassword ;
36     private Form payForm ;
37     private TextField payAmount ;
38     private DateField dateFieldPayment ;
39     private ChoiceGroup groupPayto ;
40     private ChoiceGroup groupPayFrom ;
41     private Form bizForm ;
42     private ImageItem imageItem ;
43     private ImageItem imageItem1 ;
44     private ImageItem imageItem2 ;
45     private Form sucForm ;
46     private ImageItem imageItem_Success ;
47     private Command exitCommand ;
48     private Command cancelCommand ;
```

```
49      private Command okCmdLogin   ;
50      private Command okCmd_payForm   ;
51      private Command backCmd_payForm   ;
52      private Command okCmd_BizForm   ;
53      private Command back_biz_Cmd   ;
54      private Image image1  ;
55      private Image image2  ;
56      private Ticker ticker  ;
57      private Font font ;
58      private Image image4  ;
59      private Image image5  ;
60      private Image image3  ;
61      private SimpleCancellableTask task    ;
62      private SimpleCancellableTask task1    ;
63      private SimpleCancellableTask task2    ;
64      //</editor-fold>//GEN-END:|fields|0|
65      private String answer   = "S,"; //the answer submitted
66      private String alert_notes   ; //the string for alert some
            error
67      private static final String CHANNEL1    = "WEB";
68      private static final String CHANNEL2    = "HANDHELD ";
69      NetConnection nc    = new NetConnection  ();
70
71      /**
72       * @return Returns the alert_notes.
73       */
74      public String getAlert_notes   () {
```

```
75          return alert_notes  ;
76      }
77
78      /**
79       * @param alert_notes The alert_notes to set.
80       */
81      public void setAlert_notes  (String alert_notes  ) {
82          this.alert_notes  = alert_notes  ;
83      }
84
85      /**
86       * get the String of the answer
87       */
88      private String getAnswer  (String title  , String subanswer  ) {
89          answer  = answer  + title  + "_" + subanswer  + ",";
90          System .out .println ("answer :" + answer );
91          return answer ;
92      }
93
94      /**
95       *  delay some time
96       */
97      private void TimeDelay  () {
98          for (int i = 0; i < 5000; i++) {
99              System .out .println ("I love this game ");
100         }
101     }
```

```
102
103     /**
104      * HTTP SUBMIT
105      */
106     private void httpSubmit () {
107
108         httpSubmitThread hst = new httpSubmitThread ();
109         hst.start();
110     }
111
112     /**
113      * Thread in charge of http submit
114      */
115     private class httpSubmitThread extends Thread {
116
117         public void run() {
118             try {
119
120                 // http connecting to server with post method
121                 nc.postViaHttpConnection ();
122
123             } catch (IOException e ) {
124                 e.printStackTrace ();
125             }
126         }
127     }
128
```

```
129     /**
130      * The MoBankMIDlet constructor.
131      */
132     public MoBankMIDlet () {
133     }
134     //<editor-fold defaultstate="collapsed" desc=" Generated
            Methods ">//GEN-BEGIN:|methods|0|
135     //</editor-fold>//GEN-END:|methods|0|
136
137     //<editor-fold defaultstate="collapsed" desc=" Generated
            Method: initialize ">//GEN-BEGIN:|0-initialize|0|0-
            preInitialize
138     /**
139      * Initilizes the application.
140      * It is called only once when the MIDlet is started. The
            method is called before the <code>startMIDlet</code>
            method.
141      */
142     private void initialize () {//GEN-END:|0-initialize|0|0-
            preInitialize
143     // write pre-initialize user code here
144 //GEN-LINE:|0-initialize|1|0-postInitialize
145     // write post-initialize user code here
146     }//GEN-BEGIN:|0-initialize|2|
147     //</editor-fold>//GEN-END:|0-initialize|2|
148
```

```
149     //<editor-fold defaultstate="collapsed" desc=" Generated
            Method: startMIDlet ">//GEN-BEGIN:|3-startMIDlet|0|3-
            preAction
150     /**
151      * Performs an action assigned to the Mobile Device - MIDlet
             Started point.
152      */
153     public void startMIDlet () {//GEN-END:|3-startMIDlet|0|3-
            preAction
154         // write pre-action user code here
155         switchDisplayable (null, getLoginScreen ());//GEN-LINE:|3-
                startMIDlet|1|3-postAction
156     // write post-action user code here
157     }//GEN-BEGIN:|3-startMIDlet|2|
158     //</editor-fold>//GEN-END:|3-startMIDlet|2|
159
160     //<editor-fold defaultstate="collapsed" desc=" Generated
            Method: resumeMIDlet ">//GEN-BEGIN:|4-resumeMIDlet|0|4-
            preAction
161     /**
162      * Performs an action assigned to the Mobile Device - MIDlet
             Resumed point.
163      */
164     public void resumeMIDlet () {//GEN-END:|4-resumeMIDlet|0|4-
            preAction
165     // write pre-action user code here
166 //GEN-LINE:|4-resumeMIDlet|1|4-postAction
```

```
167         // write post-action user code here
168     }//GEN-BEGIN:|4-resumeMIDlet|2|
169     //</editor-fold>//GEN-END:|4-resumeMIDlet|2|
170
171     //<editor-fold defaultstate="collapsed" desc=" Generated
            Method: switchDisplayable ">//GEN-BEGIN:|5-
            switchDisplayable|0|5-preSwitch
172     /**
173      * Switches a current displayable in a display. The <code>
             display</code> instance is taken from <code>getDisplay</
             code> method. This method is used by all actions in the
             design for switching displayable.
174      * @param alert the Alert which is temporarily set to the
             display; if <code>null</code>, then <code>
             nextDisplayable</code> is set immediately
175      * @param nextDisplayable the Displayable to be set
176      */
177     public void switchDisplayable (Alert alert, Displayable
            nextDisplayable) {//GEN-END:|5-switchDisplayable|0|5-
            preSwitch
178         // write pre-switch user code here
179
180         Display display = getDisplay();//GEN-BEGIN:|5-
                switchDisplayable|1|5-postSwitch
181         if (alert == null) {
182             display.setCurrent(nextDisplayable);
183         } else {
```

```
184             display .setCurrent (alert , nextDisplayable );
185         }//GEN-END:|5-switchDisplayable|1|5-postSwitch
186         System .out .println ("Display to:" + display .getCurrent ().
                getTitle ());
187     // write post-switch user code here
188     }//GEN-BEGIN:|5-switchDisplayable|2|
189     //</editor-fold >//GEN-END:|5-switchDisplayable|2|
190     // write pre-action user code here
191
192     // write pre-action user code here
193     // write pre-action user code here
194     //<editor-fold defaultstate="collapsed" desc=" Generated
            Getter: ticker ">//GEN-BEGIN:|14-getter|0|14-preInit
195     /**
196      * Returns an initiliazed instance of ticker component.
197      * @return the initialized component instance
198      */
199     public Ticker getTicker () {
200         if (ticker == null) {//GEN-END:|14-getter|0|14-preInit
201             // write pre-init user code here
202             ticker = new Ticker ("Welcome to MoBank ");//GEN-LINE
                    :|14-getter|1|14-postInit
203         // write post-init user code here
204         }//GEN-BEGIN:|14-getter|2|
205         return ticker ;
206     }
207     //</editor-fold >//GEN-END:|14-getter|2|
```

```
208     //<editor-fold defaultstate="collapsed" desc=" Generated
            Getter: exitCommand ">//GEN-BEGIN:|66-getter|0|66-preInit
209     /**
210      * Returns an initiliazed instance of exitCommand component.
211      * @return the initialized component instance
212      */
213     public Command getExitCommand() {
214         if (exitCommand == null) {//GEN-END:|66-getter|0|66-
                preInit
215             // write pre-init user code here
216             exitCommand = new Command("Exit", Command.EXIT, 0);
                //GEN-LINE:|66-getter|1|66-postInit
217         // write post-init user code here
218         }//GEN-BEGIN:|66-getter|2|
219         return exitCommand;
220     }
221     //</editor-fold>//GEN-END:|66-getter|2|
222
223     //<editor-fold defaultstate="collapsed" desc=" Generated
            Getter: loginForm ">//GEN-BEGIN:|68-getter|0|68-preInit
224     /**
225      * Returns an initiliazed instance of loginForm component.
226      * @return the initialized component instance
227      */
228     public Form getLoginForm() {
229         if (loginForm == null) {//GEN-END:|68-getter|0|68-
                preInit
```

```
230             // write pre-init user code here
231             loginForm = new Form("Bank Login", new Item[] { getTextAccound(), getTextPassword() });//GEN-BEGIN:|68-getter|1|68-postInit
232             loginForm.setTicker(getTicker());
233             loginForm.addCommand(getExitCommand());
234             loginForm.addCommand(getOkCmdLogin());
235             loginForm.addCommand(getCancelCommand());
236             loginForm.setCommandListener(this);//GEN-END:|68-getter|1|68-postInit
237         // write post-init user code here
238         }//GEN-BEGIN:|68-getter|2|
239         return loginForm;
240     }
241     //</editor-fold>//GEN-END:|68-getter|2|
242
243     //<editor-fold defaultstate="collapsed" desc=" Generated Getter: textAccound ">//GEN-BEGIN:|69-getter|0|69-preInit
244     /**
245      * Returns an initiliazed instance of textAccound component.
246      * @return the initialized component instance
247      */
248     public TextField getTextAccound() {
249         if (textAccound == null) {//GEN-END:|69-getter|0|69-preInit
250             // write pre-init user code here
```

```
251             textAccound = new TextField ("MoBank Account ID",
                    null, 32, TextField.ANY);//GEN-BEGIN:|69-getter
                    |1|69-postInit
252             textAccound.setLayout (ImageItem.LAYOUT_CENTER | Item
                    .LAYOUT_TOP | Item.LAYOUT_BOTTOM | Item.
                    LAYOUT_VCENTER);//GEN-END:|69-getter|1|69-
                    postInit
253         // write post-init user code here
254         }//GEN-BEGIN:|69-getter|2|
255         return textAccound;
256     }
257     //</editor-fold>//GEN-END:|69-getter|2|
258
259     //<editor-fold defaultstate="collapsed" desc=" Generated
            Getter: textPassword ">//GEN-BEGIN:|70-getter|0|70-
            preInit
260     /**
261      * Returns an initiliazed instance of textPassword component
             .
262      * @return the initialized component instance
263      */
264     public TextField getTextPassword() {
265         if (textPassword == null) {//GEN-END:|70-getter|0|70-
                preInit
266             // write pre-init user code here
267             textPassword = new TextField ("MoBank Password ", null
                    , 32, TextField.ANY | TextField.PASSWORD);//GEN-
```

```
                BEGIN:|70-getter|1|70-postInit
268             textPassword.setLayout(ImageItem.LAYOUT_DEFAULT);//
                GEN-END:|70-getter|1|70-postInit
269         // write post-init user code here
270         }//GEN-BEGIN:|70-getter|2|
271         return textPassword;
272     }
273     //</editor-fold>//GEN-END:|70-getter|2|
274
275     //<editor-fold defaultstate="collapsed" desc=" Generated
            Method: commandAction for Displayables ">//GEN-BEGIN:|7-
            commandAction|0|7-preCommandAction
276     /**
277      * Called by a system to indicated that a command has been
             invoked on a particular displayable.
278      * @param command the Command that was invoked
279      * @param displayable the Displayable where the command was
             invoked
280      */
281     public void commandAction(Command command, Displayable
            displayable) {//GEN-END:|7-commandAction|0|7-
            preCommandAction
282         // write pre-action user code here
283         if (displayable == ds_waitScrn) {//GEN-BEGIN:|7-
                commandAction|1|161-preAction
284             if (command == WaitScreen.FAILURE_COMMAND) {//GEN-
                    END:|7-commandAction|1|161-preAction
```

```
285                 // write pre-action user code here
286                 switchDisplayable (null, getPayForm ());//GEN-LINE
                        :|7-commandAction|2|161-postAction
287             // write post-action user code here
288             } else if (command == WaitScreen.SUCCESS_COMMAND) {
                    //GEN-LINE:|7-commandAction|3|160-preAction
289                 // write pre-action user code here
290                 switchDisplayable (null, getEncrypt_waitScrn());
                        //GEN-LINE:|7-commandAction|4|160-postAction
291             // write post-action user code here
292             }//GEN-BEGIN:|7-commandAction|5|165-preAction
293         } else if (displayable == encrypt_waitScrn) {
294             if (command == WaitScreen.FAILURE_COMMAND) {//GEN-
                    END:|7-commandAction|5|165-preAction
295                 // write pre-action user code here
296                 switchDisplayable (null, getPayForm ());//GEN-LINE
                        :|7-commandAction|6|165-postAction
297             // write post-action user code here
298             } else if (command == WaitScreen.SUCCESS_COMMAND) {
                    //GEN-LINE:|7-commandAction|7|164-preAction
299                 // write pre-action user code here
300                 switchDisplayable (null, getHttp_waitScrn());//
                        GEN-LINE:|7-commandAction|8|164-postAction
301             // write post-action user code here
302             }//GEN-BEGIN:|7-commandAction|9|150-preAction
303         } else if (displayable == http_waitScrn) {
```

```
304             if (command == WaitScreen.FAILURE_COMMAND) {//GEN-
                    END:|7-commandAction|9|150-preAction
305                 // write pre-action user code here
306                 switchDisplayable(null, getPayForm());//GEN-LINE
                        :|7-commandAction|10|150-postAction
307             // write post-action user code here
308             } else if (command == WaitScreen.SUCCESS_COMMAND) {
                    //GEN-LINE:|7-commandAction|11|149-preAction
309                 // write pre-action user code here
310                 switchDisplayable(null, getSucForm());//GEN-LINE
                        :|7-commandAction|12|149-postAction
311             // write post-action user code here
312             }//GEN-BEGIN:|7-commandAction|13|77-preAction
313         } else if (displayable == loginForm) {
314             if (command == cancelCommand) {//GEN-END:|7-
                    commandAction|13|77-preAction
315             // write pre-action user code here
316
317 //GEN-LINE:|7-commandAction|14|77-postAction
318             // write post-action user code here
319             } else if (command == exitCommand) {//GEN-LINE:|7-
                    commandAction|15|73-preAction
320             // write pre-action user code here
321 //GEN-LINE:|7-commandAction|16|73-postAction
322             // write post-action user code here
323             } else if (command == okCmdLogin) {//GEN-LINE:|7-
                    commandAction|17|75-preAction
```

```
324             // write pre-action user code here
325 //GEN-LINE:|7-commandAction|18|75-postAction
326             // write post-action user code here
327             }//GEN-BEGIN:|7-commandAction|19|135-preAction
328         } else if (displayable == loginScreen ) {
329             if (command == LoginScreen.LOGIN_COMMAND ) {//GEN-END:|7-commandAction|19|135-preAction
330                 // write pre-action user code here
331                 switchDisplayable (null, getBizForm ());//GEN-LINE:|7-commandAction|20|135-postAction
332             // write post-action user code here
333             }//GEN-BEGIN:|7-commandAction|21|84-preAction
334         } else if (displayable == payForm ) {
335             if (command == backCmd_payForm ) {//GEN-END:|7-commandAction|21|84-preAction
336                 // write pre-action user code here
337                 switchDisplayable (null, getBizForm ());//GEN-LINE:|7-commandAction|22|84-postAction
338             // write post-action user code here
339             } else if (command == okCmd_payForm ) {//GEN-LINE:|7-commandAction|23|86-preAction
340                 // write pre-action user code here
341                 switchDisplayable (null, getDs_waitScrn ());//GEN-LINE:|7-commandAction|24|86-postAction
342             // write post-action user code here
343             }//GEN-BEGIN:|7-commandAction|25|117-preAction
344         } else if (displayable == sucForm ) {
```

```
345             if (command == back_biz_Cmd ) {//GEN-END:|7-
                    commandAction|25|117-preAction
346                 // write pre-action user code here
347                 switchDisplayable (null, getBizForm ());//GEN-LINE
                        :|7-commandAction|26|117-postAction
348             // write post-action user code here
349             }//GEN-BEGIN:|7-commandAction|27|7-postCommandAction
350         }//GEN-END:|7-commandAction|27|7-postCommandAction
351     // write post-action user code here
352     }//GEN-BEGIN:|7-commandAction|28|
353     //</editor-fold>//GEN-END:|7-commandAction|28|
354     //<editor-fold defaultstate="collapsed" desc=" Generated
            Getter: okCmdLogin ">//GEN-BEGIN:|74-getter|0|74-preInit
355     /**
356      * Returns an initiliazed instance of okCmdLogin component.
357      * @return the initialized component instance
358      */
359     public Command getOkCmdLogin () {
360         if (okCmdLogin == null) {//GEN-END:|74-getter|0|74-
                preInit
361             // write pre-init user code here
362             okCmdLogin = new Command ("Ok", Command.OK, 0);//GEN-
                    LINE:|74-getter|1|74-postInit
363         // write post-init user code here
364         }//GEN-BEGIN:|74-getter|2|
365         return okCmdLogin;
366     }
```

```
367     //</editor-fold>//GEN-END:|74-getter|2|
368
369     //<editor-fold defaultstate="collapsed" desc=" Generated
            Getter: cancelCommand ">//GEN-BEGIN:|76-getter|0|76-
            preInit
370     /**
371      * Returns an initiliazed instance of cancelCommand
            component.
372      * @return the initialized component instance
373      */
374     public Command getCancelCommand() {
375         if (cancelCommand == null) {//GEN-END:|76-getter|0|76-
                preInit
376             // write pre-init user code here
377             cancelCommand = new Command("Cancel", Command.CANCEL
                    , 0);//GEN-LINE:|76-getter|1|76-postInit
378         // write post-init user code here
379         }//GEN-BEGIN:|76-getter|2|
380         return cancelCommand;
381     }
382     //</editor-fold>//GEN-END:|76-getter|2|
383     //<editor-fold defaultstate="collapsed" desc=" Generated
            Getter: payForm ">//GEN-BEGIN:|82-getter|0|82-preInit
384     /**
385      * Returns an initiliazed instance of payForm component.
386      * @return the initialized component instance
387      */
```

```
388     public Form getPayForm() {
389         if (payForm == null) {//GEN-END:|82-getter|0|82-preInit
390             // write pre-init user code here
391             payForm = new Form("Money Transfer", new Item[] { ›
                    getGroupPayto(), getGroupPayFrom(), getPayAmount ›
                    (), getDateFieldPayment() });//GEN-BEGIN:|82- ›
                    getter|1|82-postInit
392             payForm.addCommand(getBackCmd_payForm());
393             payForm.addCommand(getOkCmd_payForm());
394             payForm.setCommandListener(this);//GEN-END:|82- ›
                    getter|1|82-postInit
395         // write post-init user code here
396         }//GEN-BEGIN:|82-getter|2|
397         return payForm;
398     }
399     //</editor-fold>//GEN-END:|82-getter|2|
400
401     //<editor-fold defaultstate="collapsed" desc=" Generated ›
        Getter: groupPayto ">//GEN-BEGIN:|88-getter|0|88-preInit
402     /**
403      * Returns an initiliazed instance of groupPayto component.
404      * @return the initialized component instance
405      */
406     public ChoiceGroup getGroupPayto() {
407         if (groupPayto == null) {//GEN-END:|88-getter|0|88- ›
                preInit
408             // write pre-init user code here
```

```
409             groupPayto  = new ChoiceGroup ("Pay To", Choice.POPUP);//GEN-BEGIN:|88-getter|1|88-postInit
410             groupPayto.append("VISA123456789 ", null);
411             groupPayto.append("PHONEBILL123 ", null);
412             groupPayto.setSelectedFlags(new boolean[] { false, false });
413             groupPayto.setFont(0, getFont());
414             groupPayto.setFont(1, getFont());//GEN-END:|88-getter|1|88-postInit
415         // write post-init user code here
416         }//GEN-BEGIN:|88-getter|2|
417         return groupPayto;
418     }
419     //</editor-fold>//GEN-END:|88-getter|2|
420
421     //<editor-fold defaultstate="collapsed" desc=" Generated Getter: groupPayFrom ">//GEN-BEGIN:|91-getter|0|91-preInit
422     /**
423      * Returns an initiliazed instance of groupPayFrom component.
424      * @return the initialized component instance
425      */
426     public ChoiceGroup getGroupPayFrom() {
427         if (groupPayFrom == null) {//GEN-END:|91-getter|0|91-preInit
428             // write pre-init user code here
```

```
429             groupPayFrom   = new ChoiceGroup ("Pay From", Choice.
                    EXCLUSIVE );//GEN-BEGIN:|91-getter|1|91-postInit
430             groupPayFrom.append("Account2831 ", null);
431             groupPayFrom.setSelectedFlags (new boolean[] { false
                    });
432             groupPayFrom.setFont(0, getFont());//GEN-END:|91-
                    getter|1|91-postInit
433         // write post-init user code here
434         }//GEN-BEGIN:|91-getter|2|
435         return groupPayFrom;
436     }
437     //</editor-fold>//GEN-END:|91-getter|2|
438
439     //<editor-fold defaultstate="collapsed" desc=" Generated
            Getter: dateFieldPayment ">//GEN-BEGIN:|93-getter|0|93-
            preInit
440     /**
441      * Returns an initiliazed instance of dateFieldPayment
             component.
442      * @return the initialized component instance
443      */
444     public DateField getDateFieldPayment() {
445         if (dateFieldPayment == null) {//GEN-END:|93-getter
                |0|93-preInit
446             // write pre-init user code here
447             dateFieldPayment = new DateField("Payment Date",
                    DateField.DATE, java.util.TimeZone.getTimeZone("
```

```
                America/Edmonton"));//GEN-BEGIN:|93-getter|1|93-postInit
448             dateFieldPayment.setDate(new java.util.Date(System.currentTimeMillis()));//GEN-END:|93-getter|1|93-postInit
449         // write post-init user code here
450         }//GEN-BEGIN:|93-getter|2|
451         return dateFieldPayment;
452     }
453     //</editor-fold>//GEN-END:|93-getter|2|
454
455     //<editor-fold defaultstate="collapsed" desc="Generated Getter: backCmd_payForm">//GEN-BEGIN:|83-getter|0|83-preInit
456     /**
457      * Returns an initiliazed instance of backCmd_payForm component.
458      * @return the initialized component instance
459      */
460     public Command getBackCmd_payForm() {
461         if (backCmd_payForm == null) {//GEN-END:|83-getter|0|83-preInit
462             // write pre-init user code here
463             backCmd_payForm = new Command("Exit", Command.BACK, 0);//GEN-LINE:|83-getter|1|83-postInit
464         // write post-init user code here
465         }//GEN-BEGIN:|83-getter|2|
```

```
466         return backCmd_payForm  ;
467     }
468     //</editor-fold>//GEN-END:|83-getter|2|
469
470     //<editor-fold defaultstate="collapsed" desc=" Generated
        Getter: okCmd_payForm ">//GEN-BEGIN:|85-getter|0|85-
        preInit
471     /**
472      * Returns an initiliazed instance of okCmd_payForm
         component.
473      * @return the initialized component instance
474      */
475     public Command getOkCmd_payForm   () {
476         if (okCmd_payForm   == null) {//GEN-END:|85-getter|0|85-
            preInit
477             // write pre-init user code here
478             okCmd_payForm   = new Command ("Ok", Command .OK, 0);//
                GEN-LINE:|85-getter|1|85-postInit
479         // write post-init user code here
480         }//GEN-BEGIN:|85-getter|2|
481         return okCmd_payForm  ;
482     }
483     //</editor-fold>//GEN-END:|85-getter|2|
484
485     //<editor-fold defaultstate="collapsed" desc=" Generated
        Getter: bizForm ">//GEN-BEGIN:|94-getter|0|94-preInit
486     /**
```

```
487         * Returns an initiliazed instance of bizForm component.
488         * @return the initialized component instance
489         */
490        public Form getBizForm() {
491            if (bizForm == null) {//GEN-END:|94-getter|0|94-preInit
492                // write pre-init user code here
493                bizForm = new Form("Business List", new Item[] {
                       getImageItem(), getImageItem1(), getImageItem2()
                       });//GEN-BEGIN:|94-getter|1|94-postInit
494                bizForm.setCommandListener(this);//GEN-END:|94-
                       getter|1|94-postInit
495            // write post-init user code here
496            }//GEN-BEGIN:|94-getter|2|
497            return bizForm;
498        }
499        //</editor-fold>//GEN-END:|94-getter|2|
500        //<editor-fold defaultstate="collapsed" desc=" Generated
               Getter: okCmd_BizForm ">//GEN-BEGIN:|101-getter|0|101-
               preInit
501        /**
502         * Returns an initiliazed instance of okCmd_BizForm
                component.
503         * @return the initialized component instance
504         */
505        public Command getOkCmd_BizForm() {
506            if (okCmd_BizForm == null) {//GEN-END:|101-getter|0|101-
                   preInit
```

```
507             // write pre-init user code here
508             okCmd_BizForm = new Command("Ok", Command.OK, 0);// GEN-LINE:|101-getter|1|101-postInit
509         // write post-init user code here
510         }//GEN-BEGIN:|101-getter|2|
511         return okCmd_BizForm;
512     }
513     //</editor-fold>//GEN-END:|101-getter|2|
514
515     //<editor-fold defaultstate="collapsed" desc=" Generated Getter: sucForm ">//GEN-BEGIN:|104-getter|0|104-preInit
516     /**
517      * Returns an initiliazed instance of sucForm component.
518      * @return the initialized component instance
519      */
520     public Form getSucForm() {
521         if (sucForm == null) {//GEN-END:|104-getter|0|104-preInit
522             // write pre-init user code here
523             sucForm = new Form("Transaction Success", new Item[] { getImageItem_Success() });//GEN-BEGIN:|104-getter|1|104-postInit
524             sucForm.addCommand(getBack_biz_Cmd());
525             sucForm.setCommandListener(this);//GEN-END:|104-getter|1|104-postInit
526         // write post-init user code here
527         }//GEN-BEGIN:|104-getter|2|
```

```
528         return sucForm ;
529     }
530     //</editor-fold>//GEN-END:|104-getter|2|
531     //<editor-fold defaultstate="collapsed" desc=" Generated
            Getter: back_biz_Cmd ">//GEN-BEGIN:|116-getter|0|116-
            preInit
532     /**
533      * Returns an initiliazed instance of back_biz_Cmd component
             .
534      * @return the initialized component instance
535      */
536     public Command getBack_biz_Cmd    () {
537         if (back_biz_Cmd   == null) {//GEN-END:|116-getter|0|116-
                preInit
538             // write pre-init user code here
539             back_biz_Cmd   = new Command ("Back ", Command .BACK , 0);
                    //GEN-LINE:|116-getter|1|116-postInit
540         // write post-init user code here
541         }//GEN-BEGIN:|116-getter|2|
542         return back_biz_Cmd  ;
543     }
544     //</editor-fold>//GEN-END:|116-getter|2|
545     //<editor-fold defaultstate="collapsed" desc=" Generated
            Getter: alert ">//GEN-BEGIN:|131-getter|0|131-preInit
546     /**
547      * Returns an initiliazed instance of alert component.
548      * @return the initialized component instance
```

```
549         */
550         public Alert getAlert() {
551             if (alert == null) {//GEN-END:|131-getter|0|131-preInit
552                 // write pre-init user code here
553                 alert = new Alert("alert", "The Transaction is failed! Please consult with your admin!", getImage4(), null);//GEN-BEGIN:|131-getter|1|131-postInit
554                 alert.setTimeout(Alert.FOREVER);//GEN-END:|131-getter|1|131-postInit
555             // write post-init user code here
556             }//GEN-BEGIN:|131-getter|2|
557             return alert;
558         }
559         //</editor-fold>//GEN-END:|131-getter|2|
560 
561         //<editor-fold defaultstate="collapsed" desc=" Generated Getter: loginScreen ">//GEN-BEGIN:|133-getter|0|133-preInit
562         /**
563          * Returns an initiliazed instance of loginScreen component.
564          * @return the initialized component instance
565          */
566         public LoginScreen getLoginScreen() {
567             if (loginScreen == null) {//GEN-END:|133-getter|0|133-preInit
568                 // write pre-init user code here
```

```
569             loginScreen = new LoginScreen(getDisplay());//GEN-BEGIN:|133-getter|1|133-postInit
570             loginScreen.setLabelTexts("MoBank ID", "Password ");
571             loginScreen.setTitle("Login MoBank ");
572             loginScreen.setTicker(getTicker());
573             loginScreen.addCommand(LoginScreen.LOGIN_COMMAND);
574             loginScreen.setCommandListener(this);
575             loginScreen.setBGColor(-3355444);
576             loginScreen.setFGColor(0);
577             loginScreen.setUseLoginButton(true);
578             loginScreen.setLoginButtonText("Login ");//GEN-END:|133-getter|1|133-postInit
579         // write post-init user code here
580         }//GEN-BEGIN:|133-getter|2|
581         return loginScreen;
582     }
583     //</editor-fold>//GEN-END:|133-getter|2|
584     //<editor-fold defaultstate="collapsed" desc=" Generated Getter: imageItem ">//GEN-BEGIN:|138-getter|0|138-preInit
585     /**
586      * Returns an initiliazed instance of imageItem component.
587      * @return the initialized component instance
588      */
589     public ImageItem getImageItem() {
590         if (imageItem == null) {//GEN-END:|138-getter|0|138-preInit
591             // write pre-init user code here
```

```
592             imageItem = new ImageItem("Statement", getImage1(),
                    ImageItem.LAYOUT_DEFAULT, "Balance Statement");//
                GEN-LINE:|138-getter|1|138-postInit
593         // write post-init user code here
594         }//GEN-BEGIN:|138-getter|2|
595         return imageItem;
596     }
597     //</editor-fold>//GEN-END:|138-getter|2|
598
599     //<editor-fold defaultstate="collapsed" desc="Generated
        Getter: image1">//GEN-BEGIN:|139-getter|0|139-preInit
600     /**
601      * Returns an initiliazed instance of image1 component.
602      * @return the initialized component instance
603      */
604     public Image getImage1() {
605         if (image1 == null) {//GEN-END:|139-getter|0|139-preInit
606             // write pre-init user code here
607             try {//GEN-BEGIN:|139-getter|1|139-@java.io.
                IOException
608                 image1 = Image.createImage("/bs.png");
609             } catch (java.io.IOException e) {//GEN-END:|139-
                getter|1|139-@java.io.IOException
610                 e.printStackTrace();
611             }//GEN-LINE:|139-getter|2|139-postInit
612         // write post-init user code here
613         }//GEN-BEGIN:|139-getter|3|
```

```
614         return image1 ;
615     }
616     //</editor-fold>//GEN-END:|139-getter|3|
617
618     //<editor-fold defaultstate="collapsed" desc=" Generated
            Getter: imageItem1 ">//GEN-BEGIN:|141-getter|0|141-
            preInit
619     /**
620      * Returns an initiliazed instance of imageItem1 component.
621      * @return the initialized component instance
622      */
623     public ImageItem getImageItem1() {
624         if (imageItem1 == null) {//GEN-END:|141-getter|0|141-
                preInit
625             // write pre-init user code here
626             imageItem1 = new ImageItem("Transfer ", getImage2(),
                    ImageItem.LAYOUT_DEFAULT, "Money Transfer ", Item.
                    PLAIN );//GEN-BEGIN:|141-getter|1|141-postInit
627             imageItem1.addCommand(getOkCmd_BizForm());
628             imageItem1.setItemCommandListener(this);
629             imageItem1.setDefaultCommand(getOkCmd_BizForm());//
                    GEN-END:|141-getter|1|141-postInit
630         // write post-init user code here
631         }//GEN-BEGIN:|141-getter|2|
632         return imageItem1 ;
633     }
634     //</editor-fold>//GEN-END:|141-getter|2|
```

```
635
636     //<editor-fold defaultstate="collapsed" desc=" Generated
            Getter: imageItem2 ">//GEN-BEGIN:|143-getter|0|143-
            preInit
637     /**
638      * Returns an initiliazed instance of imageItem2 component.
639      * @return the initialized component instance
640      */
641     public ImageItem getImageItem2() {
642         if (imageItem2 == null) {//GEN-END:|143-getter|0|143-
                preInit
643             // write pre-init user code here
644             imageItem2 = new ImageItem("Payment ", getImage3(),
                    ImageItem.LAYOUT_DEFAULT, "<Missing Image >");//
                    GEN-LINE:|143-getter|1|143-postInit
645         // write post-init user code here
646         }//GEN-BEGIN:|143-getter|2|
647         return imageItem2;
648     }
649     //</editor-fold>//GEN-END:|143-getter|2|
650
651     //<editor-fold defaultstate="collapsed" desc=" Generated
            Getter: image2 ">//GEN-BEGIN:|142-getter|0|142-preInit
652     /**
653      * Returns an initiliazed instance of image2 component.
654      * @return the initialized component instance
655      */
```

```
656     public Image getImage2 () {
657         if (image2 == null) {//GEN-END:|142-getter|0|142-preInit
658             // write pre-init user code here
659             try {//GEN-BEGIN:|142-getter|1|142-@java.io. ›
                    IOException
660                 image2 = Image.createImage ("/mt.png");
661             } catch (java.io.IOException e ) {//GEN-END:|142- ›
                    getter|1|142-@java.io.IOException
662                 e.printStackTrace ();
663             }//GEN-LINE:|142-getter|2|142-postInit
664         // write post-init user code here
665         }//GEN-BEGIN:|142-getter|3|
666         return image2;
667     }
668     //</editor-fold>//GEN-END:|142-getter|3|
669 
670     //<editor-fold defaultstate="collapsed" desc=" Generated ›
        Getter: image3 ">//GEN-BEGIN:|144-getter|0|144-preInit
671     /**
672      * Returns an initiliazed instance of image3 component.
673      * @return the initialized component instance
674      */
675     public Image getImage3 () {
676         if (image3 == null) {//GEN-END:|144-getter|0|144-preInit
677             // write pre-init user code here
678             try {//GEN-BEGIN:|144-getter|1|144-@java.io. ›
                    IOException
```

```
679                 image3 = Image.createImage("/pm.png");
680             } catch (java.io.IOException e) {//GEN-END:|144-getter|1|144-@java.io.IOException
681                 e.printStackTrace();
682             }//GEN-LINE:|144-getter|2|144-postInit
683         // write post-init user code here
684         }//GEN-BEGIN:|144-getter|3|
685         return image3;
686     }
687     //</editor-fold>//GEN-END:|144-getter|3|
688 
689     //<editor-fold defaultstate="collapsed" desc="Generated Method: commandAction for Items ">//GEN-BEGIN:|8-itemCommandAction|0|8-preItemCommandAction
690     /**
691      * Called by a system to indicated that a command has been invoked on a particular item.
692      * @param command the Command that was invoked
693      * @param displayable the Item where the command was invoked
694      */
695     public void commandAction(Command command, Item item) {//GEN-END:|8-itemCommandAction|0|8-preItemCommandAction
696         // write pre-action user code here
697         if (item == imageItem1) {//GEN-BEGIN:|8-itemCommandAction|1|146-preAction
698             if (command == okCmd_BizForm) {//GEN-END:|8-itemCommandAction|1|146-preAction
```

```
699                 // write pre-action user code here
700                 switchDisplayable (null, getPayForm ());//GEN-LINE
                        :|8-itemCommandAction|2|146-postAction
701             // write post-action user code here
702             }//GEN-BEGIN:|8-itemCommandAction|3|8-
                    postItemCommandAction
703         }//GEN-END:|8-itemCommandAction|3|8-
                postItemCommandAction
704     // write post-action user code here
705     }//GEN-BEGIN:|8-itemCommandAction|4|
706     //</editor-fold >//GEN-END:|8-itemCommandAction|4|
707
708     //<editor-fold defaultstate="collapsed" desc=" Generated
            Getter: http_waitScrn ">//GEN-BEGIN:|148-getter|0|148-
            preInit
709     /**
710      * Returns an initiliazed instance of http_waitScrn
             component.
711      * @return the initialized component instance
712      */
713     public WaitScreen getHttp_waitScrn() {
714         if (http_waitScrn == null) {//GEN-END:|148-getter|0|148-
                preInit
715             // write pre-init user code here
716             http_waitScrn = new WaitScreen (getDisplay ());//GEN-
                    BEGIN:|148-getter|1|148-postInit
717             http_waitScrn.setTitle ("Processing ");
```

```
718             http_waitScrn.setCommandListener(this);
719             http_waitScrn.setFullScreenMode(true);
720             http_waitScrn.setImage(getImage4());
721             http_waitScrn.setText("Transaction processing ...
                    ...");
722             http_waitScrn.setTextFont(getFont());
723             http_waitScrn.setTask(getTask());//GEN-END:|148-
                    getter|1|148-postInit
724         // write post-init user code here
725         }//GEN-BEGIN:|148-getter|2|
726         return http_waitScrn;
727     }
728     //</editor-fold>//GEN-END:|148-getter|2|
729
730     //<editor-fold defaultstate="collapsed" desc=" Generated
            Getter: task ">//GEN-BEGIN:|151-getter|0|151-preInit
731     /**
732      * Returns an initiliazed instance of task component.
733      * @return the initialized component instance
734      */
735     public SimpleCancellableTask getTask() {
736         if (task == null) {//GEN-END:|151-getter|0|151-preInit
737             // write pre-init user code here
738             task = new SimpleCancellableTask();//GEN-BEGIN:|151-
                    getter|1|151-execute
739             task.setExecutable(new org.netbeans.microedition.
                    util.Executable() {
```

```
740                 public void execute () throws Exception {//GEN-
                        END:|151-getter|1|151-execute
741                     // write task-execution user code here
742
743                     TimeDelay ();
744
745                 //httpSubmit();
746
747                 }//GEN-BEGIN:|151-getter|2|151-postInit
748             });//GEN-END:|151-getter|2|151-postInit
749         // write post-init user code here
750         }//GEN-BEGIN:|151-getter|3|
751         return task;
752     }
753     //</editor-fold>//GEN-END:|151-getter|3|
754
755     //<editor-fold defaultstate="collapsed" desc=" Generated
            Getter: font ">//GEN-BEGIN:|155-getter|0|155-preInit
756     /**
757      * Returns an initiliazed instance of font component.
758      * @return the initialized component instance
759      */
760     public Font getFont () {
761         if (font == null) {//GEN-END:|155-getter|0|155-preInit
762             // write pre-init user code here
763             font = Font.getFont (Font.FACE_SYSTEM, Font.
                    STYLE_BOLD | Font.STYLE_ITALIC, Font.SIZE_LARGE);
```

```
                //GEN-LINE:|155-getter|1|155-postInit
764         // write post-init user code here
765         }//GEN-BEGIN:|155-getter|2|
766         return font;
767     }
768     //</editor-fold>//GEN-END:|155-getter|2|
769
770     //<editor-fold defaultstate="collapsed" desc=" Generated
        Getter: image4 ">//GEN-BEGIN:|156-getter|0|156-preInit
771     /**
772      * Returns an initiliazed instance of image4 component.
773      * @return the initialized component instance
774      */
775     public Image getImage4() {
776         if (image4 == null) {//GEN-END:|156-getter|0|156-preInit
777             // write pre-init user code here
778             try {//GEN-BEGIN:|156-getter|1|156-@java.io.
                IOException
779                 image4 = Image.createImage("/processing.png");
780             } catch (java.io.IOException e) {//GEN-END:|156-
                getter|1|156-@java.io.IOException
781                 e.printStackTrace();
782             }//GEN-LINE:|156-getter|2|156-postInit
783         // write post-init user code here
784         }//GEN-BEGIN:|156-getter|3|
785         return image4;
786     }
```

```
787     //</editor-fold>//GEN-END:|156-getter|3|
788     //<editor-fold defaultstate="collapsed" desc=" Generated
            Getter: imageItem_Success ">//GEN-BEGIN:|157-getter
            |0|157-preInit
789     /**
790      * Returns an initiliazed instance of imageItem_Success
            component.
791      * @return the initialized component instance
792      */
793     public ImageItem getImageItem_Success() {
794         if (imageItem_Success == null) {//GEN-END:|157-getter
                |0|157-preInit
795             // write pre-init user code here
796             imageItem_Success = new ImageItem ("Congratulations !
                    Transaction is succeeded !", getImage5 (),
                    ImageItem .LAYOUT_CENTER | Item .LAYOUT_BOTTOM |
                    Item .LAYOUT_VCENTER | Item .LAYOUT_SHRINK | Item .
                    LAYOUT_VSHRINK | Item .LAYOUT_EXPAND | Item .
                    LAYOUT_VEXPAND , "Transaction is succeed !", Item .
                    PLAIN );//GEN-LINE:|157-getter|1|157-postInit
797         // write post-init user code here
798         }//GEN-BEGIN:|157-getter|2|
799         return imageItem_Success ;
800     }
801     //</editor-fold>//GEN-END:|157-getter|2|
802
```

```
803     //<editor-fold defaultstate="collapsed" desc=" Generated
            Getter: image5 ">//GEN-BEGIN:|158-getter|0|158-preInit
804     /**
805      * Returns an initiliazed instance of image5 component.
806      * @return the initialized component instance
807      */
808     public Image getImage5() {
809         if (image5 == null) {//GEN-END:|158-getter|0|158-preInit
810             // write pre-init user code here
811             try {//GEN-BEGIN:|158-getter|1|158-@java.io.
                    IOException
812                 image5 = Image.createImage("/success.png");
813             } catch (java.io.IOException e) {//GEN-END:|158-
                    getter|1|158-@java.io.IOException
814                 e.printStackTrace();
815             }//GEN-LINE:|158-getter|2|158-postInit
816         // write post-init user code here
817         }//GEN-BEGIN:|158-getter|3|
818         return image5;
819     }
820     //</editor-fold>//GEN-END:|158-getter|3|
821
822     //<editor-fold defaultstate="collapsed" desc=" Generated
            Getter: ds_waitScrn ">//GEN-BEGIN:|159-getter|0|159-
            preInit
823     /**
824      * Returns an initiliazed instance of ds_waitScrn component.
```

```
825         * @return the initialized component instance
826         */
827        public WaitScreen getDs_waitScrn    () {
828            if (ds_waitScrn   == null) {//GEN-END:|159-getter|0|159-
                   preInit
829                // write pre-init user code here
830                ds_waitScrn   = new WaitScreen (getDisplay ());//GEN-
                       BEGIN:|159-getter|1|159-postInit
831                ds_waitScrn .setTitle ("Signing  Digital  Signature ");
832                ds_waitScrn .setCommandListener  (this);
833                ds_waitScrn .setFullScreenMode  (true);
834                ds_waitScrn .setImage (getImage4 ());
835                ds_waitScrn .setText ("Signing  Digital  Signature  ...
                       ... ");
836                ds_waitScrn .setTextFont (getFont ());
837                ds_waitScrn .setTask (getTask1 ());//GEN-END:|159-
                       getter|1|159-postInit
838            // write post-init user code here
839            }//GEN-BEGIN:|159-getter|2|
840            return ds_waitScrn ;
841        }
842        //</editor-fold >//GEN-END:|159-getter|2|
843
844        //<editor-fold defaultstate="collapsed" desc=" Generated
               Getter: task1 ">//GEN-BEGIN:|162-getter|0|162-preInit
845        /**
846         * Returns an initiliazed instance of task1 component.
```

```
847      * @return the initialized component instance
848      */
849     public SimpleCancellableTask getTask1     () {
850         if (task1  == null) {//GEN-END:|162-getter|0|162-preInit
851             // write pre-init user code here
852             task1  = new SimpleCancellableTask    ();//GEN-BEGIN
                :|162-getter|1|162-execute
853             task1 .setExecutable  (new org.netbeans.microedition. 
                util.Executable  () {
854                 public void execute () throws Exception  {//GEN-
                    END:|162-getter|1|162-execute
855                     // write task-execution user code here
856 
857                     TimeDelay ();
858                 }//GEN-BEGIN:|162-getter|2|162-postInit
859             });//GEN-END:|162-getter|2|162-postInit
860         // write post-init user code here
861         }//GEN-BEGIN:|162-getter|3|
862         return task1 ;
863     }
864     //</editor-fold >//GEN-END:|162-getter|3|
865 
866     //<editor-fold defaultstate="collapsed" desc=" Generated 
        Getter: encrypt_waitScrn ">//GEN-BEGIN:|163-getter|0|163-
        preInit
867     /**
```

```
868      * Returns an initiliazed instance of encrypt_waitScrn ›
             component.
869      * @return the initialized component instance
870      */
871     public WaitScreen getEncrypt_waitScrn() {
872         if (encrypt_waitScrn == null) {//GEN-END:|163-getter ›
                |0|163-preInit
873             // write pre-init user code here
874             encrypt_waitScrn = new WaitScreen(getDisplay());// ›
                    GEN-BEGIN:|163-getter|1|163-postInit
875             encrypt_waitScrn.setTitle("Encrypting Data");
876             encrypt_waitScrn.setCommandListener(this);
877             encrypt_waitScrn.setFullScreenMode(true);
878             encrypt_waitScrn.setImage(getImage4());
879             encrypt_waitScrn.setText("Encrypting ... ...");
880             encrypt_waitScrn.setTextFont(getFont());
881             encrypt_waitScrn.setTask(getTask2());//GEN-END:|163- ›
                    getter|1|163-postInit
882         // write post-init user code here
883         }//GEN-BEGIN:|163-getter|2|
884         return encrypt_waitScrn;
885     }
886     //</editor-fold>//GEN-END:|163-getter|2|
887
888     //<editor-fold defaultstate="collapsed" desc=" Generated ›
            Getter: task2 ">//GEN-BEGIN:|166-getter|0|166-preInit
889     /**
```

```
890         * Returns an initiliazed instance of task2 component.
891         * @return the initialized component instance
892         */
893        public SimpleCancellableTask getTask2     () {
894            if (task2  == null) {//GEN-END:|166-getter|0|166-preInit
895                // write pre-init user code here
896                task2  = new SimpleCancellableTask    ();//GEN-BEGIN
                       :|166-getter|1|166-execute
897                task2.setExecutable  (new org.netbeans.microedition.
                       util.Executable  () {
898                    public void execute () throws Exception  {//GEN-
                           END:|166-getter|1|166-execute
899                        // write task-execution user code here
900
901                        TimeDelay ();
902                    }//GEN-BEGIN:|166-getter|2|166-postInit
903                });//GEN-END:|166-getter|2|166-postInit
904            // write post-init user code here
905            }//GEN-BEGIN:|166-getter|3|
906            return task2;
907        }
908        //</editor-fold>//GEN-END:|166-getter|3|
909
910        //<editor-fold defaultstate="collapsed" desc=" Generated
               Getter: payAmount ">//GEN-BEGIN:|172-getter|0|172-preInit
911        /**
912         * Returns an initiliazed instance of payAmount component.
```

```
913         * @return the initialized component instance
914         */
915        public TextField getPayAmount() {
916            if (payAmount == null) {//GEN-END:|172-getter|0|172-preInit
917                // write pre-init user code here
918                payAmount = new TextField("Pay Amount (USD)", null, 32, TextField.DECIMAL);//GEN-BEGIN:|172-getter|1|172-postInit
919                payAmount.setPreferredSize(-1, -1);//GEN-END:|172-getter|1|172-postInit
920            // write post-init user code here
921            }//GEN-BEGIN:|172-getter|2|
922            return payAmount;
923        }
924        //</editor-fold>//GEN-END:|172-getter|2|
925        /**
926         * Returns a display instance.
927         * @return the display instance.
928         */
929        public Display getDisplay() {
930            return Display.getDisplay(this);
931        }
932
933        /**
934         * Exits MIDlet.
935         */
```

```
936     public void exitMIDlet () {
937         switchDisplayable (null, null);
938         destroyApp (true);
939         notifyDestroyed ();
940     }
941 
942     /**
943      * Called when MIDlet is started.
944      * Checks whether the MIDlet have been already started and
          initialize/starts or resumes the MIDlet.
945      */
946     public void startApp () {
947         if (midletPaused ) {
948             resumeMIDlet ();
949         } else {
950             initialize ();
951             startMIDlet ();
952         }
953         midletPaused = false;
954     }
955 
956     /**
957      * Called when MIDlet is paused.
958      */
959     public void pauseApp () {
960         midletPaused = true;
961     }
```

```
962
963     /**
964      * Called to signal the MIDlet to terminate.
965      * @param unconditional if true, then the MIDlet has to be
             unconditionally terminated and all resources has to be
             released.
966      */
967     public void destroyApp (boolean unconditional ) {
968     }
969 }
```

NetConnection.java

```
1  /*
2   * To change this template, choose Tools | Templates
3   * and open the template in the editor.
4   */
5
6  package com.uleth.mobank.connection;
7
8  import java.io.DataInputStream;
9  import java.io.DataOutputStream;
10 import java.io.IOException;
11 import javax.microedition.io.Connector;
12 import javax.microedition.io.HttpConnection;
13
14 /**
15  *
16  * @author zhuyp
17  */
18 public class NetConnection {
19
20     private String answer = "S,"; //the answer submitted
21
22
23     private static final String URL = "http://localhost:8080/
           MobileBankServer/PortalDataServlet?";
```

```
24
25      /**
26       * HTTP CONNECTION TO SERVER
27       */
28      public void postViaHttpConnection   () throws IOException   {
29          HttpConnection http    = null;
30          DataOutputStream dos    = null;
31          DataInputStream dis    = null;
32          int rc;
33
34          String url   = URL;
35          String rawData    = "answer =" + answer ;
36          url  = url  + rawData ;
37          System .out .println ("url :" + url );
38
39          try {
40              http  = (HttpConnection  ) Connector .open (url );
41
42              // Set the request method and headers
43              http .setRequestMethod  (HttpConnection  .POST );
44              http .setRequestProperty  ("User -Agent ", "Profile /MIDP  ›
                    -2.0  Configuration  /CLDC -1.1 ");
45              http .setRequestProperty  ("Content -Language ", "UTF -8")  ›
                    ;
46              http .setRequestProperty  ("Content -Length ", String .  ›
                    valueOf (rawData .length ()));
47
```

```
48              // Getting the output stream may flush the headers
49              dos = http.openDataOutputStream();
50              dos.write(rawData.getBytes());
51
52              // Getting the response code will open the connection,
53              // send the request, and read the HTTP response headers.
54              // The headers are stored until requested.
55              rc = http.getResponseCode();
56              if (rc != HttpConnection.HTTP_OK) {
57                  throw new IOException("HTTP response code: " + rc);
58              } else {
59
60                  System.out.println("dis before");
61
62                  dis = http.openDataInputStream();
63                  String resNotes = dis.readUTF();
64                  System.out.println("dis");
65  //                this.getSuccessForm().deleteAll();
66                  System.out.println("deleteAll");
67                  System.out.println("resNotes :" + resNotes);
68  //                this.getSuccessForm().append(resNotes);
69
70                  System.out.println("HTTP DONE");
71              }
```

```
72          } catch (ClassCastException e   ) {
73              throw new IllegalArgumentException    ("Not  an  HTTP  URL" ›
                    );
74          } finally {
75              if (dis  != null) {
76                  dis.close();
77              }
78              if (dos  != null) {
79                  dos.close();
80              }
81              if (http  != null) {
82                  http.close();
83              }
84          }
85      }
86
87
88  }
```

Part of Security API Code

ECDSAPrime192Signature

package org.yunpu.crypto.ecdsa;

/**

*

* @author zhuyp

* ECDSA Digital Signature over Prime curve with 192-bit key size

*/

public class ECDSAPrime192Signature

```
1  /*
2   * ECDSA Digital Signature over Prime curve with 192-bit key
        size
3   *
4   */
5  package org.yunpu.crypto.ecdsa;
6
7  import java.io.IOException;
8  import java.io.InputStream;
9  import java.math.BigInteger;
10 import java.security.SecureRandom;
11
12 import org.bouncycastle.asn1.x9.X962NamedCurves;
```

```
13 import org.bouncycastle.asn1.x9.X9ECParameters;
14 import org.bouncycastle.crypto.AsymmetricCipherKeyPair;
15 import org.bouncycastle.crypto.digests.SHA1Digest;
16 import org.bouncycastle.crypto.generators.ECKeyPairGenerator;
17 import org.bouncycastle.crypto.params.ECDomainParameters;
18 import org.bouncycastle.crypto.params.ECKeyGenerationParameters;
19 import org.bouncycastle.crypto.params.ECPrivateKeyParameters;
20 import org.bouncycastle.crypto.params.ECPublicKeyParameters;
21 import org.bouncycastle.crypto.params.ParametersWithRandom;
22 import org.bouncycastle.crypto.signers.ECDSASigner;
23 import org.yunpu.crypto.util.InputUtils;
24
25 /**
26  *
27  * @author zhuyp
28  * ECDSA Digital Signature over Prime curve with 192-bit key
      size
29  */
30 public class ECDSAPrime192Signature {
31
32     private ECDomainParameters params;
33     private ECPrivateKeyParameters privateKey;
34     private ECPublicKeyParameters publicKey;
35     private SecureRandom random = new SecureRandom();
36     private ParametersWithRandom privateKeyWithRandom;
37     private byte[] plainText;
38     private BigInteger r;
```

```
39      private BigInteger s  ;
40      private ECDSASigner signer    = new ECDSASigner ();
41      private SHA1Digest digest    = new SHA1Digest ();
42
43      public ECDSAPrime192Signature    () {
44
45      }
46
47      /**
48       *
49       * Generate ECDSA key pair as a file and store it at
             KeyDepositLocation
50       * @param KeyDepositLocation
51       */
52      private void ECDSAPrime192GenerateKey    (String
            KeyDepositLocation   ) {
53
54      }
55
56      /**
57       *
58       * @param strPlain_Text
59       * @return a digital signature
60       * @throws java.lang.Exception
61       */
62      private BigInteger [] ECDSAPrime192Sign   (String strPlain_Text   )
            throws Exception  {
```

```
63
64
65      }
66
67
68
69      /**
70       *
71       * @param strPlain_Text
72       * @return boolean verifying the digital signature is legal
             or not
73       * @throws java.lang.Exception
74       */
75      private boolean ECDSAPrime192Verify   (String strPlain_Text   )
           throws Exception  {
76
77      }
78
79      /**
80       * Transfer data information to a String based plain text
81       * @return String based plain text
82       */
83      public String get_PLAIN_TEXT   () {
84
85      }
86
87
```

```
88 }
```

AES256Crypto

package org.yunpu.crypto.aes;

/**

* @author zhuyp

* AES256Crypto is to realize AES cryptography algorithm with key size

* of 256-bit

*/

public class AES256Crypto

```
1  /**
2   *
3   * AES256Crypto is to realize AES cryptography algorithem with
        key size
4   * of 256-bit
5   */
6
7  package org.yunpu.crypto.aes;
8
9  import java.io.IOException;
10 import java.io.InputStream;
11 import java.security.SecureRandom;
12
13 import org.bouncycastle.crypto.BufferedBlockCipher;
14 import org.bouncycastle.crypto.engines.AESFastEngine;
15 import org.bouncycastle.crypto.modes.CBCBlockCipher;
```

```
16 import org.bouncycastle.crypto.paddings.PaddedBufferedBlockCipher;
17 import org.bouncycastle.crypto.params.KeyParameter;
18 import org.bouncycastle.crypto.params.ParametersWithIV;
19
20 /**
21  * @author zhuyp
22  * AES256Crypto is to realize AES cryptography algorithem with key size
23  * of 256-bit
24  */
25 public class AES256Crypto {
26
27     private byte[] key = new byte[32];
28
29     private byte[] iv = new byte[16];
30
31     private byte[] plainText = PlainText.PLAIN_TEXT.getBytes();
32
33     private byte[] cipherText;
34
35     private BufferedBlockCipher cipher = new PaddedBufferedBlockCipher(
36             new CBCBlockCipher(new AESFastEngine()));
37
38     private ParametersWithIV piv;
39
```

```
40      private SecureRandom random    = new SecureRandom  ();
41
42
43      public AES256Crypto  () {
44      }
45
46
47      /**
48       * generate encrypt/decrypt key, store it at  ›
            KeyDepositLocation
49       * @param byteKey
50       * @param KeyDepositLocation
51       */
52      private void AES256GenerateKey  (Byte [] byteKey , String   ›
            KeyDepositLocation  ) {
53
54      }
55
56
57      /**
58       * encrypt PlainText with byteKey
59       * @param PlainText
60       * @param byteKey
61       * @throws java.lang.Exception
62       */
63      private Byte [] AES256Encrypt  (Byte [] PlainText , Byte [] ›
            byteKey ) throws Exception  {
```

```
64
65      }
66
67      /**
68       * decrypt CipherText with byteKey
69       * @param CipherText
70       * @param byteKey
71       * @return PlainText
72       * @throws java.lang.Exception
73       */
74      private byte[] AES256Decrypt (Byte[] CipherText , Byte[] '
            byteKey ) throws Exception {
75
76      }
77  }
```