uOttawa

L'Université canadienne
Canada's university

FACULTÉ DES ÉTUDES SUPÉRIEURES ET POSTOCTORALES

u Ottawa

L'Université canadienne
Canada's university

FACULTY OF GRADUATE AND POSDOCTORAL STUDIES

Wei Du

AUTEUR DE LA THÈSE / AUTHOR OF THESIS

M.Sc. (Systems Science)

GRADE / DEGREE

Systems Science

FACULTÉ, ÉCOLE, DÉPARTEMENT / FACULTY, SCHOOL, DEPARTMENT

A Mobile Signing Solution Based on Personal Transaction Protocol and J2ME

TITRE DE LA THÈSE / TITLE OF THESIS

Tet Yeap

DIRECTEUR (DIRECTRICE) DE LA THÈSE / THESIS SUPERVISOR

CO-DIRECTEUR (CO-DIRECTRICE) DE LA THÈSE / THESIS CO-SUPERVISOR

EXAMINATEURS (EXAMINATRICES) DE LA THÈSE / THESIS EXAMINERS

Yongyi Mao

Jiying Zhao

Gary W. Slater

LE DOYEN DE LA FACULTÉ DES ÉTUDES SUPÉRIEURES ET POSTDOCTORALES /
DEAN OF THE FACULTY OF GRADUATE AND POSTDOCORAL STUDIES

A Mobile Signing Solution Based on Personal Transaction Protocol and J2ME

A Thesis Submitted to the School of Graduate Studies and Research
in Partial Fulfillment of the Requirements for the Degree of

Master of Science

AT
THE UNIVERSITY OF OTTAWA

By
Wei Du
June 10, 2005

SUPERVISED BY DR. TET HIN YEAP

Your file Votre référence
ISBN: 0-494-14900-0
Our file Notre référence
ISBN: 0-494-14900-0

Abstract

E-business has been booming for years; however, transaction security is still the biggest worry in the E-business world. Public key infrastructure is introduced to protect electronic transactions, wherein digital signing plays an important role that provides authentication and non-repudiation services. A smart card-based hardware security module is able to provide signing service with strong tamper-resistant and mobility. However, since smart cards do not have I/O interfaces, a user may blindly sign a document that has been altered by a malicious application.

Since mobile phones have user-friendly interfaces, strong computing power and short-distance wireless links, etc, they could be promising candidates for signing service. Some mobile phone-based signing solutions have been proposed as alternatives to smart cards. However, most of these solutions, including Zurich University's SIM based solution and Vodafone's mPKI, stick to GSM mobile networks. So far, there is no uniform standard for signing solution.

In my thesis, a mobile phone-based signing solution that is based on the Personal Transaction Protocol (PTP) defined by MeT is proposed. However, the original PTP is designed to work in a personal environment, where all devices are trusted. My thesis proposed an improved PTP to allow it to work in a public environment. In addition, J2ME platform is chosen to be the development platform of the proposed solution. The the J2ME's applicability is discussed according to the PTP server's development requirements. Finally, a PTP server's prototype is designed and developed to demonstrate the feasibility of the proposed solution.

Acknowledgements

I would like to thank the following list of people who supported me during this journey:

- Professor TET HIN YEAP, who sparked my interest in this research and was a great supervisor.
- My family, especially my PARENTS for instilling in me the passion to learn and my wife, Hong Yu, for her long lasting understanding and support during the whole research work.
- My son, Muoshi Du, for bringing me endless joy and inspiring me to work harder and harder.
- My friends and colleagues for their support, love and patience.

List of Figures

List of Tables

List of Abbreviations

AMS - Application Management Software

CA - certification Authority

CAPI - Cryptographic Application Program Interface

CDMA - Code-Division Multiple Access

CLDC - Connected Limited Device Configuration

CSP - Cryptographic Service Provider

CTF - Cryptographic Token Framework

DA - Discovery Application

DOM - Document Object Model

EKE - Encrypted Key Exchange

GOEP - Generic Object Exchange Profile

GSM - Global System for Mobile Communications

HMAC - Hash message authentication code

HSM - Hardware Security Module

IK - Integrity Protection Key

IrDA - Infrared Data Association

J2ME - Java 2 Micro Edition

JABWT - Java APIs for Bluetooth Wireless Technology

LTP - Local Transaction Protocol

MeT - Mobile Electronic Transaction

MIDP - Mobile Information Device Profile

MK - Master Key

MNO - Mobile Network Operator

mPKI - Mobile Public Key Infrastructure

OBEX - Object Exchange

OTA - Over the Air User-Initiated Provisioning

PCD - Personal Computing Device

PDA - Personal Digital Assistant

PIN - Personal Identity Number

PKI - Public Key Infrastructure

PTD - Personal Trusted Device

PTP - Personal Transaction Protocol

RFCOMM - Radio Frequency Communications Protocol

SATSA - Security and Trust Services API for J2ME

SE - Security Element

SIM - Subscriber Identity Module

SSL - Secure Socket Layer

STK - SIM Toolkit

SWIM - SIM/WIM

TLS - Transport Layer Security

USB - Universal Serial Bus

WAP - Wireless Application Protocol

WIM - WAP Identity Module

WIM - WAP Identity Module

WPKI - Wireless Public Key Infrastructure

WYSIWYS - What you see is what you sign

XML - Extensible Markup Language

1 Introduction

1.1 Problem Background and Motivation

E-business has been booming for years and has brought huge conveniences and opportunities to consumers and businesses. However, transaction security is still the biggest worry in the E-business world. Therefore, public key infrastructure (PKI) is introduced to safeguard electronic transactions, wherein digital signing plays an important role that provides transaction participators with authentication and non-repudiation services. Traditionally, a user's private key will be kept on his computer's hard disk. In this way, the private key is subject to theft by a malicious application, a virus, for example. Even though protected with a password, the private key is still easily acquired by means of dictionary attack, since the password is always short and regular. By comparison, a separate signing device has more advantages: it is more secure and more mobile. As a typical instance, smart card tokens, which conform to PKCS 11[1] or Microsoft CSP [2], are proposed and considered as a promising signing device, since smart cards have strong tamper-resistance and cryptographically computing power. However, smart cards do not have the capability to interact with users, which may lead a user to blindly sign a document that has been altered by a malicious application residing in the same computer with the valid application requesting for a signature.

Meanwhile, with increasing computing power and popularity, mobile phones gradually become a primary alternative signing device that can support "What You See Is What You Sign." Researchers from Zurich University implemented an STK application

in a GSM phone's SIM card [3], which borrows the mobile phone's capabilities of user interaction and local wireless connectivity to enable a PC to access SSL secured Web. Vodafone Ltd. tries out an SMS-based signing solution called mPKI [4]. In fact, both of them are GSM-specific solutions that depend on the SIM STK technique. My research work is motivated by the need to find a mobile phone-centric, network-independent and universal signing solution that can work with most intelligent devices, for example, a PC, an ATM machine and even an access control unit.

In my thesis, a mobile phone-centric signing solution that is based on Personal Transaction Protocol (PTP) [5] and the Java 2 Platform, Micro Edition (J2ME), will be proposed. In the proposed solution, a J2ME application, called the PTP server, will run in a mobile phone and provide signing service to external devices via PTP. PTP defined by MeT is a well-defined, layered protocol that enables a Personal Trusted Device (PTD) to provide signing service to a Personal Computing Device (PCD). PTP is supposed to work in a personal environment where all devices or PCDs are trusted so that PTP depends on a physical connection layer to provide security protection and lacks the capability of identity authentication. One of my thesis objectives is to extend PTP with a handshake process to allow it to work in a public environment. Besides, choosing J2ME as the PTP server's development platform enables this solution to be used in any mobile phone and network. The applicability of J2ME will be discussed as another objective of my thesis. Finally, a PTP server prototype will be designed and implemented to demonstrate the essences of the proposed solution.

1.2 Contributions

The main contributions of this thesis are:

- the proposal of a PTP-based mobile signature solution that enables a mobile phone to provide signing service to various devices in local connectivity.
- the proposal of an improved PTP that is able to work safely in both public and private environments.
- the proposal of J2ME CLDC and MIDP as the development platforms for the PTP server and the analysis of how related technologies back up the PTP server's development.
- a prototype to show the design and implementation essentials of the proposed solution.

1.3 Layout of this Thesis

The document is laid out as follows:

- Chapter 2 outlines security requirements in E-commerce and the backup of PKI technologies. Then a few signing solutions are introduced and reviewed.
- Chapter 3 briefly surveys some general characteristics of current mobile phones, which are considered to be evolving into a PTD to be able to perform secure electronic transactions. Then several existing mobile phone-based signature solutions are introduced and reviewed. Further, a PTP-based mobile signature solution is proposed, and the main issues to be concerned are also identified: PTP's security weakness and the development platform of the PTP server.
- Chapter 4 investigates the security weakness of PTP applied in a public environment. Two types of security model are defined to describe different security requirements in public and personal environments. Then an improved PTP is proposed with an added handshake process. To correspondingly deal with different security models, certificate-based and password-based handshake processes are defined with interaction and message details. This chapter also simply introduces how to provide communication security with XML encryption and signature technologies.
- Chapter 5 analyzes the applicability of J2ME CLDC and MIDP as the development platform for the PTP server. Furthermore, related J2ME

technologies, including development APIs, OTA and security service, are introduced. In addition, some existing SE solutions - SWIM, dual chip and dual slot are reviewed. Then, two pluggable SE tokens are introduced and also considered as the most promising solutions.

- Chapter 6 is an overview of the design and implementation of a prototype for the proposed solution. The prototype mainly covers the PTP API framework and the core PTP primitive - PTP.sign. In addition, some deployment works, for example, code signing, are introduced. Finally, the running results are shown step by step.
- Chapter 7 concludes the whole work of this thesis and gives some suggestions for future work.

2 Public Key Infrastructure and Signing Device

2.1 E-Commerce Security and Public Key Infrastructure

2.1.1 E-Commerce Security

E-business has been booming for years and has brought huge conveniences and opportunities to consumers and businesses. However, transaction security is still the biggest worry in the E-business world, since E-Businesses running on the Internet are vulnerable to attacks, for example, interception and tamper. A secure E-business environment calls for the following security services:

- **Authentication**. The assurance to one entity that another entity is who he/she/it claims to be.
- **Integrity**. The assurance to an entity that data has not been altered (intentionally or unintentionally) between "there" and "here," or between "then" and "now."
- **Confidentiality**. The assurance to an entity that no one can read a particular piece of data except the receiver(s) explicitly intended.
- **Non-Repudiation**. The assurance to one entity that another entity cannot deny the data that he sent.

Currently, most people agree that a reliable PKI can provide all those necessary services expected by E-business.

2.1.2 Public Key Infrastructure

What is PKI? A PKI is the set of operating system and application services that make it easy and convenient to use public-key cryptography, digital signatures and digital certificates [6]. Let us have a quick look at the essential techniques of PKI.

2.1.2.1 Public Key Cryptography

Public key cryptography uses two keys - a public key known to everyone and a private or secret key known only to the recipient of the message. These keys are complementary: if you encrypt something with the public key, it can only be decrypted with the corresponding private key and vice versa. Public key systems depend on the mathematical relationship between the public and private keys. It is not feasible to derive one from the other. There are two fundamental operations associated with public key cryptography: encryption and signing.

- **Encryption**. If Bob wants to send Alice some private data, he uses her public key to encrypt it and then sends it to her. Upon receiving the encrypted data, only Alice is able to use her private key to decrypt it. Therefore, the data is privately sent to Alice.

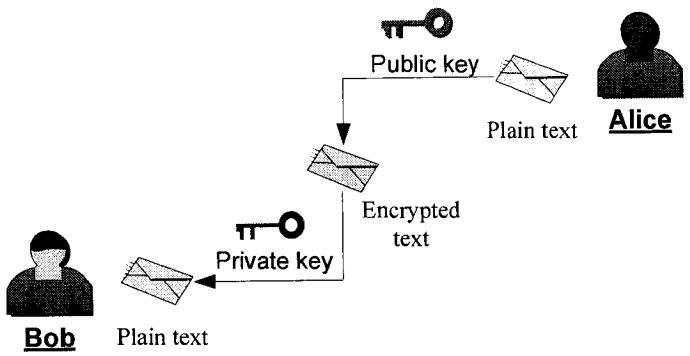


Figure 2-1 Public key encryption

- **Signing**. If Alice wants the world to know that she is the author of a message, she encrypts it using her private key and posts it publicly. The encrypted message can only be decrypted using Alice's public key. This means that it must have been encrypted using Alice's private key that she owns, so it must have come from Alice.

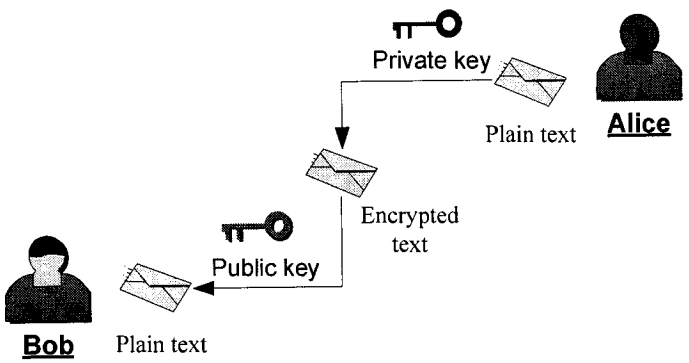


Figure 2-2 Private key signing

Some well-known public key systems include Diffie-Hellman, RSA, ElGamal and DSA.

2.1.2.2 Digital Signature

A primary application of public key cryptography is to create and verify digital signatures. A message's digital signature is created by encrypting the message's digest with the sender's private key. The signing process is shown in Figure 2-3. A digital signature has two basic capabilities.

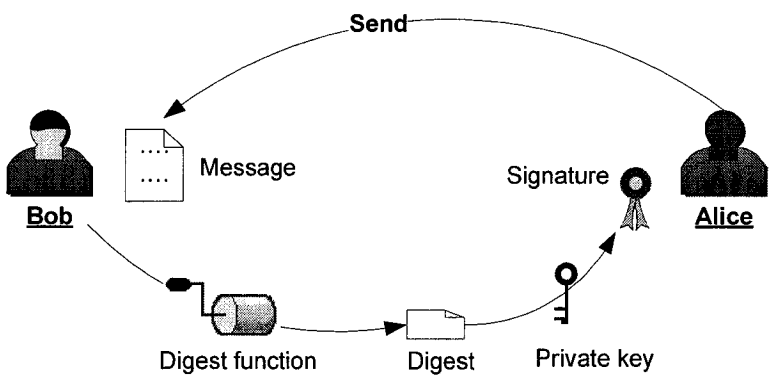


Figure 2-3 Signing process

- ***Document authentication.*** A signature can efficiently provide assurance that there has been no modification of the message, since it was digitally signed. A digest algorithm, for example, SHA-1, is used to create a digital fingerprint in the form of a hash value of a standard length, for example, 16 bits. Any change to the message invariably produces a different hash result, and it is impossible to derive the original message from knowledge of its hash value.
- ***Signer authentication.*** A signature can indicate who signed the message and is difficult for another person to produce without authorization. The signing operation of public key cryptography is used to handle this issue, i.e. the hash

value of the message is encrypted by the signer's private key and then sent to the receiver. Now the receiver can verify whether the signature is created by the correct signer and whether the document signed is the correct document.

2.1.2.3 Digital Certificate

A digital certificate is an electronic credential to show the holder's identity, which is issued by a certificate authority (CA) and is protected by the CA's signature. A certificate, based on the X.509 standard, must at least consist of the holder's name, a version number, a serial number, an algorithm identifier, the issuer, validity, the subject and public key information. In addition, a digital certificate is signed by the CA's root certificate so that a recipient can verify that the certificate is real. Combined with the digital signature, a digital certificate holder's identity can be authenticated. A Bob-authenticates-to-Alice example is shown in Figure 2.4.

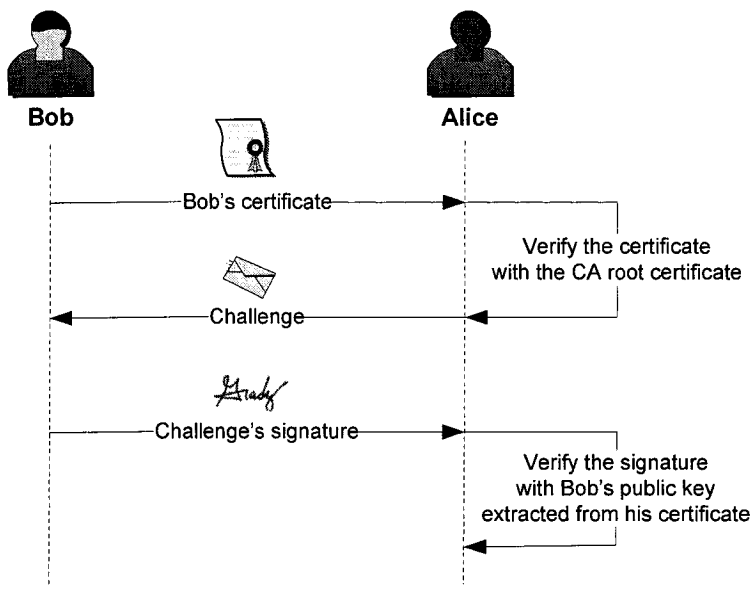


Figure 2-4 Bob authenticates to Alice with a digital certificate

2.1.3 PKI Enabling Secure Electronic Transactions

With the support of those essential techniques, PKI is able to ensure secure electronic transactions. Secure Socket Layer (SSL) protocol [8] is a typical PKI-enabled protocol that is widely applied to enable secure communications between two parties. The handshake process of SSL is simply introduced as below to show how those necessary security services are provided. From the SSL example, we can learn how PKI-related technologies meet the essential security requirements of E-commerce.

- **Authentication**. Firstly, Bob and Alice need to authenticate to each other before any real data is sent out. Suppose both Alice and Bob have their own digital certificates. Bob sends his digital certifies and starts a Bob-authenticates-to-Alice processes shown in Figure 2-4. Then Alice does the same thing. By means of digital signature and digital certificate technologies, both communication sides know exactly who they are talking to.
- **Confidentiality.** Subsequently, Bob creates a random session key encrypted with Alice's public key and sends it to Alice. Alice gets the session key after decrypting the received message with her private key. The session key will be used to encrypt all exchanged messages between Bob and Alice with secret key cryptography.
- **Integrity**. To protect the exchanged message's integrity, the message plus its digest will be encrypted together. The receiver can re-compute the message's digest and compare it with the one coming with the message to verify whether the message is modified.

- **Non-repudiation**. Actually, non-repudiation is not covered in SSL, since it not an essential need for communication security. However, both Bob and Alice can sign an important document like a contract with their private keys, and then the non-repudiation signature will be transferred just like other common messages.

2.2 Signature Device

We already know that digital signature plays an important role in PKI with regard to enabling secure electronic transactions. It directly supports authentication and non-repudiation and indirectly supports confidentiality and integrity. The most important issue when performing signing operations is keeping the signer's private key in a safe place. In most applications, a signer's private key will be kept on a hard disk of his PC. The storage of private keys is referred to as software mode in my thesis. Besides, to achieve higher security and mobility, private keys can be kept in a separate hardware device called the hardware security module (HSM).

2.2.1 Software Mode

In software mode, private keys are kept in an application or OS specific key store, where private keys are encrypted with a secret key algorithm. The secrete key to decrypt a private key is a password provided by the private key's holder. When an application needs to perform a signing operation, the cryptography API called by the application will access the key store and extract the target private key by having a user to provide his password. For example, a user is using a PC equipped with Microsoft Windows to perform online banking. The user's private key is kept in a Windows key store. When the user performs a signing operation via IE, Microsoft CAPI will acquire the user's private key from the key store and then perform the signing operation. The software mode is illustrated in Figure 2-5.

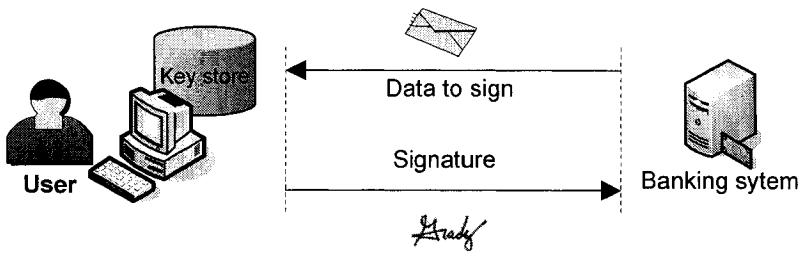


Figure 2-5 Signing operation in software mode

There are several problems associated with this traditional scheme: firstly, the key store stored on a hard disk could be easily exposed to someone else like a system administrator. Even in the case where the user uses the machine exclusively, the machine could still be infected by a virus, such as a Trojan horse program, which may send out the key store via e-mail or other means. In the case where an intruder gets a key store, it is only about how to break the password. Usually, the password set up by a common user is short and simple to guess, an therefore subject to dictionary attack. In addition, this solution does not allow moving from one PC to another. It is apparently tedious and costly to repeatedly install and uninstall the key in different machines for each user.

2.2.2 Hardware Security Module

To obtain a higher security and mobility than the software-based method described previously, the private key can be kept in a separate hardware device called a Hardware Security Modules (HSM). To be qualified as a good HSM, the device must be strongly tamper-resistant so that private keys can not be easily accessed without appropriate authorization. An HSM is not only a key store but it also performs signing operations and provides the resulting signature to the requester. In the above-mentioned example, instead

of performing the signing operation locally, Microsoft CAPI needs to request a signature from an HSM via a physical connection (Figure 2-6). In this way, the private key will not be given away so as to decrease the risk of private key leakage.

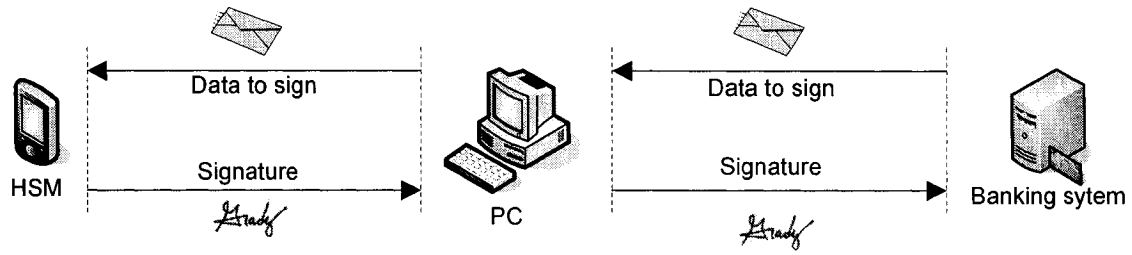


Figure 2-6 Sign process with an HSM

To further improve the security of private key storage, it is more important that an HSM separates the signature operation into a separate device, which enables a user to easily carry it around to perform secure electronic transactions anywhere. Subsequently, HSMs enables PKI to be involved in a wider application of electronic transactions. A typical case would be that an HSM can authenticate a user to a certain merchant device, such as a banking ATM or a retail vender machine, to provide non-repudiation service when a critical transaction is made. Currently, the most popular solution to a hardware security module is smart cards. However, more complex solutions have been proposed, for example, mobile phone-based solutions, whose functions are far more than security modules. Therefore, in the following text, the term "signing device" is used to represent all devices that can produce digital signatures.

2.2.3 Smart Card Solution as a Signing Device

Smart cards are small microprocessors with memory embedded inside a piece of plastic. A smart card can support multiple applications, which provide access to smart card memory or specific services. Smart cards have some form of tamper resistance against hackers attempting to extract secrets from the integrated circuit. For example, the smart card PIN will be blocked when a fixed number of invalid PINs are presented consecutively [9]. An external application communicates with a smart card via a card reader following the ISO7816 standard.

As a signing device, a user's private key will be stored in a smart card's memory, and a smart card application will perform the signing operation with the private key. To enable a smart card like a cryptography module to plug into a system conveniently, Microsoft defined a cryptographic service provider (CSP) model supported by Windows family products. In a similar way, the RSA security lab developed Cryptographic Token Interface Standard or PKCS #11, supported by UNIX-based systems and Netscape Navigator. These middleware architectures enable applications to access various security modules independent of hardware platform and implementation. To enable a new security module, only an adaptor library needs to be developed according to CSP or PKCS#11 standards. The example of a smart card as a CSP is illustrated in Figure 2-7.

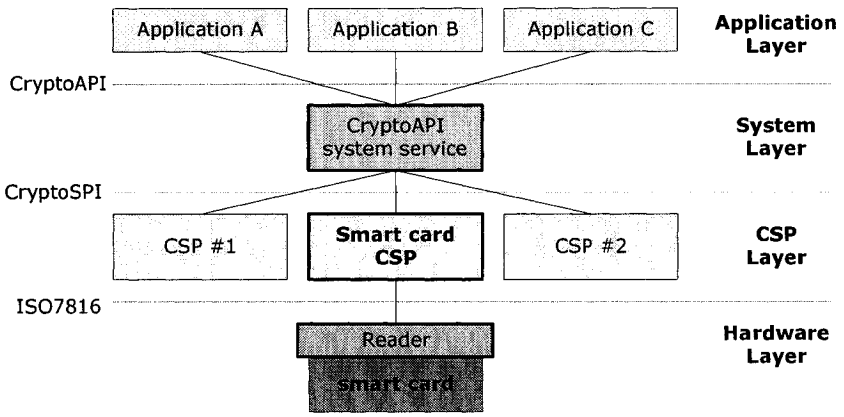


Figure 2-7 Smart card as a CSP

Schlumberger and Gemplus have both developed CSP and PKCS#11 compatible smart cards [10-11]. However, for a smart card solution, a smart card reader needs to be pre-installed on the target machine. To avoid extra cost and work raised by a smart card reader, RSA security and other companies proposed a USB token solution, which is a single smart card and reader-integrated chip with a USB interface. In this way, a user needs only to plug his USB token into a USB interface on a PC.

2.2.4 Summary

A smart card or USB token with strong tamper resistance and cryptographically computing capability is considered a most promising solution. However, it lacks a user interface to allow a user to make sure that what he signs is legitimate, which may lead to a user blindly signing a document that has been changed by a malicious application residing in the same computer, from a valid application sending the document for a signature. To solve this problem, some special signing devices [12] and PDA-based

solutions [13] were proposed. Similarly, they all have a display and buttons to show users data to sign and get users' decisions. However, because of high cost, they are not widely used. So let us summarize what the desired characteristics are that would be considered characteristics of a useful signing device:

- **Security**. It must have a secure storage with tamper-resistance to securely store private keys. In addition, it may provide other functions like certificate management, as defined in CSP and PKCS#11 standards, to support PKI-enabled protocols such as SSL.
- **Functionality**. It must be able to produce a digital signature according to the data provided. In addition, it may provide other functions like certificate management, as defined in CSP and PKCS#11, to support PKI-enabled protocols such as SSL.
- **Interactivity**. It must have a user interface to show the user data to sign and to obtain his decision.
- **Connectivity**. It must have a physical interface to enable it to be easily connected locally with signature requester devices like a PC, a banking ATM or even an access control unit. The physical interface could be a wired RS232 or a wireless Bluetooth.
- **Usability**. It must be cost effective so as not to incur much cost.

In fact, with computing power increasing, mobile phones have become a promising alternative to a signing device with the above properties. In the next chapter, the mobile phone-based signing solution will be discussed in details.

3 A Proposed Mobile Phone-Based Signing Solution

Currently, handheld intelligent devices like mobile phones and PDAs attract more researchers' attention, since they have superior computing power and user interactivity. Some mobile phone- and PDA-based solutions have been proposed, which could provide users with "What you see is what you sign" (WYSIWYS). Compared to PDAs, mobile phones are cheaper, more ubiquitous and conveniently carried around everywhere. Nowadays, mobile phones are so popular that around 30% of the population owns a mobile phone in North America. A mobile phone is one of the few things that people carry around every day. Therefore, mobile phones are considered as a promising candidate for a qualified signing device. In this chapter, the capabilities of today's mobile phone will be examined first, and then some previous mobile signature solutions will be discussed. Finally, a Personal Transaction Protocol (PTP)-based mobile signing solution will be proposed.

3.1 Today's Mobile Phones in Electronic Transaction

Wireless telecommunications have adopted 2.5G and even 3G standards in some countries, offering a wide-band network and band-width rich services. At the same time, the enhanced network and services also drive the need for more powerful mobile phones – smart phones. In 1996, Computer World described a smart phone as "a wireless phone with text and Internet capabilities [14]. It can handle wireless phone calls, hold addresses and take voice mail and can also access information on the Internet and send and receive E-mail and fax transmissions". Nowadays, some typical smart phones, such as Nokia

6600 and Sony-Ericsson P900, have departed from this definition significantly. They have more superior computing power, adopt industry-standardized OS, support Bluetooth and embrace PKI. They are not only phones for voice calls, but also intelligent devices to perform secure electronic transactions. Furthermore, they are migrating to become Personal Trusted Devices (PTD) [14].

3.1.1 Some Common Characteristics of Smart Phones

Smart phones share some common characteristics to enable them to perform secure electronic transactions. Some of them are introduced below:

- ***Superior computing power***. Intel StrongArm processors, widely adopted by Nokia, Ericsson and other phone manufactures, can run at a maximum speed of 230 MHz and 700 MHz, and have higher performance and lower cost than processors for desktop computers in the early 1990s. Superior computing power enables mobile phones to perform complicated computing.
- ***Local wireless connectivity***. Most mobile phones include an IrDA interface. Some middle-to-high end mobile phones are also equipped with a Bluetooth interface. Wireless interfaces enable mobile phones to be more involved in various electronic transactions in a proximity environment.
- ***Smart card-based security***. A GSM phone always has a smart card slot, which is exclusive to SIM cards. However, some dual chip or dual slot phones are able to accommodate an independent smart card as a security element to store confidential data. With smart card support, a mobile phone can perform more serious electronic transactions.

- ***Industry-standardized OS***. More mobile phones are adopting sophisticated industry-standardized OS, for example, Symbian and Smartphone. Rich APIs enable developers to develop more powerful applications in a shorter cycle.
- ***Development technology***. SIM STK, WAP and J2ME [15] have been widely supported by most mobile phones. Developers can easily choose the proper solutions to meet different development requirements.

3.1.2 Personal Trusted Device

Some leading phone manufactures, financial companies and telecommunication operators have united to form a special organization, Mobil Electronic Transaction Ltd. (MeT), to make related standards to promote the application of mobile phones in mobile commerce. The target of MeT is to use existing and emerging standards to build a common framework and to define an open platform for secure mobile transactions and create an implementation roadmap in order to enhance the fast adoption of trusted mobile e-business.

MeT first introduced the concept of the Personal Trusted Device (PTD). MeT believes that, since the mobile phone has been rapidly evolving into much more than a wireless phone, it is transforming into a PTD. MeT's PTD specifies essential characteristics that a mobile hand-held device should possess to be an appropriate mobile application platform to enable secure electronic transactions. Those characteristics are summarized as below.

- A PTD is personal, controlled and used by one person and carried by that person most of the time.

- A PTD has an application platform with associated user interfaces for transaction-related services such as banking and payment.
- A PTD has the security functionality required for transaction related services: secure sessions, authentication and authorization.
- A PTD contains a security element, which is used for protecting its most critical data, such as private keys.

In fact, according to MeT's description of a PTD, a PTD could be a mobile device other than a mobile phone, like a PDA or a pager. However, mobile phones will prevail over other alternatives in the field of electronic transactions with those capabilities introduced in the last section. Throughout this article, the terms PTD and mobile phone are used interchangeably to indicate mobile phones if the context allows it.

3.1.3 Security Element

According to [13], a mobile phone that is eligible to be a PTD must contain a security element to protect its most critical data, such as user's private keys. The security element (SE) is the crucial part of a PTD that determines a mobile phone's security level to perform electronic transactions. The SE is used to keep the user's private keys, and certificates and perform private key operations. In fact, the SE acts as an internal HSM inside a PTD. In terms of a PTD performing secure electronic transactions, the existence of an SE enables a three-tier security model (Figure 3-1). In this model, an external device initiates a signing request to a PTD; then the PTD transfers this request to its internal SE. Before an SE accepts a request, it must be activated by a PIN code. After that, a signature is produced by the SE and returned to the PTD. The PTD then sends it

back to the external device. In fact, the SE is a real signature service provider. The PTD acts as a bridge to connect its SE to the external device. In other words, it is the SE that enables a PTD to perform serious transactions.

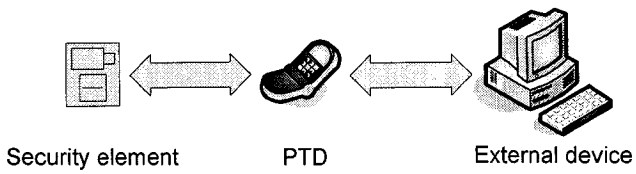


Figure 3-1 PTD's three-tier security model

Security elements may have diverse implementation forms, which can be removable or non-removable entities (Figure 3-2). In case of a removable security element, MeT requires it conforms to WIM, i.e. WAP Identity Module [16]. A non-removable SE could be implemented as an embedded hardware module or even a software module with cryptographic processing. However, no matter in which form, an SE must have strong tamper resistance to protect the private key against disclosure. Currently, smart cards are considered the best SE solution. On the one hand, an SE has strong tamper resistance and cryptographical computing power, as stated in Chapter 2. On the other hand, smart cards, called subscriber identification module (SIM) cards, have actually been applied in GSM mobile phones to identify the subscriber to the mobile network. SWIM is defined to accommodate WIM in a SIM card so that a SWIM card can both authenticate the mobile user to the GSM network and also enables mobile phone applications to perform secure electronic transactions. Other SE practices like dual chip and dual slot are also proposed and will be discussed later.

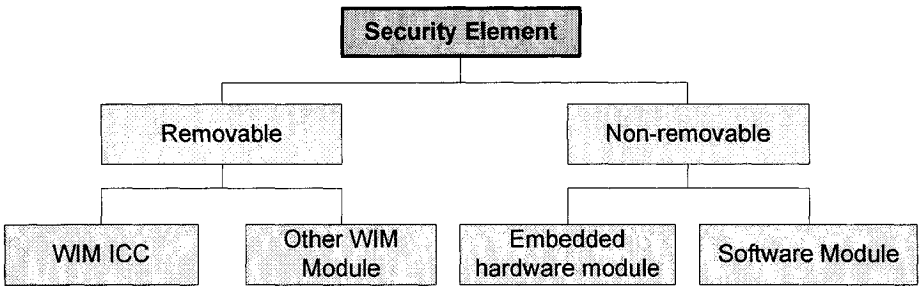


Figure 3-2 SE category hierarchy

3.2 Previous Work on a Mobile Signing Solution

According to the foregoing discussion, we know that the mobile phone has evolved into a PTD that is ready to perform secure electronic transactions. Some researchers have paid attention to use mobile phones as signing devices and some mobile signing solutions are proposed. Those solutions will be discussed in this section.

3.2.1 SIM-Based Solution

Some researchers from Zurich University proposed a SIM-based mobile signature solution [3]. An STK application is implemented in a GSM phone's SIM card to enable an application running in a PC to perform secure online transactions via SSL (Figure 3-3). The STK application drives the mobile phone's display and keyboard to interact with the user and utilizes short-distance wireless protocol, which can be IrDA or Bluetooth, to provide signing services to the PC application.

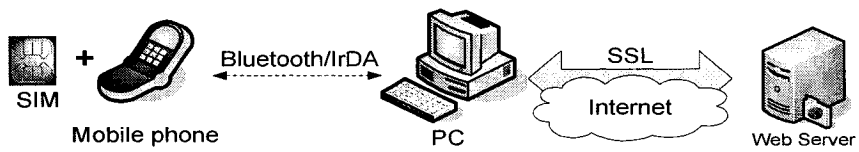


Figure 3-3 SIM-based solution

This solution solves the blindly signing problem in a pure smart card solution by showing the data to sign on the mobile phone's display and asking for users' confirmation. Further, since the private key is also kept in the SIM card, the solution has

strong tamper resistance. However, since this solution is based on the SIM STK technique, it suffers two problems: this solution can work only with GSM phones that have a SIM card; it has to fall under the resource constraints of smart cards so that it can not support complicated application logic to constitute a general signing protocol.

3.2.2 Vodafone's mPKI

Vodafone brought forward an SMS-based solution called mPKI, [4] whose architecture is shown in Figure 3-4.

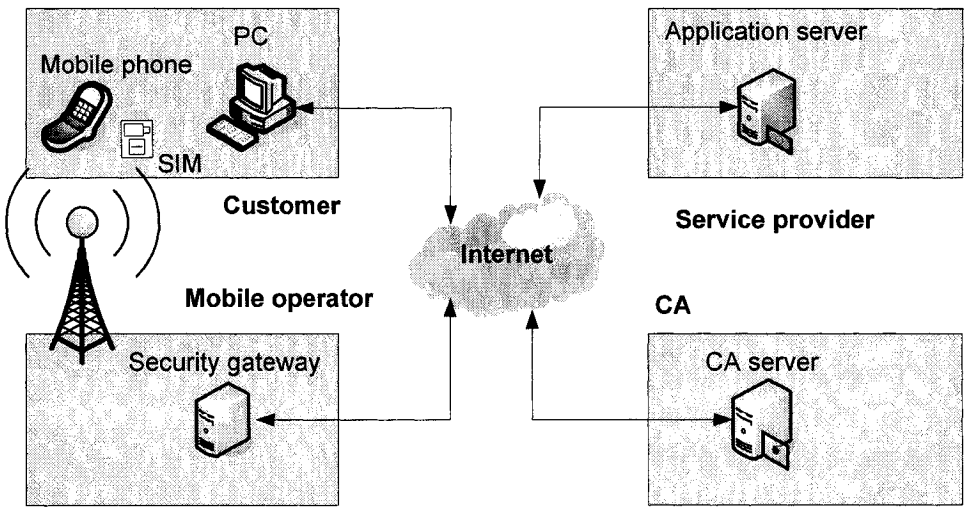


Figure 3-4 Vodafone's mPKI system

The solution can be described by the following process:

1. The client PC requests a service from an application server via the Internet, or a user directly operates a service machine, for example, an ATM.
2. The application server sends a signing request to the security gateway at the mobile operator.
3. The request is converted into an SMS message and sent to the user's mobile

phone.

4. The user receives the message, activates the SIM to sign the message and sends the signature back to the gateway.
5. The gateway verifies the signed data and sends a verification result to the application server or the service machine.

Technically, Vodafone's mPKI is still a SIM STK based GSM-specific solution. The difference with the Zurichan solution is that a mobile phone does not interact with the application server directly; instead, a security gateway provided by mobile network operators (MNO) directly interacts with both of them via SMS and undertakes the signature verification. Vodafone's mPKI is a typical MNO-centric solution: SIM cards are issued by the MNOs; security gateways are provided by the MNOs; SMS is carried by the mobile network; and payment is collected via MNO's bills to its subscribers. This solution can work in both remote and local environments. This solution has two drawbacks: it works only with GSM phones, and it is complicated and inefficient in local environments since all interactions are based on SMS.

3.2.3 Personal Transaction Protocol by MeT

MeT defines the Personal Transaction Protocol (PTP) to enable a PTD to provide signing services to applications that are executed in a Personal Computing Device (PCD) like a PC or a Set-Top Box (Figure 3-5) in a personal environment. MeT defines that PTD operates in three environments [17]: remote, local and personal environments. In fact, a personal environment is a special local environment where PTDs and PCDs stay in a short distance so that they can interact via IrDA or Bluetooth; all PCDs requesting signing services are namely considered personal devices that can always be trusted. Intended usage of PTP includes SSL/TLS authentication, S/MIME secure email, IPsec VPN and desktop login etc. A typical PTP session contains the following steps:

- Activation. The PTD is activated to use PTP with a certain PCD. This step involves creating the transport level connection and sending of device information from the PTD to the PCD.
- Exchange of object information. The PCD requests information on objects contained in the PTD. The PTD returns the objects or requested attributes.
- Signature service. The PCD sends a signature request. The PTD returns the signature.
- The PCD terminates the PTP session (the transport level connection may remain if it is needed for other purposes).

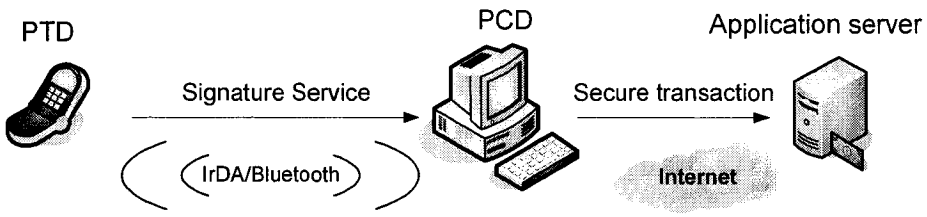


Figure 3-5 PTP application scenario

PTP is a layered protocol, which consists of the PTP service layer, the transport layer and the physical connection layer (Figure 3-6). At the PTP service layer, PTP defines five types of service primitives: session management, device information, object management and cryptographic primitives. The main primitive - PTP.sign is defined to provide PCDs with different types of signatures according to PKCS#7 [18]. At the communication layer, the Object Exchange (OBEX) protocol is adopted as transport protocols. Since PTP services are built on OBEX instead of directly on the physical connection layer, almost all physical connection interfaces can be applied, for example, IrDA, Bluetooth, USB and RS232.

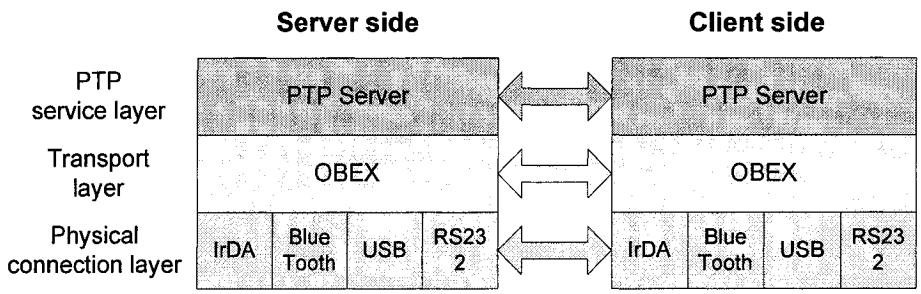


Figure 3-6 PTP Protocol Stack

PTP dose not specify any implementation details. However, PTP defines all-layer communication specifications to ensure the interoperability between PTP servers and

clients from different providers. Since PTP is supposed to work in a personal environment, PTP relies mainly on the security capabilities of the physical connection to enable a secure communication channel between a PTD and a PCD. PTP expects that a secure physical connection is already set up before or during the PTP initialization phase. In addition, PTP does not provide a mechanism to verify a PCD's identity, since PCDs are always trusted in a personal environment.

3.2.4 Summary

A few mobile phone-based signing solutions are just examined. Compared to the Zurichan SIM-based solution and Vodafone's mPKI, MeT's PTP-based solution (or PTP) has more advantages:

- PTP is a mobile phone centric solution. The PTP server is an application running in a mobile phone, which directly uses mobile phones' resources like the CPU, memory and wireless connectivity. It does not suffer the resource constraints of a SIM smart card.
- PTP is a locally connective solution via a short-distance wireless connection; Bluetooth, for example. It does not rely on SMS used mPKI with longer delays and communication costs.
- PTP is a development-platform independent solution. The PTP solution can be implemented with any development platforms on mobile phones.
- PTP is the only well-defined specification for mobile phones providing signing services. It specifies all related service primitives and defines all exchanging

message formats and even some communication details on OBEX and Bluetooth/IrDA. All of those could lead PTP to wider applications.

However, PTP is still not a perfect protocol that could be simply applied without any concerns. The crucial drawback is that PTP's security model disables it to work in a public environment, where devices requesting signing services, like a retail vendor machine, can not always be trusted. Therefore, before PTP can be migrated to a pubic environment, its security model must be improved.

3.3 Proposed Mobile Signing Solution

3.3.1 Architecture of the Proposed Solution

In my thesis, a mobile signing solution based on PTP is proposed. The proposed solution consists of two sub-systems (Figure 3-7): a PTD sub-system providing signing services and a PCD[1] sub-system requesting signing services. These two sub-systems are wirelessly connected in a short range. They talk to each other with PTP.

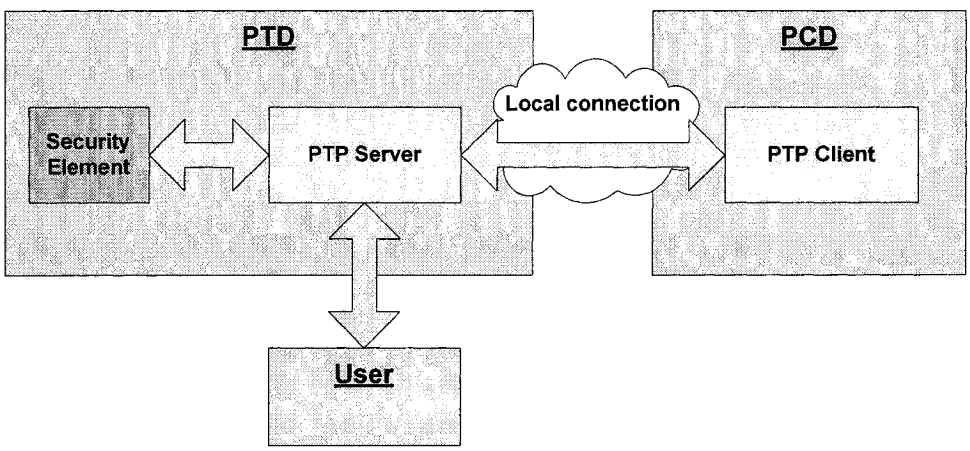


Figure 3-7 PTP-based mobile signature solution

The PTD sub-system is a PTP server application that is running in a PTD. The PTD provides the PTP server with all resources needed:

- The PTD must contain a security element to store key material and perform private key operations.
- The PTD must provide a standard short-distance wireless interface like Bluetooth to enable wireless connection with PCDs.

[1] To conform to PTP terms, we still use PCD to indicate the external device, though in most cases it will not be a personal device anymore.

The PTD must provide a J2ME CLDC- and MIDP-compatible application platform to support the development of the PTP server.

The PTP server is the core part of the solution. From an external perspective, it supports signing and other services specified in PTP to a PCD. From internal perspective, it is able to access the security element in the PTP and its functionalities.

The PCD sub-system could be any device with a PTP client running inside. A PCD could be a general intelligent machine, typically, a PC or a laptop or a special merchant device like a banking ATM or a retail vendor machine. The PTP client runs in such a PCD and interacts with the application that needs signing services in some way. With different PCDs, the PTP-based mobile signing solution can have different usage scenarios. However, the PTD system still keeps working in the same way.

3.3.2 Research scope

In my thesis, some primary issues about the PTP server in the PTP sub-system will be considered. Accurately speaking, to realize such a PTP-based system, we have to first solve two main issues.

Firstly, MeT defines the PTP to work in a personal environment where PTP needs only to deal with those trusted PCDs. PTP relies on the physical connection layer to provide most communication security. However, in a public environment, those devices requesting signing operations from PTDs cannot always be trusted spontaneously. Furthermore, the physical connection between PTDs and PCDs cannot be configured to provide communication security all the time. Therefore, the PTP's security model must be improved in order to enable it to work in a mistrusted device in a public environment.

In Chapter 4, an improved PTP's security model will be proposed to solve this issue.

Secondly, MeT PTP 1.0 focuses on the details of each communication layer of PTP. It does not recommend on which platform to implement the PTP server. Therefore, we need to carefully consider the various requirements of the PTP server and then choose an appropriate platform to ensure that it works well in practice. In Chapter 5, J2ME CLDC and MIDP will be recommended as the development platforms of the PTP server. Furthermore, related techniques to implement the PTP server will be introduced. In addition, a PTP server prototype will be designed and developed to partially show the implementation details, which will be described in Chapter 6.

In fact, except for two above-mentioned issues that will be covered in my thesis, the implementation of a security element is also an important topic to the proposed PTP-based solution. In a sense, the SE in a PTD determines the security level of all applications, including the PTP server, that depend on it. The detailed research on the SE will be saved for future works. However, some existing SE implementations will be introduced and discussed in Chapter 5.

4 An Improved Personal Transaction Protocol

4.1 Analysis of PTP Security

4.1.1 Secure Communication in a Public Environment

In a public environment, a performing signing operation needs three essential security protections: authentication, confidentiality and integrity.

- **Authentication**. In a wireless world, authentication is twofold. On one side, authentication means device pairing: the PTD and PCD must guarantee that they are talking to the intended counterpart since no cable binds them together. On the other side, that the PCD installed in a pubic environment is not always trusted. Before performing any transaction, the user needs to authenticate the PCD's identity and ensure the PCD is a valid device performing the intended service.
- **Confidentiality**. Some messages sent from a PCD to a PTD are sensitive, for example, a banking statement or a business contract. Therefore, all messages between a PTD and PCD must be kept confidential. No third party can intercept those messages.
- **Integrity**. Integrity protection ensures the messages that a PTD and PCD exchange is kept unchanged. It prevents a malicious third party from defrauding the PTD of a valid signature on a fake document, which is changed from the original one.

In a critical scenario, for example, when a user is interacting with a banking machine, all security protections are indispensable. However, in a non-critical scenario, some securities could be optional, or all of them could be omitted. In the following section, PTP's security model will be examined to see whether it can be applied in a public environment.

4.1.2 PTP's Security Model

Personal Transaction Protocol is designed to provide signing services to a personal computing device (PCD) in a personal environment. Since a PCD is a personal device, the user does not need to verify its identity. The only issue that concerns the user comes from wireless communication. It must be ensured that a PTD will talk to the intended PCD instead of a malicious third party. Meanwhile, all messages between the PTD and the PCD should be protected with confidentiality and integrity. Therefore, MeT builds PTP's communication security on a wireless communication layer. In the short-distance wireless world, IrDA and Bluetooth are mainly concerned. Since IrDA does not offer any communication security, Bluetooth is the only choice for PTP to obtain communication security. Therefore, all statements about communication security in PTP are based on Bluetooth implicitly or explicitly.

4.1.2.1 Authentication and Confidentiality

PTP completely relies on Bluetooth to enable authentication and confidentiality. Bluetooth provides built-in security mechanisms for authentication. By default, Bluetooth communication is not authenticated, and any device can talk to any other device. However, a Bluetooth device may choose to require authentication to provide a particular

service. In such case, a pairing of the Bluetooth device is necessary. Bluetooth pairing is normally done with a PIN code, which is an ASCII string up to 16 characters in length. A user is required to enter the same PIN code on both devices [19]. Then a conventional challenge-response routine is employed to authenticate each other. During Bluetooth pairing, both devices generate a 128-bit link key from the PIN code and other information. An encryption key then derived from the link key will be used to encrypt all Bluetooth messages.

4.1.2.2 Integrity

Integrity protection is offered by the PTP service layer, since Bluetooth does not provide it. A PCD determines whether integrity protection is needed during the session initialization phase. When a PCD requests a new session in PTP.Req primitive, it needs to specify the QoS parameter, which is an indication of the desired security services for this session. Currently, the only choice is integrity protection. In case the QoS parameter specifies integrity protection, both PTD and PCD will use the HMAC [20] mechanism to protect the integrity of messages. The HMAC signature will be put into request and response XML documents according to the format defined in [21]. The derivation an process of HMAC key or symmetric integrity protection key (IK) from a master key (MK) and the PTP SessionId is described as follows [5]:

1. The PCD sends an InitRequest to the PTD. The InitRequest is sent using canonical XML with no comments.
2. The PTD derives an IK for this session from MK, the SessionId and the InitRequest. The TLS PRF [TLS] shall be used as the key derivation function.
3. The PTD calculates the XML-signature MAC for the InitResponse message

using IK.

4. When receiving the InitResponse, the PCD retrieves the SessionId, MK and InitRequest and generates the IK
5. Now the PCD can verify the XML-signature on the InitResponse message. This will also implicitly verify the integrity of the InitRequest since it is a part of the key derivation function.
6. All subsequent messages are protected using IK.

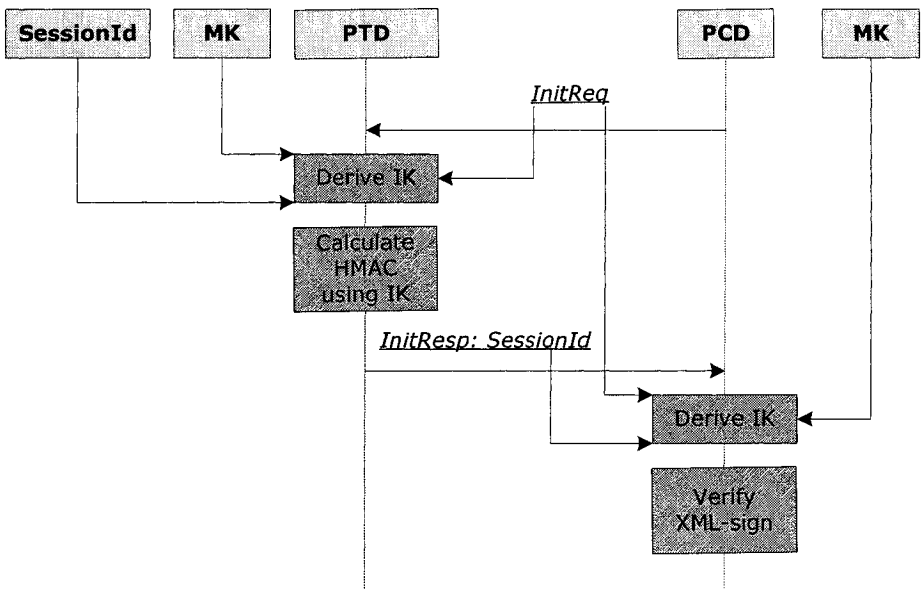


Figure 4-1 Integrity protection key derivation

However, where does the MK come from? Again, PTP has to turn to Bluetooth. PTP derives the MK from Bluetooth's link key, which can be retrieved using the HCI command "Read_Stored_Link_Key" [5]. Therefore, in terms of integrity protection, PTP still originally relies on Bluetooth.

4.1.3 PTP Security Gap in a Public Environment

According to the above analysis, we can come up with such a conclusion that PTP depends on Bluetooth to provide authentication, confidentiality and integrity protection. In fact, MeT defines PTP as working in a personal environment, where PCDs can always be trusted. In addition, since both PTD and PCD are personal devices, a secure wireless link can be set up by the user by means of pairing two Bluetooth devices. However, the case is more complicated in a public environment:

1. A device requests a signature from a PTD, for example, an ATM or a retail machine belonging to a service provider, which cannot be trusted easily. Therefore, it is not enough to ensure only a PTD to talk to its intended peer. PTP needs to support PTD to verify a PCD's identity.
2. A device, like an access control unit, sometimes does not have an input and output interface to allow a user to configure or accept a PIN code. In such case, Bluetooth pairing does not work.
3. In addition, the wireless protocol is constrained to only Bluetooth-like security-enabled protocol. The low-cost IrDA without any security is excluded from PTP applications.

Considering the weakness mentioned above of PTP in a public environment, PTP will have to be improved to solve the above problems so that it can work well in both personal and public environments.

4.2 Security Model of the Improved PTP

4.2.1 LTP Overview

The security issues, which disable PTP to work in a public environment, result from the fact that PTP depends on a physical connection channel to provide communication security. In this chapter, an improved PTP is proposed to accommodate the PTP working public environment. This improved PTP is called Local Transaction Protocol (LTP), since it can be applied in a general local environment. In fact, LTP is only different with PTP in the security model; others are the same. If the context allows, the term PTP is still used to indicate the improved PTP or LTP.

LTP builds up communication security at the service layer. The session initialization is extended to accommodate a handshake process, which deals with identity authentication, device pairing and session key exchange. Based on the shared session key, XML encryption and signatures will be adopted to enable PTP's confidentiality and integrity. In this way, LTP can get rid of the security dependence on wireless communication protocol. Meanwhile, the above-mentioned PTP's problems in a public environment can be solved successfully. The changes to the PTP security model is illustrated in Figure 4-2.

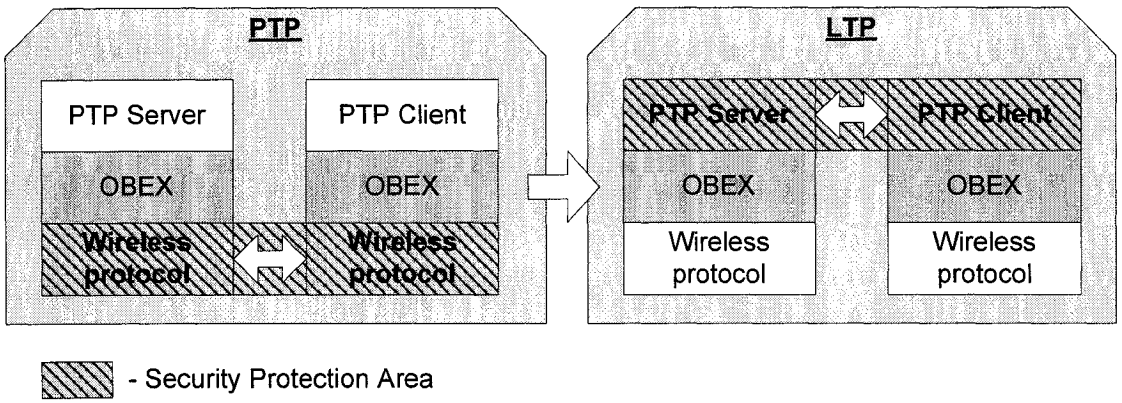


Figure 4-2 Changes to PTP security architecture

4.2.2 LPT's Security Modes

In the initialization phase of the secure wireless communication channel, the authentication process always plays a crucial role in further enabling other securities: confidentiality and integrity. In most cases, a long-term session key can be negotiated by both sides during the authentication process. As mentioned before, the authentication in a wireless work is twofold: identity verification and device pairing. Identity verification means that a PTD needs to verify a PCD's identity to determine whether the PCD is a valid service provider. On the other hand, since PTD and PCD communicate via a wireless channel, the user has to make sure that his PTD is talking to the correct PCD and vise versa, which is called device pairing. In fact, device pairing is always indispensable for a mobile signature solution in both public environment and personal environments. However, identity verification is only needed in a public environment. Therefore, the LTP's security model could be further categorized into two modes according to the environment applied: public mode and personal mode, which will be discussed in detail in the following sections.

4.2.2.1 Public Mode

This application always happens when a PTD interacts with a mistrusted PCD mounted in a public place, for example, a bank machine, a vendor machine or an access control system, where nobody is on guard in most cases. The PCD directly provides some kind of commercial service to a user with a PTD. Before the user performs any transaction with that PCD, he must make sure that the interacting PCD really performs the intended service and comes from the intended service provider. Therefore, the PCD needs to authenticate itself to the user. In this case, a PCD needs to be equipped with a digital certificate to verify its identity to a PTD.

In public mode, device paring is still indispensable. However, a certificate-based authentication process does not ensure the pairing of two wireless devices. Therefore, a user's observation is also needed to identify whether his PTD is talking to an intended PCD. The way to help a user to judge will be described in the section regarding the certificate-based handshake.

4.2.2.2 Personal Mode

This scenario happens when a PTD interacts with a device, for example, a personal computer or a set top box, that can be trusted by the user in a personal environment. Therefore, when a user's PTD starts a connection with the PCD, two devices need only to consider device pairing, i.e. they are really talking to the intended party instead of a malicious third party. In such a case, PCD does not need to equip with a certificate. A password-based authentication process is enough to ensure both PTD and PCD to connect to each other correctly. This process is similar to the Bluetooth device pairing. Both PTD and PCD share a password input by the user. They ensure a connection to the right

counterpart by means of a challenge-response process. The process is a replacement for PTP when IrDA is adopted as wireless channel or for any reason the Bluetooth is hard to configure to pair.

To realize LPT's security model, the PTP initialization primitive - ptp.init - will be extended to a handshake process, whereby PTD and PCD authenticate to each other and then a shared session key can be exchanged to provide confidentiality and integrity services. A certificate-based and password-based authentication mechanism will be incorporated in the handshake process to respectively deal with public and personal modes. These two handshake processes will be defined in the following sections.

4.3 LTP Handshake Process

The handshake process is meant to authenticate PTD and PCD to each other and furthermore negotiate a shared session key, which can provide long-term communication confidentiality and integrity between a PTD and a PCD. To make the statement easy, the whole LTP session initialization process is divided into three phases: session request, handshake process and session response. It is illustrated in Figure 4-3. Another reason to make this division is that, if the handshake process is omitted, the LPT will degenerate to PTP. In addition, corresponding to public mode and personal mode, LTP supports two types of handshake processes: the certificate-based handshake and password-based handshake.

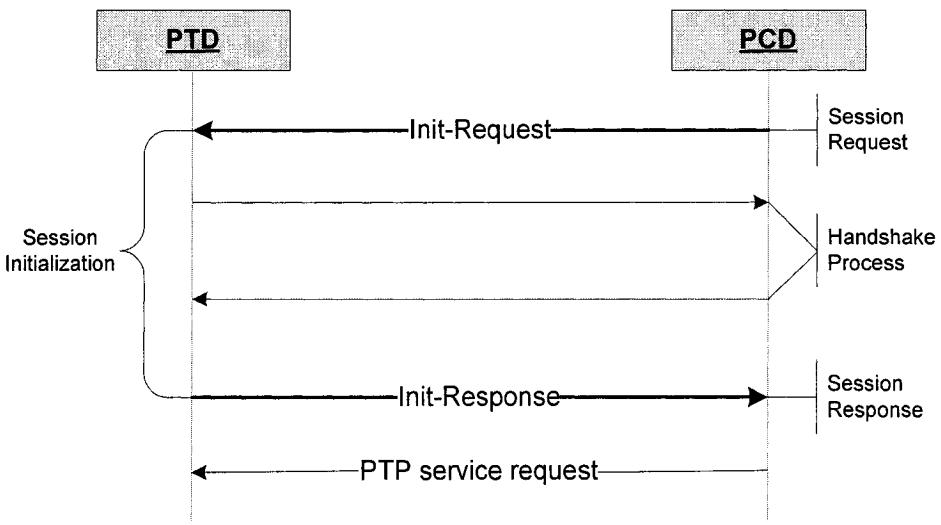


Figure 4-3 LTP session initialization

4.3.1 Session Request Initialization

In PTP, PCD performs PTP.init (initRequest) primitive to request a new session from a PTD and uses the QoS element in initRequest to negotiate security services with PTD. The QoS element contains only one element with Boolean type - "v1IntegrityProtection" - to indicate whether the integrity protection is needed. In LTP, we need to extend QoS to support the handshake configuration. Therefore, a new element , "handshake", will be added into the QoS type, and it has a complex type - "ptp:handshake". The handshake is an optional element. If it does not exist in QoS, the handshake transactions will be skipped. The handshake type contains only one element: mode. The "mode" specifies the handshake type - either public mode or personal mode. If PTD can handle the handshake requested by a PCD, it will start the handshake. Otherwise, PTD will return a "status" message to show the reason why it cannot accept the handshake process. After the session request phase, PTD and PCD will start the handshake process.

4.3.2 Handshake Process Details

4.3.2.1 Certificate-Based Handshake

The certificate-based handshake is designed for public mode, in which both PTD and PCD own valid certificates. The certificate-based handshake process is built on SSL's handshake protocol and furthermore is adjusted and reduced to fit for PTP's requirements.

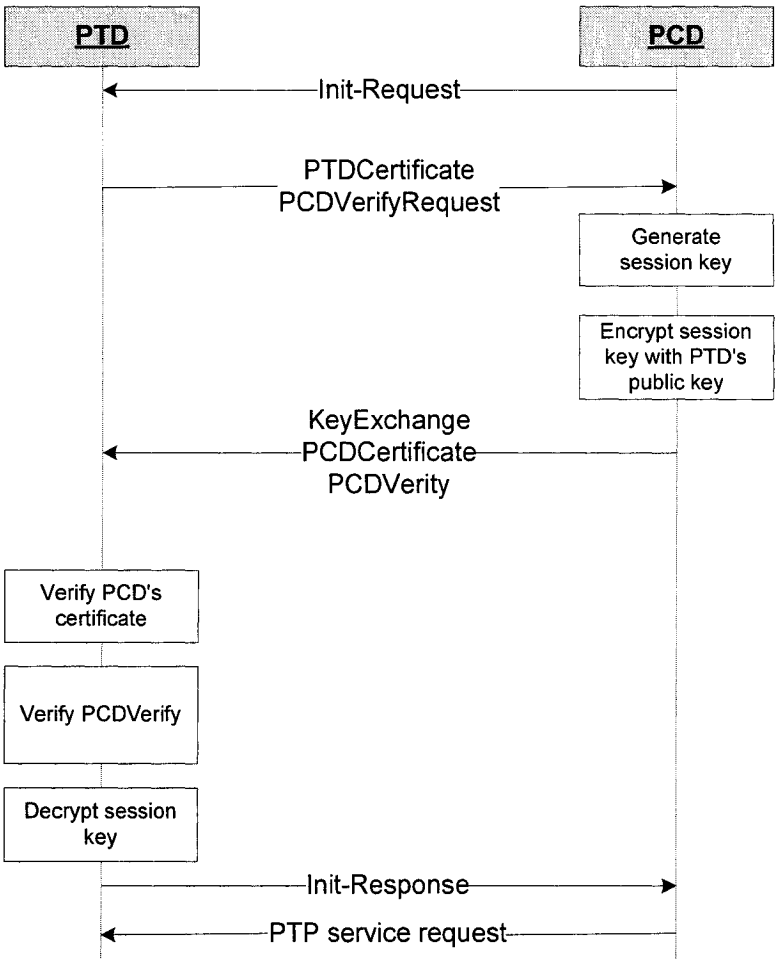


Figure 4-4 Certificate-based handshake process

The certificate-based handshake starts with an InitRequest from PCD to PTD, which specifies the handshake mode. In this case, it should be PUBLIC. If both sides are implemented correctly, PTD should response with PTDCertificate message, i.e. the PTD user's certificate. Following PTDCertificate, a PCDVerifyRequest message is sent if the PCD's authentication is needed. The PCDVerifyRequest contains a 160-bit random number.

After receiving PTD's response, PCD will generate a 128-bit random number that is used as the session key to protect the following messages' confidentiality and integrity.

Then the session key is encrypted by the PTD's public key extracted from its certificate. The session key encryption conforms to CKM_RSA_PKCS. After that, the PCD sends this encrypted session key to PTD. PCD will also send its certificate and verification information – PCDVerify, which is encrypted PCDVerifyRequest by PCD's private key. The PCDVerify generation conforms to CKM_RSA_PKCS.

Upon receiving PCD's response, PTD will verify the validity of PCD's certificate and authenticate the PCD by means of the PCDVerify. Then the PTD will decrypt the session key from KeyExchange Message. In fact, this step is used to implicitly verify whether the user is the real holder of the certificate he presented. If PTD can decrypt the session key, it means the user is the real holder of the certificate. Now the handshake finishes, and both PTD and PCD share the same 128-bit secret key. In addition, after the PTD verifies the PCD's certificate, it will show the PCD's identity information, for example, a serial number or a room number extracted from the PCD's certificate, on its display so that the user can compare the PCD's identity information showed on his PTD with that shown on PCD's nameplate, which is always a fixed part of the PCD for security reasons. This is the main way for a user to identify whether his PTD is talking to an intended PCD.

4.3.2.2 Password-Based Handshake

The password-based handshake is designed only for personal mode operation, in which only PTD owns a certificate. A shared password is used for device pairing and session key exchange. L. Buttyan proposed a protocol named SECTUS to solve a similar problem [3]. SECTUS stands for secure transfer via the user, which is developed especially for the purpose of device pairing over short-distance wireless links. In L.

Buttyan's project, SECTUS is used to exchange a long-term key to provide communication security between a SIM card inside a mobile phone and a PC. The shared password is generated by the SIM card and is transferred to the PC via the user, i.e. the user reads the password from the mobile phone's LCD and inputs it into a PC. Then a long-term key exchange process can proceed via the short password. The SECTUS protocol is modified and incorporated into the password-based handshake process.

1. Password-Based Authentication

Before we jump into the handshake transaction, the password-based authentication and key exchange protocol should be examined first. In SECTUS, the password-based authentication process adopts Challenge-Response authentication protocol, which is summarized as below.

- The authenticator sends a "challenge" message to the peer.
- The peer responds with a value calculated using a "one-way hash" function with the password.
- The authenticator checks the response against its own calculation of the expected hash value. If the values match, the authentication is acknowledged; otherwise, the connection SHOULD be terminated.

In this way, the password does not need to be sent over the wireless channel. In SECTUS, the first step is omitted. The SECTUS's password generated by PTD is a one-time password, which means that, when a PCD requests a new session, a 6-digit random number is generated as the password, and the next time the password should be different from the former one. Therefore, it is impossible for the authentication process to suffer "playback" attacks. So the challenge message could be anything known to both PTD and

PCD. SECTUS uses HMAC as the message authentication algorithm to authenticate both communication peers.

2. Session Key Exchange

After both PTD and PCD authenticate each other via Challenge-Response protocol, they possess a shared secret, which could be used to protect the communication channel. However, since we need the user to transfer the password from PTC to PCD, it should not be longer than 4 characters or 6 digits [3]. In fact, the key derived from the password, which has only 17 bits for 4 characters and 20 bits for 6 digits, is too short to maintain the encrypted messages uncovered for a longer period until the messages become valueless. A desired key size is 128 bits long at least. Therefore, another longer key needs to be safely negotiated and exchanged. If the password is used to encrypt the new key, the above-mentioned problem still exists. Bellovin and Merritt offered an Encrypted Key Exchange (EKE) protocol to safely exchange the session key, which represents a type of protocol, working in the case that both sides have a shared short password and one side has a public-key pair and enabling the safe session key exchange with the help of one peer's public key [22]. They use different algorithms; however, they share the same essences:

- A has a pubic key pair. The public key or a certificate is sent to B.
- B encrypts the new session key with A's public key.
- A decrypts the session key and starts the new secession.
- The message exchanged between A and B should be protected by the shared password with symmetric encryption, HMAC or another algorithm.

In fact, SECTUS is a variation of EKE protocol. SECTUS adopts the HMAC

algorithm to realize authentication and integrity protection.

3. Handshake Transactions

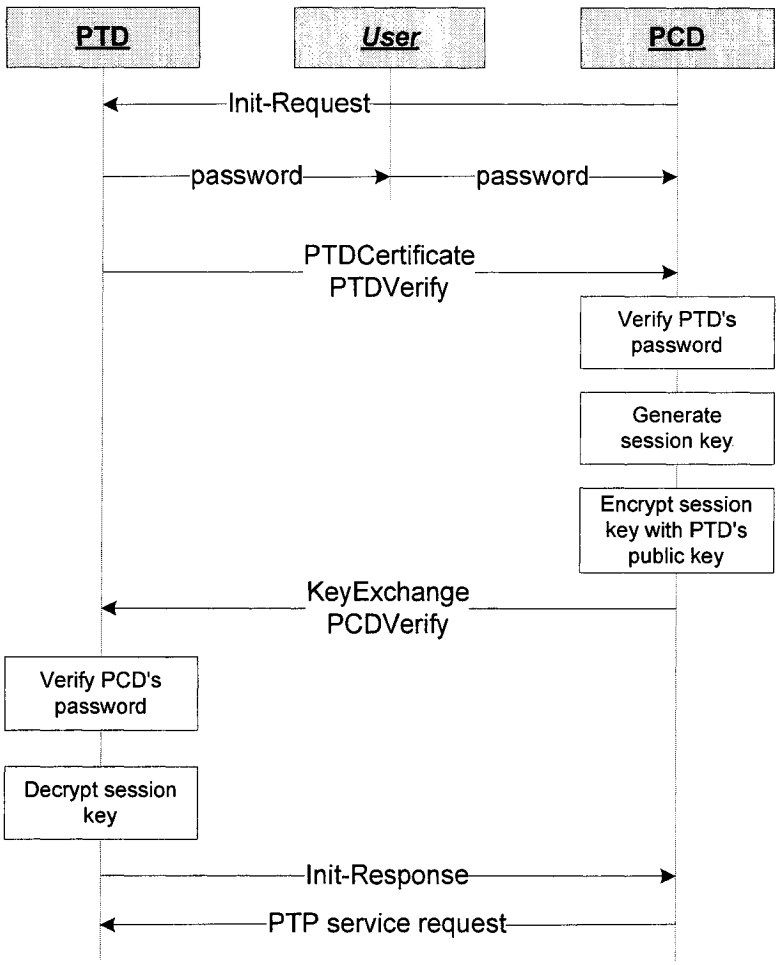


Figure 4-5 Password-based handshake process

The handshake process starts with an InitRequest from PCD to PTD, which specifies the handshake mode. In this case, it should be DPM. If both sides are implemented in the right way, PTD should response with a PTDCertificate message, i.e. the PTD user's certificate. Following PTDCertificate, a PTDVerify message is sent. The PTDVerify is the PTDCertificate's HMAC calculated by the SHA1 algorithm, which is twofold: showing the certificate really comes from the PTD with the password and

providing integrity protection for PTDCertificate.

After receiving PTD's response, the PCD calculates the HMAC of PTDCertificate with the same password to verify whether the message sender knows the password or, in other words, it is the real PTD. Unlike the certificate-based handshake process, in this step, PCD does not need to verify the user's certificate since the certificate acts only as a tool to help to transfer a session key. Then PCD generates a 128-bit random number that is used as the session key to provide the message's confidentiality and integrity. The session key will be encrypted by the public key extracted from the PTD's certificate. After that, the PCD sends this encrypted session key called KeyExchange to PTD. PCD will also send PCDPasswordVerify to PTD as the other part of Challenge-Response protocol. The PCDPasswordVerify is calculated by HMAC-SHA1, which assures the PTD that the KeyExchange is really from the real PCD.

In succession, the PTD will verify PCDPasswordVerify and decrypt the session key from KeyExchange. Now both PTD and PCD share the same 128-bit session key, and the handshake process finishes.

4.3.3 Session Response

After a PTD receives the session key created by PCD, the handshake process will enter the session response phase. In this phase, the PTD will send an InitResponse with a new session ID to the PCD, which indicates the completion of the session initialization process. Since the session key has been exchanged, InitResponse and the following messages will be provided with confidentiality and integrity protection. Then all PTP services can be called.

4.4 Handshake Message Definitions

In Section 4.2, the Certificate-based and Password-based handshake processes have been described. In this part, the message format will be defined as well. Conforming to PTP, messages use an XML format, which is defined by an XML schema. In this way, multiple messages in one-way transmission can be combined in a complex XML, so that the two different handshake processes can be incorporated into one standard protocol with the same process and message types. This idea is illustrated in Figure 4-6.

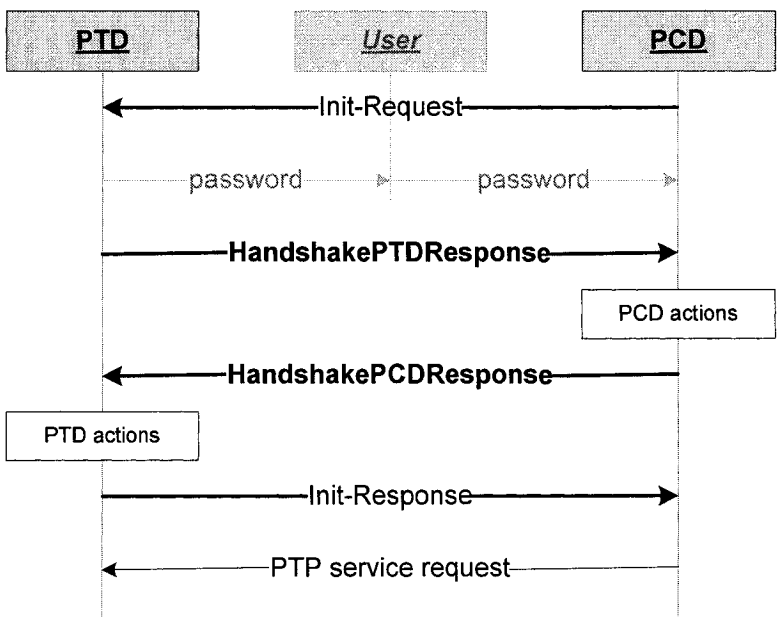


Figure 4-6 Standardized handshake process

In the standardized handshake process, the interaction between InitRequest and InitResponse is abstracted as two message's exchange: HandshakePTDResponse and HandshakePCDResponse. They are both derived from ptp:Response type, including a

status attributes to show the status of the response.

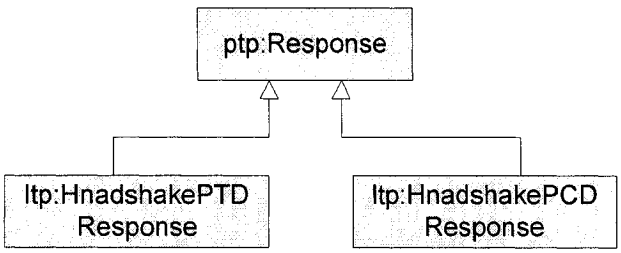


Figure 4-7 Handshake message object hierarchy

ltp:HandshakePTDResponse, ltp:HandshakePCDResponse and related sub-types will be defined as well as the extended QoS and ltp:Handshake mentioned in the handshake initialization and finalization phases.

4.4.1 ptp:QoS and ltp:Handshake

As mentioned in session request phase, a "handshake" element with

```
<complexType name="QoS">
  <annotation>
    <documentation xml:lang="en">
    ...
    </documentation>
  </annotation>
  <sequence>
    <element name="v1IntegrityProtection" type="boolean"/>
    <element ref="handshake" type="ltp:Handshake minOccurs="0"/>
  </sequence>
</complexType>
```

```
<complexType name="Handshake">
  <annotation>
  <documentation xml:lang="en">
  This type is intended for negotiations relating to the handshake process. When sent
  inside QoS in an InitRequest, this indicates the choice of a PCD.
    </documentation>
  </annotation>
  <sequence>
    <element name="mode" type="string"/>
  </sequence>
</complexType>
```

"ltp:Handshake" type will be added into ptp:QoS type. The extended ptp:QoS and ltp:Handshake are defined as follows.

4.4.2 ltp:HandshakePTDResponse

HandshakePTDResponse represents the handshake message sent from PTD to PCD. To support the need of both certificate-based and password-based handshakes, ltp:HandshakePTDResponse should consist of three elements: PTDCertificate, PCDVerifyRequest and PTDVerify. The relationship between HandshakePTDResponse and its elements are illustrated in Figure 4-7.

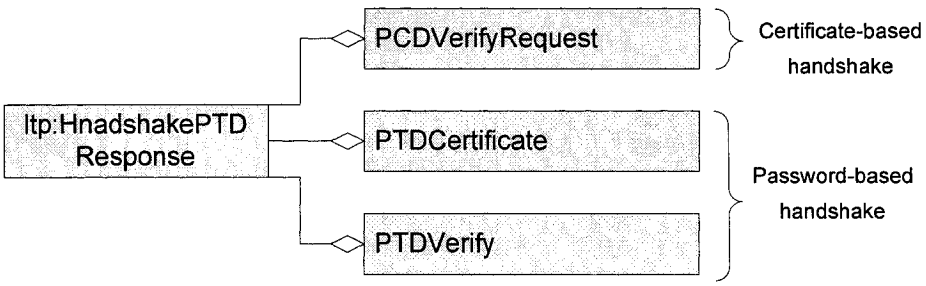


Figure 4-8 Relationship between HandshakePTDResponse and its elements

The PCDVerifyRequest elements work for certificate-based handshakes; PTDCertificate and PTDVerify work for password-based handshakes. PCDVerifyRequest is a 28 bytes random number encoded in base64 binary format. According to PTP, since only x.509 certificates are supported, PTDCertificate follows the x.509 format encoded in the base64 binary format. In PTP, RSA is the only supported public key algorithm [5].

```
<complexType name="HandshakePTDResponse">
  <annotation>
    <documentation xml:lang="en">
    This type is intended for PTD to send a handshake response.
    </documentation>
  </annotation>
  <sequence>
    <element name="PTDCertificate" type="base64Binary" minOccurs="1"/>
    <element name="PCDCertificateRequest" type="base64Binary"/>
    <element name="PTDVerify" type="base64Binary"/>
  </sequence>
</complexType>
```

4.4.3 ltp:HandshakePCDResponse

HandshakePCDResponse represents the handshake message sent from PCD to PTD. To support the need of both certificate-based and password-based handshakes, ltp:HandshakePCDResponse should consist of three elements: KeyExchange, PCDCertificate and PCDVerify. The relationship between HandshakePCDResponse and its elements are illustrated in Figure 4-9.

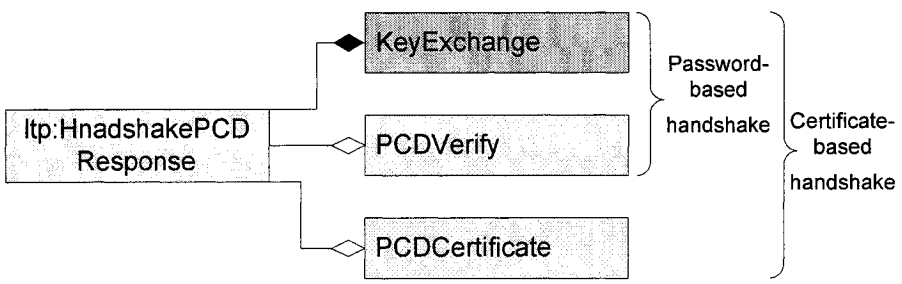


Figure 4-9 Relationship between HandshakePCDResponse and its elements

The KeyExchange, PCDCertificate and PCDVerify elements work for certificate-based handshakes; the KeyExchange and PCDVerify work for password-based handshakes.

```
<complexType name="HandshakePCDResponse">
  <annotation>
    <documentation xml:lang="en">
    This type is intended for PTD to send a handshake response.
    </documentation>
  </annotation>
  <sequence>
    <element name="KeyExchange" type="hexBinary" minOccurs="1"/>
    <element name="PCDCertificate" type="base64Binary"/>
    <element name="PCDVerify" type="hexBinary"/>
  </sequence>
</complexType>
```

4.5 LTP Security Implementation

After defining the interactions and message format in the handshake process, we need to examine how to implement the security services: authentication, confidentiality and integrity protection. The authentication service is included in the LTP handshake process, where PTD and PCD identify each other by means of certificates or a shared password. Therefore, we will focus on how to implement the confidentiality and integrity protection in LTP.

4.5.1 Integrity

In PTP, the XML signature is adopted to provide communication messages with integrity protection. In both Request and Response types, a ds:Signature element, referenced from the XML Digital Signature standard, is defined to hold the message's signature. PTP uses HMAC-SHA1 to generate a signature. The HMAC key is derived with the same key generation function as the one mentioned in Chapter 3. The only difference in LTP is that the 128-bit session key, exchanged in the handshake process, is used as the master key instead of the link key derived from the communication channel.

4.5.2 Confidentiality

PTP does not involve communication confidentiality, which is provided by Bluetooth protocol. In LTP, since all messages are in XML format, XML encryption is a straightforward solution [23]. To apply XML Encryption to LTP messages, some notes are outlined as below:

- All elements in XML messages are encrypted, including ds:Signature.
- The XML encryption adopts the DES (Advanced Encryption Standard) algorithm. The 128-bit session key is used as the encryption key.
- No encryption key information is included in encrypted XML, since the session key is already known to both PTD and PCD.

5 J2ME technologies Backing up the Development of the PTP Server

In this chapter, several popular development technologies on mobile phones will be compared, and then the reasons why J2ME CLDC and MIDP are chosen as the PTP server's development platform will be presented. And related techniques supported in the development, deployment and security protection will be discussed. In the end, a few SE implementations will be introduced and analyzed.

5.1 Mobile Phone Development Techniques

In the face of increasing competition and declining revenue per subscriber, mobile network operators are pinning their hopes on value-added services to increase usage and customer loyalty. Subsequently, different parties in mobile commerce have proposed and implemented various techniques to develop such services. Those techniques have different characteristics and respective application territories. Some popular techniques are SIM Toolkit (STK), Wireless Application Protocol (WAP) and Java 2 Platform, Micro Edition (J2ME).

5.1.1 STK

An STK application running in the SIM card of a GSM phone could drive the handset's display, utilize handset local resources and access the GSM network. Generally, STK applications use SMS as service bearers. STK enables MNOs to quickly deploy value-added services via the SIM card in spite of the differences in customers'

handsets. Therefore, the STK technique is widely accepted by GSM operators. STK technology has an apparent drawback: STK is a GSM-specific technology, so it cannot be a common solution for all mobile phones.

5.1.2 WAP

WAP is a browser-based solution used to provide text content to wireless devices like cell phones and PDAs. WAP incorporates existing web protocols and furthermore optimizes them to meet the requirements of the low bandwidth, high latency conditions in a wireless environment. WAP 1.2 starts to embrace Wireless Public Key Infrastructure (WPKI) to enable secure transactions. Further, WAP 2.0 adopts standard WEB techniques like XHTML, TCP, and SSL etc, which make WAP more powerful and secure. Now WAP has been adopted and supported by most telecommunication operator and handset manufacturers. A major disadvantage of WAP is the lack of local processing capability, even with the help from WMLScript. The WAP browser cannot freely access the OS resource, for example, cryptography and the Bluetooth protocol stack.

5.1.3 J2ME

J2ME is the Java platform for consumer and embedded devices such as mobile phones, PDAs, TV set-top boxes and in-vehicle telematics systems etc. The J2ME architecture defines configurations, profiles and optional packages as elements for building complete Java runtime environments that meet the requirements for a broad range of devices and target markets [24]. The combination of CLDC and MIDP is especially defined for mobile phones and low-end PDAs. CLDC defines a small-footprint Java virtual machine called KVM and a set of compact class libraries for devices with

intermittent network connections and low power consumption. MIDP defines the whole lifecycle of a MIDlet suite, which consists of discovery, installation, update, invocation and removal [25]. MIDP also provides a rich user interface, extensive network connectivity, local data storage, etc., which enables MIDlets to be competent for a broad range of enterprise mobile applications. J2ME optional APIs are defined to meet field-specific requirements like cryptography, Bluetooth and web service. Nowadays, most middle-to-high end mobile phones, like Nokia 6600 and Motorola E380 [26], have supported CLDC 1.1 and MIDP 2.0.

Generally, STK, WAP and J2ME are the main development techniques. STK is only applied in GSM mobile phones with an embedded SIM card. WAP and J2ME are the two dominant development techniques supported in most mobile phones. WAP is a browser-based solution - all transactions are performed by means of a WAP browser. It is easy to make a new service available to mobile phones without any deployment work. However, a WAP browser is devoid of local processing - it cannot freely access a mobile phone's resources, for example, cryptographic or Bluetooth API. Currently, J2ME seems to be the most promising development platform for mobile phones. J2ME is a cross-OS platform, which has been supported in all primary OS like Symbian, Smart Phone and Linux. With the support of CLDC, MIDP and rich optional APIs, J2ME is competent for GUI, network communication, cryptography, Bluetooth, etc.

5.2 Build a PTP Server on J2ME

An appropriate development platform is crucial to build a PTP server that can work well in practice. The platform needs to meet both functional and systematic requirements. Functionally, to back up PTP's implementation, the application platform needs to support OBEX protocol, provide cryptographic algorithms, access security elements and parse XML messages. Systematically, the platform needs to provide necessary mechanisms to ensure that the PTP server can be easily deployed and provide the necessary security features.

According to the introduction of mobile phone development techniques in Chapter 3, we know that the J2ME CLDC and MIDP is an appropriate platform to implement PTP server. On one side, J2ME has strong local processing capabilities with rich APIs. By means of JABWT and SATSA, J2ME standard optional packages, J2Me can meet the PTP server's functional requirements on OBEX communications, cryptography and SE access. Although J2ME has not defined a standard XML API yet, there are some compact and powerful third-party APIs available [27]. On the other hand, as a member of the Java family, J2ME applications could run on any J2ME-enabled mobile phones with different OSs. Furthermore, MIDP defines the Over the Air Provisioning (OTA) standard [25], by means of which an MIDP application called MIDlet can be conveniently downloaded to a mobile phone via mobile networks. In addition, CLDC and MIDP provide security protection to ensure that MIDlets run securely in a control fashion. As a whole, J2ME platform provides the development of the PTP server with comprehensive support (Figure 5-1). Those techniques will be discussed in the following sections.

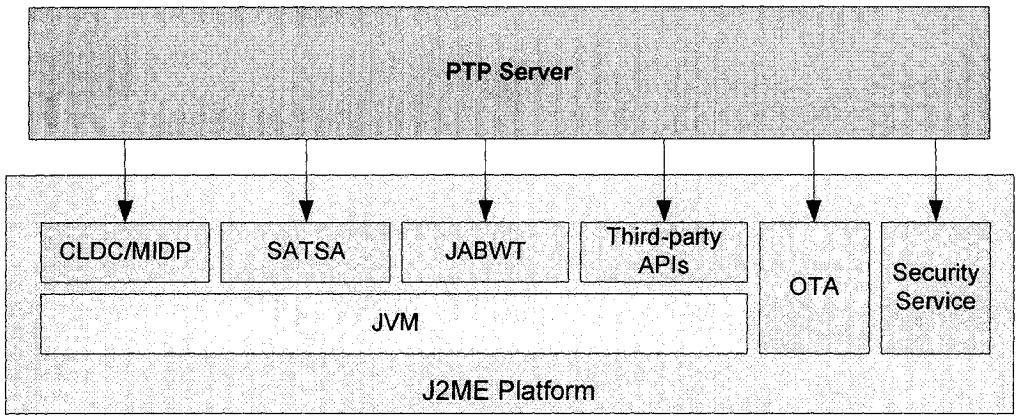


Figure 5-1 J2ME platform backing up the PTP server

5.3 J2ME Development Techniques

Crucial techniques enabling the PTP server to run on J2ME will be discussed in this section.

5.3.1 Security and Trust Services API

Security and Trust Services API for J2ME (SATSA) is defined to provide J2ME applications with a consistent interface to access smart cards, general cryptography and signing services offered by a security element. SATSA consists of four optional packages that can be implemented independently [28]:

- SATSA-APDU and SATSA-JCRMI packages support communication with smart card applications using the APDU protocol and Java Card RMI protocol. [28]
- The SATSA-CRYPTO package provides a subset of the J2SE cryptography API, which supports basic cryptographic operations, such as signature verification, encryption, and decryption etc. [28]
- The SATSA-PKI package provides a Java interface to a security element, which supports digital signature and perform users, credential management.

The PTP server mainly relies on PKI and CRYPTO packages to perform PKI functions and cryptographic operations. PTP's object management primitives - PTP-StoreObject and PTP-ObjectInformation are defined to access and store user's certificates. These functions can be supported by the UserCredetialManager class in the PKI package. It enables a J2ME application to access a certificate or certificate URI in a

certificate store and perform adding and removing operations. PTP-Sign is the core primitive to provide signing services. For signing content, PTP-sign must work in WYSIWYS mode to enable users to confirm the data to sign. However, it is not necessary for authentication usage. CMSMessageSignatureService, another class in the PKI package, has two methods: authenticate and sign, which correspondingly support authentication usage in non-WYSIWYS mode and signing usage in WYSIWYS mode. In addition, during the handshake process discussed in Chapter 4, the PTP server also needs to perform some cryptographic operations like SHA1 digest and DES encryption/decryption. Those algorithms can be found in java.security and java.crypto packages. [24]

In fact, SATSA does not independently provide support for those PKI-related and cryptographic operations. It acts only as an interface to adapt J2ME applications to access real sources, for example, OS's cryptographic library or PKI functions provided by an SE.

5.3.2 OBEX Communication and JABWT

PTP employs Object Exchange (OBEX) protocol to transport messages between the client and the server. OBEX is a protocol developed by the Infrared Data Association for "pushing" or "pulling" objects to and from clients and servers. OBEX works just like a session-based binary HTTP protocol. OBEX can run over various physical communication protocols, which enable PTP to work with IrDA, Bluetooth and even TCP/IP etc. In PTP, PTD acts as an OBEX server and PCD acts as an OBEX client. The PCD will "put in" its request and "get" its response. Java APIs for Bluetooth Wireless

Technology (JABWT) are defined to provide J2ME applications with a consistent Bluetooth and OBEX development environment [29]. JABWT consists of two optional packages: Bluetooth API and OBEX API, which can be implemented independently. OBEX API screens all details of the underlying protocol from the user. Only a connection string is needed to indicate a specific protocol and other connection information. A standard OBEX connection string has three parts: {protocol}:[{target}][{params}]. In the case of OBEX over Bluetooth RFCOMM, the {protocol} is defined as btgoep because this is the implementation of the Generic Object Exchange Profile (GOEP). The {target} and {params} respectively indicate the Bluetooth device's address, service and security requirements. Examples of a client and server connection string are shown below:

```
Client: btgoep://0050C000321B:12
Server:btgoep://localhost:12AF51A9030C4B2937407F8C9ECB238A
```

Currently, Bluetooth API has been implemented in some smart phones like Nokia 6600 and Sony-Ericsson P900. However, OBEX API has not been supported yet.

5.3.3 XML Parsing

Since PTP messages are in XML format, the PTP server needs to be able to parse an XML document. When a PTP client sends a PTP request in XML, the PTP server needs to identify the service primitive's name and parameters from the XML request. So far, JCP has not defined a standard XML parser for J2ME. However, some open source parsers are available from third parties. Some of them are shown as below.

Table 5-1 Open source XML parsers

Name	License	Size	Type
ASXMLP 020308	Modified BSD	6 kB	push, model
kXML 2.0 alpha	EPL	9 kB	pull
kXML 1.2	EPL	16 kB	pull
Xparse-J 1.1	GPL	6 kB	model

Xparse-J 1.1 is chosen in my thesis, since it has a 6k footprint and supports the XML document model, which simplifies program development.

5.4 Over the Air User-Initiated Provisioning

MIDP 2.0 defines the "Over the Air User-Initiated Provisioning" (OTA) specification that allows MIDlet suites to be easily deployed onto user's mobile phones via a mobile network or other connections [25]. MIDP 2.0 requires an MIDP-compliant device to provide a discovery application (DA) and application management software (AMS) to combine to discover, download and install MIDlet suites. The DA is always the device's resident browser (e.g., i-mode or WAP), and the AMS is a special native application coming with the J2ME platform. The OTA process performs as following:

- The user uses DA to search a MIDlet suite and finally locate a link to an application descriptor, i.e. a JAD file.
- The user selects the link to inform the AMS to download the JAD.
- The AMS downloads the JAD and gets all of the information about the MIDlet suite, including the vendor, name, size and the location of the JAR file. These JAR files will be presented to the user, thereby allowing the user to select and install the chosen JAR file.
- The AMS starts to download the JAR file and install it. In case the JAR file is signed, the AMS has to verify the JAR file's signature to decide whether the JAR file really comes from the correct vendor claimed in JAD. All exceptions will be presented to the user.

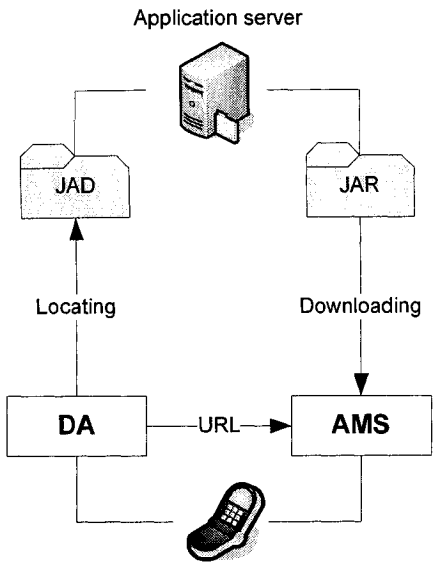


Figure 5-2 J2ME's OTA process

5.5 Security for MIDP Applications

J2ME CLCD and MIDP combine to provide a security model to ensure a MIDlet suite to work in a safe and controlled way. The security model is defined at two different levels: low-level security and application-level security.

5.5.1 Low-Level Security

An application running in a virtual machine must not harm or crash a device. Therefore, the CLDC specification requires that a Java virtual machine conforming to the CLDC standard must reject invalid class files [30]. This is guaranteed by the class file verifier, which ensures that the bytecodes and other items stored in class files cannot contain illegal instructions that could be executed in illegal order and cannot contain references to invalid memory locations or memory areas that are outside the Java object memory. Since the conventional J2SE class file verifier is not ideal for small, resource-constrained devices, the new class file verifier operates in two phases:

1. ***Off-device verification***. After compiled, class files have to be run through a preverifier tool in order to remove certain bytecodes and augment the class files with additional StackMap attributes to speed up runtime verification.
2. ***In-device verification***. At runtime, a verifier component of the virtual machine uses the StackMap attributes generated by the preverifier to perform the actual class file verification efficiently.

With this two-phase verification, the new runtime verifier in Sun's KVM requires only about 10 kilobytes and 100 bytes of dynamic RAM at runtime.

5.5.2 Application-Level Security

Application-level security means that a Java application can access only those libraries, system resources and other components that the Java application environment allows it to access.

In CLDC, application-level security is accomplished by using a metaphor of a closed "sandbox." An application must run in a closed environment in which the application can access only those libraries that have been defined by the configuration, profiles and other classes supported by the device.

MIDP 2.0 extends the "sandbox model" by introducing the concept of trusted applications [25]. A trusted MIDlet suite is allowed to access protected APIs, as indicated by the domain policy that consists of a set of permissions. An untrusted MIDlet suite must execute in the mistrusted domain where access to protected APIs either is not allowed or is allowed with explicit user permission. A MIDlet suite can only be trusted when it is signed using X.509 PKI. The signature of JAR and the certificate signing the JAR will be added to the application descriptor (JAD file) as attributes. A MIDlet will be mapped to the protection domain bound to the root certificate used to verify the signature. MIDP 2.0 requires that an implementation must support the X.509 certificate, RSA and SHA-1 algorithms to generate signatures.

5.6 Security Element Implementations

As mentioned in Chapter 3, the implementation of SE in the PTD or mobile phone is the crucial part that determines the security level of the PTP server. Since it is a

relatively independent topic, the detailed research will be saved for future works. However, to give the reader a complete view of the proposed mobile signature solution, some existing SE implementations will be introduced and discussed.

Although security elements could be in various forms, even software, they must have strong tamper resistance. Currently, smart cards are the best choice of SEs since they are able to protect against hackers attempting to extract secrets from the integrated circuit inside the smart card. Several smart card-based SE solutions - SWIM, dual chip and dual slot- have been proposed and applied in practice. In addition, two pluggable solutions - external token and software SE - will be discussed.

5.6.1 SWIM

SWIM is a GSM-specific SE implementation: WIM is integrated in the SIM card, a smart card used to identify the subscriber to the GSM network, inside a GSM mobile phone. This solution can be deployed by upgrading each subscriber's SIM card to support WIM functionality. The major advantage is that subscribers donot need to change their phones. Therefore, it is cheap and convenient for users to accept. SWIM is also welcomed by mobile network operators, since they end up controlling the security of mobile commerce by dominating SIM cards' issue. In fact, service providers, especially financial companies, prefer more neutral solutions to avoid interference from third parties. In addition, SWIM does not work for CDMA phones without a SIM-like smart card inside.

5.6.2 Dual Chip and Dual Slot

The dual chip and dual slot solutions avoid using SIM cards by adding an extra

smart card chip or slot. Mobile phones with dual chips have one extra chip, other that SIM card to provide signing services. Mobile phones with dual slots are equipped with a second card reader slot; therefore, users can easily change smart cards without uncovering mobile phones. Both solutions try to get rid of mobile network operators' control. They are welcomed by financial companies. However, the major drawback is that users have to change their phones if they want to use a secure mobile service, which will seriously hinder the progress of switching to new services.

5.6.3 Pluggable SE Tokens

Based on the above introduction to the SWIM, dual chip and dual slot solutions, we know that a promising solution is to enable mobile service providers to independently deploy their SEs and to make sure that users do not need to invest in new mobile phones to switch to a secure mobile service. Recently, various pluggable SE solutions have been proposed to solve these issues. They are hardware or software tokens that can be dynamically "plugged" onto mobile phones when needed.

A hardware token is a separate device with an integrated smart card reader, which can be connected to a mobile phone via a standard interface like USB, or RS232. The idea is borrowed from the USB token in the PC world. Each service provider can issue its own token to the consumers of its service. Meanwhile, this solution defers a user's decision to pay for the security element until it is needed. However, the drawback is that the solution compromises convenience, since the consumer needs to take care of two separate devices. In some non-critical scenario, for example, access control in a company or university, software tokens can be considered. Since only software is involved, it is cheap and easy to produce and deploy. Service providers can easily create a software SE

to work with their applications. A software token is best implemented at the operating system level. The key material should be kept in the mobile phone's persistent storage, which can be the mobile phone's own non-volatile memory or external extended memory, for example, the secure digital card. In this case, the private key must be encrypted with a longer PIN code. However, the software token has inferior security to a smart card-based SE.

Pluggable SEs, including hardware and software tokens, solve those problems from SWIM. On the one hand, they enable mobile service providers to independently deploy their own SEs. On the other hand, they allow a user to freely choose his favorite phone model without thinking about the matter of SE. Therefore, pluggable SEs are preferred in my research work.

6 A Prototype for a J2ME-Based Mobile Signature Solution

The PTP-based mobile signature solution is mainly composed of a PTP server and a PTP client. As mentioned in Chapter 5, the PTP server will be built on a J2ME platform. Since a complete implementation of the proposed solution involves a lot of work, a prototype will be designed and implemented instead to demonstrate the essential points of the proposed solution. In this prototype, only the core primitive PTP-sign will be covered. The PTP session initialization process and communication security will be omitted. In addition, a PTP client used to test the PTP server will also be implemented in the prototype with only essential features. The interactions in the prototype are illustrated in Figure 6-1 to give readers a feel of what the prototype could do.

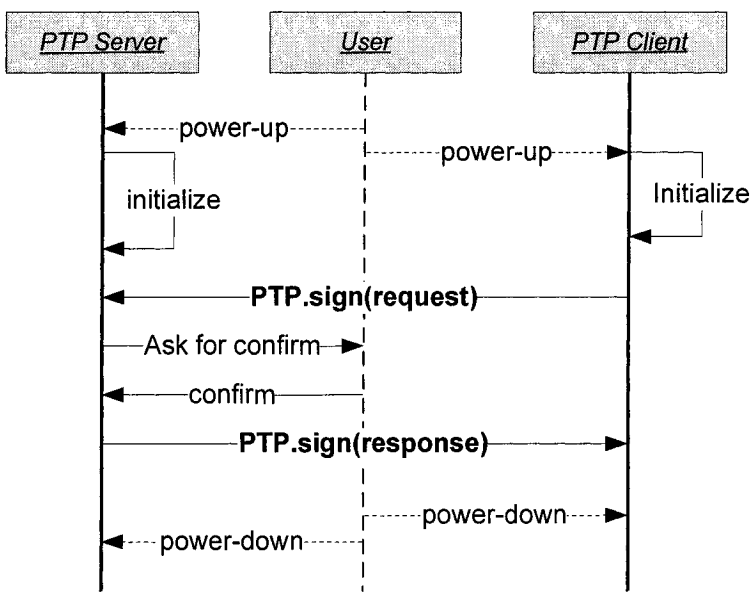


Figure 6-1 Interactions between a user, the PTP server and the PTP client

6.1 System Design

In this section, the main design issues will be discussed. Attention will be given to the PTP server side, since it is the focus of my research work. Although only a prototype will be implemented in the end, a nearly close to complete system design will be covered for future works. For example, a deliberate server-side API framework is proposed to make the design more modularized, flexible and easy to extend.

6.1.1 PTP Server

The PTP server is implemented as a J2ME MIDlet. It needs to handle all interactions with users, but also deal with all PTP service requests from PTP clients. As a network application, the PTP server will be designed as a multi-threading application, as shown in Figure 6-2. When a PTP server starts, a main thread called the PTP console is created to perform all system initializations first. Then the main thread will initialize the PTP API framework, which will start a new thread called the PTP thread. It will wait and serve all incoming PTP requests. The PTP thread is indispensable to allow for user interface response, since PTP services contain complicated cryptographic computation and lengthy network communications that may block the system reaction for a while. When the user issues a "stop" instruction via the user interface, the main thread will inform the PTP thread to terminate and then exit.

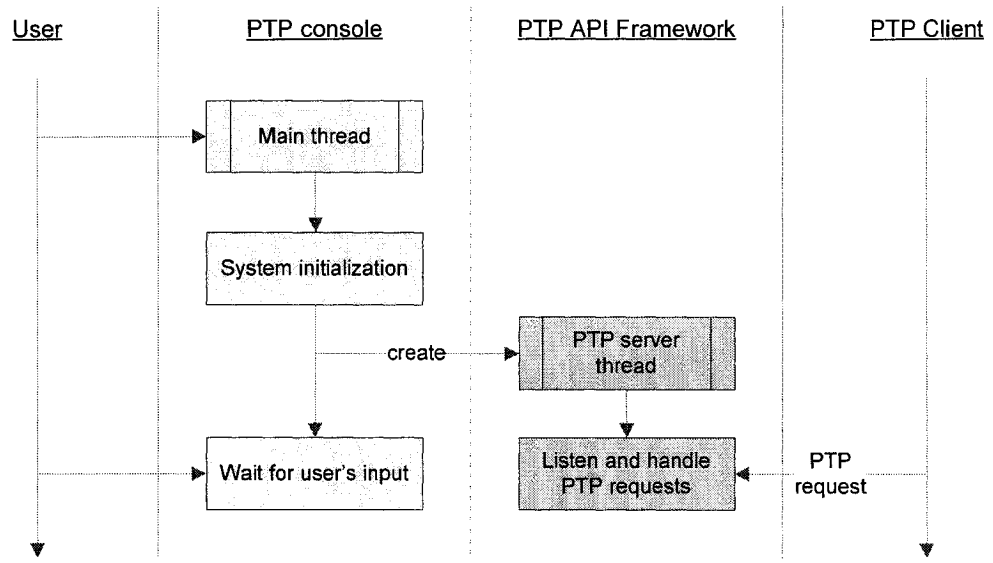


Figure 6-2 Multi-threading PTP server

6.1.2 PTP API Framework

In the implementation of PTP API, a framework is needed to intercept and interpret all service requests from a PTP client and execute corresponding PTP functions. The framework undertakes all fundamental works to handle a PTP request. The PTP server framework includes four main components: PTP listener, communication handler, security handler and command handler. The dependent relationship is shown in Figure 6-3s. When the PTP framework is initiated, a PTP listener is created to listen to the PTP service request from the PTP client. The PTP request for information will go through the OBEX handler, security handler and command handler. Finally, a specific PTP API will handle that request and return a PTP response, which will go back through the route, as shown in Figure 6-4.

- **PTP listener**. It is an OBEX listener that is initialized by the PTP thread. When a PTP request comes, it sends an OBEX connection to the communication handler to process the request. According to PTP, all OBEX requests are sent as put action.
- **OBEX handler**. It deals with all OBEX communication details and XML conversion. It extracts PTP request data from the OBEX request and converts it to an XML DOM object by means of the Xparse-J XML parser. When a PTP response is sent out, it does inverse operations.
- **Security handler**. It transparently handles all details of communication security. As mentioned in Chapter 4, after the PTP initialization process, a session key is created. The security handler will use this session key to create XML signatures and encrypt XML. When it reads an encrypted XML DOM object, it will decrypt it and verify its signature to decide whether the XML is tampered. Then it passes a new XML DOM in plain-text to the next step. Contrarily, when an XML DOM needs to be sent out, it will create an XML signature and then encrypt the whole XML.
- **Command handler**. It acts as a command dispatcher. It analyses the top element's name of an incoming XML and then calls a corresponding PTP function. After the PTP function finishes, it returns the PTP response in XML format to its caller.

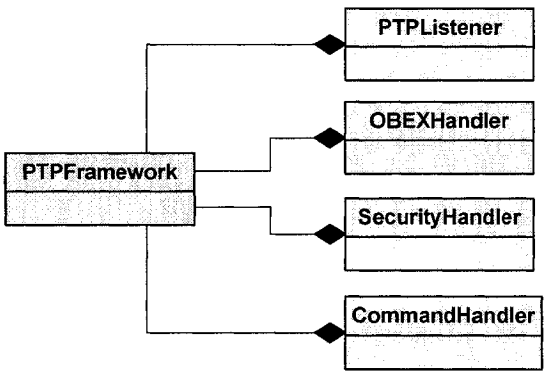


Figure 6-3 Dependent relationship of the PTP framework and its components

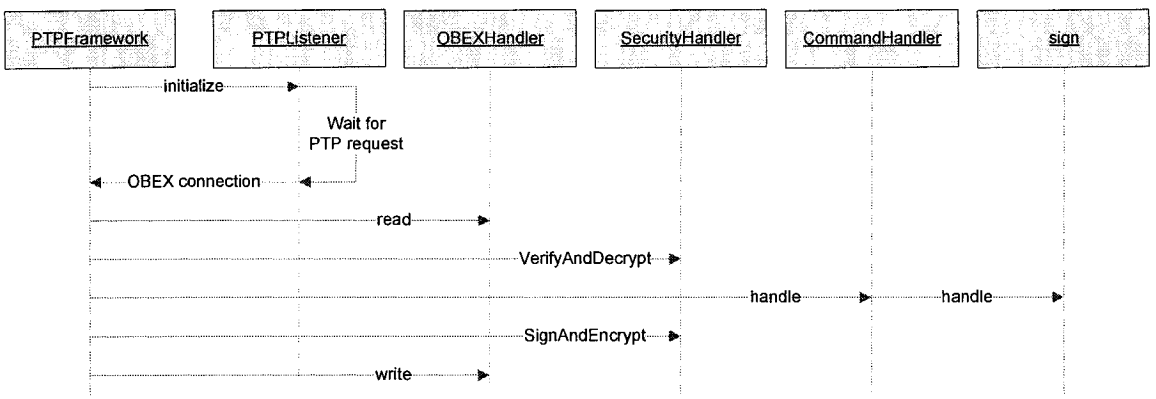


Figure 6-4 Sequence diagram of a PTP request going through the PTP framework

6.1.3 PTP-sign Service Primitive

The PTP-sign is the core service primitive in PTP. For demonstration purposes, it is the service one implemented in the prototype. According to PTP, it needs to support WYSIWYS and non-WYSIWYS modes. When the PTP-sign works in both WYSIWYS mode, it is supposed to provide non-repudiation service. The document to sign must be presented to the user to ask for confirmation. When the PTP-sign works in non-

WYSIWYS mode, it is intended to take part in an authentication process like WTSL. Both signing modes will be implemented in the prototype.

6.1.3.1 PTP service command

Under the PTP API framework, each PTP service primitive will be correspondingly implemented as a class, called the service command, with the same name as the service primitive. In order to work with the command handler, all commands following the command pattern implement the same PTPCommand interface (Figure 6-5), which has only one method. The process method takes a parameter and a return a value both with the XML document type. For example, the command sign is to realize PTP-sign primitive.

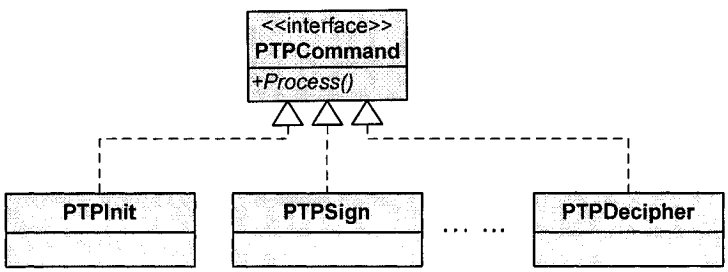


Figure 6-5 All PTP service commands implement the PTPCommand interface

6.1.3.2 Implementation of CMSMessageSignatureService class

The PTP-sign relies mainly on the signing functions of the CMSMessageSignatureService class from Security and Trust Services API. SATSA defines this particular class to generate application-level digital signatures that conform

to the Cryptographic Message Syntax (CMS) format by means of an SE implementation. CMSMessageSignatureService splits the generation of formatted digital signatures between two methods, namely the sign and authentication. The sign method is associated with the non-repudiation key, and the authentication method is associated with keys marked for the authentication key or other usage. In addition, these two methods interact with both PTP-sign's WYSIWYS and the non-WYSIWYS mode. Therefore, the PTPSign needs only to act as an adaptor of CMSMessageSignatureService, i.e. simply calling authentication or the sign method.

As mentioned in Chapter 5, since an appropriate SATSA implementation is not currently available, CMSMessageSignatureService needs to be implemented in the prototype. To make things simple, the signature service is implemented directly in either authentication or the sign method rather than in an SE, as shown in Figure 6-6. Key materials will be pre-kept in resource files that can be loaded by CMSMessageSignatureService when needed. Bouncy castle for J2ME is used to provide all cryptographic algorithms like SHA1 and RSA. [31]

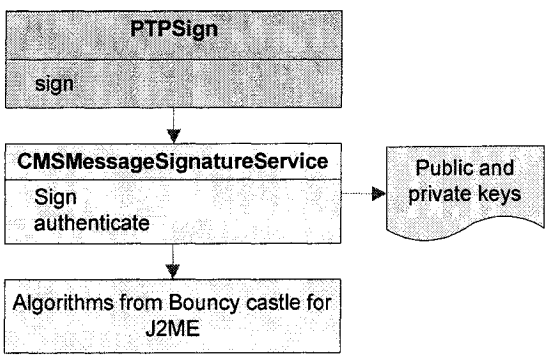


Figure 6-6 PTPSign implementation

6.1.4 PTP Client

A PTP client is needed to test the PTP server prototype. To minimize the implementation work, the PTP client will be included in the same MIDlet with the PTP server, sharing most common codes. In addition, since both the PTP client and the server are MIDP applications, we can directly develop and test OBEX communication between the PTP client and the server by means of the OBEX support offered by Sun's MIDP emulator.

6.2 Prototype Implementation

6.2.1 Development Platform

PC	Intel Pentium 4 2.66GHz 512 M of RAM Samsung SP0411N 40GB IDE Disks D-Link DEF-530TX PCI Fast Ethernet Adapter ENVISION EN7100si LCD Monitor
OS	Microsoft Windows XP sp2
JDK	Sun Java 2 SDK, SE v1.4.2_06
J2ME emulator and tool	Sun J2ME Wireless Toolkit 2.2
IDE	Borland JBuilder X Enterprise Trial 10.0.176.0

6.2.2 Source Code List

signService package

- SigServiceMIDlet.java	Prototype main class controlling MIDlet's life cycle
- ServerGUI.java	Server-side user interface handling all input and output interaction with the user
- ClientGUI.java	Client-side user interface handling all input and output interaction with the user
- ClientProcessor.java	A client-side separate thread to perform PTPSign request

PTP/server package	
- PTPFramework.java	PTP framework controller class
- PTPListener.java	PTP framework listener class
- OBEXHandler.java	PTP framework OBEX handler class
- SecurityHandler.java	PTP framework security handler class. Note: Since security implementation is omitted in this prototype, the functions in SecurityHandler return only the same XML object without any processing
- CommandHandler.java	PTP framework command handler class
PTP/server/service package	
- PTPSign.java	PTP sign service implementation class
PTP/client package	
- PTPSign.java	PTP client-side sign request class
SATSA package	
- CMSMessageSignatureService.java	SATSA CMSMessageSignatureSevice class implementation
Icons	
- signService.png	MIDlet icon

6.2.3 Make signService.jar

Jbuilder's archive builder wizard is used to make signService.jar. Following the wizard's instructions step-by-step, a MIDlet archive project can be created. When building it, the archive project will automatically the compile source codes, preverify the byte code created and then obfuscate the jar file with RetroGuard. After "make project" is performed, a signService.jar with 135,453 bytes is created. According to the configuration of the archive project during creation, a MANIFEST.MF file and a signService.jad file are created, as shown in the following.

MANIFEST.MF

```
Manifest-Version: 1.0
MIDlet-Name: Mobile signing service
MIDlet-1: SigServiceMIDlet, , SigService.SigServiceMIDlet
MicroEdition-Configuration: CLDC-1.1
MIDlet-Icon: /icons/sigService.png
MIDlet-Permissions: javax.microedition.io.Connector.*
MIDlet-Vendor: Wei Du
MIDlet-Version: 1.0
MicroEdition-Profile: MIDP-2.0
```

signService.jad

```
MIDlet-1: SigServiceMIDlet, , SigService.SigServiceMIDlet
MIDlet-Icon: /icons/sigService.png
MIDlet-Jar-Size: 135453
MIDlet-Jar-URL: signService.jar
MIDlet-Name: Mobile signing service
MIDlet-Vendor: Wei Du
MIDlet-Version: 1.0
MIDlet-Permissions: javax.microedition.io.Connector.*
```

6.2.4 Code Signing

To make the MIDlet of mobile signature prototype trustworthy, it must be signed. J2ME Wireless Toolkit 2.2 is used to sign the signService MIDlet. The detailed signing procedure is shown below:

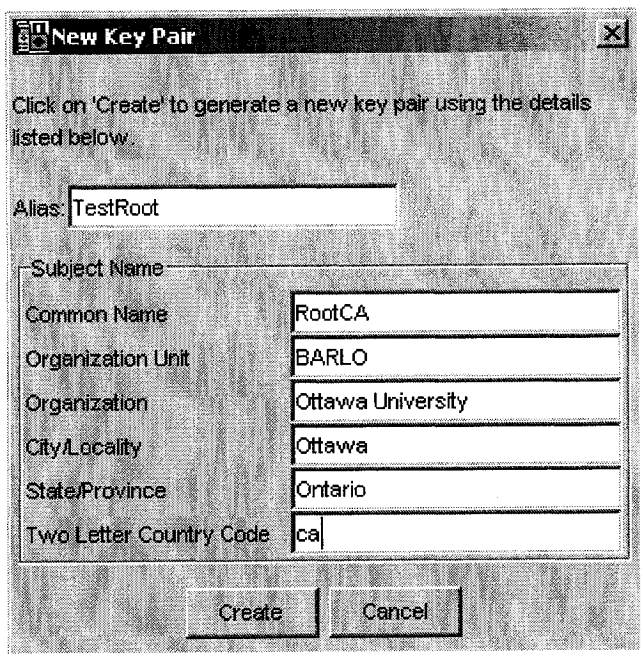
1. Activate the "sign MIDlet suite" function

   In KToolbar, first open the signService project, select "sign" from the project menu and then the "sign MIDlet suite" windows will pop up.

2. Generate a key pair and a certificate

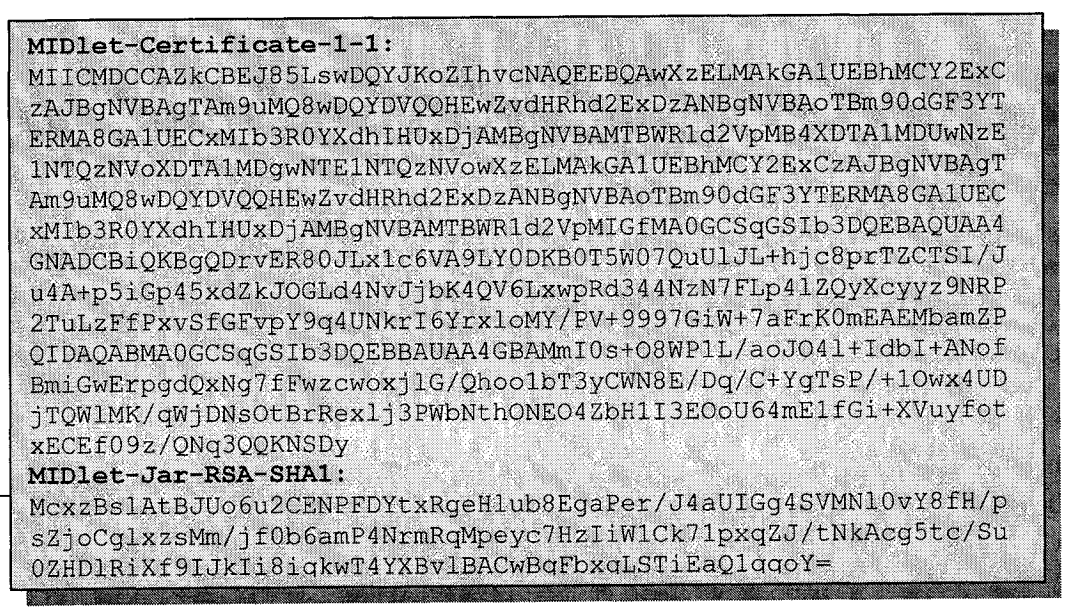
   Select "new key pair..." from the Action menu and input all of the

necessary information for a new root certificate in the pop-up window as below. When clicking "Create", a pop-up window appears to ask you to select a security domain. Select "Trusted" and click "OK". Now a new key pair and a root certificate are created.



3. Sign MIDlet suite

Now it is ready to sign the signService MIDlet. Select "Sign MIDlet suite" from the action menu. A pop-up window will tell you "MIDlet suite signed."



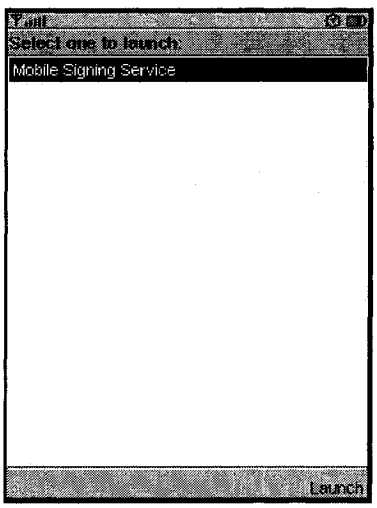
Now you open the signService.jad; you will find that two items are added into it, which are the root certificate and the MIDlet's signature.

6.3 Running and Result

After the MSS MIDlet suite is signed, it is ready to run in the Wireless Tool kit (WTK) emulator. Each running step and result will be shown below.

1. Launch the MSS MIDlet suite

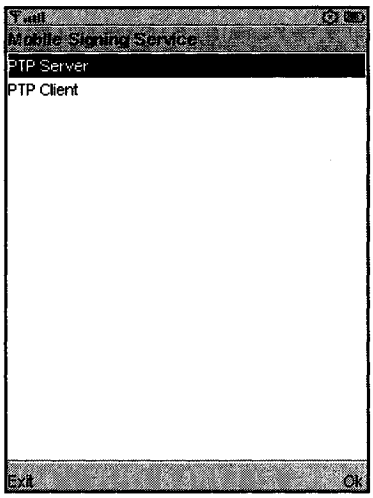
In addition to supporting OTA provisioning via HTTP, WTK 2.2 provides another easy way to launch a MIDlet suite locally. To do so, simply click the "Run" button on the KToolbar, and then an emulator will be started with the currently active MSS MIDlet suite. Click "launch" to start the only mobile signing service MIDlet.



2. Choose the PTP server or the PTP client

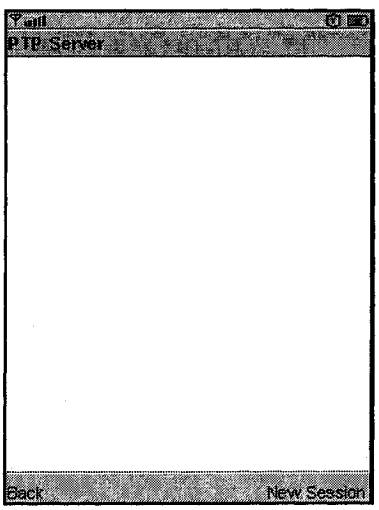
Since both the PTP server and the client are implemented in the same MIDlet, the following screen will show you two choices: PTP server and PTP client. Now you have

to choose which part you want to run in the current emulator.



3. Activate the PTP server

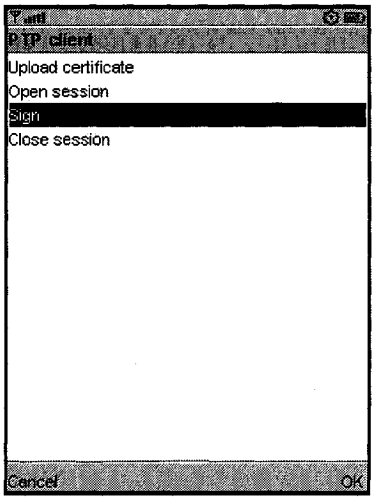
Suppose you choose to active the PTP server first. You will see a blank server console. Now the PTP server is ready to serve.



4. Activate the PTP client

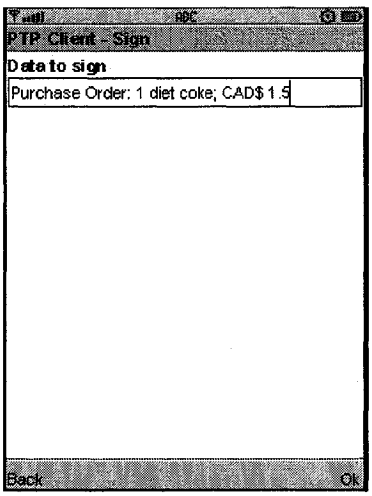
To activate a PTP client, you have to repeat steps 1-2 to launch another MSS

instance. Then you choose to activate the PTP client. A function list will show up in the following screen. However, only the "sign" function is implemented.



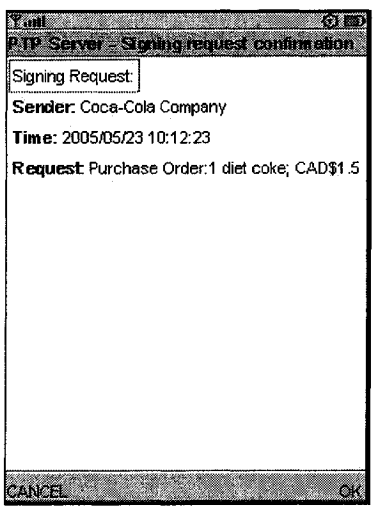
5. PTP client: Request a signing request

Click "Sign" and press "OK" to access the sign function. In the following screen, you can put anything in the "data to sign" input box and click "OK" to send out a signing request. In practice, data to sign, for example, a purchase order, will be generated by the client software.



6. PTP server: Confirm the signing request

When the PTP server receives the signing request, it will show the request to the user to ask for a confirmation. If the user approves the request, it will respond to the client with a signature; otherwise, it will simply ignore the request.



7. PTP client: Receive the requested signature

In the case where a user confirms the signing request, the PTP client will receive the signature of the request data. The PTP client will verify the signature and proceed with the following operation. In the prototype, if the signature is verified, "Verification: OK" will show up on the screen. Now the PTP-sign procedure finishes.

7 Conclusion and Suggestions for Future Work

7.1 Conclusion

A PTP-based mobile phone signature solution, which can provide signing services to other devices via local wireless connectivity, is proposed. This solution is expected to allow one's mobile phone to become a mobile authenticator and signer. The primary issue concerned in this thesis is to enable a PTP application to work in a mistrusted environment. Therefore, a PTP application is extended to accommodate a handshake process in the initialization phase. The improved PTP supports two types of handshake processes: certificate-based and password-based handshakes. The certificate-based handshake is designed to enable a mobile phone to authenticate the identity of a mistrusted signing service requestor, such as an ATM machine. The password-based handshake is meant to be compatible with the original PTP application working in a personal environment, where only device pairing is considered. It also helps a PTP application to get rid of the dependency on underlying physical communication protocol for device pairing. This handshake enables a PTP application to provide security protection over various wireless protocols, even for IrDA. Both handshake processes are defined by transaction and message formats. The improved PTP uses XML encryption and signature technologies to provide PTP messages with confidentiality and integrity protection.

J2ME CLCD and MIDP platforms are proposed as the development platform to implement the proposed PTP server. According to the analysis of the functional and system requirements, the J2ME platform is proven to be a suitable platform for the

implementation of PTP servers. Related development and deployment of security technologies are discussed in detail. In addition, a prototype is provided to demonstrate the development of a PTP server on a J2ME platform. The PTP API framework and the core PTP service - PTP.sign are implemented.

7.2 Future Works

There are a number of outstanding issues to be investigated in future works:

1. In my prototype, only the PTP API framework and PTP.sign were considered in the design. For a complete solution, a PTP server with a complete handshake process should be implemented.
2. Performance is a crucial factor to enable the mobile phone-based signature solution to be practical. Currently, the prototype is running on a MIDP simulator offered by Sun. Therefore, it is necessary to test the implementation in a real mobile phone to verify its performance.
3. The SE is an important part of the solution; it determines the security level of the system. In my thesis, some SE alternatives are discussed. In future works, the pluggable SE should be studied further. Especially, software SE should be considered in more detail using the Cryptographic Token Framework (CTF) [32].

References

[1] RSA Security, *PKCS #11: Cryptographic Token Interface Standard v2.2*, 2004

[2] Microsoft, *Microsoft Cryptographic Service Providers,* http://msdn.microsoft.com/library/default.asp?url=/library/en-us/seccrypto/security/microsoft_cryptographic_service_providers.asp

[3] Michael Rohs and Harald Vogt, *Smart Card Applications and Mobility in a World of Short Distance Communication*, CASTING Project, ETH Zurich, Distributed Systems Group and Swisscom AG Bern, Corporate Technology

[4] Vodafone, PKI goes Mobile, http://www.silicon-trust.com/pdf/secure_4/30_techno_3_1.pdf

[5] Mobile Electronic Transaction Ltd., *Personal Transaction Protocol Version 1.0, 2002*

[6] Microsoft, An Introduction to the Windows 2000 Public-Key Infrastructure, http://www.microsoft.com/technet/archive/windows2000serv/evaluate/featfunc/pkiintro.mspx

[7] *Digital Signature Guidelines Tutorial*, http://www.abanet.org/scitech/ec/isc/dsg-tutorial.html

[8] Netscape Communications, *The SSL Protocol*, Version 3.0, 1996

[9] Andrew Nash, William Duane, Celia Joseph and Derek Brink, *PKI – Implementing and Managing E-security*, Osborne/McGraw-Hill, 2001.

[10] Schlumberger Limited, *Higher Capacity Cryptoflex Smart Card Brings Increased Security to E-Commerce Transactions and IT Network Acces,* http://newsroom.slb.com/press/newsroom/index.cfm?baid=2&prid=3241

[11] Gemplus Limited, GemSAFE an Entrust Ready solution, http://www.gemplus.com/products/gemsafe_entrust_ready/

[12] Margus Freudenthal, Sven Heiberg and Jan Willemson, *Personal Security Environment on Palm PDA*, http://ieeexplore.ieee.org/iel5/7224/19469/00898891.pdf?arnumber=898891

[13] D. W. Davies. *Use of the signature token to create a negotiable document.* In Advances in Cryptology: Proceedings of CRYPTO'83, pages 377 - 382, New York, USA, 1984.Plenym Publishing.

[14] Sharon Gaudin, *Smart phones Quickstudy*, http://www.computerworld.com/printthis/1999/0,4814,43505,00.html, Computer World

[15] G. Richter, *Evaluation and Implementation of Secure Mobile Commerce Systems*, http://www.cs.joensuu.fi/~icleju/EssaysContributions/WAP/SecurityProblemsinMcommerceusingWAP.htm

[16] WAP Forum, *Wireless Identity Module Specification*, http://www1.wapforum.org/tech/terms.asp?doc=WAP-260-WIM-20010712-a.pdf

[17] Mobile Electronic Transaction Ltd., *PTD Definition Version 2.0, 2002*

[18] RSA Security, *PKCS#7: Cryptographic Message Syntax Standard V1.5*, 1993

[19] The FreeBSD Documentation Project, *FreeBSD Handbook*, Chapter 25, Advanced Networking

[20] H. Krawczyk et al., *HMAC: Keyed-Hashing for Message Authentication*, RFC 2104

[21] D. Eastlake 3rd et al.., *XML-Signature Syntax and Processing*, The Internet Society & W3C, 2002

[22] Bellovin, S.M., Merritt, M., 1992. *Encrypted key exchange: password-based protocols secure against dictionary attacks*. In: Proceedings of the 1992 IEEE Computer Society Conference on Research in Security and Privacy, pp. 72-84.

[23] Donald Eastlake et al., *XML Encryption Syntax and Processing*, The Internet Society & W3C, 2002

[24] Sun Microsystems, *J2ME Data Sheet*, http://java.sun.com/j2me/docs/j2me-ds.pdf

[25] JSR 118 Expert Group, *Mobile Information Device Profile Specification*, Version 2.0, 2002

[26] Sun Microsystems, *J2ME Device List*, http://developers.sun.com/techtopics/mobility/device/device

[27] Jonathan Knudsen, *Parsing XML in J2ME*, http://developers.sun.com/techtopics/mobility/midp/articles/parsingxml/, 2002

[28] JSR 177 Expert Group, *Security and Trust Services API for Java™ 2 Platform*, Micro Edition, 2003

[29] Motorola Wireless Software, Applications & Services, *Java APIs for Bluetooth Wireless Technology*, 2002

[30] Sun Microsystems, Inc., *Connected Limited Device Configuration Specification*, Version 1.1, 2003

[31] Bouncycastle.org, *Bouncy Castle Cryptography 1.29*, http://www.bouncycastle.org/docs/docs1.5/index.html

[32] Symbian Ltd, *Symbian OS Version 8.0 Product description*,

http://www.symbian.com/technology/SymbianOSv8_funcdesc_2.1.pdf