# Mobile Personal Authentication using Fingerprint

**Cheng Po Sum**

A Thesis Submitted in Partial Fulfilment
of the Requirements for the Degree of

Master of Philosophy

in

Department of Computer Science & Engineering



July 2003

# Table of Contents

## List of Figures

## List of Tables

## Acknowledgments

I would like to give my sincere thanks to my supervisor Dr. Y. S. Moon for giving me advice to explore the topics and help me to solve the difficulties in writing the thesis. Dr. Moon gave me a lot of opinions so that I gain much valuable experience from him. Dr. Moon, as a supervisor, also becomes my friend during my Master of Philosophy (MPhil.) study. He gives advice to me time to time from research field to my future.

I would like to thank my colleagues, K.C. Chan, T.Y. Tang, K.F. Fong, M.L. Ho, H.M. Tang and K.F. Jang for giving me an unforgettable campus life. They also gave me a lot of valuable discussions and solutions when I met some problems in my research. They let me feel that I am not working by myself.

I would like to thank my family and friends, especially my parents, for supporting my graduate study. They gave me encouragement on helping and pushing me to finish the whole research study.

## 摘要

現時，個人電子助理和手機等個人流動設備變得越來越普及。由於工業界推出了很多新的工業設計和技術 ，流動設備可支援更多先進的應用。如生物辨識和多媒體處理，這些應用雖然需要複雜的計算 ，但它們也開始移植到流動設備上。在眾多的應用之中，流動付款對流動設備是一種比較重要的應用。在流動付款系統中，用戶能夠在任何時間和地方也能靈活地作出付款或其它相關的處理。但是 ，認證成爲流動付款系統裏一個重要安全問題 ，尤其是牽涉到金錢上的商業行爲。所以我們建議用一個較低成本而又安全的流動系統 ，系統裏面包括有指模辨認和智能卡的應用 ，以提高流動保安系統的安全性。

爲了評估我們的流動保安系統 ，我們開發了使用告票付款系統 (PTPS)。在告票付款系統中，我們設計了不同的安全模式，它可以讓不同的人或組織在一個安全的渠道上進行溝通。我們做了不同的實驗去評估安全模式在流動平台上的表現。結果顯示 ，我們的告票付款系統能夠有效率地在流動保安系統上運作及達到理想的表現。

## Thesis Abstract

Nowadays, personal mobile devices, such as PDAs and cell phones, are becoming more and more popular. With the new advancement in industrial technologies, mobile devices can support more advanced applications. Computational intensive applications such as those for biometric and multimedia purposes are now deployed to these devices. Among these different applications, mobile payment is one of the most important applications in mobile platform. With the ubiquitous characteristic of mobile devices, mobile payment provides flexibility to users as they can access the devices anywhere at any time. However, authentication is a critical security concern in the mobile payment system. It is especially important for those commercial activities involving monetary transfers. Hence a low cost prototype secure mobile device, which is capable for handling fingerprint and smart card, is proposed here to enhance the security of the mobile systems.

To evaluate the performance of our proposed mobile security system, a Penalty Ticket Payment System (PTPS) is developed on top of our constructed device. Different secure modules are implemented in PTPS so that different parties can communicate with each other via a security channel. Experiments are conducted to evaluate the performance of the secure modules in mobile platform. Results show that our mobile security system can be deployed to PTPS efficiently and it can achieve real time performance.

# 1. Mobile Commerce

## 1.1 Introduction to Mobile Commerce

Mobile commerce is one of important commercial activities nowadays. It often involves the use of mobile devices when making transactions. Mobile commerce is quite different from the traditional electronic commerce. Electronic commerce is a more general term to define the type of commercial activity conducted in an electronic world. However, mobile commerce emphasizes the wireless way of conducting transactions. Therefore mobile commerce is a subset of electronic commerce. The use of mobile devices in mobile commerce makes a significant change in our computer paradigm. The ubiquitous environment in our real life has a great impact on our living style. We can use the mobile device to interact with other devices or computers anywhere at anytime. However this significant change makes a great challenge to the security issue. The uses of mobile devices are quite different from fixed-location devices (e.g. desktop personal computers). There will be security holes without a well designed authentication technique and control access schemes [23]. We will have a more detailed discussion about the security issue in a later part of the thesis.

The ways of conducting transactions in mobile commerce give more flexibility to the users. With the mobile devices, users can enjoy the mobile services provided from the suppliers in real time. For example, a user can buy or sell a stock at a certain price immediately. Or he can make a bill payment that will overdue a few minutes later. Mobile users can receive and interact with the services content at an instant in different ways like HTML (HyperText Markup Language) [17] and SMS (Short Message Service) [53]. Therefore with more and more applications deploying to the mobile world, customers can enjoy a higher quality of services.

Mobile commerce applications become more and more popular and important because the number of mobile users has grown rapidly from four million in 2000 to twelve million in 2002 [42]. There are many kinds of mobile commerce applications that provided to the customer. According to [52], mobile commerce applications can be divided into three main categories, digital content delivery, telemetry service and transaction management.

**Digital content delivery**

Digital content delivery refers to the transfer of requested information to the user. Information including weather, stock price, e-book, ticket availability and traffic status are some of the examples. Digital information can be delivered to a user who owns a network enabled mobile device. When more advanced equipment like high revolution LCD display and faster processor is used, information like MP3, video and digital map can be delivered via higher speed networks.

**Telemetry service**

Telemetry service is a new way of mobile application. Telemetry technology allows a user to control home appliances from a mobile device via a wireless network. However, this type of service is not very popular because IP-enabled home appliances are still a very new idea that requires the participation of numerous appliance vendors and the drafting of mutually agreed operation protocols.

**Transaction management**

Transaction management is one type of mobile commerce service that is easier to deploy to the user. Online shopping and mobile purchasing are two examples. In transaction management service, a user can enjoy real time information browsing, product selection and purchasing. All steps can be completed via some small portable devices, giving great convenience to users to manage their transactions.

Among the above three categories of mobile commerce applications, security concerns in transaction management is the most important one because it involves a large amount of monetary transactions. Therefore we will further discuss the mobile payment systems for handling monetary transfers.

## 1.2 Mobile commerce payment systems

In traditional desktop-based electronic commerce, many payment systems are already available. However, as mentioned in [51], direct porting of such systems to mobile devices are not suitable. The unique characteristic of mobile commerce causes many difficulties in integrating with the traditional electronic payment systems.

Therefore, new types of mobile payment systems have been proposed to solve the problem. Basically, there are three types of mobile payment systems. They are software electronic coins, hardware electronic coins and background account payment system.

**Software electronic coins payment system**

Figure 1.1 shows the software electronic coins payment system. This type of payment system uses software to represent monetary value. Customers can purchase electronic coins and download them to mobile devices. A coin can be stored as a file with a serial number and expiry day. Whenever a customer employs a mobile device for a purchase operation using an electronic coin, the merchant software will retrieve the coin from the mobile device and forward it to the issuing bank. The bank will then check the validity of the serial number of the coin. If the coin is valid, the bank will deduct the purchasing value from the coin and transfer the amount to the merchant's account.

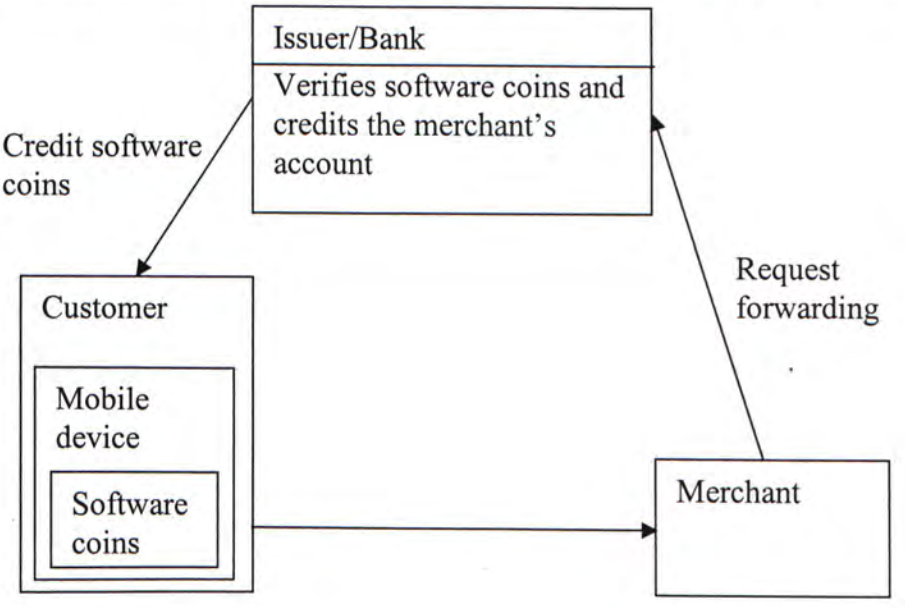



Figure 1.1: Software electronic coins payment system

The disadvantage of this system is that the coin needs to be generated externally and then downloaded to the mobile device. This procedure is not convenient to the customer because the customer needs to go to the issuing bank for crediting the electronic coin. Speaking of security issue, the uniqueness of the electronic coin mainly depends on its serial number. Without a proper encryption to protect the serial number, it can be easily duplicated and used illegally. Besides, the customer is completely anonymous to the bank and merchant. That means no parties can identify who is using the electronic coin.

Therefore, the security level of authentication is weak, making it suitable for small amount of monetary transfers.

**Hardware electronic coins payment system**

Figure 1.2 shows a hardware electronic coins payment system. In this type of payment system, a hardware token like a smart card is often used to store the electronic coin. Using a smart card gives a great convenience to the customer because a smart card is easy to carry. In conducting a transaction, a customer only needs to connect his/her smart card, which is provided by the merchant, to the card terminal. The software from the merchant, which is provided from the card issuer, will check the validity of the smart card and deduct the purchasing amount from the card. The transaction log will then be sent to the bank and the bank will credit the merchant's account. There are several advantages in using this approach. Merchant can conduct transactions with customers directly and transfer the amounts to the issuing bank. Customer can add credit to the balance inside the smart card with an Automatic Teller Machine (ATM) directly. Besides, a hardware token (smart card) is difficult to duplicate [64] and it gives more protection to the customers. But the security protection in those systems is still not enough because no authentication procedure is involved. Using a hardware electronic coin like the Mondex card, does not require any authentication before conducting a transaction. That means anyone who picks up the card can use it for purchasing directly without any restrictions.

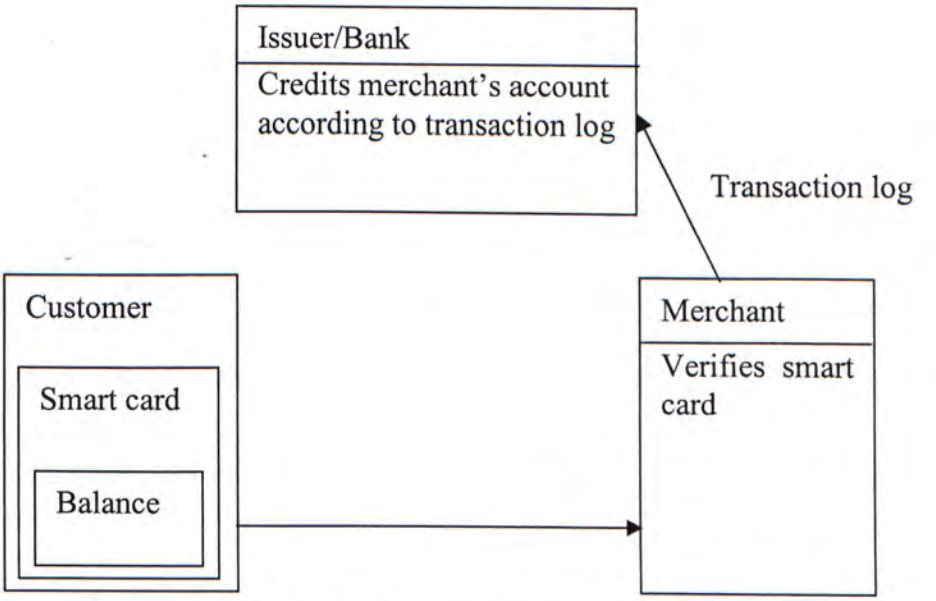



Figure 1.2: Hardware electronic coins payment system

**Background account payment system**

Figure 1.3 shows a background account payment system. In a background account payment system, a trusted third party who stores the account of the customer is required to act as a mediator between the customer and the merchant. When the customer receives the invoice from the merchant, he or she will send an authentication and authorisation messages to the merchant. The merchant will then forward those messages to the trusted third party. The trusted third party will verify the message and settle the transaction. Famous background account systems like the Secure Electronic Transactions (SET) belong to this category. In SET, a transaction is encrypted and digitally signed to prevent unauthorized attacks. The advantage of using this type of payment system is that the account is stored in a trusted third party. Unauthorized persons cannot access the account directly. In order to attack this type of system, an attacker needs to send a fake message to the trusted third party. In this way, the security of the system depends on the protection mechanism for the authentication and authorisation messages.

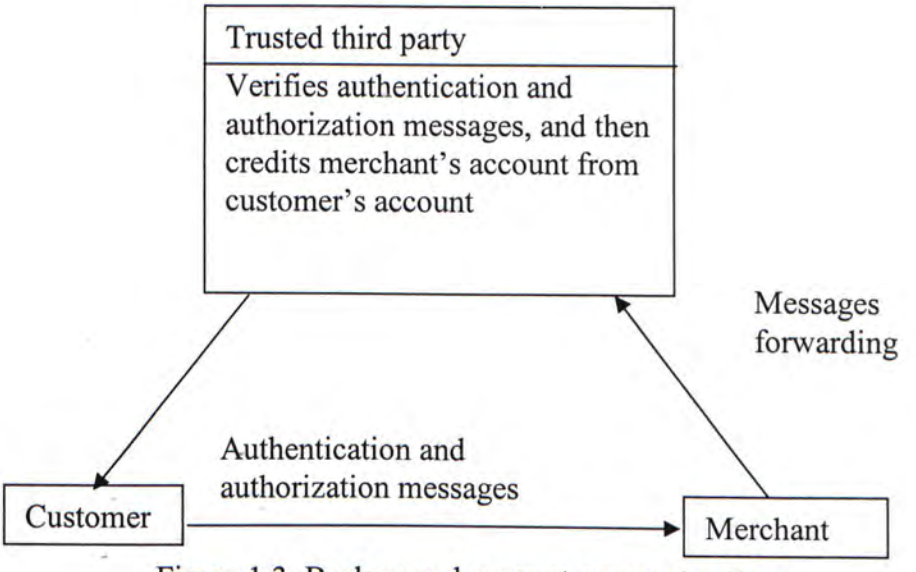



Figure 1.3: Background account payment system

## 1.3 Security in mobile commerce

In the previous section, we have discussed different kinds of mobile commerce payment systems. Security in those payment systems is very important because it involves monetary values. Without a proper security measure, payment systems can be easily attacked. There are many known types of attack to payment systems [7]. To

protect against these attacks, security requirements on payment authentication, integrity, non-repudiation and confidentiality must be further considered [64].

**Authentication**

Authentication is to ensure the identities of different parties that involved in the transactions. There are different types of authentication methodologies for different applications, for examples, password, smart card and biometric. Each methodology has its strengths and weakness. We will have a detailed discussion in a later part of this thesis.

**Integrity**

Integrity is to ensure that no unauthorized principals can modify the content of the transaction. Information inside the transaction including payer and payee identities, purchase details and other related information should ALL be protected.

**Non-repudiation**

Non-repudiation is to ensure that once a customer has committed a transaction, he or she cannot deny it. A signature from the originator needs to be included in the transaction so as to provide a non-repudiation feature.

**Confidentiality**

Confidentiality is to ensure that transaction data will only be disclosed to the relevant parties during storage or transmission processes. A secret communication channel is needed during the transfer of data. Different pieces of information inside a single transaction are separately disclosed to different parties (e.g. SET) on a need-to-know basis. This requirement adds further complication to the communication mechanism.

In order to achieve the above four security requirements, a security technology, Public Key Infrastructure (PKI), is deployed in the world of electronic commerce. There are lots of cryptography technologies inside PKI. The main components include symmetric cryptography, asymmetric cryptography, digital signature and digital certificate. They are described in Figure 1.4 to 1.7 respectively [61][4].

In symmetric cryptography, a secret key is used to encrypt the plain text into cipher text. During data transmission, the cipher text is not viewable from an unauthorized party. Therefore, it provides data confidentiality. On the receiver side, the cipher text is decrypted using the same secret key to obtain the plain text. Both parties that involve in the process need to share the same secret key. Therefore, this method cannot guarantee authentication and data integrity.

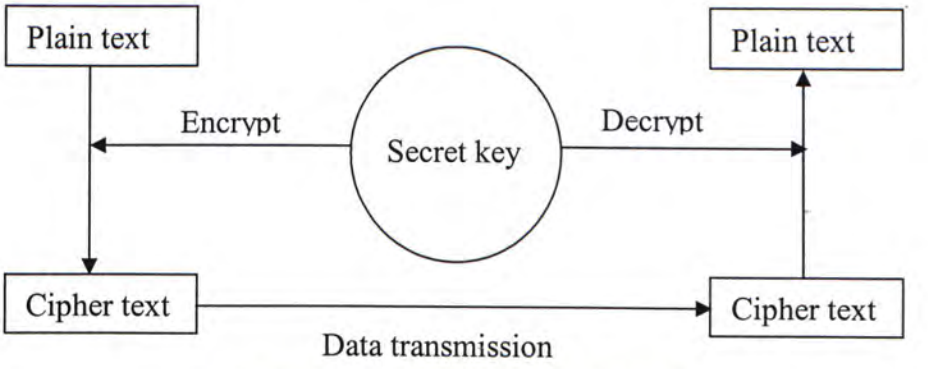



Figure 1.4: Symmetric cryptography

In asymmetric cryptography, two keys are used for encryption and decryption. Therefore it can provide key confidentiality. The two keys are mathematically related and generated by RSA algorithm [48]. Data encrypted with a public key can only be decrypted using a private key and vice versa. On the other hand, a private key is accessible by the owner only while a public key can be distributed to the public. Therefore, data encrypted with a public key can provide data confidentiality while a private key cannot. The advantage of this design is that no sharing of a key is necessary. Each party only needs to keep a private key for decrypting secret messages that are encrypted with corresponding public key.

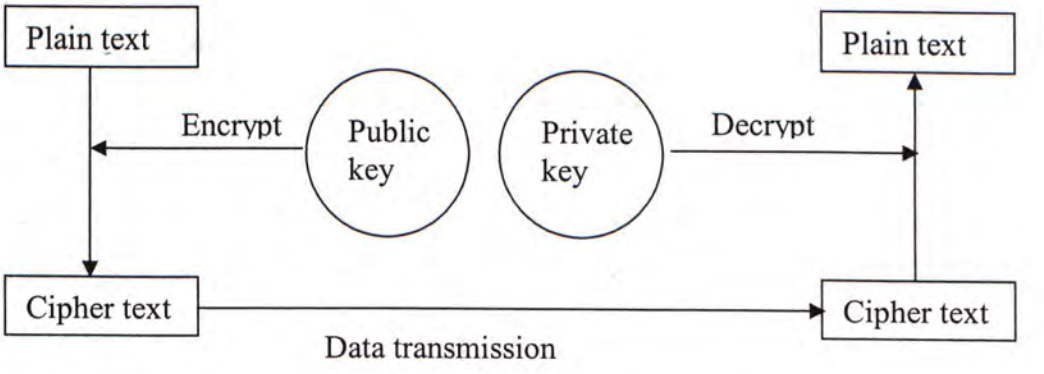



Figure 1.5: Asymmetric cryptography

In creating a digital signature, the plain text in a message is first hashed using a hash function [15]. The hashed plain text is signed by a private key to produce a signature. After the signature has been transmitted to the receiver side, the receiver uses the corresponding public key to decrypt the signature in order to get the original hashed data, say H1. The receiver now produces another hashed data using the same plain text as the sender, say H2. By checking whether the two hashed data, H1 and H2, are equal or not, the receiver can verify whether the message has been "properly signed". Digital signature, thus, provides authentication and data integrity features since a digital signature can only be produced using its owner's private key. Besides, a digital signature also provides the non-repudiation feature because the sender cannot deny his/her signed document using his/her private key.

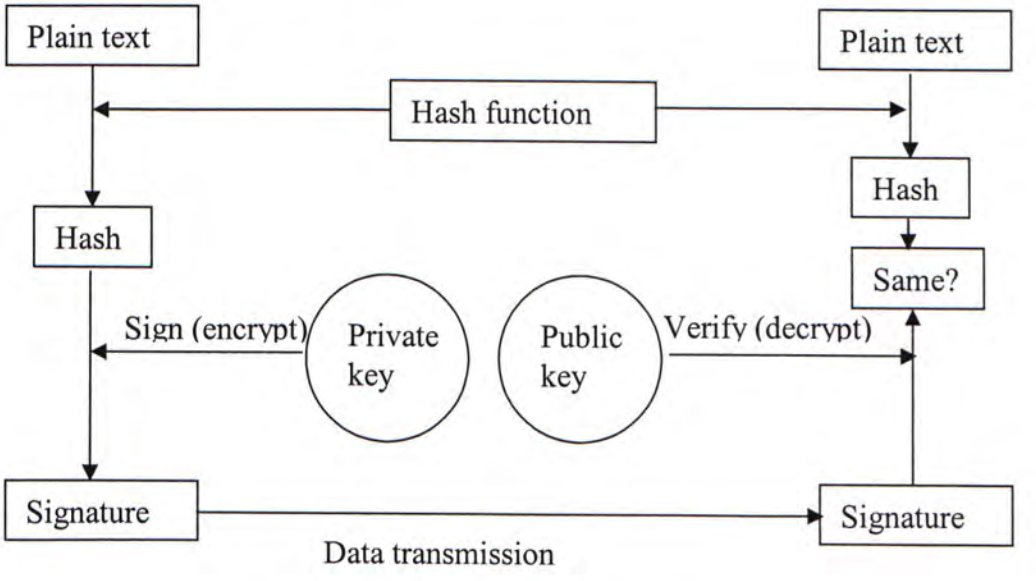



Figure 1.6: Digital signature

A digital certificate is used to ensure that the entity's public key belongs to the entity it claims to be. There are four main fields in a digital certificate. The first one is the owner's identity that stores the related owner information. The second one is the owner's public key. The third one stores the certificate authority's (CA) information. The last one is the CA's signature that is produced by signing CA's private key on the hashed owner's information as shown on the left of Figure 1.7. A CA is a trusted party who is responsible to generate public keys for verified owners. One party can verify another one by CA's public key as shown on the right of Figure 1.7.

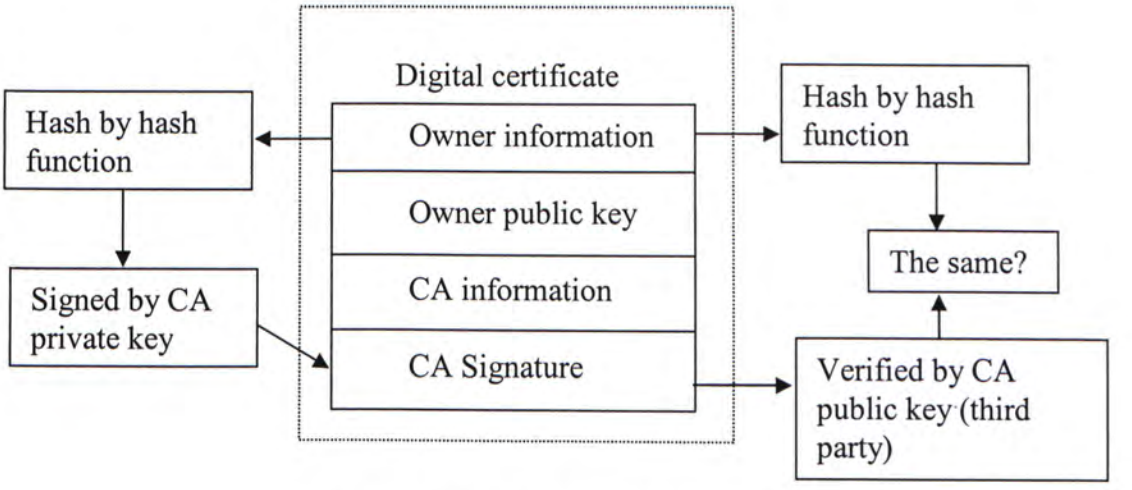



Figure 1.7: Digital certificate

The above four main components in a PKI system provide a complete framework for various security applications. Each party using this framework can communicate with the others in the secure channel. In the next chapter of this thesis, we will discuss the advantages and disadvantages of different authentication techniques that can be applied to mobile commerce applications.

# 2. Mobile authentication using Fingerprint

## 2.1 Authentication basics

Authentication is an important procedure in a secure transaction. If the authentication layer of the system is not reliable, even strongly encrypted data can become useless because no reliable identification of the involved parties can be made. In this way, an unauthorized party may have chance to attack valuable information. Although PKI can help us to solve different security requirements (which are mentioned in chapter one), it cannot replace the need for secure authentication mechanism. The reliability of PKI is strongly dependent on the authentication mechanism.

The protection of the private key is very important in PKI because the private key is used for identifying the party involved in a transaction. If we do not have a suitable authentication mechanism to protect the private key, PKI can become untrustworthy since anyone can copy the private key and pretend to be the owner in order to conduct a transaction.

Modern authentication mechanisms are based on three common design considerations: "Something-You-Know" (knowledge factor), "Something-You-Have" (possession factor) and "Something-You-Are (biometric factor)" [44][4][61].

A "Something-You-Know" architecture refers to remembering some passwords or PINs. A lot of applications, like ATM card and e-banking, use this strategy nowadays. In this model, a secret password is shared between the user and the host. The advantage is that the implementation of this architecture is not complicated and it can be easily integrated into different applications. However, the disadvantage is that the whole authentication relies only on the password. If the user discloses the password accidentally, an unauthorized person can steal the owner's identity easily. Moreover, passwords made up of strings of numerals and characters that can be guessed by try-and-error. Moreover, since passwords are stored in a security system, the system shares some security responsibility. Cases exist in the past where systems were attacked and passwords were stolen.

Different from the "Something-You-Know" architecture, the "Something-You-Have" architecture refers to some physical objects that the user owns. Those physical objects include hardware token cards or software digital certificates. The implementation of a hardware token method is often used with a hardware card, e.g. a smart card. The smart card stores the key used for authenticating its owner. A smart card reader is needed for reading information from smart card and sending the information to a host computer. Although it is difficult to attack the information stored inside a smart card [47], information can be stolen during the data transmission to/from the smart card reader from/to the host. Therefore, some smart cards are capable of applying encryption to its data for protection during transmission. This authentication technique is more secure than the "Something-You-Know" architecture because an unauthorized person will have to steal or duplicate the certificate or hardware token before he or she can access the valuable information. The software certificate may also be stolen if the user's PC is hacked. As a result, the host can only identify the mathematically strong objects [44] assigned to the users but not the users themselves directly!

The last authentication mechanism depends on some biometric features, like fingerprint, face, iris, voice, etc. of a user. It is a strong authentication technique because only the owner possesses the biometric features himself assuming unique physical characteristics of a human body. It is very difficult for an unauthorized person is very difficult to forge or duplicate the same features belonging to a user. Traditionally, biometric data processing, which is usually considered as a branch of signal processing, requires extensive computation in order to achieve accuracy and reliability. Fingerprint authentication is one of the most popular and mature techniques [35][43][33] among all biometric authentication techniques. Numerous biometric researches have been conducted in fingerprint authentication. New technologies based on electronic capacitance, pressure, thermal radiation have enabled the production of many low cost and small fingerprint sensors. Thus, fingerprint authentication will become one of the most popular techniques deployed to the mobile applications in the future.

As mentioned above, there are three main types of authentication factors. An authentication process can combine different factors to achieve better security. For example, an authentication system can use a smart card together with biometrics. If a user cannot present his/her smart card and validate himself/herself through biometric

tests, he/she will be rejected by the authentication system. This type of system is called hybrid architecture. Hybrid design provides a flexible architecture in which different security techniques are fused together to achieve a better security. With the combination of strong authentication technology and PKI, a high level of security in mobile commerce can be achieved. In this thesis, we will focus on the investigation of such a development. Therefore, we will introduce the fingerprint biometric knowledge in the next section.

## 2.2 Fingerprint basics

A fingerprint matching system contains several components. Its operations are made up of two phases, the enrolment phase and the authentication phase. Figure 2.1 shows a generic fingerprint matching system.

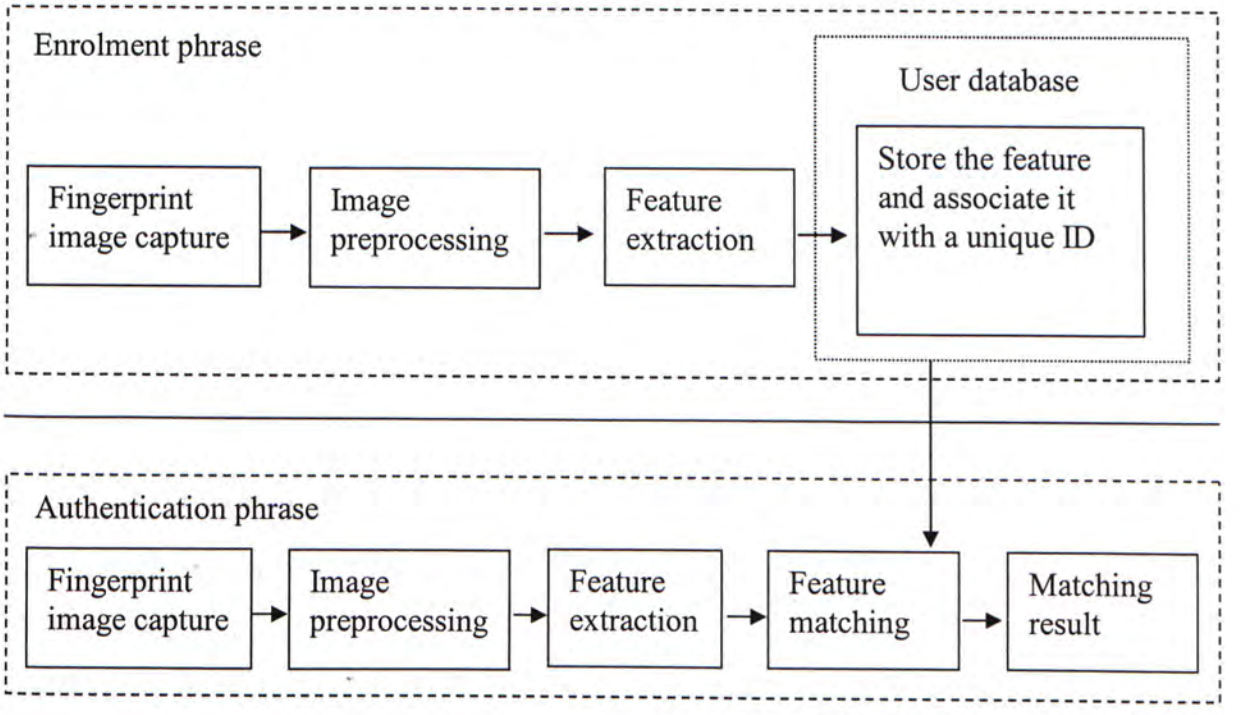



Figure 2.1: Fingerprint matching system

The enrolment phase contains four stages: fingerprint image capture, image pre-processing, fingerprint feature extraction and fingerprint registration. Fingerprint image capture refers to the capturing of a user's fingerprint from a fingerprint sensor. There are different kinds of fingerprint sensors. The physical size, fault tolerance and resolution of a sensor characterize its properties as well as the main features for evaluation. Image pre-processing refers to the enhancement of a fingerprint image. It is an image

processing technique to improve the quality of the image like contrast adjustment and feature reconstruction. Fingerprint feature extraction is a mathematical process applied to a fingerprint image. This process extracts the unique features from the captured fingerprint image. Feature extraction is an important step because it is closely related to the accuracy of the authentication system. The last step is fingerprint registration. It adds a user's unique features to the database and assigns a unique ID to the corresponding record.

The authentication phase contains five steps. The first three steps are similar to the enrolment phrase. The fourth step is feature matching in which the system retrieves the registered fingerprint features from the database and compares it with the features extracted from a live fingerprint template. In the last step, the system calculates the matching result so as to determine whether the two templates belong to the same person or not.

In a lot of fingerprint systems, the matching result is represented by a score, say zero to hundred. The threshold score value for matching the fingerprints are often determined by the calculation of False Rejection Rate (FRR) and False Acceptance Rate (FAR) [1]. FRR is the rate of rejecting authorized users. FAR is the rate of accepting unauthorized users. To analyse the characteristic of FRR and FAR, a fingerprint database is needed to evaluate the fingerprint verification algorithm. The test result can be regarded as a reference to measure the security level of the system.

As we have mentioned above, feature extraction is the process that is closely related to the accuracy of a fingerprint verification system. Therefore, we must extract the features of the fingerprint accurately. A fingerprint feature is the characteristic of ridgelines on the finger so that each finger has its unique set of features [49]. As there are noise and distortion when capturing a fingerprint image, direct comparison of two fingerprint images using cross-correlation is not practical. Hence, special information that will not be influenced too much by noise and distortion has to be selected from a fingerprint image. These features include core point, orientation field and minutiae. A core point is the uppermost point of the innermost curving ridge as shown in Figure 2.2. Most fingerprint images will only contain one core point so that it can be used for fingerprint alignment. An orientation field is the directional structure of the ridgeline on

the fingerprint image as shown in Figure 2.3. Minutiae are the ridgeline termination or bifurcation. The end of a ridgeline forms a termination and the merge of two ridgelines forms a bifurcation as shown in Figure 2.2. The matching of two fingerprint images is the calculation on the similarity of those fingerprint information.

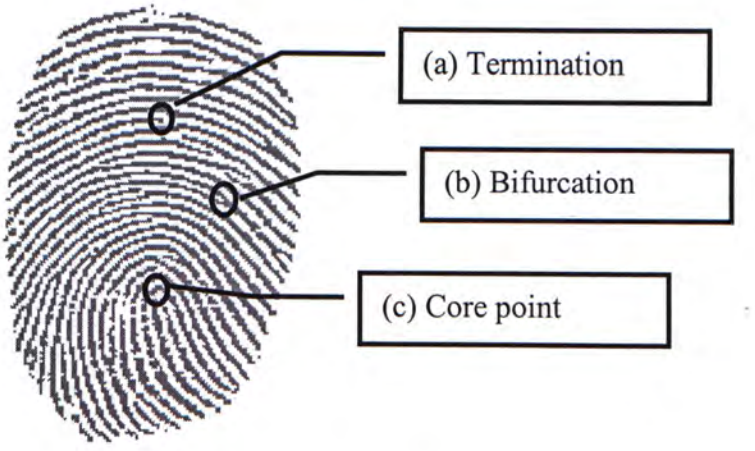



Figure 2.2: (a) Termination (a) Bifurcation (c) Core point

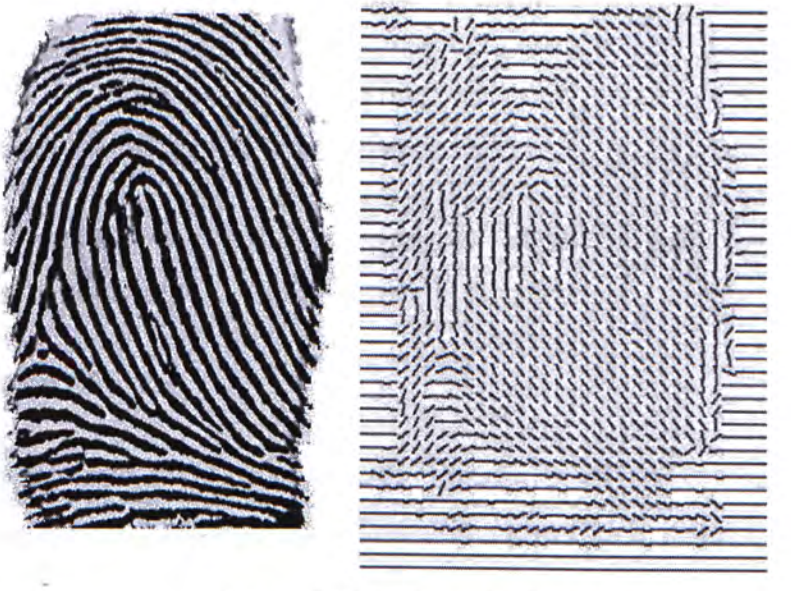


Figure 2.3: Orientation field

There are two types of fingerprint systems, identification and verification systems. In identification, a database is needed to store the minutiae information of all pre-registered users. When a user presents his or her fingerprint on the sensor, the system extracts the features and identifies the matched fingerprint in the database. It is a one to N matching problem. In verification, the system verifies the fingerprint of the user that

he or she claims to be. Before a user presents his or her fingerprint, he or she needs to inform the system his or her own identity to be verified by the system. In this way, fingerprint verification is a one to one matching and the procedure is much simpler than identification.

## 2.3 Fingerprint authentication using mobile device

The process of fingerprint authentication and the characteristic of mobile device cause a great challenge to the deployment of fingerprint authentication system on mobile devices. Fingerprint feature extraction is a complicated process that involves a lot of mathematical calculations. The process can be divided into several parts [34]. They are image enhancement, image segmentation and orientation, core point location and minutiae extraction.

### (a) Image enhancement/filtering

This part aims at enhancing the image quality before feature extraction. Common techniques like the application of a directional filter [29] and image normalization [22] are often employed.

### (b) Image segmentation and orientation

This part computes the orientation of the fingerprint image and identifies the Region of Interest (ROI) [21].

### (c) Core point location

This part locates a core point [5] for fingerprint alignment and matching. It is computed from the orientations of the ridgelines obtained from step (b).

### (d) Minutiae extraction

This part traces the ridgeline with the helps of orientation information and finds out the bifurcation and termination features.

In general, the authentication process is an $n^3$ process, assuming that fingerprint is an n by n (pixel) image. Steps (a) to (d) involve intensive floating-point calculations, especially approximations of trigonometric functions.

Although the fingerprint authentication process is complicated, it can run on desktop PC in real time because desktop PC has a very fast processing speed. We do not need much optimisation before constructing the verification software. However, when the authentication process executes in a mobile device, the situation becomes different because the software and hardware design of mobile device are quite different from the desktop PC [30]. Devices such as mobile phones and Personal Digital Assistants (PDAs) share a number of unique features and limitations. They include processing power, floating-point computation capability, battery and other physical constraints.

**Processing power**

With the advancement in electronics technology, the computational capability of mobile devices increases exponentially, enabling them to execute a wide range of applications. An embedded processor with large market share is the ARM [57][55], a RISC [45] based processor. Table 2.1 shows the speed comparison on different ARM based embedded processors. From the table, we can observe that even the speed of the fastest embedded processor is only comparable to speeds of some very obsolete desktop processors. The speed limitation is due to the consideration of power consumption and heat dissipation on a mobile device. The power consumption of desktop processor is quite large when comparing with an embedded processor. Therefore, the battery of a mobile device cannot meet such demands. Besides, the heat dissipation of desktop processors is very high and it is senseless to add a fan on a mobile device. Therefore, desktop processors are not suitable for mobile devices.

| Processor/Family | Production Year | Clock MHz |
|---|---|---|
| ARM7 | 1998 | 100 – 133 |
| ARM9 | 2000 | 180-250 |
| StrongArm | 2000 | 206 |
| XScale | 2002 | 400-1000 |

Table 2.1: ARM architecture embedded processor speed comparison table

**Floating-point co-processor**

A floating point co-processor is a dedicated processor for floating arithmetic calculations such as floating addition, floating substraction, sine, log and etc. It operates

together with the main processor and is actually resident in desktop PC processor. But the floating-point co-processor is a luxury in a mobile device because the extra co-processor draws substantial extra power from the battery. Therefore, most embedded processors do not have hardware floating-point units. Instead, software floating-point arithmetic is used.

There are two methods in handling software floating-point. They are software emulation and software library [37]. In the software emulation approach, when floating-point is detected during compilation, the compiler will automatically generate assembly code to emulate specific floating-point co-processor hardware. The advantage of this design is that the software emulator supports all the floating-point operations available in the emulated hardware. In the software library approach, all the required floating-point mathematical operations are simulated using integer operations. When a programmer encounters a floating-point operation, he or she will make a function call to a floating point library instead. The advantage of this design is that we can customize the simulation software for an embedded system. Both approaches are also very popular for an embedded system and the software developers can make their own choice depending on their applications.

**Battery**

Battery technology development advances at a rate far slower than the processing units in mobile devices [50]. When more advanced functions are added to the mobile devices, including the multimedia and biometric applications, battery capacities hardly seem enough for the mobile devices. Therefore, some battery driven system design methodologies have been developed [27] to increase the battery capacity of the device. In such a design, the efficiency is closely related to the power drawing procedures.

**Physical constraints**

The portable feature of a mobile device limits its size and weight. These constraints cause lots of restriction in the design of a mobile device. The display needs to be small due to its physical size. Extra peripherals such as external sockets, network modules are also limited.

As discussed above, fingerprint authentication is a challenge to mobile devices. When implementing such an application in an embedded device, we encounter difficulties due to slower CPU speeds, absence of cache and most important of all, absence of a floating-point unit. While optimising the authentication, we cannot afford to sacrifice reliability. Thus, a fast, low cost and accurate methodology must be developed for fingerprint matching for embedded applications. In the next chapter, we will discuss the methodology on how *to design a mobile fingerprint authentication device.*

# 3. Design of Mobile Fingerprint Authentication Device

## 3.1 Objectives

Before we design the hardware and software components of a mobile device, we have to determine the application requirements. In a fingerprint system, as mentioned above, the system can be divided into identification and verification applications. Identification application identifies a person from a group of individuals. The size of database can range from a few persons (home use) to hundreds of persons. The performance of such a system is database size dependent. Access control (e.g. door lock) is a typical identification example. Figure 3.1 shows the block diagram of a common access control system. Enrolment is conducted when a user first registers a fingerprint in the system. The core part of the system is the authentication device. It needs to handle different connections such as Ethernet, RS232 and Wiegand (an industrial standard security protocol sending bit strings of specified length and content) to communicate with the enrolment machine, database server and security controller.

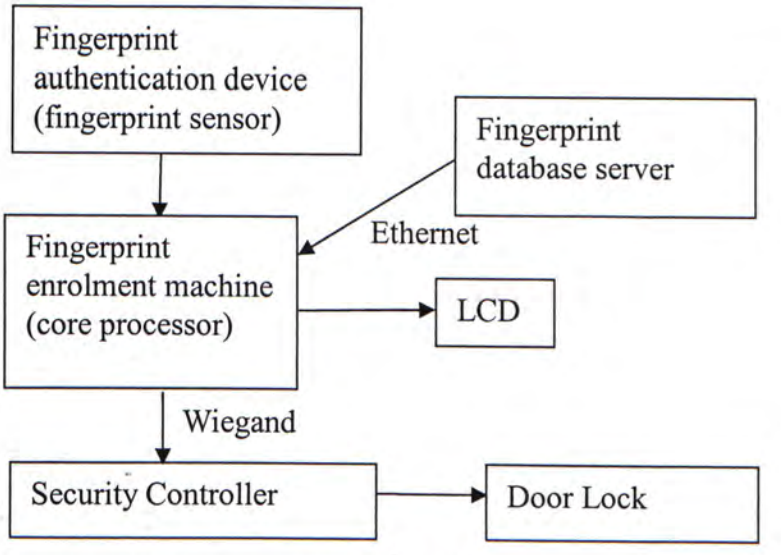



Figure 3.1: Block diagram of physical access control system

With the limited features of a mobile device, it is difficult to conduct identification using slow speed CPU in a very large database. To solve this problem, we can use a smart card. Every user in the system is issued with a smart card in which his identity is stored. When the system needs to verify a user, the user needs to present his smart card and verify his identity using his fingerprint. With this approach, the mobile authentication device does not need to perform one to N matching. Figure 3.2 shows the

block diagram of a fingerprint and smart card authentication system. Similar to Figure 2.1, this figure also contains enrolment phase and authentication phase. In enrolment phase, the features of the captured fingerprint will be extracted and registered in the smart card. In authentication phase, the live fingerprint template will be transferred and matched with the pre-registered one in the smart card. Since this approach is more practical for mobile device, we decided to adopt this approach in our implementation.

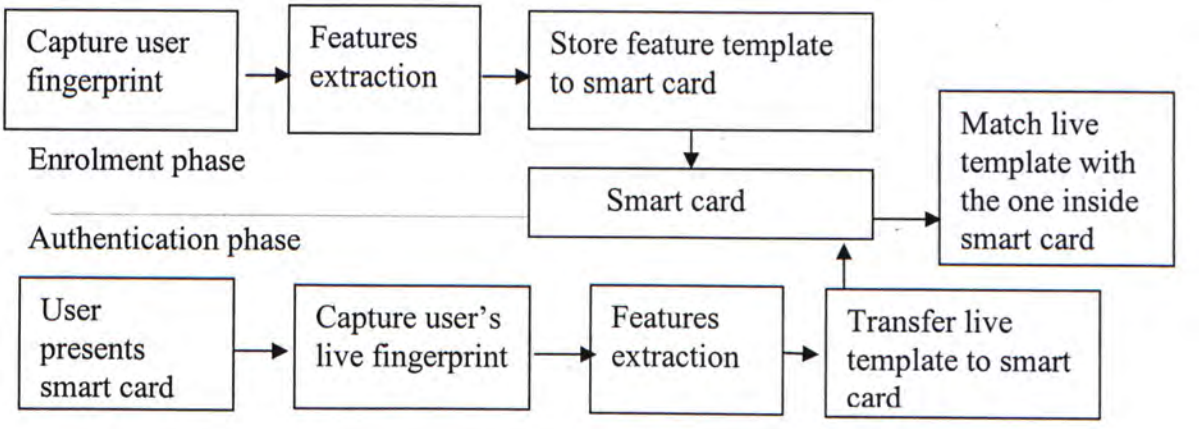



Figure 3.2: fingerprint and smart card authentication system

Like other embedded systems, the design of a mobile personal fingerprint authentication device requires the consideration of cost, development time and runtime performance. Since real time response is a critical factor in this case, we will focus on the searching of an appropriate implementation strategy that can yield such performance.

Before we design the mobile device, we need to understand what an embedded system is. In [65], Wolf defines an embedded system as "Any sort of device which includes a programmable computer but itself is not intended to be a general-purpose computer is said to be an embedded system." Traditionally, an embedded system is often regarded as a micro controller residing in a special purpose device. Because of its dedicated purpose(s) and limited resources, there is limited or even no operating system support. In the event that extensive computation is required, a dedicated processor [24] is also included to ensure a satisfactory performance. Such systems can be found in numerous industrial applications, particular in the area of control engineering. With the advancement in semi-conductor technology and operating system miniature, we are seeing the births of new kinds of embedded systems. A glance of an embedded Linux web site clearly reveals this [32]. Embedded systems have been widely used in factories

and even offices and households. Well known examples include set-top-boxes and routers. The advent in personal and mobile computing has pushed embedded systems further to the personal applications. Mobile telephones, personal digital assistants are some typical examples. In short, embedded systems have quickly become a part of modern man's life.

## 3.2 Hardware and software design

Designing an embedded system involves both a hardware part and a software part. The hardware part includes the core hardware platform that the system used. It can be the choice of processors, memories and other related peripherals. The software part is the design of software layer(s) implemented in the embedded system. The software architecture needs to be designed carefully because it affects the efficiency of the whole system. It controls the processor and memory resources as well as the communication protocols between the processor and the peripherals. Therefore, a bad design will affect the stability and reliability of the system.

### 3.2.1 Choice of hardware platform

When developing an embedded application, there are a number of choices of the type of processors. Those domain specific processors [31] can be generally divided into three areas [46], System-on-Chip Processor, Digital Signal Processor and Application Specific Integrated Circuit Processor.

#### a) System-on-Chip (SoC) Processor

In the past, embedded applications used an 8-bit micro-controller as the core part of the design. An 8-bit processor micro-controller is designed for low-cost applications such as servomotor control and simple robot control. Today, the 8-bit processor is still very popular for low-end applications because they are relatively cheap and easy to control. With more advanced applications deployed by the embedded systems, an 8-bit processor seems to be too slow to support the advanced applications. Therefore, a new technology embedded processor is being developed to suit the applications. We call it a System-on-Chip (SoC) processor. SoC will play an important part in the next generation of computing. It is a new type of hardware architecture with complex hardware that

integrates with different Intellectual Property (IP) cores. As shown in Figure 3.3, the IP cores may include SDRAM controller, peripheral I/O controller, Ethernet controller, LCD interface and etc. Those IP cores communicate with the Central Processing Unit (CPU) via internal bus. The variety of IP cores allows a single SoC to be programmed for different applications. The new architecture of SoC enables the reusability of hardware modules and therefore shortens the design time for new systems. In fact, a SoC processor is similar to a general-purpose desktop processor without floating-point support because of cost and power consumption considerations. The requirement that that all processes be performed on a SoC helps to reduce power consumption, minimize chip areas and simplify the hardware and software development process [46].

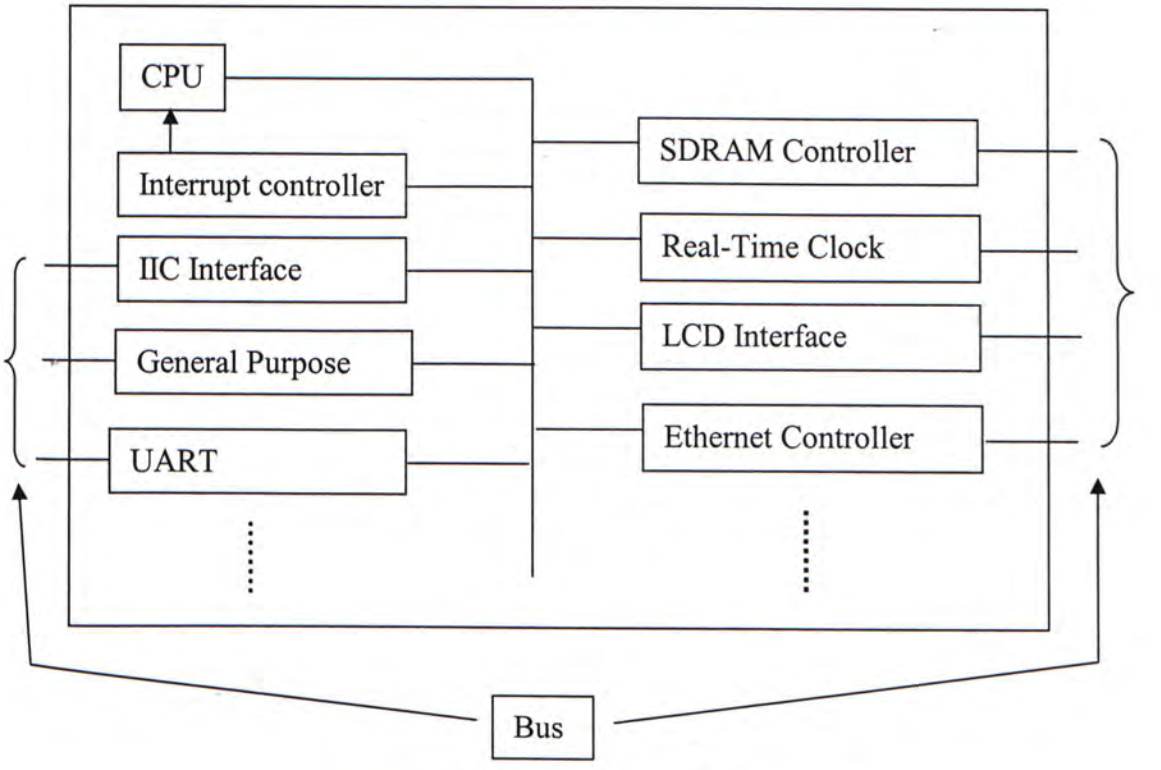



Figure 3.3: Architecture of SoC processor

In SoC circuit design, system complexity will grow faster than Very Large Scale Integrated (VLSI) circuit [26] complexity because the functionality of embedded system based on SoC can be implemented in software. With the help of software implementation, a SoC can deploy to different applications. Let us take the ARM [19][18] series of processors as an example. They are 32-bit RISC processors with pipelined features. One of the famous ARM-based processors is the Intel PXA250 application processor. This SoC is equipped with an Intel XScale core microprocessor with different IP cores. It can be used for embedded system with a higher performance,

lower power consumption and on-the-fly frequency scaling functionality [18]. The on-chip controllers can increase the communication performance between the core processor and the peripherals and hence reduce the overhead on the design of the hardware architecture of the embedded systems. With such design, the PXA250 can be deployed to different areas of embedded application that range from personal devices, such as MP3 player, handheld device, to advanced applications on network routers, automobile and high speed cell-phone [14].

**b) Digital Signal Processor (DSP)**

DSP is a processor that targets on dedicated application (target on algorithm) with low cost and power consumption. It contains specialized hardware for fast computation and parallel programming. DSP usually deals with arithmetic manipulation such as multiplication and addition. Since these two operations often occur simultaneously, as shown in Figure 3.4, it has to include a parallel hardware so that multiplication and addition can perform in parallel way. This design will significantly increase the speed of the DSP for arithmetic operations. Besides, it can be equipped with different memories and bus architectures targeting high performance, extremely low cost and low power [46].

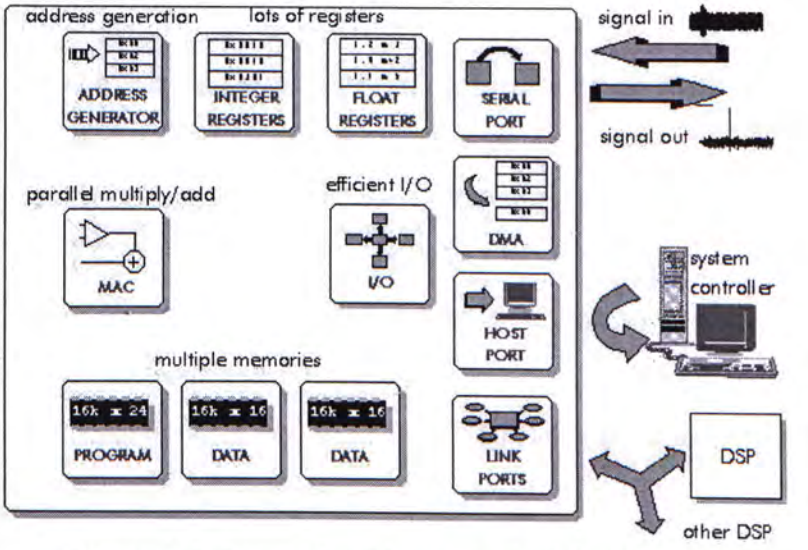



Figure 3.4: General architecture of a DSP [8]

There are quite a lot of applications that are using DSPs nowadays. Therefore, they are still important components in today's embedded systems. DSP is usually used in data domain systems. It is often used in applications in which data transport is the

dominant requirement. When a data stream enters, the DSP needs to process and give result immediately. If the hardware is not fast enough, significant loss of data will occur and the system will fail. Application examples are multimedia decoding and telecommunication. They require the process of input stream and give output immediately. This type of application has a demand for high computation performance on high throughput rather than short latency deadline. Besides, DSPs are also used in applications that often involve a lot of arithmetic calculations, such as communications, audio and video processing, GPS and navigation. With the use of DSP for dedicated processing, the runtime performance of the embedded application can be greatly improved.

The speed of a DSP has exponentially increased in the past 20 years [8]. The typical representatives of DSPs are those from Texas Instruments (TI). Both the TI C5000 [59] and C6000 series [60] are widely used in the industry. A C5000 series DSP is equipped with a 16-bit fixed-point processor. It is widely used in low-end applications such as MP3 encoding/decoding algorithm. For the C6000 series DSP processors, they support hardware caching and floating-point operations to increase the flexibility and performance for the high-end applications such as complex image processing algorithm.

**c) Application Specific Integrated Circuit (ASIC)**

ASIC is a chip that can be designed by an engineer with no particular knowledge of semi-conductor physics or semi-conductor process. It is an integrated circuit designed for a specific task. There are two types of ASIC, Field Programmable Gate Array (FPGA) and Standard Cells. A FPGA design allows more flexibility for changes and faster turnaround time. As the hardware itself can be programmable for different logics, development and feedback time can be cut short. But the cost of FPGA is expensive when comparing with Standard Cells design. It is because transistors can be wasted if the design does not fully utilize the capabilities of a FPGA chip.

For application design using Standard Cells, every transistor is used inside the hardware. The manual routing on the Standard Cells can give better utilization and lower cost [9]. In contrast to a FPGA design, a Standard Cells based design needs a longer fabrication time and therefore a lot of simulations are needed to ensure its

correctness. However, the production cost is relatively low when compared to FPGA because a smaller die size can be achieved in Standard Cells.

We have now discussed three types of embedded system processors. Their performances and programmability vary according to different processors. In Table 3.1, it shows that ASIC has no (Standard Cells) or very low (FPGA) programmability. Although the runtime performance of ASIC is very high, the programmability issue increases its development cost for application development. In the contrast, DSP and SoC processor allow more programming flexibility so that they are more suitable to implement complex software algorithms. Therefore, we decide to conduct experiments on DSP and SoC processor for performance evaluation when creating our design.

| Processor type | ASIC | DSP | SoC |
|---|---|---|---|
| Performance | High | Medium | Medium to Low |
| Programmability | None or very low | Low | Very high |

Table 3.1: Performance and Programmability of domain specific processor

## 3.3 Experiments

The objective of the experiments is to evaluate the runtime performance of an image-processing algorithm on DSP and SoC. The algorithm is a fingerprint verification algorithm [28][67] that involves a lot of floating-point computation. We divide the algorithm into four parts in order to evaluate the runtime performance of the processors.

### 3.3.1 Design methodology I – DSP

#### 3.3.1.1 Hardware platform

The low end DSP that we used is TI TMS320C52 fixed-point DSP. It is a C52 core based DSP chip for low-end device in the C5X family. It is a 16-bit fixed-point DSP operates on 40MIPS only. It has 544 words of on-chip data RAM and 512 words of on-chip data/program ROM.

#### 3.3.1.2 Software platform

For software, we used the Code Composer from TI TMS320C5X DSP as the development platform. The Code Composer is a compiler tool to convert the C language to the machine language that is readable by the TMS320C5X series DSPs. It provides a wide range of function for DSP development such as assembly programming, address decoding, DSP memory resource management and etc.

#### 3.3.1.3 Implementation

DSP software development can be divided into several fundamental tasks. They are algorithm, coding specification, compiling and optimisation, simulation, testing, debugging and integration [20]. Among all those tasks, we only focus on the core part. The core part includes algorithm compilation, optimisation and simulation.

Our colleagues have previously developed the fingerprint verification algorithm [66]. In order to implement the algorithm on the DSP platform, two methods are available:

**a) Translate all the C source codes to assembly language manually**

This method can result in a very high performance on the fingerprint algorithm. However, it is a very difficult task because the translation of few thousand lines of code to assembly language needs lots of time and effort. Moreover, it is difficult to achieve in a short period of time.

**b) Translate all the C source codes to machine language under the Code Composer software**

To shorten the integration time, we decided to use the Code Composer so that we can simulate the environment of DSP code and create a task environment. With this approach, we can simulate the interface of other software and hardware components that will minimize bugs found during integration.

Although we used the Code Composer as our development software, we still had to make certain modification to our software when integrating to the DSP platform. Below are the simple steps for migrating the fingerprint verification algorithm to DSP:

- Remove all the file I/O functions because the DSP does not support reading and writing of files.

- In order to get the fingerprint image, we need to preload the image in the memory space so that it can be used during the execution of the algorithm.

- Replace all the dynamic memory allocations to static memory allocations because dynamic memory allocation is a very time consumption procedure in DSP processing and we should avoid it in the implementation.

- The memory of the DSP is not large enough to hold the 256x256 fingerprint image and therefore we should reduce the image size to 128x128 in our experiment.

In order to verify the performance of the DSP on both floating-point and fixed-point implementations, we have implemented both approaches using the same DSP so as to review the effect of floating-point calculation on a DSP processor.

#### 3.3.1.4 Experiment and result

Figure 3.5 shows the result on the number of instructions needed for the DSP to execute the core fingerprint verification modules using fixed and floating-point techniques. From the result, we observe that if we compile a floating-point program using Code Composer on a fixed-point DSP, the performance is extremely low when compared to fixed-point implementation. We can conclude that the performance of the floating-point algorithm, which is translated from the Code Composer to DSP platform, is not acceptable. As a result, the Code Composer can help us to shorten the development time, but the result may not be useful if real-time performance is required.

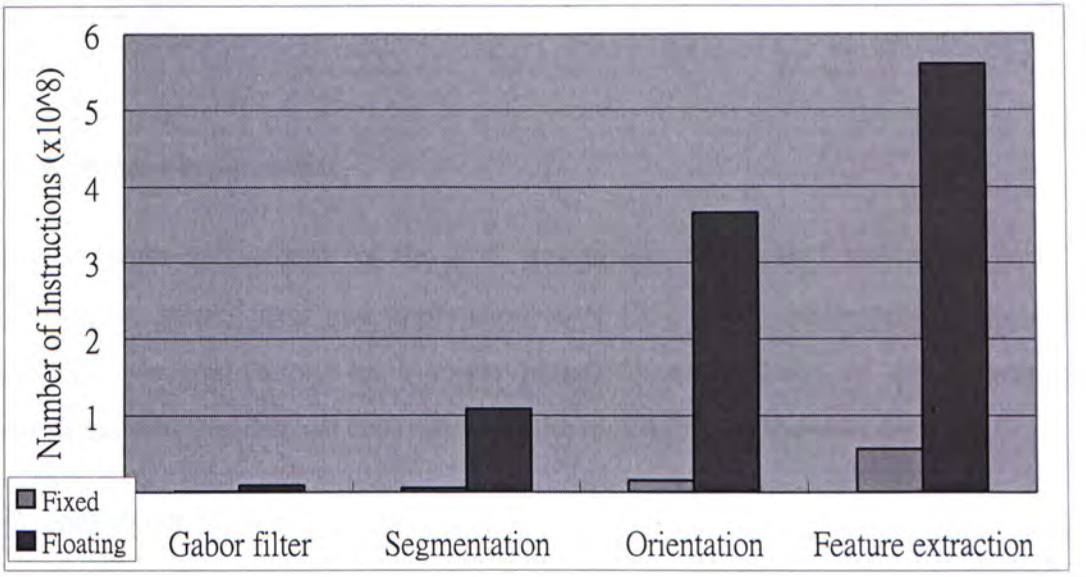



Figure 3.5: Comparison on the number of instruction on TMS320C52 of fixed and floating point in the approach to fingerprint verification implementation

The total number of instructions needed for the fixed-point fingerprint algorithm is 76774545 (~76 million) while floating-point fingerprint algorithm needed 1044426351 (~1044 million) instructions. As the speed of the DSP is operating at 40MIPS, the time required for fingerprint verification is about 1.9 second and 26.1 second on fixed-point and floating-point implementation respectively.

The result of this experiment shows that Code Composer cannot efficiently convert the floating-point algorithm for a fixed-point DSP processor. In addition, the fixed-point implementation needs 1.9 second for 128x128 pixels image. Real time cannot be achieved when using full size image (256x256 pixels). From the result, we discovered that the compiler for the DSP has difficulty in optimising algorithm. Manual modification of the algorithm, such as the involvement of assembly programming, is needed in order to have a better performance.

### 3.3.2 Design methodology II – SoC

#### 3.3.2.1 Hardware components

In SoC development, the hardware platform we chosen is Compaq IPAQ H3600 which is a Windows CE based mobile device and we have converted it into Linux operating system. The IPAQ mobile device is equipped with an Intel StrongARM

(206Mhz) embedded processor for running the fingerprint verification algorithm in our system.

#### 3.3.2.2 Software components

The software architecture of the SoC design can be divided into three layers, hardware layer, kernel layer and application layer [25]. With this approach, we can develop multiple applications on a single kernel for extensibility of the embedded fingerprint system. Besides, we can maximize the reusability on the software code.

#### 3.3.2.3 Implementation

Linking the host application and hardware is the Application Program Interface (API). We have chosen to use embedded Linux as the operating system because its open source code allows us to customize the software architecture in our system. With the use of Linux kernel, we can standardize the API for device drivers, scheduling and messaging services so as to save a lot of development time and effort.

For the application layer, we have to port the fingerprint algorithm for the StrongARM platform. As mentioned before, direct cross-complication of the algorithm is possible but it results in poor runtime performance (greater than 20 seconds in verification of one single). Therefore, we started to develop a software reengineering process to port the software to its new environment. Basically, the re-engineering process is made up of five steps:

a) Whenever possible, replace floating-point data by integers;

b) Replace the remaining floating-point calculations by fixed-point calculations;

c) Buffer all reusable complex calculations;

d) Avoid the use of subroutines;

e) Recompile using an optimised compiler.

#### 3.3.2.4 Experiment and result

The above procedure appears routine and straightforward. Yet, there are numerous domain specific conversions that must be aware of. In our case, the precisions of the fixed-point system as well as good approximations of the trigonometric functions are major concerns [11]. We have to be careful to ensure that the reengineered software yields similar results to the original software. We estimated the errors by applying the reengineered software to a 300-person database and analysis the resultant False Acceptance Rates and False Rejection Rates to make sure that they resembled those of the original software. The 4000 lines of C code took two to three months for reengineering. Ultimately, the verification time was cut to one second or less.

It took two months to finish the cost development, porting the embedded Linux OS to StrongARM development board and writing drivers to interface the fingerprint sensors. Nevertheless, the experience and the source code are reusable.

### 3.4 Observation

In the above experiments, we have made two case studies to evaluate the development and performance of two different DSP systems for multimedia and image processing applications. It seems that the DSP development requires significant adoption effort when porting algorithm to their platforms. Even with optimised compilers, assembly code implementations are still very important when optimal performance is sought. With the new SoC processors, which are equipped with some DSP-featured hardware, new strategies for designing embedded systems emerge. The new strategy that enables efficient development is made from the combinations of hardware and software technologies. With an operating system ported to the SoC processors, reuse of code and libraries can be achieved. This approach can separate the hardware design and software design of an embedded application and achieve the parallelism in development.

# 4. Implementation of the Device

## 4.1 Choice of platforms

After the analysis of the design requirements of our fingerprint authentication mobile device [40], SoC is taken as the best choice for implementation in terms of both hardware and software considerations. Its hardware flexibility and software programmability benefit the design and implementation of mobile device. With the flexibility from SoC design, we can extend the use of the authentication device to different applications like door lock systems and time attendance systems.

## 4.2 Implementation Details

### 4.2.1 Hardware implementation

Figure 4.1 shows the block diagram of our mobile device. It is made up of three main components. The Intel Lubbock is one of the main components and it is the core of our base platform for development. The Atmel finger chip and the Gemplus smart card reader are the other two components that are added as peripherals for authentication purpose.

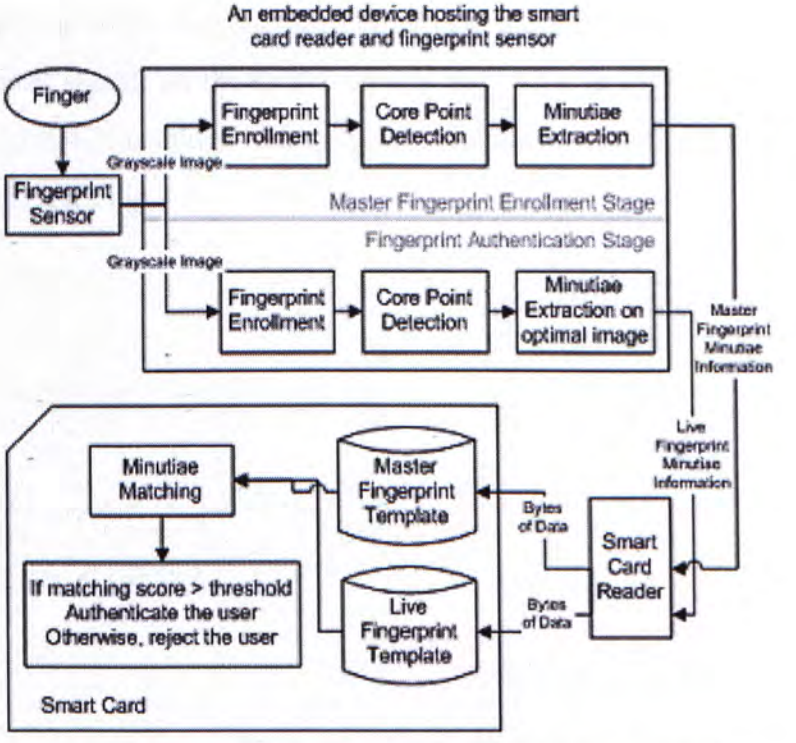



Figure 4.1: Proposed mobile device for authentication

#### 4.2.1.1 Atmel FingerChip

The finger chip operates on 3.3V with a small power consumption of 20mW with 1MHz clock cycle. The finger chip uses the temperature difference between the sensor and the fingerprint to measure the data value. If the temperature difference is too small, a temperature stabilization feature will be activated to ensure sufficient contrast exists in the captured fingerprint image.

We connect the finger chip via the GPIO of the base platform. The GPIO communication allows us to gain full control of the pins in order to communicate to the finger chip directly. Table 4.1 shows the hardware pin connections between the Atmel finger chip and the Lubbock device.

| Lubbock GPIO | LED21 | 32 | 7 | 9 | 10 | 12 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|---|---|---|---|---|---|
| Finger chip pins: | RST | PCLK | De0 | De1 | De2 | De3 | Do0 | Do1 | Do2 | Do3 |

RST: Reset pin, PCLK: Input Clock, De0-3: Even pixel, Do0-3: Odd pixel

Table 4.1: Pins connection between the Atmel finger chip and the Lubbock device

The Reset pin of the finger chip is connected to LED21 of the Lubbock to indicate finger chip reset signal. As the LED operates from 1.5V to 2V only, an extra circuit, as shown in Figure 4.2, is added to increase the voltage range in order to trigger the reset signal of the finger chip.

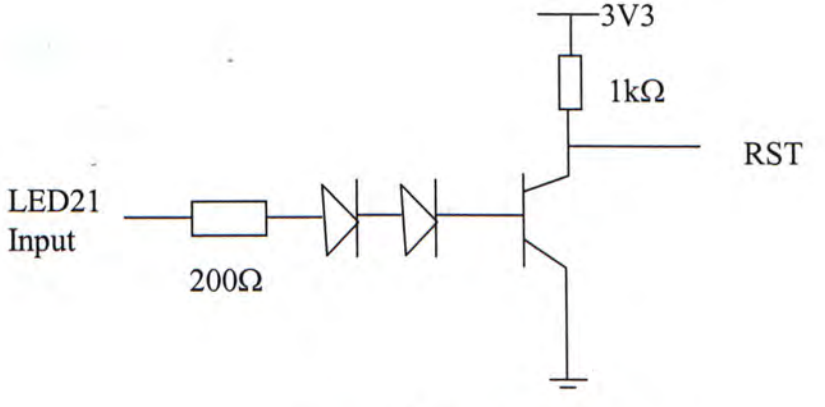



Figure 4.2: Reset pin circuit

#### 4.2.1.2 Gemplus smart card and reader

A smart card is actually a card-sized computer with a microprocessor and some memory chips packaged in the form of a paper card. The card can be programmed to suit different applications. The smart card that we use contains a Java virtual machine. It allows Java programs to run on a smart card, we choose an 8-bit processor with 20Kbytes on-card memory to reduce the cost of the device. For connection, we use the FF serial port of the Lubbock [18] to connect smart card with it.

### 4.2.2 Software implementation

#### 4.2.2.1 Operating System

On the top of our hardware, we choose embedded Linux as our operating system. We choose Linux because it is an open source operating system. We can study the source code and modify it by ourselves without any restrictions. Linux is a powerful and sophisticated system with lots of management facilities, organized device driver layer and well documented. Linux contains so many features and makes it too resource demanding when it is ported to a mobile device. Therefore, people started to develop the embedded version of Linux. Embedded Linux is a trimmed down version of Linux in which the operating system is customized to fit most embedded system requirements.

#### 4.2.2.2 File System

File system is very important to an embedded system because it stores all the system files and application programs in the device. There are several types of file systems and each of them has its advantages and disadvantages.

a) RAMDISK

Ramdisk [16] is a file system that represents a portion of memory as a hard drive. The advantage of ramdisk file system is that the access speed is very fast compared to a hard drive. The content, however, will be lost if the power of the device is cut off. Ramdisk is often used in the early stages of development in which it is often used as an INITRD [63] feature in Linux. INITRD allows a small file system to be loaded with the same mechanism as the procedure of

loading the kernel. That means if you can load the kernel to the device, you can also load the INITRD and the developer can test the device driver inside the ramdisk immediately.

b) CRAMFS

It is a compressed read only file system. It compresses a file one page at a time so that it allows random page access. It is often used in system files which stores read only partition to prevent the system from file corruption.

c) JFFS2

JFFS2 is developed from the Journaling Flash File System (JFFS). JFFS provides the storage media with the crash and power down protecting mechanism. Besides, version two (JFFS2) supports the compression that can save about fifty percent of the storage space.

After comparing the above file systems, we decided to use JFFS2. Table 4.2 shows the partition design in the Lubbock development board. The first partition is bootloader that is responsible for establishing power on communication between the host and the target board. The second partition is used for storing the parameters setting of bootloader so that we can load the setting each time we power on the board. The third partitions stores the Linux kernel that controls the entire device drivers. The last partition is the user space partition that stores all the files inside the file system. We use separate partition for the Linux kernel and the user file because we want to separate the system file from the user space. With this approach, we can easily upgrade the kernel without affecting the files stored inside the user space.

| Partition name | Flash address | Memory address | Length | Entry point |
|---|---|---|---|---|
| Bootloader | 0x04000000 | 0x04000000 | 0x80000 | 0x00000000 |
| Bootloader parameter | 0x05F80000 | 0x05F80000 | 0x40000 | 0x00000000 |
| Linux kernel | 0x04080000 | 0x04080000 | 0xC0000 | 0x00000000 |
| User file | 0x04140000 | 0x04140000 | 0x1200000 | 0x00000000 |

Table 4.2: Flash drive partition design of mobile device

#### 4.2.2.3 Device Driver

Device driver plays an important role in the Linux I/O system [10]. It controls all the hardware devices that exist in the system. A device driver provides a bridge between the user applications and the hardware so that the user programs can access the hardware with a standard API. With the device driver, the programmer can control the hardware without a need to fully understand the details of the hardware I/O operations. The device driver hides all the complicated communication. Therefore, device driver can influence the efficiency of the whole system if it is not well designed. Figure 4.3 shows the block diagram of the Linux I/O system. We can see that device driver is a layer that is the closest to the hardware. Any I/O from the hardware devices needs to pass through the driver before going to the upper software layer.

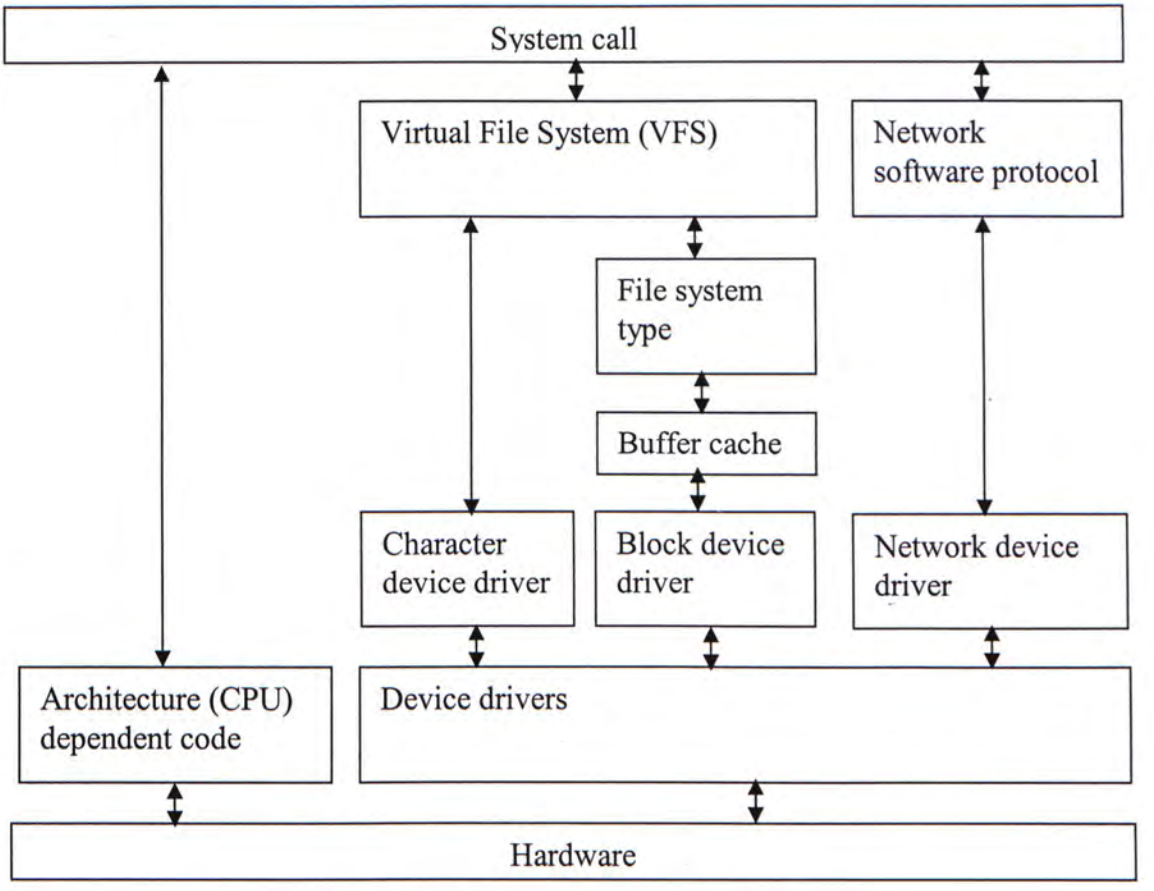



Figure 4.3: Overview of Linux I/O system

There are three main classes of device drivers in Linux operating system. They are character device driver, block device driver and network device driver.

a) Character device driver

Hardware devices that can be accessed as characters (streams of bytes) are belonging to this category. Such as serial port or serial console, we can write/read a byte to/from those character devices. In Linux, it uses file node and major number [3] to represent the device.

b) Block device driver

Different from character device, block devices can read/write a block of data to/from the hardware. This type of device can usually be accessed randomly and cache is needed to reduce the writing cost. It uses the same mechanism as the character device driver to separate the different kind of devices.

c) Network device driver

A network device driver is quite different from character and block device drivers. Linux uses name to represent different network device like eth0, eth1 and etc. Network devices often communicate with the other hosts by sending/receiving package via different networks.

In our system, there are two main hardware peripherals. They are Atmel FingerChip and Gemplus smart card reader. Both hardware devices need a software driver implementation so that it can be interfaced with the user applications.

**The Atmel FingerChip driver**

As the finger chip outputs the fingerprint image pixels as a stream of bytes, we can use a character driver to implement the software controller for the finger chip. Figure 4.4 shows the block diagram of character device driver components for Atmel FingerChip.

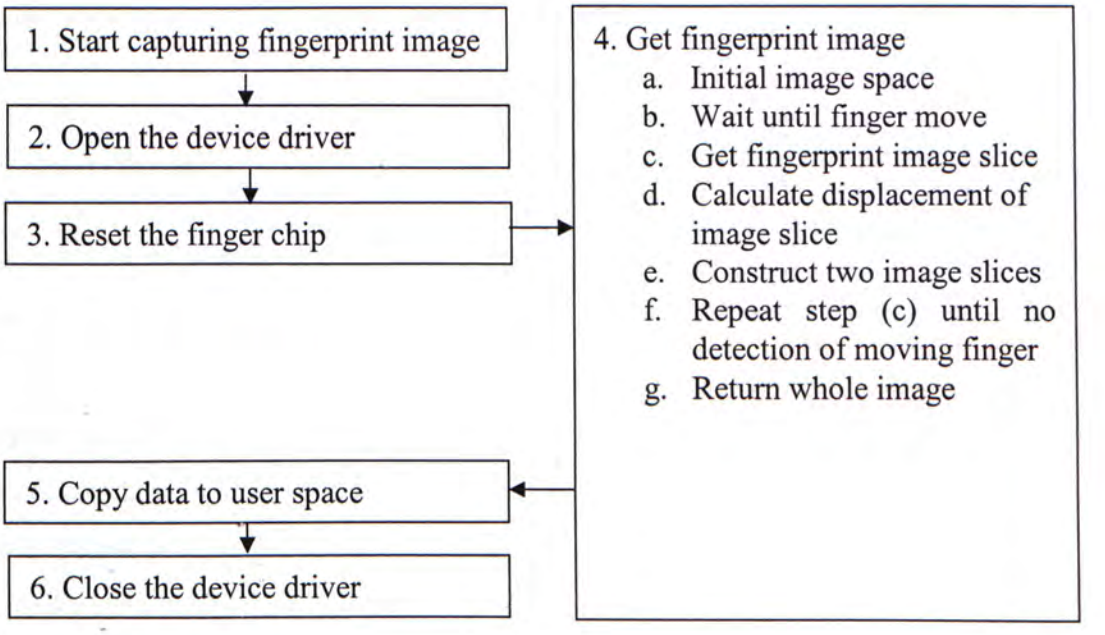



Figure 4.4: Atmel FingerChip device driver components

As the size of the fingerprint sensing area contains only 280x40 (width x height) pixels, a full image is actually obtained by joining several image slices together. The technique used in alignment the image slices is shown in Figure 4.5. We first calculate the horizontal and vertical displacement of two slices by comparing the similarity of the

pixel value along the alignment region. Then we reconstruct the whole image using a total of 11 aligned slices.

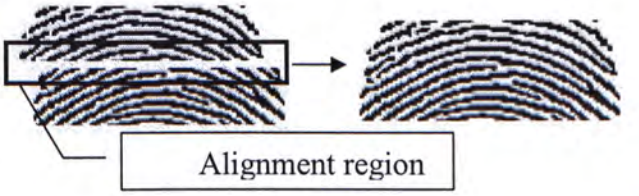



Figure 4.5: Image slices alignment technique

**The Gemplus smart card reader driver**

We do not need a device driver for the Gemplus smart card reader because it is directly connected to serial port in which a serial port device driver is already available in the standard Embedded Linux operating system. The software that we need is the communication protocol control software between the serial port and the smart card reader. The smart card we use is an ISO-7816-4 compliant smart card that is compatible to the PC/SC [36] standard. Therefore, we can use the free source code from [36] to establish the communication control between the mobile device and the smart card reader.

#### 4.2.2.4 Smart card

In today's e-commerce world, a credit card is one of the popular payment methods. Many people use a credit card because it is convenient for them to make transactions. One disadvantage is that the storage space in credit card is very limited and it is impossible to store the transaction records. In addition, security protection in credit card is not strong enough. Credit card contains a magnetized strip that is not difficult to duplicate. People may use an illegal reader to read the information in the credit card and duplicate the information in a fault card.

To solve the security and storage problem, a smart card is proposed to cover the disadvantage of a credit card. The name "smart" in smart card means it has intelligence to process data. A smart card is not only a memory card; it also contains a processor for data manipulation.

The type of card we chosen is Java card. Java card also contains a processor with Java Virtual Machine (JVM). With JVM, it is possible to execute Java bytecodes inside the card. Then we can write Java program and execute it inside the smart card.

**Design and implementation of the Java card**

In order to support fingerprint authentication, we need to implement some routines inside the Java card and hence to communicate with the host. The communication protocol between the host and smart card is called Application Protocol Data Unit (APDU). Figure 4.6 shows the general APDU command sequence. Table 4.3 shows the design of the Application Protocol Data Unit (APDU) inside the smart card.

| CLA | INS | P1 | P2 | Lc | Data | Le |
|---|---|---|---|---|---|---|

CLA: Application class ID
INS: Instruction ID for specific application
P1, P2: Specific definition for single instruction ID
Lc: Number of bytes transfer to the smart card
Data: Bytes sequence send to the smart card
Le: Number of bytes return from the smart card

Figure 4.6 General APDU byte sequence

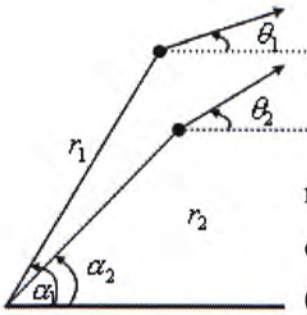
| Command | CLA | INS | P1 | P2 | Lc | Data | Le |
|---|---|---|---|---|---|---|---|
| Register master fingerprint template | 0x00 | 0x38 | 0x00 | 0x00 | 0x181 | Master fingerprint template | ---- |
| Download live fingerprint template | 0x00 | 0x43 | 0x00 | 0x00 | 0x81 | Live fingerprint template | ---- |
| Match live template with master template | 0x00 | 0x44 | 0x00 | 0x00 | ---- | Matched score | 0x02 |

Table 4.3: APDU for fingerprint matching inside smart card

Because the storage size of a smart card is very limited, we need to compress the fingerprint features in the host before loading them to the smart card. The routine for downloading master and live fingerprint templates (features) from a host to smart card is trivial. We define two variables in the Java applet to store two templates. The Lc column of Table 4.3 shows that the size of the master template is larger than the live

template. This is because the master template is used for matching reference. The master template needs to contain more minutia information in order to yield more accurate matching result. In contrast, a live template will be used once only and therefore we can reduce the number of minutiae in the live template to achieve a faster matching and transferring time.

In the fingerprint template matching routine, we need to perform a point-to-point matching in the smart card. This is significant because the confidential master template data never leaves the smart card. Each feature vector in the template contains the distance to the core point (r), minutiae orientation ($\alpha$) and the rotated orientation ($\theta$) as shown in Figure 4.7. For two minutiae, if the differences of their distances and orientations are within some threshold values, i.e. $|r1-r2| < 8$ and $|\alpha1-\alpha2| < 35$, they are regarded as matched. We will then assign marks to the matched live minutiae according to their Euclidean distance. Moreover, marks will be adjusted according to the difference between their rotated orientations. If the difference is smaller than a defined threshold value, i.e. $|\theta1-\theta2| <$ threshold, the mark will be multiplied. [66]




$r_i$ is the distance to core$_i$

$\alpha_i$ is the orientation of a minutia i

$\theta_i$ is the orientation of a rotated minutia i

Figure 4.7: Two minutiae in polar coordinates

We use the following routine to match two fingerprints:

```
For i=1 to Minutiae_Number N
{
    For j=1 to Minutiae_Number N
    {
        If (orientation between minutia i and j <=35 AND
            distance to core point between minutia i and j <= 8) {

            Regard j as "Matched"
            Mark   = a fixed value – orientation x distance difference
        }
        If (rotated orientation between minutia i and j <= threshold)
            Mark is multiplied
    }
    If (more than one j's are marked)
    {
        Resolve the conflict by choosing the j with the highest mark
    }
    Delete the corresponding j in the selection set of Minutiae in the live template
}
```

The matching score will be between 1 and 100. If the score is higher than a predefined value, the user is authenticated as the valid cardholder. Otherwise, the user may repeat the authentication process to verify his identity again.

#### 4.2.2.5 Fingerprint software

For the fingerprint software library, we have developed a fixed-point fingerprint verification system [28] that can be used in embedded system applications. The software can be divided into two parts: feature extraction and feature matching. The feature extraction of the software can be either executed on a desktop PC or on the mobile device.

#### 4.2.2.6 Graphical user interface

A Graphical User Interface (GUI) is important in our secure mobile device because we need to display captured fingerprint image and user messages on the LCD display of

the mobile device. There are several GUI toolkits available for the embedded Linux environment:

**a) PocketLinux**

PocketLinux is a Java based windowing system. It uses open source implementation of Java virtual machine, named Kaffe. The virtual machine can be scaled down to suit different embedded devices. It has two low level layers for handling framebuffer display and touch screen. On the top of the low level layers, it contains a PocketLinux class that has implemented its own widgets. Then it uses XML for programming the GUI of the display. Since Java is pretty slow even it is trimmed down, its runtime performance is not very efficient.

**b) Microwindows**

Microwindows aims at supporting a graphical windowing environment in an embedded device. It mainly contains two layers: Microwindows and Nano-X. Microwindows is a device dependent layer for controlling framebuffer drawing. Nano-X is a device independent layer that builds on top of Microwindows layer. It aims at providing a unique API for writing GUI. As the core of Microwindows is not designed very well and it causes flashing problem when we draw widgets. It is also not suitable for embedded device.

**c) Qt/Embedded**

QT is a windowing system providies full features including window manager, application launcher and input methods. The Qt/Embedded version is designed for an embedded device. It uses C++ language for GUI programming and abstracts the C object layer. Therefore, it is very efficient in running on the embedded device.

After the analysis on the pros and cons of different embedded GUI toolkits, we decide to choose QT. Although QT/Embedded have already provided a complete framework for embedded GUI programming, some low-level software integrations such as touch screen compatibility need to be redesigned in order to make it compatible for our device.

Since QT does not support the touch screen of Intel Lubbock touch screen driver, we need to write our own version of touch screen routine to handle the signal from the kernel. Figure 4.8 shows the routine for touch screen used inside QT. We define the structure as shown in Figure 4.9 to receive the touch screen information from the hardware.

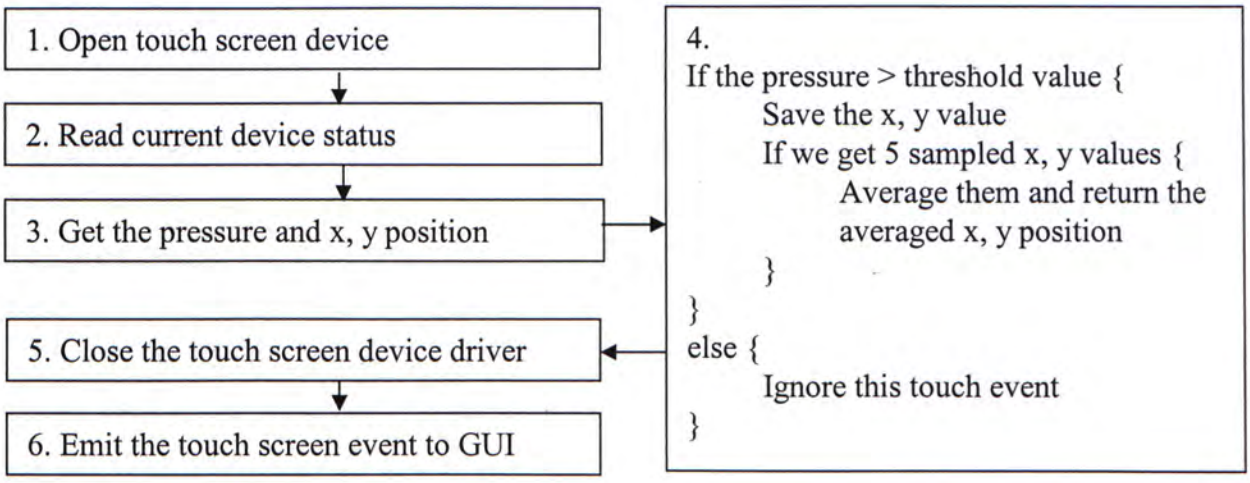



Figure 4.8: Touch screen interface between QT and device driver

We use a threshold value to filter out some touch screen events because the touch screen is very sensitive for external signals. Noise can appear even when there is no real touch event. For sampling technique, we use averaging so that we can get a more accurate result. After we have solved the touch screen problem, we can compile the toolkits using the XScale compiler and QT/Embedded library is successfully integrated to our mobile platform.

```
typedef struct {
        unsigned short pressure;
        unsigned short x;
        unsigned short y;
        unsigned short pad;
        unsigned short stamp;
} TS_EVENT;
```

Figure 4.9: Structure of touch screen raw data

## 4.3 Results and observations

### a) Fingerprint matching accuracy of our secure mobile device

The accuracy of our proposed system depends on the feature extraction and matching processes. The feature extraction process is discussed in detail in [28]. The matching algorithm that is executed in a smart card is difficult to test. Therefore, we evaluated its performance through simulation experiments on a PC with Redhat Linux 9.0 and JDK1.4.1 installed. A fingerprint database that contained 383 different fingerprints supplied the test data. Each record in the data was created when a person scanned his or her finger 3 times through a fingerprint sensor, resulting in 1149 fingerprint images in total.

The Java minutiae matching program used in this experiment was exactly similar to the one used in our smart card. Only byte and short Java primitive types of variables were used in the matching program.

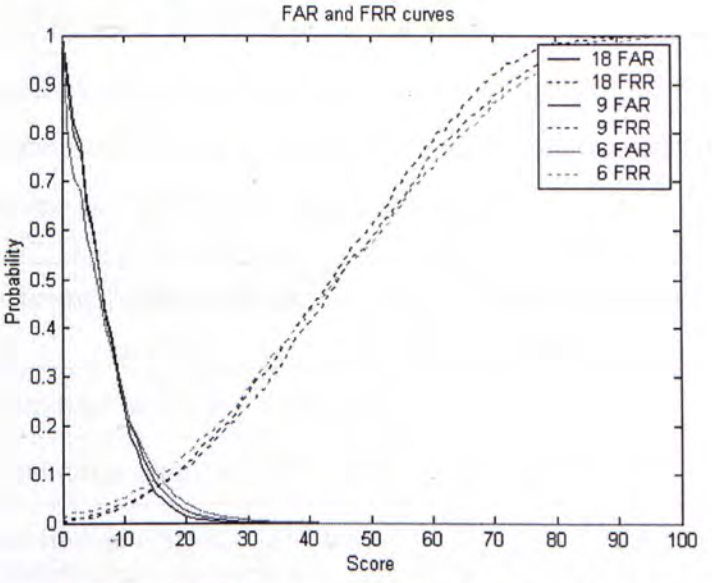



Figure 4.10: FAR and FRR curves of 6, 9 and 18 matching minutiae used

The 1149 fingerprint images were registered and their minutiae data were written into templates in binary format correspondingly. Using a methodology that optimises the number of minutiae used in a matching process [28], we varied the number of matching minutiae, choosing 6, 9 and 18, and obtained the False Accept Rates (FAR) and False Reject Rates (FRR) as shown in Figure 4.10. The results of the test on the 1149 fingerprint images showed that the Equal Error Rates (EER) rates were about 9.6%,

7.9% and 7.2% when 6, 9 and 18 minutiae were used for matching respectively. The test results are summarized in Table 4.4.

| Number of Minutiae used | EER (%) |
|---|---|
| 6 | 9.6 |
| 9 | 7.9 |
| 18 | 7.2 |

Table 4.4: EERs of the matching algorithm against different number of minutiae

**b) Speed of the whole authentication algorithm**

In another experiment, we tested the speed of the complete authentication system. We used different numbers of minutiae in the matching: 9, 18 and 25 minutiae. Since the storage requirement of each feather vector is 12 bytes and the header of each template (image quality, fingerprint type, number of minutiae, etc) is 17 bytes, the sizes of the 9, 18 and 25 minutiae templates are 125, 233 and 317 bytes respectively. During authentication, the features are extracted in the mobile device and downloaded to the smart card in a 128 bytes/transfer fashion. Hence, the time of transferring a 25-minutia template file is about triple that of transferring a 9-minutia template file. Table 4.5 summarized the average transfer time of different sizes of template.

| Different Template Sizes | Average Transfer Time (s) |
|---|---|
| 9 minutiae template (125 bytes) | 0.70 |
| 18 minutiae template (233 bytes) | 1.28 |
| 25 minutiae template (317 bytes) | 1.89 |

Table 4.5: Average times of downloading template data to a smart card

| Different Minutiae used | Average Time used in matching(s) |
|---|---|
| 9 minutiae | 1.10 |
| 18 minutiae | 3.28 |
| 25 minutiae | 6.49 |

Table 4.6: Average matching times using different number of minutiae

The runtime of the matching algorithm is O(N2), where N is the number of minutiae used in the matching. The average times of different minutiae used in the matching algorithm are listed in Table 4.6. The time for minutiae extraction is a function of the number of minutiae used and is less than 10 milliseconds. This amount is negligible when compared to the time used in data transfer and minutiae matching in the smart card. It is clear that the number of minutiae used greatly affects the times of template transfer and minutiae matching on the smart card.

The above analysis, with experimental result support, shows that it is feasible to conduct the fingerprint authentication using an 8-bit smart card processor on the embedded Linux. The whole fingerprint authentication can be completed in around 2 seconds in our proposed system if 9 matching minutiae. In this way, our system achieves the dual goals of speed and reliability.

# 5. An Application Example – A Penalty Ticket Payment System (PTPS)

## 5.1 Requirement

Figure 5.1 shows the procedure for handling a traffic offence in an open street. It involves a driver, his driving licence, a policeman, penalty, .... The procedure is a legal case so that it must be handled formally. Moreover, if the offence is minor, the penalty can be settled by fixed sum of money. Under such a situation, monetary transfer becomes necessary.

In the beginning, a driver's car is first stopped by a policeman. The policeman verifies the driver's identity by checking the driver's photo on the license. But at the same time, the driver may not know whether the policeman is genuine or not. To settle this matter, authentication becomes necessary. If a driver agrees to pay the penalty, monetary transfer becomes necessary. Such a complicated matter can be nicely handled using our intelligent mobile smart card reader.

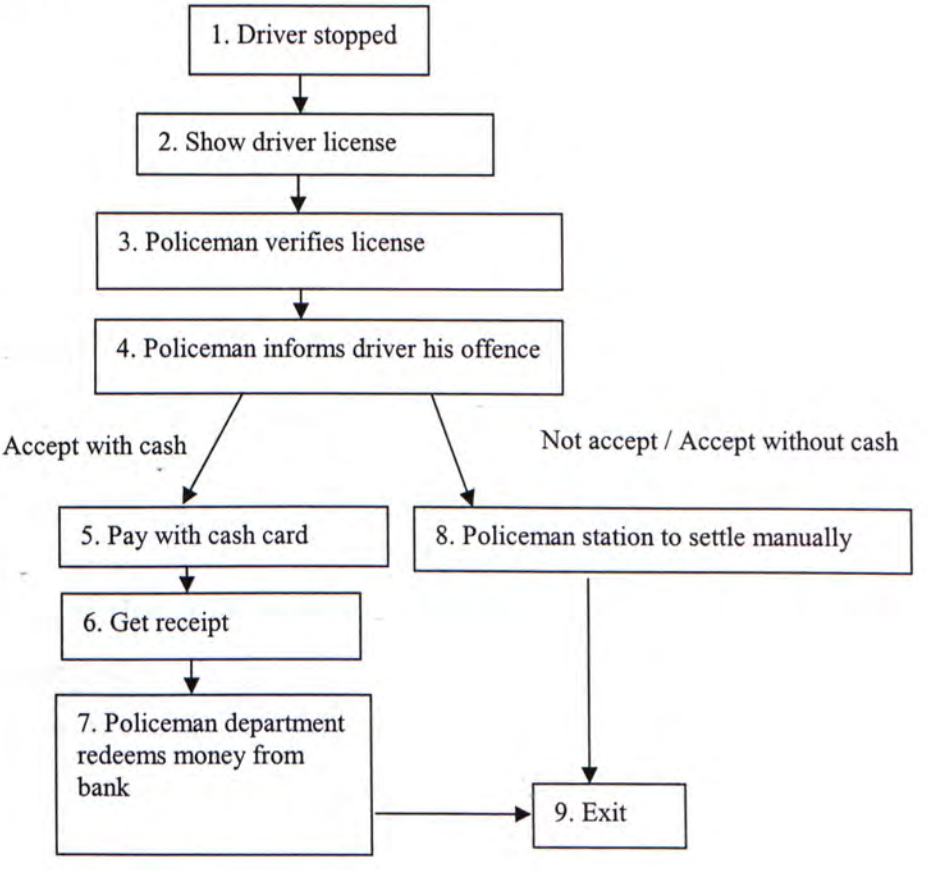



Figure 5.1: General situation when the driver involve in a traffic offence

## 5.2 Design Principles

Before we design the penalty ticket system, we need to determine the parties involved in the system. There are four parties, namely, the driver, policeman, bank and trust organization. Figure 5.2 shows the relationships of all the parties involved in the penalty ticket system. The policeman carries the intelligent mobile smart card reader with him. The ideal intelligent mobile smart card reader should look like a PDA equipped with a smart card reader. From hereon, we shall refer to the device simply as a PDA.

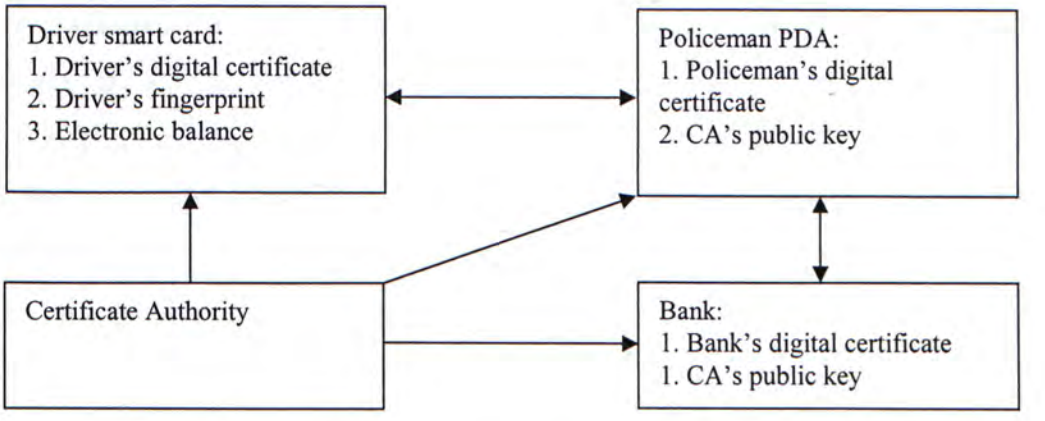



Figure 5.2: Parties involved in PTPS

In Figure 5.2, we can see that the Certificate Authority (CA) builds up a channel to all other parties. The CA is the party responsible for issuing digital certificate. It is a trust organization that the client can obtain a digital certificate from. When two clients invoke any transaction, they can verify each other by verifying their certificates issued by the same CA. Therefore without the certificate issued by the same CA, two parties cannot verify each other. In other case, it is reasonable to assume that the common CA is the government.

The driver is another party invoked in the system. He is issued a driver licence that is a smart card containing his digital certificate, fingerprint and cash balance. Although the digital certificate can help the policeman to verify the validity of the smart card, the policeman cannot verify whether the smart card belongs to the driver or not. This is the basic authentication problem encountered in many different applications [13]. In order to verify the identity of the driver, the policeman can verify the driver's live fingerprint with the one registered in the smart card. Then both the identities of smart card and

driver can be verified. The cash balance stored inside the smart card is used for paying the penalty due to the traffic offence.

The policeman is another party who issues traffic offence to the driver. The fingerprint sensor in his PDA is used to capture the live fingerprint from the driver. The PDA's smart card reader is used to accept the driver licence (smart card) from the driver. The PDA also contains a CA's public key for verifying the genuineness of the driver licence using the digital certificate stored inside the smart card.

The bank is the party directly responsible for handling the monetary transfer. All money transactions between the PDA and the smart cards are recorded in the PDA. After one day's work, the policeman will bring this PDA back to the police station where the PDA will be connected to a network that links the bank. The bank will decode each transaction and transfer the amount to the police's account. We will assume that the driver can "buy" credit in the form of electronic coins from the bank and store the coins in his driver licence, the smart card. When necessary and under his authorization, the driver can shift the coins to the policeman's PDA.

To implement the PTPS, we need to ensure the security issues following the steps below:

**a) Driver verifies policeman**

In this step, the driver needs to have a secure and reliable mechanism to verify whether the policeman should be trusted or not. He can do this by checking the policeman's digital signature stored inside the PDA. This verification will be a further safeguard when the driver transfers electronic coins to the PDA. Figure 5.3 shows this normal verification procedure.

After the transfer of the coins, the policeman's information as well as other details of the offence will be transferred to the driver's smart card using a MD5 hashing method or similar. The hashed information that is equivalent to a receipt for the penalty fee will be stored in the card's memory, say H1. Secondly, the CA signature will also be transferred to the card that will perform a RSA decryption using CA's public key to make sure that the receipt has been "signed" by a proper authority. Inside the card, the

resultant decrypted data, say H2 will be compared with H1. If H1 equals to H2, it means that the digital certificate of the policeman is valid and he has the right to access the driver's smart card licence.

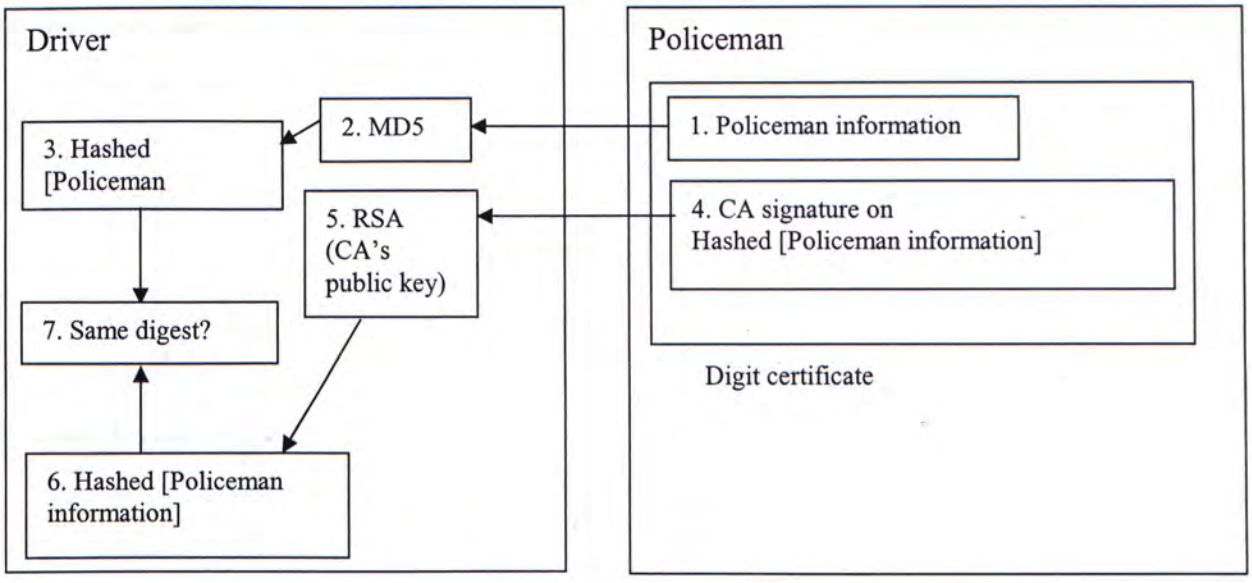



Figure 5.3 Driver verifies policeman using digital signature

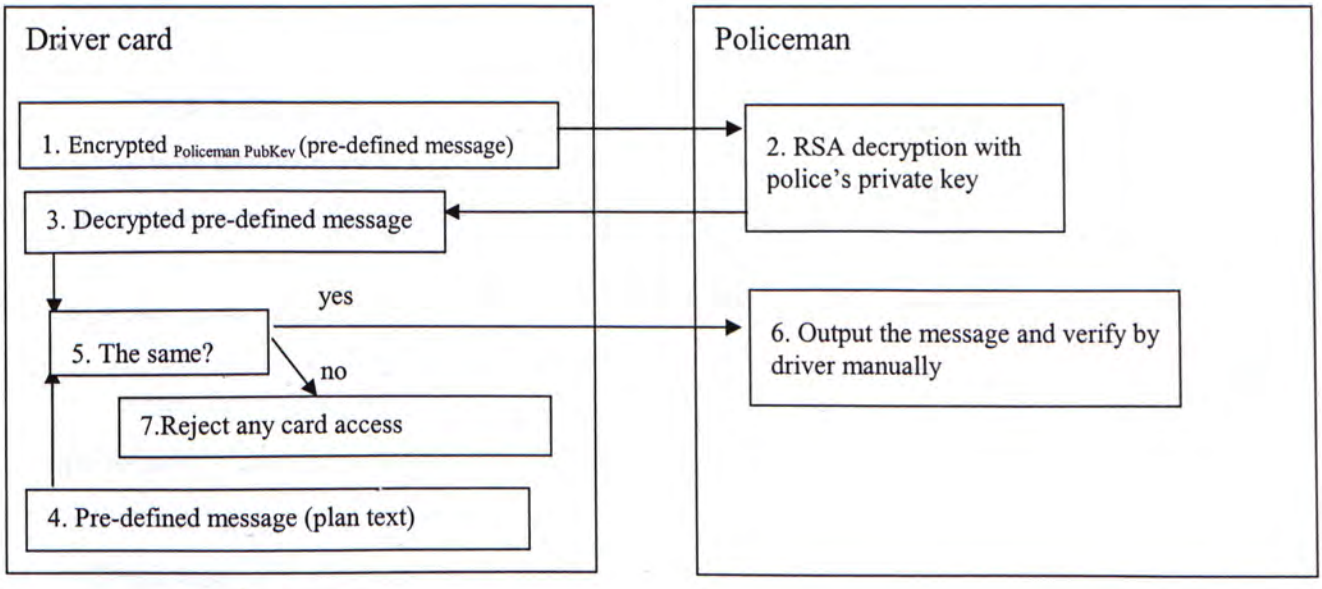



Figure 5.4 Driver verifies policeman using a low speed processor smart card

Unfortunately, the above procedure is not feasible in our implementation because the smart card only has a 5MHz processor that is not fast enough to perform the MD5 hashing and RSA decryption. There are some fast enough smart cards equipped with special built-in PKI hardware to perform those cryptographic operations. We have not chosen such cards because they are too expensive. Therefore, we redesign the driver

verification procedure in order to let a low speed processor to verify the policeman's PDA.

Figure 5.4 shows the block diagram of how a driver can verify a policeman using a low speed smart card. To verify the policeman, it is the same as verifying whether the policeman has a valid private key or not. Therefore, in every driver smart card, we decide to add a pre-defined message that is encrypted with policeman's public key. The encryption is done when the smart card driver licence is created. Now, both the encrypted and original message chosen by the driver are stored inside the card. When the smart card is inserted into the policeman's PDA, the encrypted message will be transferred to the PDA that will decrypt the message using the police's private key and send the decrypted message back to the smart card. The verification procedure then is only a text base comparison and a 5MHz processor is capable of performing such an operation. If the verification fails, that means the policeman cannot present a correct private key and any card access will be rejected. If the verification succeeds, the message will be sent back to the police's PDA that will display the message to let the driver have a manual check of his/her own recorded message.

This scheme can achieve the same security level as if a digital signature verification technology has been used. One drawback of this approach is that the driver will have to go to the card issuing office whenever he or she wants to use a message.

**b) Policeman verifies driver**

After the driver has successfully verified the identity of the policeman, it is the policeman's time to verify whether the driver card belongs to the driver. This step is important because the driver may present a driver card that does not belong to him. When issuing a new driver card to a driver, the driver needs to register his fingerprint so that his personal information is stored inside the driver card. No one can claim to be the owner of the driver smart card unless his fingerprint matches the stored record inside the smart card. Once the fingerprint is captured by the policeman's PDA, the fingerprint's features will be extracted by the PDA and sent to the smart card for matching as described previously.

**c) Transfer of Payment**

After both the driver and the policeman have verified the identity of each other, the policeman can start preparing the offence payment. Figure 5.5 shows the block diagram of the payment preparation procedure. Firstly, the policeman's PDA prepares a penalty record by collecting information such as payment ID, driver ID, and payment amount from the driver smart card as well as the policeman's PDA itself. Then, the electronic coins inside the cash card are transferred to the policeman's PDA. If the transfer is successful, the PDA will encrypt the payment record using the bank's public key. There is an option that a tiny printer is connected to the PDA for printing a payment receipt for the driver. The policeman's PDA encrypt the payment information in the form of a receipt.

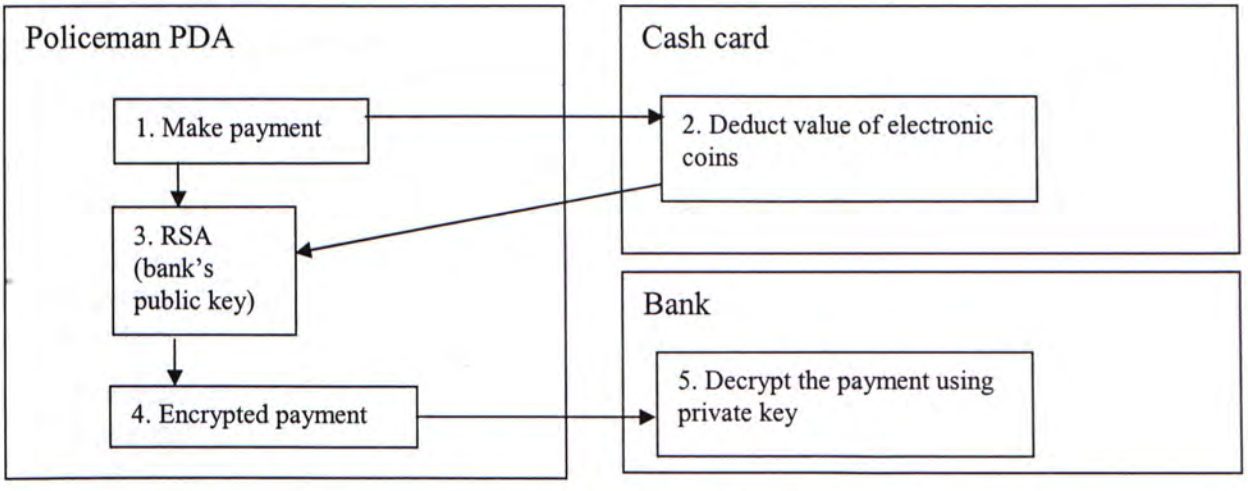



Figure 5.5 Payment preparations

**d) PDA verifies policeman**

Before the policeman can use the PDA, the policeman needs to present his policeman card to the PDA and to be verified using his fingerprint. This can prevent unauthorized person to steal the PDA for illegal use. The authentication mechanism is almost the same as the one in part b.

## 5.3 Implementation

To implement PTPS, we use the OpenSSL [58] as the core library and integrate it with the fingerprint verification and smart card modules in our secure mobile device, i.e. the policeman's PDA. OpenSSL is a robust open source library for the purpose of

cryptography operations. It provides a variety of cryptography functions that can be used to develop different secure applications. Famous protocols such as Secure Sockets Layer [54] and Transport Layer Security [62] are also supported in OpenSSL. To support the PTPS system, we have developed a software library on top of OpenSSL. The software library is divided into two core modules; they are the card module and the embedded device module. The card module is responsible for the handling of the digital certificate and the embedded device module is responsible for the cryptography operations conducted inside the policeman's PDA.

**a) Driver card implementation**

To support PTPS, the driver card needs to include three sources of information; they are the digital certificate, fingerprint and electronic balance of the driver. As we have already developed a smart card that support fingerprint verification (in section 4.2.2.4), therefore we can add new modules to support the remaining functions.

| Command | CLA | INS | P1 | P2 | Lc | Data | Le |
|---|---|---|---|---|---|---|---|
| Register master fingerprint template | 0x00 | 0x38 | 0x00 | 0x00 | 0x181 | Master fingerprint template | ---- |
| Download live fingerprint template | 0x00 | 0x43 | 0x00 | 0x00 | 0x81 | Live fingerprint template | ---- |
| Match live template with master template | 0x00 | 0x44 | 0x00 | 0x00 | ---- | Matched score | 0x02 |
| Get balance | 0x00 | 0x30 | 0x00 | 0x00 | ---- | Balance | 0x02 |
| Credit balance | 0x00 | 0x32 | 0x00 | 0x00 | 0x02 | Credit value | ---- |
| Debit balance | 0x00 | 0x31 | 0x00 | 0x00 | 0x02 | Debit value | ---- |
| New Message | 0x00 | 0x40 | 0x00 | 0x00 | 0x14 | Pre-defined message | ---- |
| Verify PDA | 0x00 | 0x41 | 0x00 | 0x00 | ---- | ---- | ---- |
| Put certificate | 0x00 | 0x36 | 0x00 | 0x00 | 0x460 | Certificate | ---- |
| Get certificate | 0x00 | 0x37 | 0x00 | 0x00 | ---- | Certificate | 0x460 |

Table 5.1: APDU of the driver card

Table 5.1 shows the APDU of the driver card. Seven operations are added to the smart card to support PTPS. Get balance, credit balance and debit balance commands are used to handle the electronic coins inside the smart card. Put certificate and get certificate commands are used to handle the digital certificate of the driver. New message command is used to update the pre-defined message (as mentioned in section 5.2) stored inside the smart card. Verify PDA is used to verify the validity of the policeman PDA using the original and pre-defined message.

**b) Secure software library implementation**

We have developed the six software modules for PTPS based on OpenSSL. The secure software modules provide a complete secure infrastructure in our system.

1) GEN_CERT

This module generates a digital certificate. Before we execute this module, a subject file that contains the owner information is needed to construct a digital certificate. When this module is executed, it first generates a pair of RSA keys (public key and private key). Then it uses the subject file and the public key to generate the digital certificate. This digital certificate now contains the owner name, owner organization, owner public key and related information. In the last step, it converts the certificate to the X.509 [6][2] format. The X.509 standard is a popular standard used in digital certificate for representing data fields and distribution of pubic keys. With this module, we can generate certificates for different parties in PTPS such as driver and the policeman department.

2) SIGN_CERT

Before a digital certificate can be used, it needs to be digitally signed by the CA. When this module is executed, it adds the CA's subject name to the issuer field of the certificate. Then it signs the certificate with the CA's private key and finally outputs the signed certificate to the system.

3) ADD_INFO

Besides the standard data fields defined in X.509 format, we can add extra information to the digital certificate for specific usage. For example, in PTPS, we can add the driver ID to the certificate so that any modification of the driver ID will cause failure in verification. Data added to the certificate cannot be changed as it is encrypted by the CA's private key.

4) ADD_ENCR

This module allows the addition of encrypted message in the certificate. The encrypted message is mainly used for the verification of policemans's PDA in our system. When it is executed, it reads the plain text message and encrypts the message with policeman's private key. Then the encrypted message is added to the digital certificate using the ADD_INFO module.

5) VERIFY_PDA

This module verifies the police PDA using the encrypted message stored in the driver card. When this module is executed, it reads the certificate from the driver card and uses the policeman's private key to decrypt the message. The decrypted message will then be sent back to the driver card for in-card verification.

6) VERIFY_DRIVER

This module verifies the driver using the digital certificate stored inside the driver card. When this module is executed, it reads the certificate from the driver card and uses CA's public key to verify the validity of the certificate. This module only contributes part of the driver verification process because after the certificate verification is finished, the fingerprint verification process will be called by the system to verify the fingerprint of the driver.

The above six modules are integrated into our secure mobile device to enable PTPS. Figure 5.6 shows the flow diagram of how a digital certificate is created for different parties. Firstly, the CA itself needs to generate its own certificate so that other parties can get the public key from that certificate for verification. This process will involve GEN_CERT, ADD_INFO and SIGN_CERT modules one by one. The certificate of policeman department will use the same procedure as CA. But for the

certificate of the driver, ADD_ENCRY module is needed between ADD_INFO and SIGN_CERT so that pre-defined message can be encrypted and stored inside the driver's certificate.

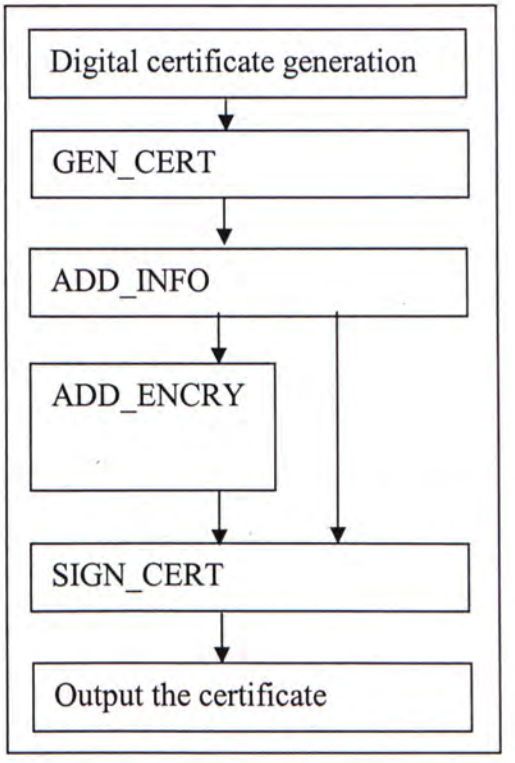



Figure 5.6: Certificate generation process

Figure 5.7 shows the software and hardware architecture of PTPS in the secure mobile device. It describes the communication between the fingerprint and smart card modules with the PTPS secure modules. As shown in the diagram, we separate the software layer into an application layer and a library layer. The library layer contains the fingerprint module and smart card module. These two modules are the core software libraries that are responsible for communicating with the hardware device and conducting verification using fingerprint sensor and smart card. We separate this library layer from the application layer so that we can develop different applications based on our secure mobile device. On top of the library layer, the application layer contains all the related modules for PTPS. With this approach, the application development can be isolated from the lower layer so that we can modify the hardware layer and library layer without affecting the application layer. Therefore, portable application can be achieved.

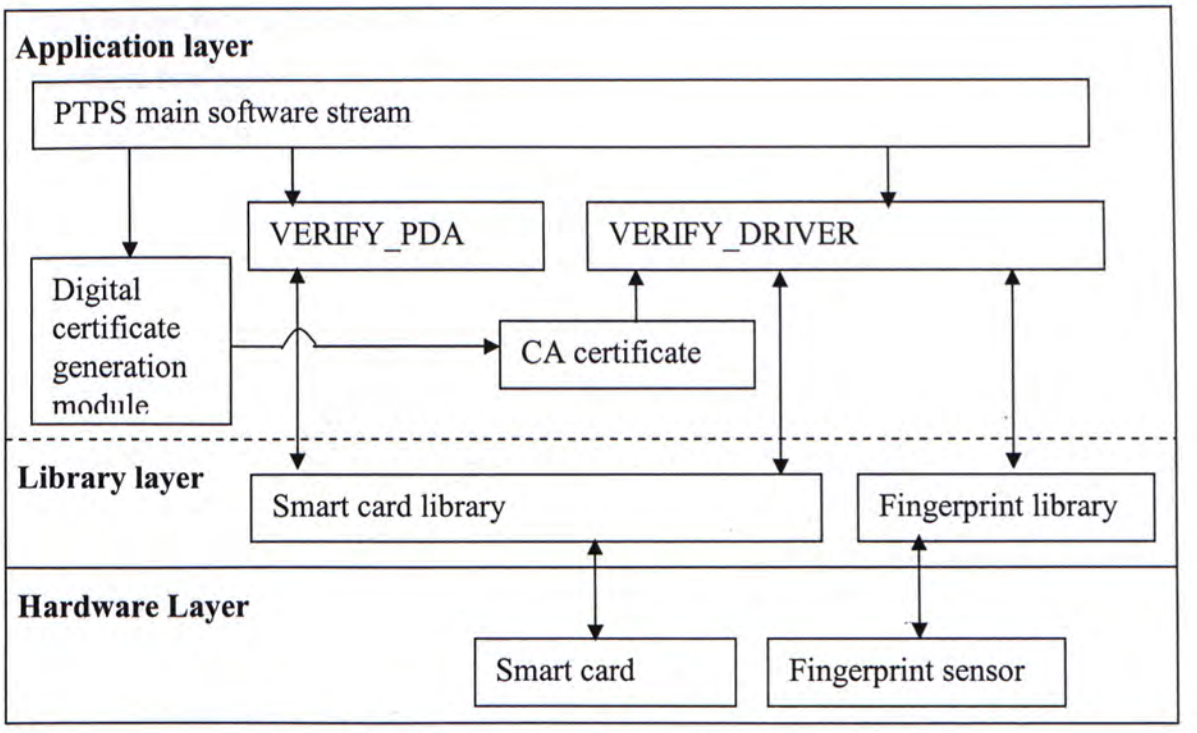



Figure 5.7: Software and hardware architecture of PTPS

## 5.4 Results and Observation

Since PTPS is used in a mobile device, speed is a critical factor. Experiments on different modules are conducted to evaluate the performance. Figures 5.8 to 5.13 show the runtime performance of secure modules GEN_CERT, ADD_INFO, ADD_ENCRY, SIGN_CERT, VERIFY_DRIVER and VERIFY_PDA, respectively. Each module is executed a hundred times and the average result is calculated. We simulate the driver smart card as a normal file so that we can ignore the influence of smart card I/O overhead. Result shows the generate certificate is the most time consuming part among all the secure modules, because a time consuming process, the RSA key pairs generation, is involved in this step. ADD_INFO, ADD_ENCRY and SIGN_CERT are part of the certificate generation process. Those modules can achieve acceptable performance even if they are executed in a mobile platform (SA1110 and PXA250 processors). As certificate generation is only executed once for issuing a new driver card at the beginning, it is not a must to achieve real time performance. If necessary, it can be done in a desktop PC for efficiency consideration. The VERIFY_DRIVER and VERIFY_PDA, modules are often involved because the policeman and different drivers need to use them for mutual authentication. The most time consuming parts of these two

modules are for reading and decrypting of information in certificates. Experiments show that these two modules can achieve real time performance in a mobile platform.

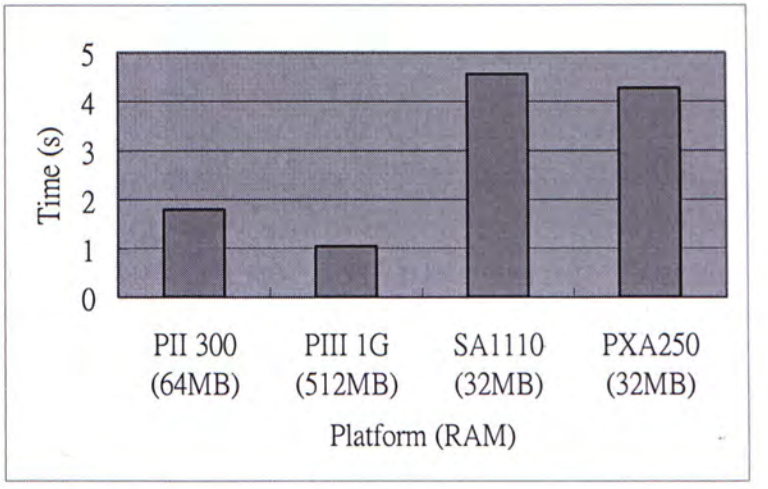



Figure 5.8: Performance of certificate generation in different platforms

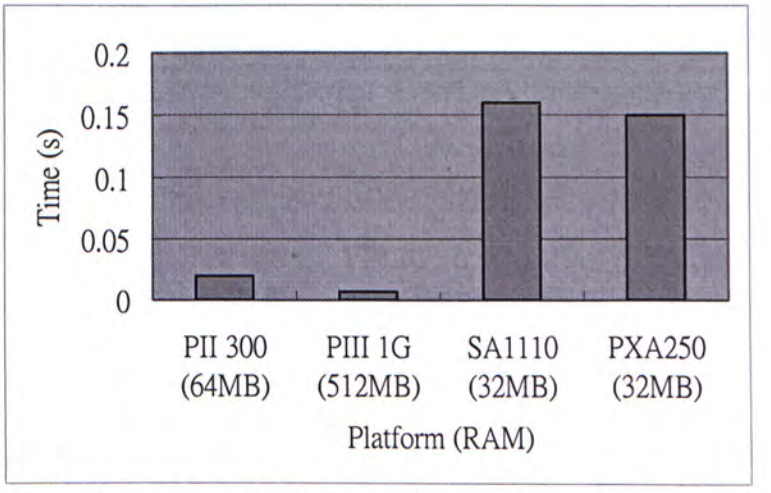



Figure 5.9: Performance of adding certificate information in different platforms

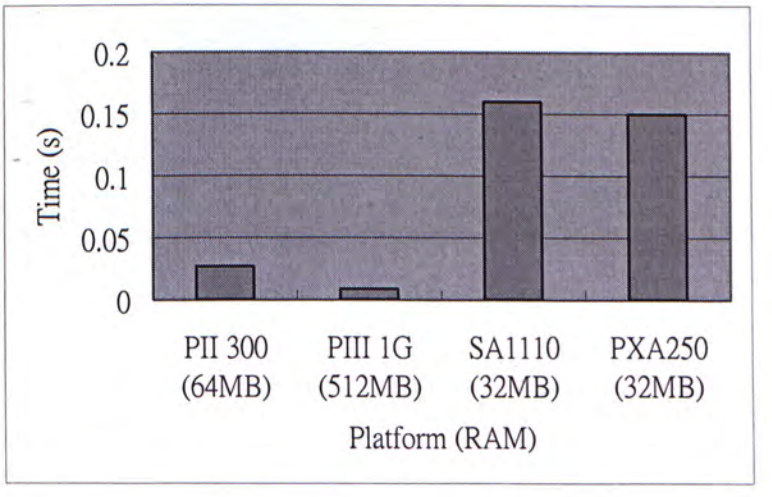



Figure 5.10: Performance of adding encrypted information in different platforms

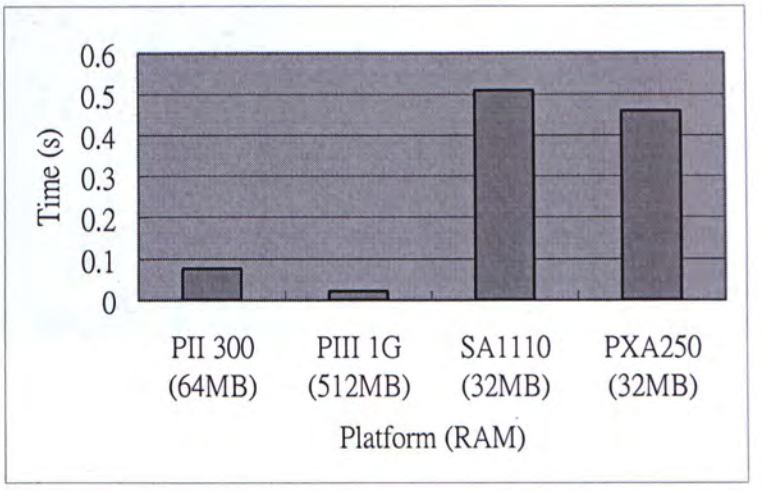



Figure 5.11: Performance of signing certificate in different platforms

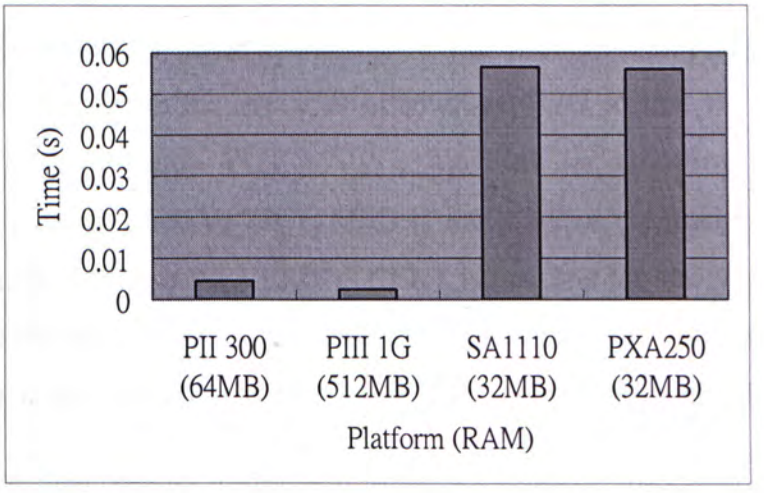



Figure 5.12: Performance of verifying driver in different platforms

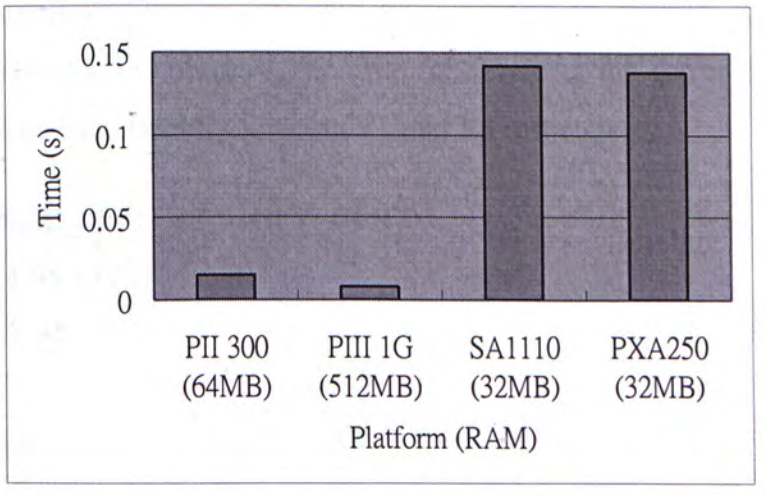



Figure 5.13: Performance of verifying PDA in different platforms

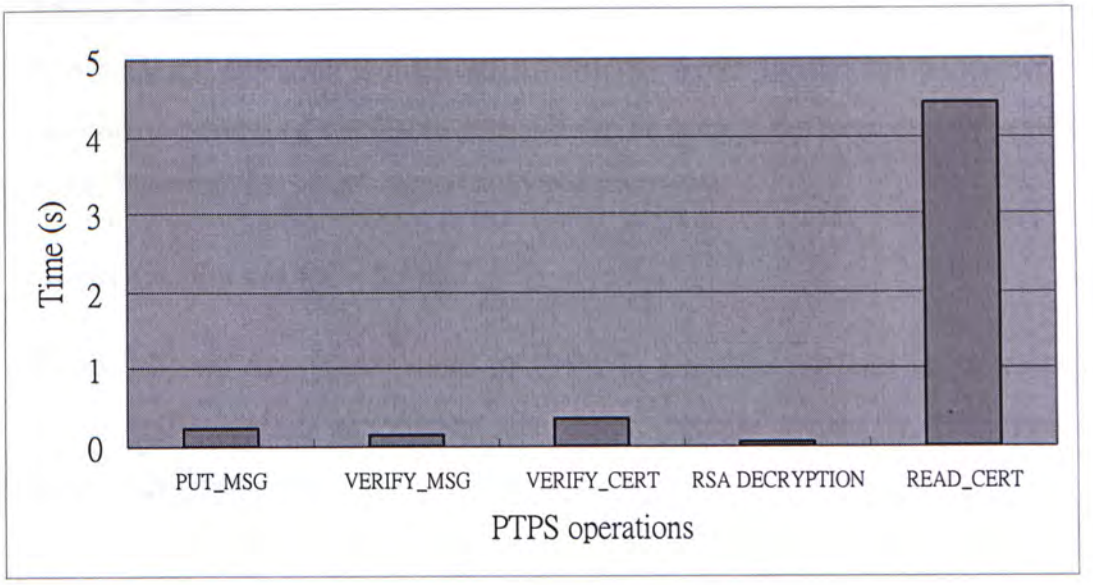



Figure 5.14: Overall performance of PTPS

Figure 5.14 shows the performance of PTPS in our secure mobile platform without file simulation. It includes the overhead of smart card I/O so that we can evaluate the overall system performance. PUT_MSG is the time for uploading the pre-defined message to the driver card. VERIFY_MSG is the time for comparing the pre-defined message inside the driver card. VERIFY_CERT is the time for verifying the certificate. RSA_DECRYPTION is the time for decrypting the encrypted pre-defined message and READ_CERT is the time for reading the certificate from the driver card.

The results show that all the operations except READ_CERT operation can execute in less than one second. The main overhead of READ_CERT is due to the reading of certificate from the smart card in the PDA. The certificate is about 1120 bytes and a few second is needed for I/O process. The time for verifying the driver and verifying the policeman can be calculated by equation V1 and V2 respectively.

V1: READ_CERT + VERIFY_CERT
= 4.45 + 0.35
= 4.80s

V2: READ_CERT + RSA_DECRYPTION+PUT_MSG+VERIFY_MSG
= 4.45 + 0.0558+0.23+0.1502
= 4.886

Although the time for both V1 and V2 needs about 5 seconds, we can observe that the READ_CERT operation is duplicated. After the driver verifies the policeman, we can pre-store the retrieved certificate so that it can be used in the procedure of verifying the driver. Therefore the actual mutual authentication time:

$$4.80s + 4.886s - 4.45s = 5.236s$$

In general, real time performance of PTPS in a mobile platform is shown to be achievable and therefore an efficient and secure mobile system is developed for handling traffic offence applications.

# 6. Conclusions and future work

In this thesis, we have investigated different infrastructures for constructing mobile payment systems. From the investigation, we observe that even if PKI is deployed to the payment systems, security still cannot be guaranteed if a system does not have a good authentication mechanism. Therefore we propose a fingerprint and smart card combined authentication technique in which fingerprint templates are matched inside smart card to protect user privacy. We studied the various methods for implementing the architecture using embedded systems. Through experiments, we have shown that SoC is a better approach for constructing our secure mobile device because SoC has the flexibility to support multiple and diversified applications as well as the ease to accommodate additional hardware devices. The whole authentication can be built using existing mobile platforms so that it can be integrated to the popular mobile payment systems very easily.

To evaluate the performance of our designed secure mobile device, we have developed PTPS for use in traffic offences. The system allows a policeman and the driver to settle the traffic offence payment efficiently. PTPS combines different secure software modules to support digital certificate, digital signature and mutual authentication on our proposed embedded system. Experiments were also conducted to evaluate the performance of PTPS. Results show that satisfactory performance can be achieved.

Engineering can always be conceived as a combination of art and science to solve problems. In our particular case, the science aspect comes from the computer hardware, the system software. We have contributed the art aspect by choosing a flexible platform (SoC + Embedded Linux) and the application software (Fingerprint Verification, Smart Card Programming, PKI Programming, System Integration) to solve a very difficult problem (Personal Authentication for Mobile Payment). Our contribution is obvious from our demonstrable results.

Because of advancement in technologies and the emergence of new artful concepts, engineering design never stops. Such advancements, in turn, impact our software design. In future work, we can foresee at least improvement in two areas. First, redesign the

secure mobile device so that its physical size can be further reduced to a portable version. Secondly, redesign of the complete system using the concept of hardware and software co-design for the embedded system. We will pause here.

## 7. References


1. A. Jain, R. Bolle, and S. Pankanti, Biometrics: Personal Identification in Networked Society, *Kluwer Academic Publishers*, 1999

2. Adams, C., Cain, P., Pinkas, D. and R. Zuccherato, "Internet X.509 Public Key Infrastructure Time-Stamp Protocol (TSP)", RFC 3161, August 2001

3. Alessandro Rubini, Jonathan Corbet, Linux Device Drivers, 2nd Edition, *O'Reilly & Associates, Inc* 2001

4. Andrew Nash, William Duance, Celia Joseph, Derek Brink, PKI implementation and Management E-Security, *McGraw-Hill*, 2001,pp299-375

5. Asker M. Bazen and Sabih H. Gerez, Extraction of Singular Points from Directional Fields of Fingerprints, *Mobile Communications in Perspective, Annual CTIT Workshop*, Enschede, The Netherlands, February 2001

6. Bassham, L., Polk, W. and R. Housley, "Algorithms and Identifiers for the Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation Lists (CRL) Profile", RFC 3279, April 2002

7. Bellare, M., et al., Design, Implementation and Deployment of a Secure Account-Based Electronic Payment System, *Research Report RZ 3137, IBM Research Division*, June 1999

8. Berkeley Design Technology, *http://www.bdti.com*

9. Bob Zeidman, President, The Chalkboard Network, Introduction to ASIC Design, Embedded System Conference, San Francisco 2002

10. Chi-Wei Yang, Paul C. H. Lee, Ruei-Chuan Chang, Reuse Linux Device Drivers in Embedded Systems, *In Proceedings of the 1998 International Computer Symposium (ICS'98)*, Tainang, Taiwan

11. Darrell Dunn, Intel scale up StrongARM with new XScale platform, Aug, 2000

12. Dave Jaggar, ARM Architecture and Systems, *IEEE Micro*, Vol.17, No. 4, Jug-Aug. 1997, pp9-11

13. David D. Zhang, Biometric Solutions For Authentication In An E-World, *Kluwer Academic Publications*, 2002

14. Digital Signal Processing Design Using the TMS320C5X, Instructor's Manual

15. Hash function, *http://www.rsasecurity.com/rsalabs/faq/2-1-6.html*

16. How to use Ramdisk for Linux, http://www.linuxfocus.org/English/November1999/article124.html

17. HyperText Markup Language (HTML) Home Page, *http://www.w3.org/MarkUp/*

18. Intel PXA250 and PXA210 Application Processors, *Developer's Manual*, February 2002

19. Intel StrongARM SA-1110 Microprocessor, *Developer's Manual*, June 2000

20. Introduction to VLIW Computer Architecture, *Philips Semiconductors*

21. Jain, A.; Lin Hong; Bolle, R., On-line fingerprint verification, *Pattern Analysis and Machine Intelligence*, IEEE Transactions on, Volume: 19 Issue: 4, April 1997 pp302 –314

22. Jain, A.; Lin Hong; Yifei Wan, Fingerprint image enhancement: algorithm and performance evaluation, *Pattern Analysis and Machine Intelligence*, IEEE Transactions on, Volume: 20 Issue: 8 , Aug 1998 pp777 –789

23. Jalal Al-Muhtdi, Anand Ranganathan, Roy Campbell, M. Dennis Mickunas, A Flexible, Privacy-Preserving Authentication Framework for Ubiquitous Computing Environments, *22nd International Conference on Distributed Computing Systems Workshops*, 2002

24. Jennifer Eyre, The Digital Signal Processor Derby, *Berkeley Design Technology Inc. IEEE Spectrum*, Jun. 2001, pp62-68

25. Jennifer Eyre, The Digital Signal Processor Derby, Berkeley Design Technology Inc. IEEE Spectrum, Jun. 2001, pp62-68

26. John P. Uyemura, Introduction to VLSI Circuits and Systems, Wiley, 2002

27. K. Lahiri, A. Raghunathan, and S. Dey, Battery-driven system design: A new frontier in low power design, in *Proceeding Joint Asia and South Pacific Design Automation Conference / International Conference VLSI Design*, pp261-267, Jan. 2002.

28. K.C. Chan, Y. S. Moon, P. S. Cheng, Fast fingerprint verification using sub-regions of fingerprint images, to be appeared in IEEE CSVT, 2003

29. Kamei, T.; Mizoguchi, M., Image filter design for fingerprint enhancement, *Computer Vision, 1995. Proceedings*, International Symposium on, 21-23 Nov 1995, pp109 –114

30. Kapil Raina, Anurag Harsh, mCommerce Security, A Beginner's Guide, *McGraw-Hill* 2002

31. Liang-Gee Chen, The Technology Trend of Embedded Processor, *Department of Electrical Enginnering*, National Taiwan University

32. Linux Device, the Embedded Linux Portal, *http://www.linuxdevices.com*

33. Liu, S.; Silverman, M.; A practical guide to biometric security technology *IT Professional*, Volume: 3 Issue: 1, Jan.-Feb. 2001 pp27 –32

34. Maio, D.; Maltoni, D., Direct gray-scale minutiae detection in fingerprints, *Pattern Analysis and Machine Intelligence*, IEEE Transactions on, Volume: 19 Issue: 1, Jan 1997 Page(s): 27 –40

35. Miller, B., Vital signs of identity, *Spectrum, IEEE,* Volume: 31 Issue: 2, Feb. 1994, pp22-30

36. MUSCLE, *http://www.linuxnet.com/info.html*

37. NetWinder Floating Point Notes, *http://www.netwinder.org/~scottb/notes/FP-Notes.html*

38. Neudoerffer, D., 5 Steps to Secure Mobile Data, *http://techupdate.zdnet.co.uk/story/0,,t481-s2125672,00.html*

39. Overview of the SET Protocol, *http://www.seas.upenn.edu/~tcom500/commerce/set.htm*

40. P.S. Cheng, Y.S. Moon, Z.G. Cao, K.C. Chan, T.Y. Tang, *Enabling Fingerprint Authentication in Embedded Systems*, 1st International Workshop on Signal Processing for Wireless Communication 2003, Vol. II, pp. 228-231

41. Pandey, R, Advances in DSP Development Environments, *ELECTRO '96. Professional Program. Proceedings*, 1996, pp299-301

42. Peter Thayer, Security Aspects of Wireless Devices, *San Francisco Embedded Systems Conferences*, 2001

43. Phillips, P.J.; Martin, A.; Wilson, C.L.; Przybocki, M.; An introduction evaluating biometric systems, *Computer*, Volume: 33 Issue: 2, Feb. 2000, pp56 – 63

44. Promoting Web-Enabling eBusiness, Authentication Architectures, Technologies, and Commercial Implementations, *Technical White Paper Series, eFORCE Inc.*

45. RISC Processor Architecture, http://www.ece.iit.edu/~jstine/ece529/handouts/skew.ppt

46. Rolf Ernst, Embedded System Architectures, NATO Science Series: Applied Sciences, volume 357 of *System-Level Synthesis*, pp1-43, Il Ciocco, August 1999

47. Ross Anderson, Markus Kuhn, Tamper Resistance – a Cautionary Note, 2nd USENIX Workshop on Electronic Commerce Proceedings, Oakland, California, 1996, pp 1-11

48. RSA key generation, *http://www.cs.usask.ca/grads/jil072/crypto/RSA-1/img0.htm*

49. S. Pankanti, S. Prabhakar, and A. K. Jain, On the Individuality of Fingerprints, IEEE Transactions on PAMI, Vol. 24, No. 8, pp1010-1025, 2002.

50. S. Ravi, A.Raghunathan and N. Potlapally, Securing Wireless Data: System Architecture Challenges, *International Symposium on System Synthesis (ISSS)* ,October 2002

51. S. Schwiderski-Grosche and H. Knospe, Secure mobile commerce, *Electronics and Communication Engineering Journal*, October 2002

52. Senn, J.A, The Emergence of M-Commerce, *Computer*, Volume: 33 Issue: 12, Dec. 2000, pp148-150

53. SMS (Short Message Service) – Technical Overview, *http://www.cswl.com/whiteppr/tech/sms.html*

54. SSL 3.0 Specification, http://wp.netscape.com/eng/ssl3/

55. Steve Furber, ARM System-on-Chip Architecture, second edition, *Addison-Wesley*, 2000

56. Steve Furber, ARM System-on-Chip Architecture, second edition, *Addison-Wesley*, 2000

57. The ARM architecture, http://www.arm.com/armtech/ARM_Arch?OpenDocument

58. The OpenSSL project, http://www.openssl.org

59. TI C5000 series DSP, http://www.go-dsp.com/xdspgt/swfs/swfs/start-c5000.html

60. TI C6000 series DSP, http://www.go-dsp.com/xdspgt/swfs/main-c6000.html

61. Tom Austin, PKI: A Wiley Tech Brief, *John Wiley & Sons, Inc.*, 2001,pp3-21

62. Transport Layer Security, http://www.ietf.org/html.charters/tls-charter.html

63. Unix man pages: initrd, http://www.rt.com/man/initrd.4.html

64. Vesna Hassler, Security Fundamentals for E-Commerce, *Artech House*, 2001, pp76-100, 369-383

65. Wayne Wolf, *Department of Electrical Engineering Princeton University*

66. Xiping Luo, Jie Tian, Yan Wu, "A minutiae matching algorithm in fingerprint verification", Proceedings. 15th International Conference on Pattern Recognition, 2000, Volume: 2, 2000, pp1038-1041

67. Y. S. Moon, F. T. Luk, T. Y. Tang, K.C. Chan, C. W. Leung," Fixed-Point Arithmetic for Mobile Devices-A Fingerprint Verification Case Study," Proceedings of the SPIE 2002 Seattle, Advanced Signal Processing Algorithms, Architectures, and Implementations XII, volume 4791, pp144 -149