國立交通大學

資訊科學系

碩 士 論 文

WAP 憑證轉換工具設計與實作

Design and Implementation of WAP Certificate Converter Toolkit

研 究 生：顏志明

指導教授：袁賢銘　教授

中華民國 九十四 年六月

WAP 憑證轉換工具設計與實作

研究生：顏志明　　　　　　　　　　　　　指導教授：袁賢銘 教授

國立交通大學資訊科學研究所

摘要

行動安全(mobile security)一直是在手持裝置上開發行動商務(m-commerce)程式的一項重要議題。對此，無線運用協定(Wireless Application Protocol, WAP)已訂製一系列規範標準，包括 WTLS (Wireless Transport Layer Security), WPKI (Wireless Public-Key Infrastructure), WIM (Wireless Identity Module), and WMLScript Crypto Library. 在這些標準中，一項重要的組件就是憑證(certificate)的使用。然而，為了符合無線環境的限制，WAP 重新定義了數種憑證格式，同時既有網際網路上通用的 X.509 憑證，也無法在未經修改情況下，直接下載到支援 WAP 的行動裝置上使用。

在本篇論文中，我們設計和實作出一個 WAP 憑證轉換工具，提供方便的程式及操作介面，可以將網際網路上通用的 X.509 憑證轉換成 WAP 相容憑證格式，並套用在支援 WAP 的行動裝置上。我們所開發的這個 WAP 憑證轉換工具，可以用來協助目前網際網路上發行 X.509 憑證的憑證中心(Certification Authority, CA)，將他們發出的憑證，根據使用者的需要，動態且容易地轉換成 WAP 憑證。就增進行動安全和考慮現行 X.509 憑證轉換成本而言，這套工具提供了相當程度的便利性與經濟性，可以用來幫助行動商務程式之開發。

Design and Implementation of WAP Certificate Converter Toolkit

Student: Chih-Ming Yan Advisor: Shyan-Ming Yuan

Department of Computer and Information Science

National Chiao Tung University

Abstract

Mobile security has been a key factor for development of m-commerce applications on hand-held devices. To enhance the mobile security, WAP (Wireless Application Protocol) has defined several standards such as WTLS (Wireless Transport Layer Security), WPKI (Wireless Public-Key Infrastructure), WIM (Wireless Identity Module), and WMLScript Crypto Library to regulate it. One important component of those standards is the use of certificate. However, for fitting the limitations in the wireless environment, WAP introduces several certificate formats in those specifications and the existing Internet X.509 certificates can not download directly to WAP-enabled handsets without any modification.

In this paper, we design and implement a WAP Certificate Converter Toolkit to provide the convenient interfaces for doing the conversion from the existing Internet X.509 certificates to WAP compatible certificates that can be deployed in WAP-enabled mobile devices. The toolkit we provide can be used by the Internet CA's (Certification Authority) to dynamically and easily transform their issued certificates into WAP certificates by the need of users. As far as enhancement of mobile security and conversion cost from Internet X.509 certificates are concerned, this toolkit is convenient and economic to a large extent and can be used to help the development of m-commerce applications.

Acknowledgement

回首二年研究所生涯，學習著實收穫良多。其中最要感謝的是我的指導教授袁賢銘老師，不論在課業或研究領域上，他總是給予學生最大的自由空間，去想像發揮其創意，這對於喜歡自我安排個性的我十分受用，也因此在這二年學習規劃裡，沒有絲毫時間被浪費掉，日子過得忙碌而紮實。除了袁老師之外，當然要感謝的人很多。曾經帶領過我作研究的蕭存喻、吳瑞祥和葉秉哲三位學長，經常跟我討論論文並提供意見的高子漢學長，謝謝你們的指導跟建議。還有感謝我的同學們：葉倫武，你總是幫我解決實驗室雜七雜八的問題；於之鈞，有你陪我談天論地，心情再煩也會變得開朗；柯憲昌，我們一起做了研究討論，也激盪出不少的想法；沈上謙，從你身上看到如何積極規劃安排自己；吳仁凱，經常和我談資訊人員的心酸，也許真的有一天可以考慮去賣香雞排；高啓涵，我們二人總是一起搭檔報告論文；還有另外二位女同學朱文如、林慧雯，妳們是實驗室的唯二朵花。認識你們，增添二年研究所生活不少色彩。最後當然要感謝我的父母，沒有你們無私對孩子的愛，就沒有今天的我。謹以這篇論文奉獻給我的雙親。

Table of Contents

List of Figures

List of Tables

Chapter 1 Introduction

1.1. Motivation

With rapid growth and development of mobile devices, such as mobile phones, Personal Digital Assistant (PDA) and Pocket PC, today most people possess at least one of those mobile devices. The pervasiveness and mobility of hand-held devices lead to more and more demand to do business on them. And one of the major concerns needed to be solved in mobile e-business is the security issue.

In WAP, it provides two kinds of security for this goal. One is communication security, defined in WTLS (Wireless Transport Layer Security) and TLS (Transport Layer Security). The other is end-to-end security, specified in WPKI (Wireless Public Key Infrastructure), WIM (Wireless Identity Module), and WMLScript Crypto Library. Both of them use certificates as a mean of proving identity and doing encryption operations. A certificate is a digital credential that binds the identity of a subject to a public key. Certificates are usually issued by a trusted third party named Certification Authority (CA) in a public key infrastructure. Nowadays, most CA's like VeriSign and Thawte issue certificates based on X.509 format.

However, the existing Internet X.509 certificate is not suitable to directly bring into the mobile devices without any modification because of the limitations of mobile devices such as lower computing power and less memory and storage. Although the WAP has introduced several certificate formats tailed for mobile devices in different situations, the overload and convenience to create WAP compatible certificates by the existing CA's is still a problem. So, we develop a WAP Certificate Converter Toolkit to help the CA's to easily transform the Internet X.509 certificates they issued into WAP compatible certificates for the mobile devices.

1.2. Objectives

The purpose of this thesis is to enhance the security for increasing the mobile e-business development. We will focus on providing a convenient toolkit for the existing Internet CA's to produce WAP compatible certificates for mobile applications. The objectives that we want to achieve are as follows:

- We want to be able to produce WAP compatible certificates across different WAP versions. WAP has defined a set of protocols as the standard for mobile devices manufacturers and developers to follow in the mobile world, and is still a continuous job since 1997. Until now, there are several major changes in the WAP history that result in different requirements over WAP versions. For example, before WAP 1.2, the security support in WAP is only through WTLS that just ensures the data transport between mobile devices and WAP gateways, and the CA certificates stored in the mobile devices for server authentication must be the WTLS format. Because not all of mobile devices keep up with the latest version of WAP, we need to consider the compatibility for different WAP versions. In practice, we can solve this problem by requesting a mobile device to provide its supporting WAP capability and accordingly transferring the certificate needed to it on the fly.

- We want to be able to produce WAP compatible certificates from the existing CA's that issue X.509 format certificates with little overhead. In the wired world, the Internet has developed a lot of complete and matured protocols to engage in e-business. And most Internet CA's have already built their public key infrastructures in accordance with those standards. Even for WAP, it models its protocols by modifying the existing Internet protocols to fit into mobile environment. So, to request the existing CA's to build another mechanism for issuing WAP compatible certificates will increase the cost of implementation and maintenance. A feasible strategy is to provide an adaptive layer to transform the existing X.509 certificates into WAP compatible ones. The major goal of our converter toolkit is to supply such an adaptive layer to assist CA's in doing the transformation easily for mobile applications.

- We want to be able to produce WAP compatible certificates by different people with a convenient way. We target two kinds of people to use our converter toolkit. One is the programmers and the other is the operators. For programmers, we provide a set of application programming interfaces (APIs) packed in the form of library to be used by them. For the operators, we supply a visual graphic user interface (GUI) that is friendly to do the certificates conversion and can see each field of the resulting certificate in a variety of views.

1.3. Thesis Organization

In Chapter 2, we discuss the background of WAP and its security mechanisms. In Chapter 3, we introduce the Internet X.509 certificate format, WAP certificate formats, and signature algorithms used to apply on WAP certificates. In Chapter 4, we discuss the design and implementation of our proposed converter toolkit. It will cover the conversion paths from the X.509 certificates to WAP compatible ones and the components of two interfaces – library and GUI – in detail. In Chapter 5, we conduct three experiments to verify what we claim in the Objectives section and evaluate the results. In Chapter 6, we give the conclusions and show some future works.

Chapter 2 Background

2.1. WAP Concept

The Wireless Application Protocol (WAP) is a set of protocols made by the WAP Forum which is founded by Ericsson, Motorola, Nokia and Phone.com (formerly Unwired Planet) in June 1997. It aims to be the mobile equivalent of traditional Internet by using the existing Internet standards, mainly TCP/IP and WWW, and extending them for the wireless networks and mobile devices. In contrast to wired networks, wireless networks have several fundamental limitations. They have less bandwidth that can cause poor performance, high latency and less connection stability. Similarly, compared to desktop computers, mobile devices also have limitations to be considered, including less memory and powerful CPU, smaller displays, limited input facilities, and restricted power consumption.

2.1.1. WAP 1.x

Programming Model

Based on the request-response pattern of WWW, the WAP programming model extends it to three-tier architecture: client—gateway—server (see Figure 2-1). The middle proxy performs a bridge between the wireless network and the wired Internet (see Figure 2-2). In the side between the client and the proxy, they communicate with each other via the wireless network like GSM (Global System for Mobile Communications). On the other side, the proxy talks to the server by using wired Internet. In addition to the role of networking protocol switch, the gateway also performs the content encoder and decoder in between to make the data transfer more compact in the wireless network.

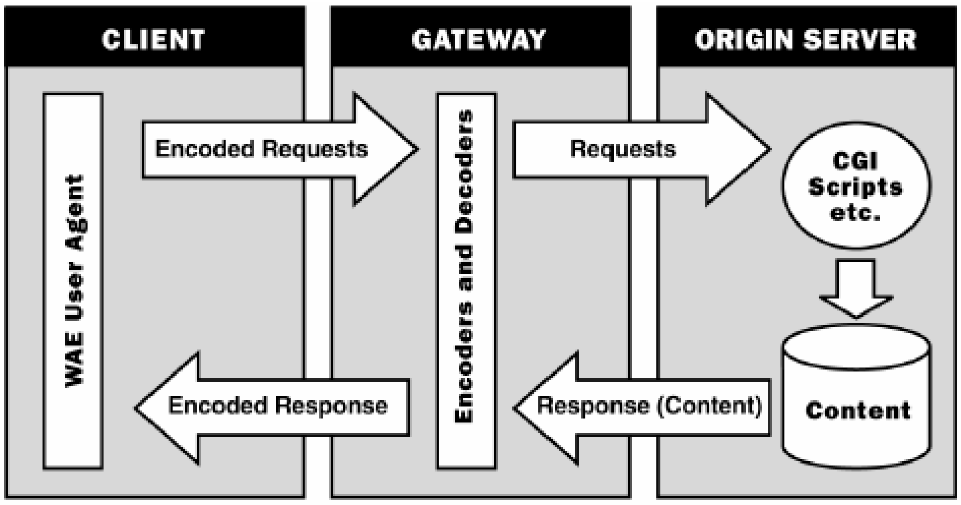


Figure 2-1: The WAP Programming Model1.x (Source: [1])

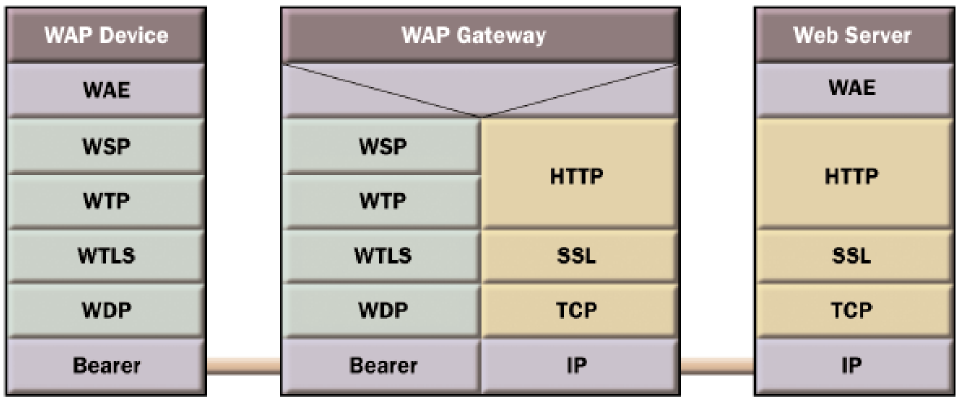


Figure 2-2: WAP Gateway as Protocol Switch (Source: [2])

Protocol Stack

WAP 1.x means the combination of WAP 1.1 and WAP 1.2, made in June 1999 and December 1999 respectively. Actually, the architectures of WAP 1.1 and WAP 1.2 are the same (see Figure 2-3).

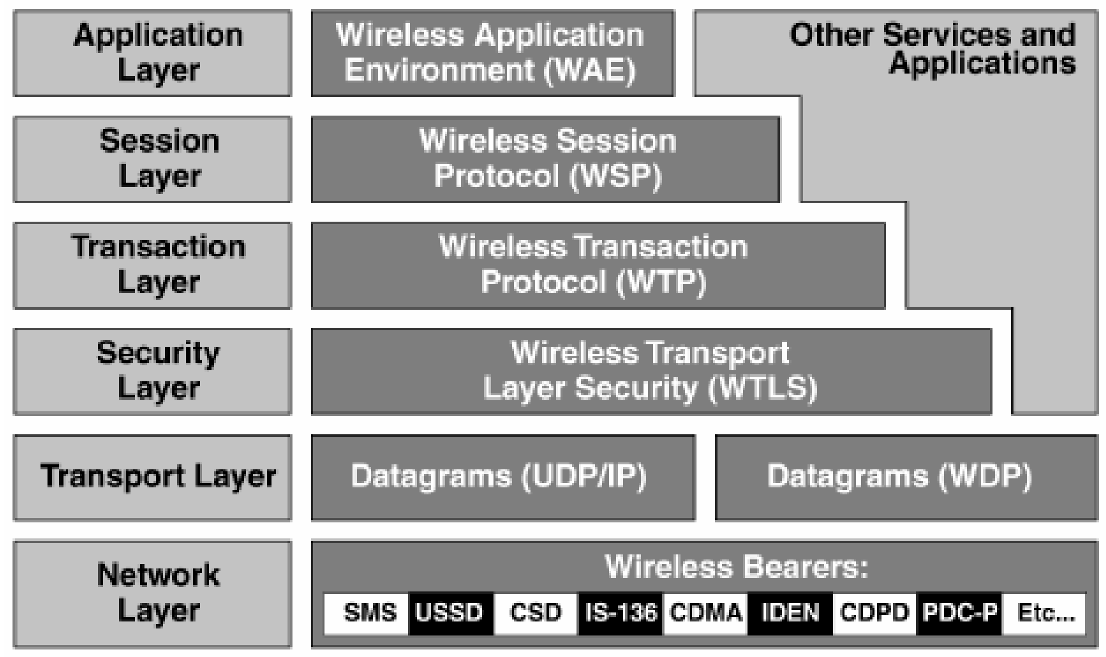


Figure 2-3: WAP 1.x Protocol Stack and TCP/IP equivalent (Source: [1])

In Figure 2-3, it shows the WAP 1.x protocol stack and the equivalent TCP/IP parts. It takes the layered-architecture and modifies from the Internet TCP/IP protocol. Each layer can only communicate with its upper layer and lower layer and not all the layers are required. For example, the WTLS provides the security layer for data transfer, but it is optional. The major difference between WAP 1.1 and WAP 1.2 is the enhancement of end-to-end security that was originally lacked in WAP 1.1. The addition of protocols in WAP 1.2 about security includes WPKI, WIM, and WMLScript Crypto Library.

2.1.2. WAP 2.0

Programming Model

In WAP 2.0, the WAP gateway is not a required component any more. The mobile client can make a HTTP request directly to the web server without through the WAP gateway (see Figure 2-4). However, if you want to offer other mobile services like location, privacy, and push based services, a WAP gateway is still suggested (see Figure 2-5)

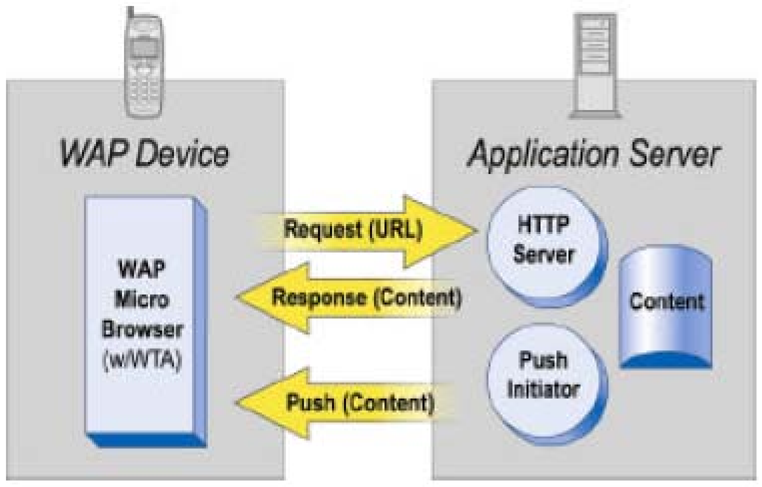


Figure 2-4: The WAP Programming Model 2.0 (1) (Source: [3])

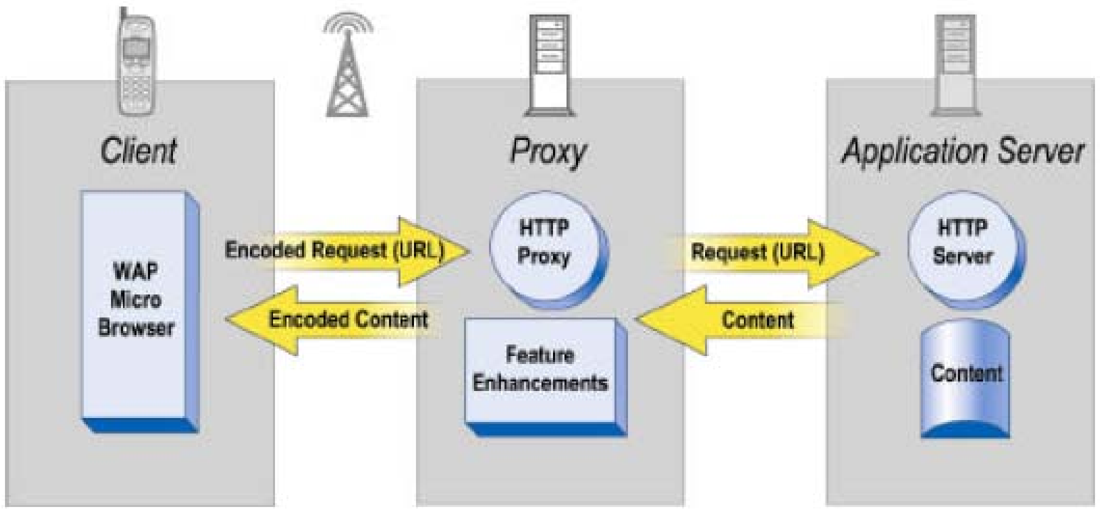


Figure 2-5: The WAP Programming Model 2.0 (2) (Source: [3])

Protocol Stack

WAP 2.0, published in January 2002, gives a big change to WAP and brings the wireless world closer to the Internet. It refines the WAP protocol stack by replacing the four protocols below the WAE layer in WAP 1.x. with the optimized Internet protocols, including HTTP, TLS, TCP, and IP (see Figure 2-6). Besides, it uses

XHTML/JavaScript in replace of original WML/WML as the new user interface language of mobile devices.

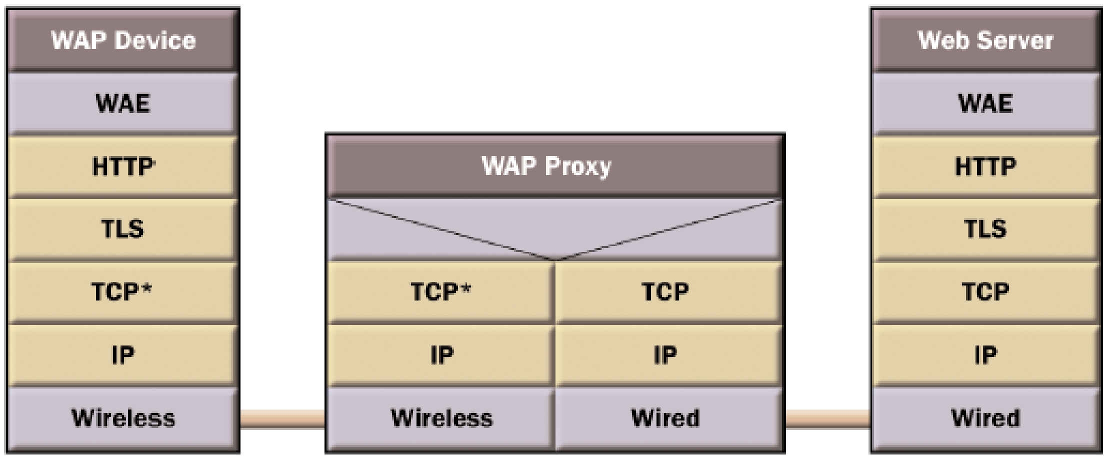


Figure 2-6: WAP 2.0 Protocol Stack and Architecture (Source: [2])

One of the problems WAP 2.0 wants to address is the security hole caused by WTLS and the gateway [9]. In WAP 1.x with WTLS/TLS, the gateway needs to decode the user's sensitive data in WTLS format and then encode it into TLS format for sending to the web server. In such way, it will give a chance for the gateway to peek user's sensitive data even though the tunnel between the client and gateway is safe. So, in WAP 2.0, the gateway is no longer performs the role of switch between the wireless and wired network. This improvement decreases the processing overhead in the gateway and effectively enhances the security in the tunnel.

2.2. WAP Security Mechanisms

The security services WAP provided so far can be classified into two categories: one is communication security, supplied by WTLS; the other is end-to-end security, provided by WPKI, WIM, and WMLScript Crypto Library. Communication security requires the client to authenticate the server and encrypt the transfer data between the client and server. The end-to-end security is achieved by requesting a user certificate to be stored in the mobile device, and using it as the proof of identity and for encryption operations.

2.2.1.WTLS

WTLS (Wireless Transport Layer Security) [4] is the security layer for data transfer in the WAP 1.x architecture. It is adapted from the TLS (Transport Layer Security) version 1.0 and specifically designed to provide privacy, data integrity, and authentication within the constraints of wireless networks. WTLS itself consists of five protocols (see Figure 2-7).

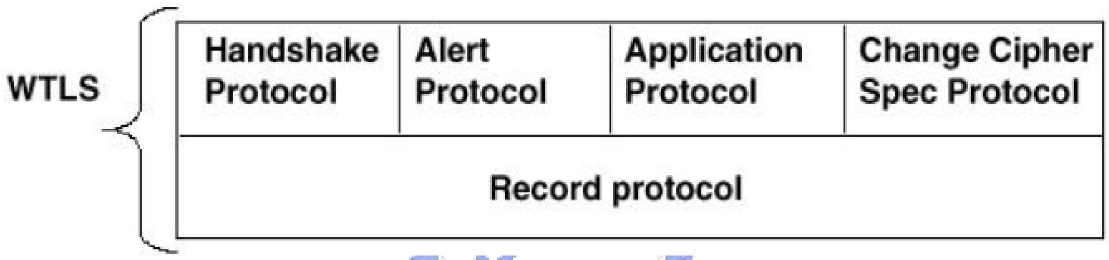


Figure 2-7: WTLS Internal Architecture (Source: [10])

The WTLS Record Protocol is a layered protocol, which gets messages to be transmitted from the upper layers, optionally compress the data, applies a MAC, encrypts, and transmits the result. On the contrary, the received data from the lower layer is decrypted, verified, decompressed, and then passed to the higher layers. The Record Protocol takes care of the data integrity and authentication. Above the Record Protocol, there are four protocols. The Handshake Protocol is responsible for negotiating the security parameters for a secure session. Security parameters include protocol version, compression method, and information on the use of authentication and public key techniques to generate a shared secret. The Alert Protocol is used to specify the alert messages that convey the security of messages and a description of this alert. The Application Protocol is the interface for the upper layers of WTLS to access the WTLS layer. The Change Cipher Spec Protocol defines the messages that are sent during the handshake phase after the security parameters have been agreed on [10].

In addition, there are three classes of security level between the client and server specified in the WTLS (see Table 2-1).

Security Level	Feature
Class 1	Anonymous key exchange is used for creation of an encrypted channel between the client and server. The client needs to take care if the public key sent by server is belonging to the server it really wants to communicate with.
Class 2	Provide server authentication with the use of server certificate. The server needs to send the server certificate to let the client to authenticate its identity.
Class 3	Provide both client authentication and server authentication. Before building a secure connection, both of the client and server need to make sure the identity with each other by means of exchanging their certificates.

Table 2-1: classes of security level in WTLS

2.2.2.WPKI

Just as WTLS is the optimized version of TLS with respect to mobile environments, WPKI (Wireless Public Key Infrastructure) [5] is an optimized extension of a traditional public key infrastructure for the wireless networks. WPKI requires the same components used in the traditional PKI (Public Key Infrastructure), including a Certification Authority (CA), a Registration Authority (RA), a repository to store certificates and Certification Revocation Lists (CRL), and an end-entity application (EE) [23]. However, the EE and RA are implemented differently and a new component called the PKI Portal is introduced. The EE is implemented in WPKI as an embedded module of the micro browser that runs in a WAP device. And the EE relies on the WMLScript Crypto Library to do the cryptographic operations. The PKI Portal typically acts as the role of RA and interoperates with the WAP devices on the wireless network and the CA on the wired network respectively (see Figure 2-8).

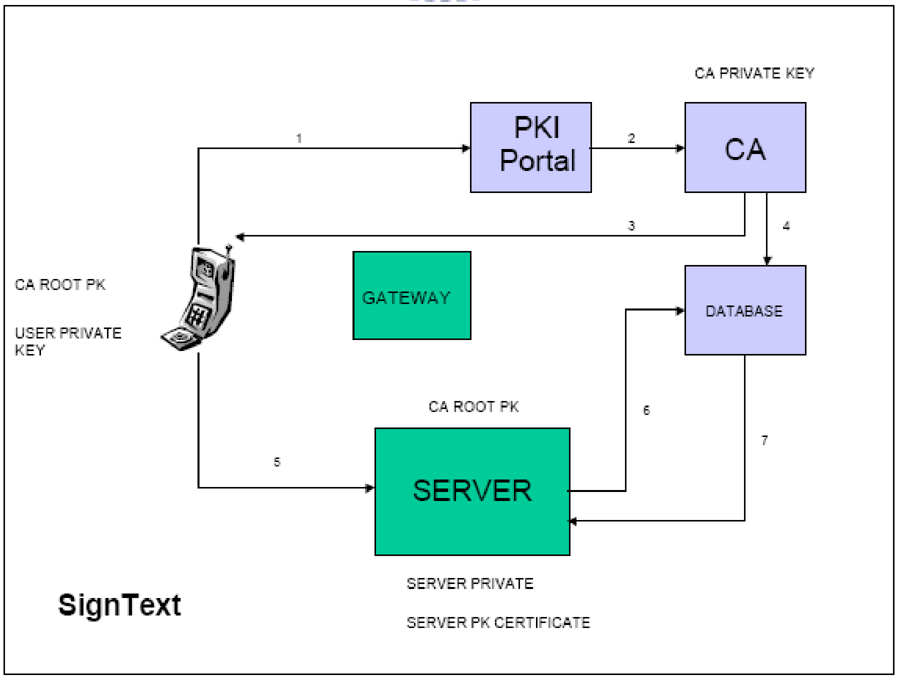


Figure 2-8: SignText and WPKI (Source: [5])

Other wireless-specific adaptation in WPKI, compared to PKI, is summarized in the following table.

	WPKI	PKI
Syntax notation and encoding rule	WTLS Presentation Language and binary encoding	ASN.1/DER/BASE64
Certificate Format	1.WTLS Server/Root CA certificates[in WAP client devices]: WTLS certificate 2.Client/Root CA certificates[in servers]: Internet X.509(RFC2459) 3.Client/Root CA certificates[in WAP client devices]: WAP Profiled X.509	Internet X.509 (RFC2459)

Cryptographic Algorithms and Key Length	1.RSA(1024 bits or more)
	2.ECC(160 bits or more)
		No limit

Table 2-2: Comparison of WPKI and PKI

2.3. ME Security Functionality

WAP puts two security facilities on the Mobile Equipment (ME), usually meaning the mobile devices, to activate the security functionality in the client end of WPKI. They are WIM and WMLScript Crypto Library.

2.3.1. WIM

WIM (Wireless Identity Module) [6] is a module that is tamper-resistant and can securely store sensitive data, such as certificates and private keys, and perform the computation of security operations, like generating random numbers and signatures. The objects and information data described in WIM follow the format of PKCS #15 (Public-Key Cryptography Standards #15). A WIM implementation can be an external smart card or a component embedded within the SIM (Subscriber Identity Module) card. The WIM is intended to perform both client and server authentications in WTLS and prove the user identity for end-to-end security.

Communication between WIM and the entities using it is via the command-response protocol in the form of service primitives. A service primitive describes the semantics of one service and its parameters. When WIM is implemented in a smart card or SIM card, the service primitives will be implemented as card commands. A card command is described by using APDU (Application Protocol Data Units) that is the standard form of smart card command defined in the ISO7816-4. There are two types of APDU. One is command APDU whose structure is listed in Table 2-3. The other is response APDU and its structure is shown in Table 2-4.

Part	**Field**	**Length**	**Description**
Header (mandatory)	CLA	1 byte	type of command
	INS	1 byte	command
	P1	1 byte	parameter 1
	P2	1 byte	parameter 2
Body (conditional)	Lc	1 byte	specify the length of data
	Data	variable	data
	Le	1 byte	indicate the maximum length of data expected in response

Table 2-3: Structure of command APDU

Part	**Filed**	**Length**	**Description**
Body (conditional)	Data	variable	response data for the command APDU
Trailer (mandatory)	SW1	1 byte	status1 after execution of the command APDU
	SW2	1 byte	status2 after execution of the command APDU

Table 2-4: Structure of response APDU

2.3.2.WMLScript Crypto Library

The WMLScript Crypto Library [7] is a part of WMLScript that is a script language, like the JavaScript for HTML, to manipulate the WML (Wireless Makeup Language) for better user interaction. This library is intended to use with WIM to provide cryptographic operations. Currently, the library only supports digital signature functionality by the WMLScript function, Crypto.signText, which asks the user to sign a string of text. The prototype of Crypto.signText is listed in the following table.

Function Name	**Parameter 1**	**Parameter 2**	**Parameter 3**	**Parameter 4**	**Return**
Crypto.signText	stringToSign	options	keyIdType	keyeId	signedString

Table 2-5: Prototype of Crypto.signText

- **stringToSign**: String

  This parameter is the string to be signed. Before being signed, the string should be converted to the same encoding if it contains different character sets. The recommended encoding is UTF-8.

- **options**: Integer

  Both of this and next parameters represent several option values that are ORed together into an integer. There are three options in this parameter: INCLUDE_CONTENT, INCLUDE_KEY_HASH, and INCLUDE_CERTIFICATE. The first option, if it is set, will include the string to be signed in the result. The second option, if it is set, will include the hash value of the public key corresponding to the signature key in the result. The third option, if it is set, will include the certificate or a URL of the certificate in the result.

- **keyIdType**: Integer

  This parameter indicates the type of a key identifier specified in next parameter. There are three options. NONE means the browser may use any key and corresponding certificate in WIM. USE_KEY_HASH means the next parameter will be the SHA-1 hash value of the user public key. TRUSTED_KEY_HASH means the next parameter is supplied with the SHA-1 hash value of a trusted CA public key and the browser must use a signature key issued by the trusted CA to sign the string.

- **keyId**: String

  The specified key based on the previous parameter. It can be an empty string or a string including the SHA-1 hash value of one public key or multiple, concatenated SHA-1 hash values of several public keys.

- **return**: String

  If the signText operation is successful, the resulting string will be the Base64 encoding of SignedContent structure defined in [7].

Chapter 3 Certificates Analysis

As we known, most existing Internet CA's issue certificates in the format of X.509. Among WAP versions, it introduces several certificates accustomed for the wireless environment as well. In this chapter, we will elaborate each of them in terms of content meaning, syntax description, and encoding rule. After that, we introduce the signature algorithms that are supported in the WAP specification to apply on WAP compatible certificates.

3.1. X.509 Certificate

3.1.1. Content

An X.509 certificate [11] binds a public key to a subject represented by a naming convention called Distinguished Name (DN). Such a certificate is also named an identity certificate in that it is used to authenticate the identity of the subject. A Distinguished Name is a global name composed of the combination of Common Name (CN), Organization Unit (OU), Organization Name (O), Locality Name (L), State Name (S), and Country (C). Currently, there are four versions of X.509 certificate. Version 1 specifies the basic fields; version 2 introduces the unique identifiers of subject and issuer, and CRL (Certificate Revocation List); version 3 adds the notation of extensions; version 4 supports for the attribute certificate that is not binding a public key to a DN but to one or more attributes. Table 3-1 shows the common fields of all versions of X.509 certificate [25].

Field	**Description**
version	the X.509 version, 1 to 4
serial number	a unique number, assigned by the CA to identify this certificate
signature algorithm identifier	the algorithm used by the CA to sign the certificate
issuer	the Distinguished Name of the entity that signs the certificate
period of validity	the begin and end times of the certificate in valid
subject	the Distinguished Name of the entity that owns the public key
subject's public key	the information about the public key of the subject
signature	the signature of the certificate

Table 3-1: Common fields of all versions of X.509 certificate

3.1.2. ASN.1

The X.509 standard [11] describes a certificate using the notation known as Abstract Syntax Notation One (ASN.1). As the name indicated, ASN.1 is not tied to any programming language and is designed to abstractly describe messages to be exchanged between communicating applications on different computer systems. ASN.1 consists of a set of well-defined primitive data types and methods to construct more complex data types from the primitive data types. Here, we only list some important data types that are used in the X.509 certificate in Table 3-2.

Data Type	**Description**
Boolean	two possible values are TRUE and FALSE
INTEGER	any positive or negative integer whatever its length
CHOICE	like the union type in C language, providing several alternatives to choose one
SEQUENCE	like the struct type in C language, containing data items in order
SEQUENCE OF	same to SEQUENCE except the type of each data item is the same
SET	similar to SEQUENCE except that the data items are no ordered
SET OF	same to SET except the type of each data item is the same
OBJECT Identifier	a global naming convention used to point to an object within a predefined hierarchy of naming space
GeneralizedTime	a way to model a date and time by means of a character string: four digits for the year, two digits for the month, two digits for the day, two digits for the hour, two digits for the minute, and optionally a dot or comma and two digits for the second. By default, this way describes the local date and time. You can also use a positive (+) or negative (-) delay with respect to the universal time coordinate (UTC) to express the

	local date and time. If this format wants to describe the UTC, just append the ‘Z’ to the end of the character string to indicate it.
UTCTime	same to GeneralizedTime except only two digits for the year
BIT STRING	a binary string of ‘0’ and ‘1’
Octet String	same to BIT STRING, emphasizing on an octet as a unit; an octet has 8 bits.
PrintableString	the alphabet that can be printable includes spaces, uppercase and lowercase letters, digits and the symbols “’”, “(“, “)”, “+”, “,”, “-“, “.”, “/”, “:”, “=”, and “?”.
UniversalString	the UCS (Universal multiple-octet coded Character Set) standard define 2^{31} cells (each cell contains a single character) to accommodate all the alphabets of all the languages in the world. It plans 128 groups of 256 planes of 256 rows of 256 cells. Currently, only the first plane (38,885 cells) called Basic Multilingual Plane (BMP) is allocated. A character will be encoded by four bytes (UCS-4) [29].
BMPString	represent the first plane of UCS. It uses two bytes to encode a character.
UTF8String	use “UCS Transformation Format, 8-bit form” (UTF-8) to efficiently encoding a character of BMPString into variable length.

Table 3-2: ASN.1 data types used in X.509 certificate

3.1.3.DER

Although ASN.1 provides a good way to describe messages in an abstract manner, the content of messages requires to be encoded by some rules for transfer between communicating applications. Several encoding rules for ASN.1 exist [29].

Basic Encoding Rules (BER) is the first encoding rules of ASN.1 and the basis for the other encoding rules. BER has the format of a 3-tuple <Type, Length, Value>, TLV for short. T indicates a data type in the ASN.1. For each type, there exists an adjacent encoding rule. L points out the number of bytes that V occupies.

Based on BER, there are another two encoding rules – Canonical Encoding Rules (CER) and Distinguished Encoding Rules (DER). The reason why CER and DER are created is BER allows too many encoding options for the same value such that the intermediate relaying parties may re-encode the transfer data with different options, which is not accepted for secured data like digital signature. So, CER and DER put constraints on BER to make the encoding of the same value with no degree of freedom. The major difference between CER and DER is CER allows the indefinite-length format while DER only uses a definite-length format. In practice, DER is the mostly used encoding rules for X.509 certificates.

3.1.4.Base64

The result of DER encoding of an X.509 certificate is binary content that is not human readable and not suitable to transmit over a text-only medium like e-mail. Base64 is a way to convert unreadable characters of 8-bit ASCII character set into readable characters. The principle of Base64 is as follows. For each three bytes (8*3=24 bits) of the binary content, cut them into 4 chunks of 6-bit and encode each chunk into a Base64 character ($2^6 = 64$ characters ,‘A’ to ‘Z’, ‘a’ to ‘z’, ‘0’ to ‘9’, ‘+’ ,

‘/’, plus ‘=’) [14].

3.2. WAP Certificates

There are several kinds of certificates specified in WAP for different purposes. The WTLS provides the communication security for data transfer in WAP 1.x and defines three classes of security level. WTLS Class 2 requires sever authentication and thus brings the format of WTLS certificate. Moreover, the WPKI supports the end-to-end security for the need of m-commerce to ensure the not-repudiation service. In WPKI, certificates are intended to be stored in the WIM which is usually embedded in the mobile device. How to download CA certificates into WIM correctly and make user certificates more efficiently stored in WIM become two important issues. For these issues, WAP creates a profile of the X.509 certificate accustomed to the wireless environments and defines several certificate formats for the downloading mechanisms.

3.2.1. Profiled X.509 Certificates

To make the X.509 certificates to be used in the wireless networks, WAP has set profiles for X.509 certificates of different roles [8]. For the CA, it defines the least required fields for certificates stored in WIM or sent over-the-air via WAP protocols. For the user, it defines two types of certificates stored in WIM for user authentication and digital signature, respectively. In general, the WAP Profiled X.509 certificates of CA and user contain the same common fields as existing Internet X.509 certificates. The fields specific to WAP are listed in the following table.

Field	**CA**	**User**
certificate serial number	less than 8 bytes	
signature algorithm	● sha1withRSAEncryption ● ecdsa-with-SHA1	
subject public key	RSA, ECC	
key length	● RSA key: at least 1024 bits ● ECC key: at least 160 bits	
keyUsage	N.A. (not available)	● RSA key: digitalSignature, keyEncipherment, nonRepudiation ● ECC key: keyAgreement nonRepudiation
subjectKeyIdentifier	● RSA: SHA-1 hash of the modulus of public key of the subject ● ECC: SHA-1 hash of the x-coordinate of the elliptic curve point	
authorityKeyIdentifier	same to subjectKeyIdentifier	the subjectKeyIdentifier of the issuer
basicConstraints	isCA	N.A.
extKeyUsage	id-kp-codeSigning	N.A.

Table 3-3: WAP Profiled X.509 certificates of CA and user

3.2.2.WTLS Certificate

Before WAP 2.0, WTLS is the only way to create secure connections for data transfer. In the process of building a secure connection between the client and server, the client will need to identify the server to ensure which server it is talking to. One of ways to prove identity is using a certificate. The WTLS introduces a compact, binary encoding certificate called WTLS certificate for server authentication.

Basically, the fields in the WTLS certificate are similar to the common fields of X.509 certificates. However, the WTLS certificate omits the extensions of the X.509 certificate and uses a more efficiently binary encoding rule than the DER to reduce the size in large. The presentation language of data structure in WTLS is explained as follows [4]:

- The basic block size is one byte (8 bits) and the byte ordering for a multi-byte value is big-endian that the leftmost bit is the most significant bit.
- A comment is enclosed by the "/*" and "*/".
- An optional component is denoted by enclosing it in "[[]]" double brackets.
- An opaque means one byte which the data is not interpreted.
- A vector (single dimensioned array) is a stream of homogeneous data elements, expressed in the form of "T T'[n];" . T means a data type. T' is a new type that is a fixed length vector of type T. The size of T' is n bytes, where n is a multiple of the size of T.
- A variable length vector is denoted by "T T'<floor..ceiling>", where flooring and ceiling represent the minimum and maximum length of bytes, inclusively.
- The basic numeric data type is unsigned by unit8. Extended numeric data types like unit16, unit24, and unit32 are the fixed length vectors of uint8.
- They type enum is represented by "enum {e1(v1), e2(v2),…, en(vn), [[n]]} Te;".

The e1(v1) means the first element and its value of enum type Te which may take n bytes as the maximum length. Without explicitly specifying n, the space of Te will be the number of bytes that can contain the largest value of its elements.

- A structure type that is much like the struct of C language is defined by:

```
struct{
    T1 f1;
    T2 f2;
    …
    Tn fn;
} [[T]];
```

A WTLS certificate is defined as a structure type [4]. The syntax and meaning are listed in the following:

```
struct {
    ToBeSignedCertificate to_be_signed_certificate;
    Signature signature;
} WTLSCertificate;
```

```
struct {
    uint8 certificate_version;
    SignatureAlgorithm signature_algorithm;
    Identifier issuer;
    uint32 valid_not_before;
    uint32 valid_not_after;
    Identifier subject;
    PublicKeyType public_key_type;
```

```
        ParameterSpecifier parameter_specifier;
        PublicKey public_key;
} ToBeSignedCertificate;

select( SignatureAlgorithm )
{
        case anonymous: { };
        case ecdsa_sha:
                digitally-signed struct {
                        opaque sha_hash[20];
                }
        case rsa_sha:
                digitally-signed struct {
                        opaque sha_hash[20];
                }
} Signature;
```

The certificate_version is always 1. The supporting signature algorithms are SHA1/RSA and SHA1/ECDSA. The public key types are RSA and EC. The content of the subject and issuer is organized as follows:

<servicename>; <organization>; <country>[; <commonname>[; <extension>
[; <extension>[…]]]]

where:

"; " is a 2-character field separator.

<servicename> is equivalent to Organization Unit (OU).

<organization> is equivalent to Organization (O)

<country> is equivalent to Country (C)

<commonName> is equivalent to Common Name (CN)

<extension> is an attribute in the form of <type>=<value>

3.2.3.Hashed Certificate

Either in WTLS or in TLS, to verify the server identity by means of server authentication needs the trusted CA certificate that issued the server certificate, directly or indirectly. In order to provide integrity, the trusted CA certificate is downloaded in self-signed format. However, this way does not provide authentication. To provide the authentication of the trusted CA certificate, the WPKI has specifies two downloading mechanisms and each mechanism brings a new certificate format [5].

The first mechanism is called out of band hash verification and the certificate syntax is as follows:

```
struct {
    uint8 version;
    CertDisplayName displayName;
    Certificate trustedCACert;
    opaque cainfo_url <0..2^8-1>;
    HashAlgorithm hash_alg;
} TBHTrustedCAInfo;
```

The presentation language of the hashed certificate is same to the WTLS certificate. The version must be 1 and the displayName is a name for display on the WAP client device. The trustedCACert is the original CA certificate that can be an X.509 certificate or WTLS certificate. The cainfo_url contains the CA's URL for the client to get further information about it. The hash algorithm is SHA-1.

The essential of this method is that firstly wrapping the CA certificate into a

TBHTrustedCAInfo structure and then hashing it with SHA-1 to calculate a hash value of 30 decimal digits. The steps to work out the hash value are as follows:

Step 1: Apply SHA-1 to TBHTrustedCAInfo to get the SHA-1 digest of 160 bits.

Step 2: Take the leftmost 80 bits of the SHA-1 digest and divide them into five groups, each group with 16 bits.

Step 3: Each of the five groups can be expressed as a 5 decimal digits (from '00000' to '65535'). Then, for each group, calculate the check digit by

(a) Double the values of digits located at the odd positions from the left, leaving the digits at even positions unchanged, and then combine all of the digits, newly double digits and unchanged digits, into a new group of decimal digits.

(b) Add the individual digits of the new group. The sum must be a number ending in zero (30, 40, 50, etc.). If the sum is not ending in zero, then the check digit (0~9) is added to make it end in zero. So, each original group of 5 decimal digits will get a check digit to append in the end.

Step 4: Combine the five groups of 6 decimal digits; we get the hash value of 30 decimal digits.

3.2.4.Signed Certificate

The second mechanism for authentication of the trusted CA certificate is called signature verification method. This method is simple. It also uses the same presentation language as WTLS. To make a signed certificate, wrapping the CA certificate into a TBSTrustedCAInfo structure, signing it to get the signature, and finally putting both the TBStrustedCAInfo and signature into the SignedTrustedCAInfo structure. The syntax is stated as follows [5].

```
struct {
    TBSTrustedCAInfo tc_info;
    Signature signature;
} SignedTrustedCAInfo;


struct {
    uint8 version;
    CertDisplayName displayName;
    Certificate trustedCACert;
    opaque cainfo_url <0.. 2^8-1>;
    Certificate signerCert;
    SignatureAlgorithm sig_alg;
} TBSTrustedCAInfo;
```

Several fields of the signed certificate are same to those of the hashed certificate. The only difference is that a signed certificate will take the signer's certificate with it for the client to verify the signature in the SignedTrustedCAInfo structure.

3.2.5.Resp Certificate

The user certificates defined in WAP are WAP Profiled X.509 user certificates [8]. They can be saved in a LDAP directory or a database, or delivered to the mobile device. If the user certificate has to be delivered, the WPKI specifies three possible downloading types for it. WPKI use the certificate format defined in the following to accommodate these downloading types [5].

```
struct {
    unit8 version;
    CertRespType type;
    select (type) {
        case cert_info:
                CertDisplayName display_name;
                Identifier ca_domain;
                Identifier subject;
                opaque url<0..255>;
            case cert:
                CertDisplayName display_name;
                Identifier ca_domain;
                Identifier subject;
                X509Certificate cert;
            case referral:
                opaque url<0..255>;
                uint32 seconds_to_wait;
        }
} CertResponse;
```

The first type just provides the URL about where the user certificate is located. This is the preferred downloading type of the user certificate due to the less storage in the client device. The second type will take the whole user certificate to be downloaded. The third type is only used for requesting a new user certificate from a CA and the CA responds it with the waiting time to retrieve the user certificate again.

3.3. Signature Algorithms

This section will introduce two signature algorithms – RSA and ECDSA-- supported in the WPKI. Both of these two algorithms use the SHA-1 as the hash algorithm to generate the hash value of the message before creating the signature.

3.3.1.SHA-1 Hash Algorithm

In cryptography, a hash algorithm is a function that takes a message of arbitrary length as input, and outputs a fixed-size unique value, namely hash value or digest. Such a function has several good properties. First, for any message of large size, the length of output is short and fixed. Second, for any two different messages, there will be two unique hash values. Third, it is very hard to derive the original message from a hash value. Therefore, a hash value can be used to serve as a representation of the input message.

Secure Hash Algorithm 1 (SHA-1) [12], based on MD4, is the hash algorithm proposed by the U.S. National Institute for Standards and Technology (NIST), and adopted by WAP to calculate the hash numbers for hashed certificates. The hash value SHA-1 will produce is 160 bits of length. In addition to provide the identity of a message, SHA-1 often coordinates with the signature algorithm to provide data integrity. The algorithm details of SHA-1 and the difference between SHA-1 and MD4 can be found in [26] [27].

3.3.2.RSA Signature Algorithm

RSA signature algorithm is the first and most popular signature algorithm invented by R. L. Rivest, A. Shamir, and L. Adleman in 1978 [15]. It is belonging to the public-key cryptosystem whose security is based on the hardness of solving the factoring problem for a very big prime number.

To create and verify a signature, you first need to have a key pair -- private key and public key. Below is the algorithm to create a RSA key pair [26].

Step 1: Generate two large distinct primes p and q with roughly same size

Step 2: Calculate $n = p*q$ and $\varphi(n) = (p-1)*(q-q)$, where $\varphi(n)$ means the number of primes less and equal to n.

Step 3: Choose a random positive integer e, less than φ(n) and relative prime to φ(n).

Step 4: Compute $d = e^{-1} \bmod \varphi(n)$, where $e * e^{-1} \equiv 1 \ (\bmod\ \varphi(n))$

Step 5: (n, e) is the public key; (n, d) is the private key

With the key pair, we can generate a signature by a private key and verify it with the corresponding public key. The algorithms are as follows. Suppose m represents the message to be signed by the private key and H means the hash algorithm.

1. Signature Production:

    (a) Calculate m' = H(m), where m' is the hash value of m

    (b) Compute $s = (m')^d \bmod n$

    (c) s is the signature for m

2. Verification:

    (a) Calculate $m' = s^e \bmod n$

    (b) Compute m'' = H(m), where m is the message attached with the signature s

    (c) compare m' to m''; if not equal, verify failed.

3.3.3.ECDSA Signature Algorithm

Digital Signature Algorithm (DSA) is another signature algorithm that is adopted as the Digital Signature Standard by NIST. Like RSA signature algorithm, it also needs a key pair before doing signature related operations. The details about DSA key pair generation, and signature creation and verification can be found in [16].

Elliptic Curve DSA (ECDSA) is the elliptic curve analogue of the DSA by using the elliptic curve cryptosystems (ECC) as the underlying basis. The advantages of ECC include smaller key sizes, less bandwidth requirements, lower power consumption. So, ECC shows its attractiveness to limited computing environments such as mobile devices. The background about ECC and the algorithms of ECDSA are also clearly explained in [16].

Chapter 4 Toolkit Framework and Implementation

In last chapter, we have explored the Internet X.509 certificate and WAP compatible certificates in detail and discussed the signature algorithms supported by WAP. In this chapter, we will introduce the design and implementation of our proposed converter toolkit for WAP. The ultimate goal of our toolkit is to enhance the mobile security for m-commerce.

To provide the WAP compatible certificates for different purposes across several WAP versions and to assist the existing Internet CA's easily porting their issued X.509 certificates to wireless networks with little afford, we design a converter toolkit that provides two interfaces – a library and a GUI – to decode the content of Internet X.509 certificates and then encode into WAP compatible certificates. Our target certificates include both CA certificates and user certificates. And the supporting signature algorithm we choose to implement is RSA. This is because most Internet CA's issue X.509 certificates by using RSA signature algorithm.

4.1. Certificates Conversion Paths

Before we do the conversion, we need to have the Internet X.509 certificates first. In addition to using the existing Internet X.509 certificates as the source of conversion, out converter toolkit can also generate the WAP Profiled X.509 certificates for CA and user, respectively.

With the X.509 certificates, we can make them as input to the converter toolkit to do the conversion for outputting the desired WAP certificates. In principle, the certificates conversion paths can be divided into two categories (See Fig. 4-1).

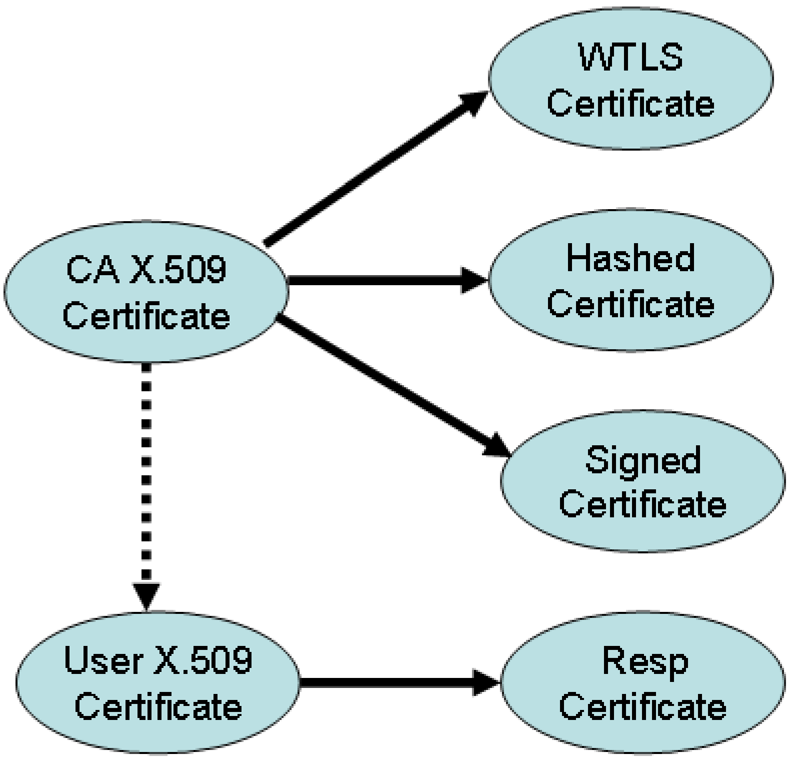


Figure 4-1: WAP certificates conversion paths

The first kind of conversion paths is for the CA X.509 certificates that are self-signed by their own private keys. For a CA X.509 certificate, it can be converted into three different WAP compatible certificates. One is the WTLS certificate, which is specified in the WTLS specification for providing server authentication to build a secure connection between the client and server. The other two are essentially the wrapping of CA X.509 certificates for the downloading mechanisms defined in the WPKI specification.

The second kind of conversion paths is for the user X.509 certificates that are issued by the CAs. For a user X.509 certificate to be delivered to the mobile device, it needs to be converted to the downloading format specified in the WPKI specification. As described in section 3.2.5, the user X.509 certificate can be wrapped into three kinds of forms. Out converter toolkit implements the URL form which is suggested in the WPKI.

4.2. Framework Design

We implement the WAP certificate converter toolkit in Java (JDK 1.4) and build it upon the Java Cryptography Architecture. The whole framework is shown in Figure 4-2. In this section, we first describe the components of our toolkit and the utility libraries underlying it. And then, in next two sections, we discuss the implementation details of our converter toolkit.

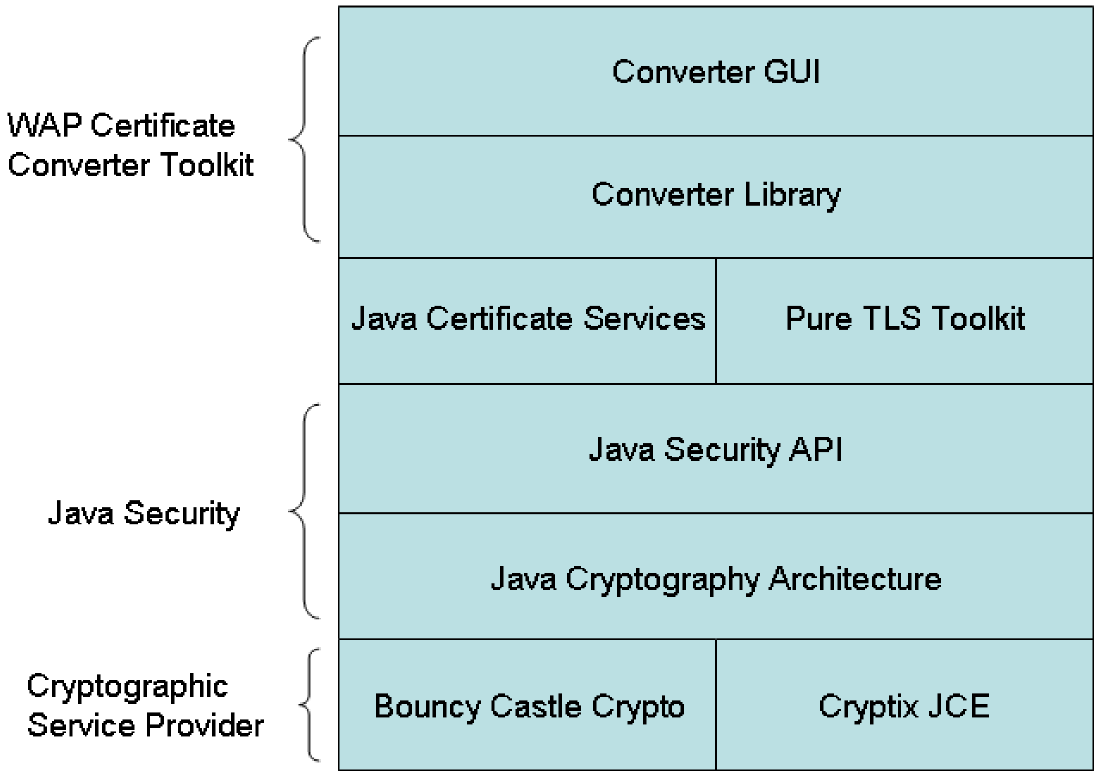
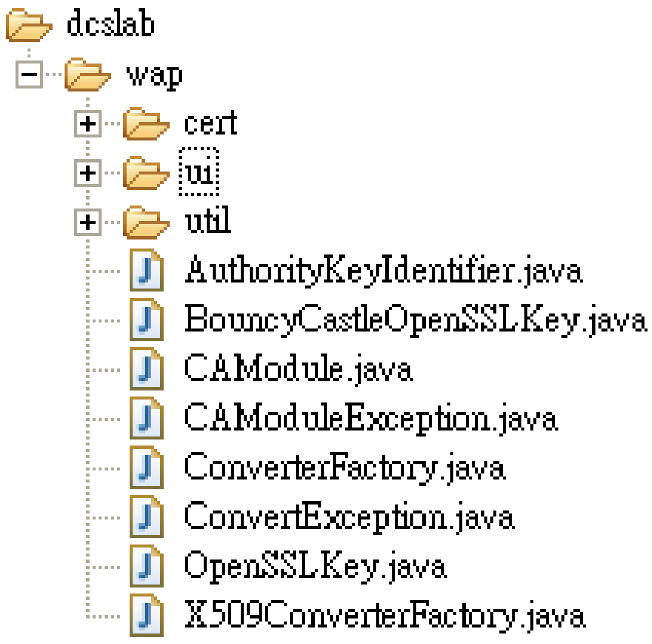

WAP Certificate Converter Toolkit
Java Security
Cryptographic Service Provider
Converter GUI
Converter Library
Java Certificate Services
Pure TLS Toolkit
Java Security API
Java Cryptography Architecture
Bouncy Castle Crypto
Cryptix JCE

Figure 4-2: Architecture of WAP Certificate Converter Toolkit

4.2.1. WAP Certificate Converter Toolkit

Our proposed WAP Certificate Converter Toolkit is made up of a library and a friendly GUI application based on the library to provide a variety of views of all certificates talked in the Chapter 3. It contains four Java packages, shown in Figure 4-3. The library consists of dcslab.wap and dcslab.wap.cert two packages. The GUI application is implemented in the dcslab.wap.ui package. The dcslab.wap.util package provides convenient facilities supporting for both the library and the GUI application.



```
dcslab
 wap
   cert
   ui
   util
   AuthorityKeyIdentifier.java
   BouncyCastleOpenSSLKey.java
   CAModule.java
   CAModuleException.java
   ConverterFactory.java
   ConvertException.java
   OpenSSLKey.java
   X509ConverterFactory.java
```

Figure 4-3: packages of WAP Certificate Converter Toolkit

4.2.2.Java Cryptography Architecture

One of the famous features in Java language is the tightly binding with security. Two architectures about security defined in Java are the Java Security Architecture and Java Cryptography Architecture. The former regulates the security mechanism of Java platform while the latter emphasizes on the cryptographic functionality for Java language [24].

Java Cryptography Architecture and its extension, Java Cryptography Extension, provide a service provider framework that they define the interfaces of cryptographic API, the Java Security API, and leave the implementation of those API to the cryptographic service providers. Currently, in Java platform, Sun has supplied a few cryptographic service providers such as Sun, SunJCE, Sun JSSE, and SunJGSS to provide default implementation of Java Security API.

4.2.3.Bouncy Castle Crypto

Even though Sun has already supplied several cryptographic service providers, it still lacks many demanding security functionalities. For example, we can not create an X.509 certificate from the standard Java Security API, not to mention other low-level cryptographic manipulations. Moreover, the source code of Sun's cryptographic implementation is not completed, even it can be downloaded freely, and can not be modified with arbitrary. Therefore, we need other cryptographic service providers to overcome above deficiencies.

The Bouncy Castle Crypto package is an open source project that provides a Java implementation of cryptographic algorithms [17]. It makes up for the insufficiency of Sun's service providers. In addition to easily modifying it to fit our need, it also provides quite a few convenient low-level API to manipulate cryptographic algorithms and X.509 certificates. Our converter toolkit makes use of these API to deal with ASN.1 syntax, DER encoding, and X.509 certificates.

4.2.4.Cryptix JCE

Cryptix JCE is another Java cryptographic service provider [18]. It is the chosen service provider by the PureTLS toolkit (see next section). The Java Certificate Services (see below section) also uses it to transform the binary format of X.509 certificates to the BASE64 format.

4.2.5. PureTLS Toolkit

PureTLS Toolkit is the Java version of SSL implementation developed by Eric Rescorla [28]. It provides the API for Java Certificate Services to create the RSA key pair and the X.509 certificate request in PKCS #10 format. Besides, it also provides the utility for our converter toolkit to wrap a string of Distinguished Name into an X509Name object.

4.2.6. Java Certificate Services

Java Certificate Services (JCS), developed by Vladimir Silva, provide the management of X.509 certificates, including certificate request creation (CSR), CSR signature, user certificate creation, and self-signed (CA) certificate creation [13]. JCS has implemented a simple Internet CA and can create Internet X.509 certificates for CA and user. Our converter toolkit modifies it for generating the WAP Profiled X.509 certificates of CA and user.

4.3. Converter Library

4.3.1. Building Blocks

From the previous introduction, we take advantage of the underlying libraries, especially the Bouncy Castle Crypto package, to process the stuff about X.509 certificates. However, to convert X.509 certificates to WAP compatible certificates, we need to be able to deal with the syntax and binary encoding of WAP certificates. So, we implement the presentation language specified in WTLS specification as Java classes in the dcslab.wap.cert package to provide the high-level value access and low-level data encoding.

A data type defined in the presentation language will extend the abstract class AbstractEncoder that implements the Encodable interface and supply the implementation of method getData() with binary encoding of its value (see Figure 4-4). The binary encoding of a data type is a 2-tuple <length, value>. The length is either explicitly defined with the data type or the least space to contain the largest value of the data type.

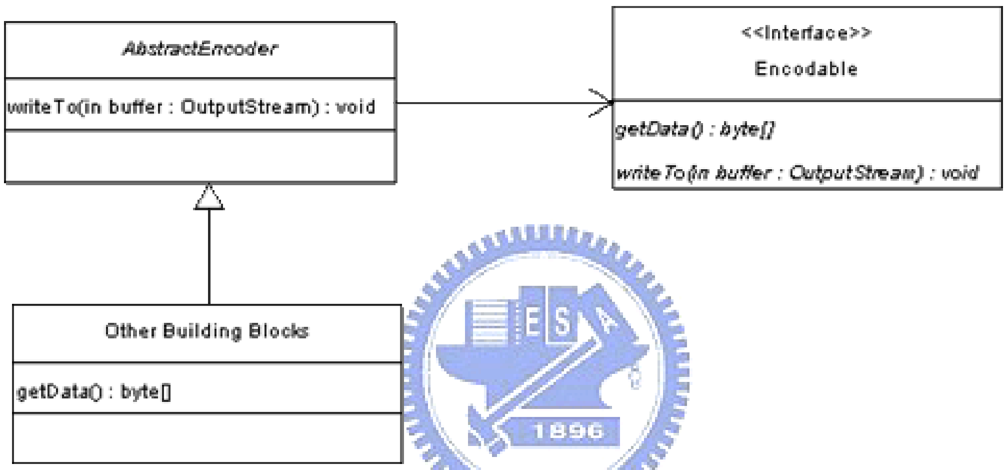


Figure 4-4: Class Diagram of Building Blocks

In addition, we provide four wrapper classes for WAP certificates as the unified interface for outside access. They are WTLSCert, HashedCert, SignedCert, and RespCert. The essential information encapsulated in these wrapper classes is listed in the following table.

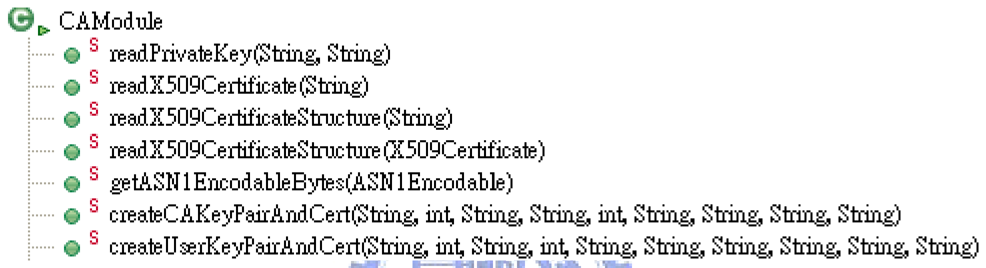
Wrapper Class	**WAP certificate**	**MIME Type**
WTLSCert	WTLSCertificate	application/vnd.wap.wtls-ca-certificate
HashedCert	TBHTrustedCAInfo	application/vnd.wap.hashed-certificate
SignedCert	SignedTrustedCAInfo	application/vnd.wap.signed-certificate

RespCert	CertResponse	application/vnd.wap.cert-response

Table 4-1: Wrapper classes for WAP certificates

4.3.2.CAModule

This class and its supporting classes provide the functionality of creation of key pairs and X.509 certificates, parsing of X.509 certificates, and DER encoding of ASN.1 syntax. CAModule acts as a CA and the Figure 4-5 shows its public methods.



```
CAModule
  S readPrivateKey(String, String)
  S readX509Certificate(String)
  S readX509CertificateStructure(String)
  S readX509CertificateStructure(X509Certificate)
  S getASN1EncodableBytes(ASN1Encodable)
  S createCAKeyPairAndCert(String, int, String, String, int, String, String, String, String)
  S createUserKeyPairAndCert(String, int, String, int, String, String, String, String, String, String)
```

Figure 4-5: public methods of CAModule

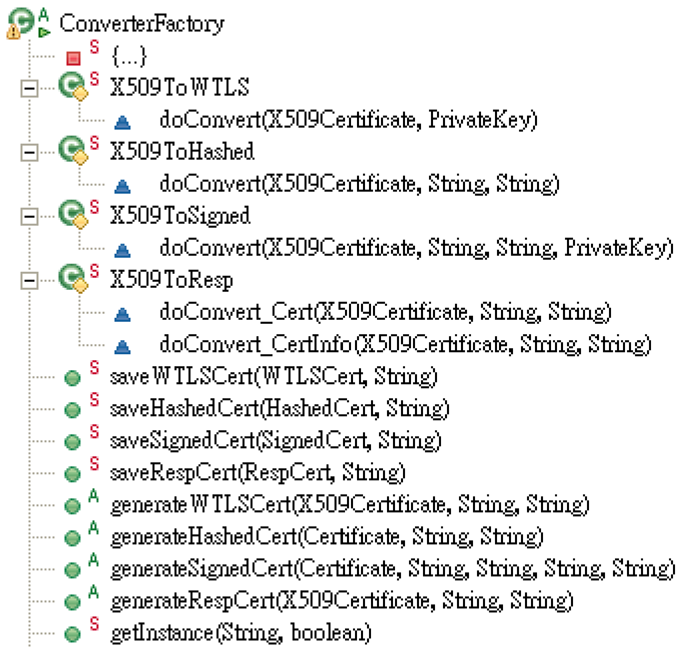
We use the RSA algorithm to generate a key pair and the key length is either 1024 bits or 2048 bits that corresponds to the requirement of WAP. The signature algorithm of the X.509 certificate is SHA-1/RSA. The process of creating a new CA X.509 certificate is as follows. Step 1, we create a public/private key pair with a random number as the seed. Step 2, we wrap the string of CA name into a X509Name object and use it, the generated public key, and other information to fill out the TBSCertificate structure. Step 3, we apply SHA-1 hash algorithm on the TBSCertificate structure to get a digest of 160-bit length and then sign the digest with RSA signature algorithm to get the signature. Step 4, the combination of TBSCertificate structure and the signature is what so called an X.509 certificate. Step 5, the content of an X.509 in Step 4 is DER encoding; that is, certain binary format. To make it human readable and can be transmitted through textual medium, we transform it into the text form by Base64 encoding. Then, we insert the header "-----BEGIN CERTIFICATE-----"before it and append the footer "-----END CERTIFICATE-----"after it. The resulting text is usually what we apply for an X.509 certificate from the Internet CA.

The creation of a user X.509 certificate is similar to that of a CA X.509 certificate. However, there are two differences. First, the CA certificate is self-signed; that is, the fields of subject and issuer are with the same value and the whole certificate is signed by the subject's private key. Second, the extension fields such as keyUsage, extKeyUsage, basicConstraints, subjectKeyIdentifer, and authorityKeyIdentifier have different meanings between a CA certificate and a user certificate.

4.3.3.ConverterFactory

The task of CAModule is to create X.509 certificates of CA and user. Once we have the X.509 certificates, we can convert them into WAP compatible certificates by the need. The conversion is done via the ConverterFactory class (see Figure 4-6). ConverterFactory is designed as an abstract class that its subclasses have to implement four abstract methods. They are generateWTLSCert(), genereateHashedCert(), generateSignedCert(), and generateRespCert(). The first three methods are to convert a CA X.509 certificate into WTLSCert, HashedCert, SignedCert, respectively. The last method is used to convert a user X.509 certificate into a RespCert.

A conversion path means the transforming process from one certificate to another. For each conversion path, we design a dedicated class for the conversion job and put it as an inner static class in the ConverterFactory. The reason why we do such design is we hope to concentrate the conversion capabilities on the ConverterFactory abstract class so that the concrete classes inheriting from it can easily and flexibly choose the conversion classes to fulfill the abstract methods. For example, the X509ConverterFactory class is a subclass of ConverterFactory and implements the four abstract methods by using the four conversion classes embedding in the ConverterFactory.



```
ConverterFactory
  S {...}
  S X509ToWTLS
      doConvert(X509Certificate, PrivateKey)
  S X509ToHashed
      doConvert(X509Certificate, String, String)
  S X509ToSigned
      doConvert(X509Certificate, String, String, PrivateKey)
  S X509ToResp
      doConvert_Cert(X509Certificate, String, String)
      doConvert_CertInfo(X509Certificate, String, String)
  S saveWTLSCert(WTLSCert, String)
  S saveHashedCert(HashedCert, String)
  S saveSignedCert(SignedCert, String)
  S saveRespCert(RespCert, String)
  A generateWTLSCert(X509Certificate, String, String)
  A generateHashedCert(Certificate, String, String)
  A generateSignedCert(Certificate, String, String, String, String)
  A generateRespCert(X509Certificate, String, String)
  S getInstance(String, boolean)
```

Figure 4-6: ConverterFactory class

4.4. Converter GUI

The Converter Library provides the API for programmers to do the entire conversion job. To use it needs writing programs. For convenience of use and for operators who are not good at programming, we design a GUI application to make the conversion more easily. The GUI is not only based on the Converter Library, but also further provides a variety of views for a certificate. There are four views of a certificate that our GUI can support. The field view can see the high-level value of each field of the certificate. The syntax view can show the data structure of a field. The binary view can watch the low-level binary encoding of the field. The Base64 view can look the text form of the certificate.

The major components of the GUI application are depicted in Figure 4-7.

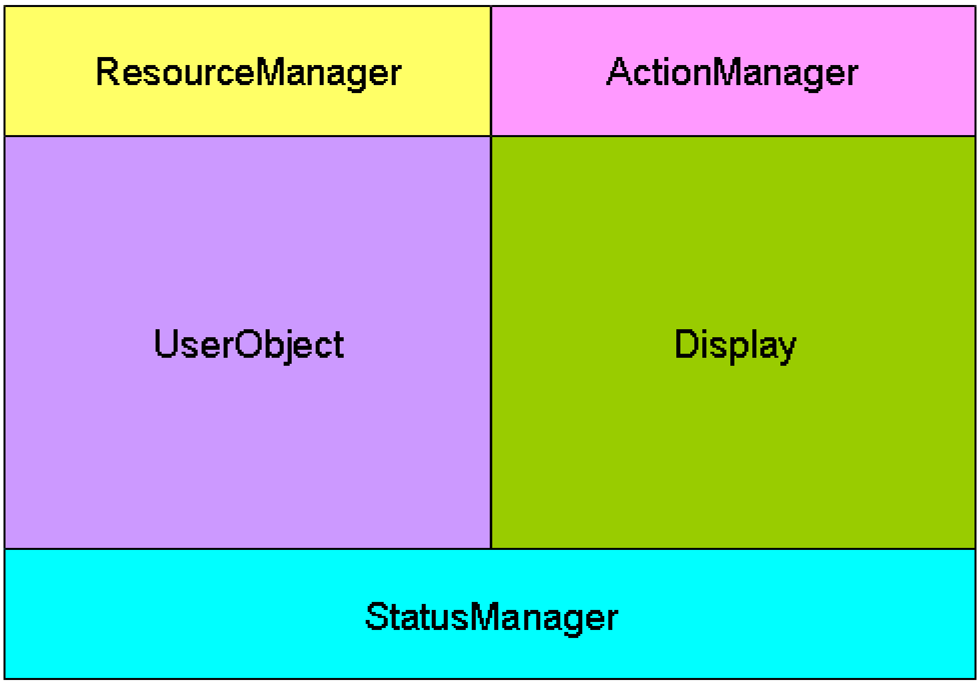


Figure 4-7: major components of Converter GUI

4.4.1.ResourceManager

ResourceManager is responsible for the creation of menus, popup menus, toolbar, menu items, and buttons. It reads a configuration file that records the information of all resources and generates above described UI stuff to hook in the main frame.

4.4.2.ActionManager

ActionManager is the manager of the states of all action objects. Basically, for a GUI application, several UI items may trigger the same action object when they are activated. To reflect the states of those UI items linking with the same action object in time, ActionManager will maintain a state database to record the recent status of all action objects.

4.4.3.StatusManager

StatusManager takes responsibility for managing the status bar. The job it performs is to show the helpful information in the status bar when the mouse pointer moves on the UI items.

4.4.4.UserObject

The UserObject is a data object set in a tree node. Actually, it is a data container that encapsulates the certificate object. The inheritance hierarchy of UserObject and the containing certificate objects (in the parentheses) are shown in Figure 4-8. When a tree node is selected, the UserObject it links with will be passed to the Display (see next section) to show the detailed information of the certificate.

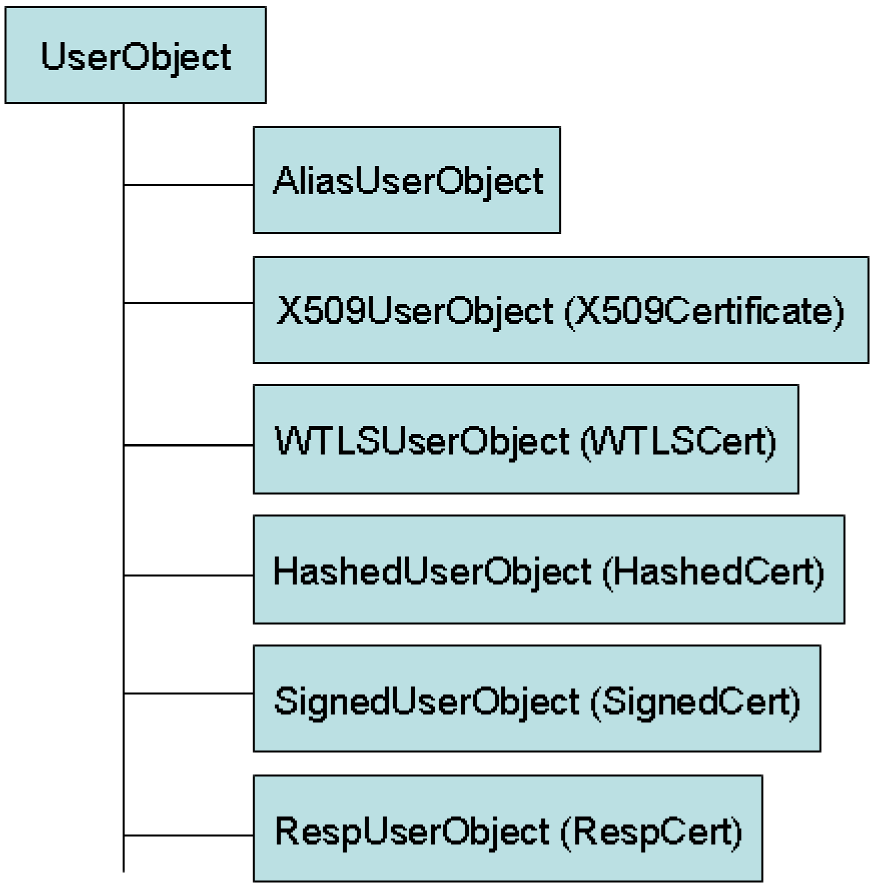

UserObject
AliasUserObject
X509UserObject (X509Certificate)
WTLSUserObject (WTLSCert)
HashedUserObject (HashedCert)
SignedUserObject (SignedCert)
RespUserObject (RespCert)

Figure 4-8: inheritance hierarchy of UserObject

4.4.5. Display

The Display is just a collection concept. In fact, it is the displaying panel of field view, syntax view, binary view, and Base64 view. For each certificate object, there will have a corresponding Display to it. Each time the selected tree node passed the UserObject to its corresponding Display, the information of the certificate object within the UserObject will be retrieved and shows in the right regions of the displaying panel.

Chapter 5 Experiment and Evaluation

To evaluate our proposed converter toolkit, we conduct three experiments. The first experiment is to create WAP compatible certificates from X.509 certificates. We use the GUI version of our toolkit to do this experiment. The second experiment is to test the correctness of those certificates from the first experiment. We will import the WAP certificates created by the first experiment into a simulated WAP cell phone to see whether those WAP certificates can be accepted. In the final experiment, we demonstrate a small m-commerce application that will make a digital signature via the Crypto.signText() method with a user private key.

5.1. Experiment 1: Certificates Conversion Paths

This experiment wants to verify the conversion paths described in section 4.1. Those conversion paths will create WAP compatible certificates.

5.1.1. Environment Configuration

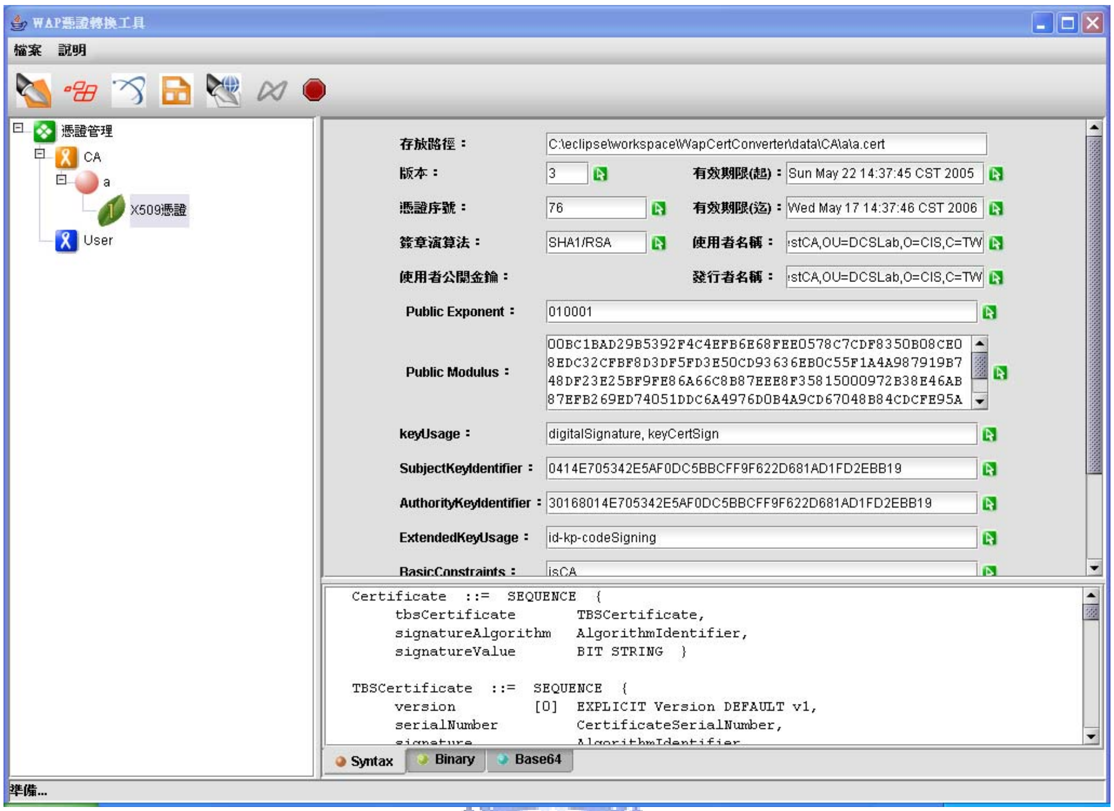
To do the conversion, we need to have the Internet X.509 certificates of CA and user. For simplicity, we create them directly from the GUI application. In fact, the X.509 certificates produced by our toolkit have already been the WAP Profiled ones. Even so, they still can be used as the sources of the conversion.

5.1.2.Experiment Result

The test items and results are shown in the following table.

Test Item	**Description**	**Result**
create a CA X.509 certificate	the CA X.509 certificate is used as the source of conversion	see Figure 5-1
create a user X.509 certificate	the user X.509 certificate is used as the source of conversion	see Figure 5-2
convert to WTLS certificate	convert the CA X.509 certificate into a WTLS certificate	see Figure 5-3
convert to Hashed certificate	convert the CA X.509 certificate into a Hashed certificate	see Figure 5-4
convert to Signed certificate	convert the CA X.509 certificate into a Signed certificate	see Figure 5-5
convert to Resp certificate	convert the user X.509 certificate into a Resp certificate	see Figure 5-6

Table 5-1: Experiment 1 Result

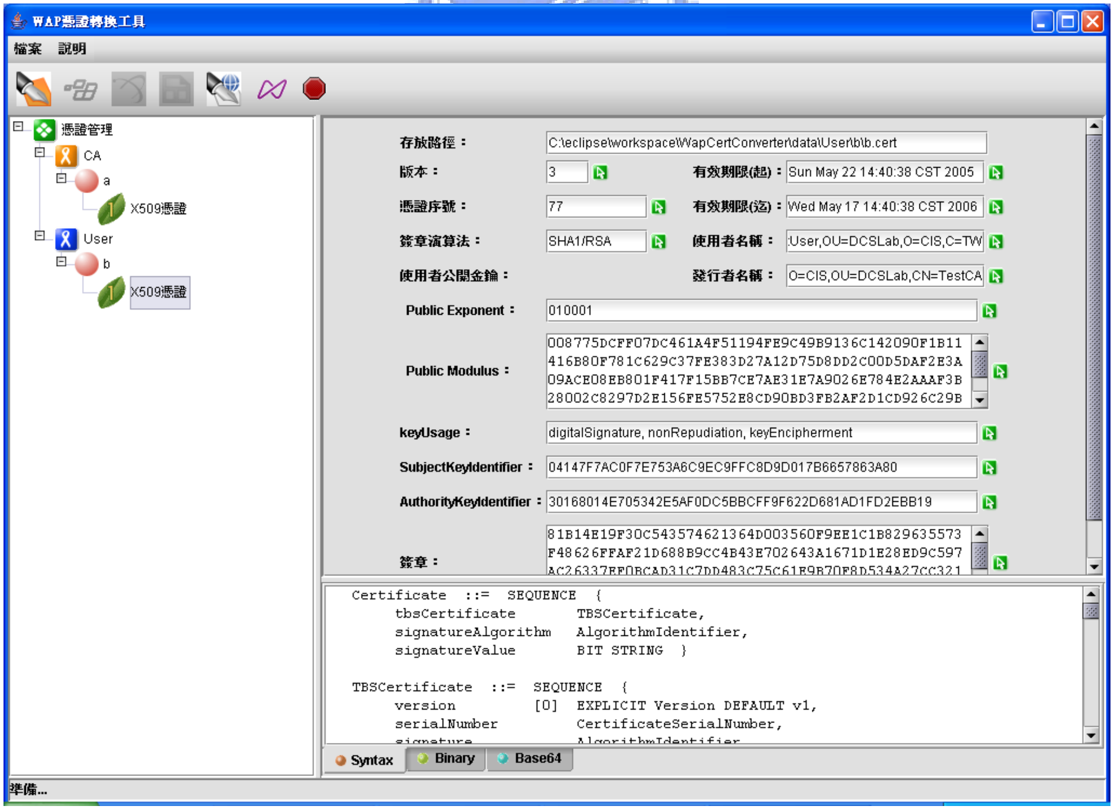


存放路徑：	C:\eclipse\workspace\WapCertConverter\data\CA\a\a.cert		
版本：	3	有效期限(起)：	Sun May 22 14:37:45 CST 2005
憑證序號：	76	有效期限(迄)：	Wed May 17 14:37:46 CST 2006
簽章演算法：	SHA1/RSA	使用者名稱：	stCA,OU=DCSLab,O=CIS,C=TW
使用者公開金鑰：		發行者名稱：	stCA,OU=DCSLab,O=CIS,C=TW
Public Exponent：	010001		
Public Modulus：	00BC1BAD29B5392F4C4EFB6E68FEE0578C7CDF8350B08CE0 8EDC32CFBF8D3DF5FD3E50CD93636EB0C55F1A4A987919B7 48DF23E25BF9FE86A66C8B87EEE8F35815000972B38E46AB 87EFB269ED74051DDC6A4976D0B4A9CD67048B84CDCFE95A		
keyUsage：	digitalSignature, keyCertSign		
SubjectKeyIdentifier：	0414E705342E5AF0DC5BBCFF9F622D681AD1FD2EBB19		
AuthorityKeyIdentifier：	30168014E705342E5AF0DC5BBCFF9F622D681AD1FD2EBB19		
ExtendedKeyUsage：	id-kp-codeSigning		
BasicConstraints：	isCA		

```
Certificate  ::=  SEQUENCE  {
     tbsCertificate       TBSCertificate,
     signatureAlgorithm   AlgorithmIdentifier,
     signatureValue       BIT STRING  }

TBSCertificate  ::=  SEQUENCE  {
     version         [0]  EXPLICIT Version DEFAULT v1,
     serialNumber         CertificateSerialNumber,
     signature            AlgorithmIdentifier,
```

Figure 5-1: create a CA X.509 certificate

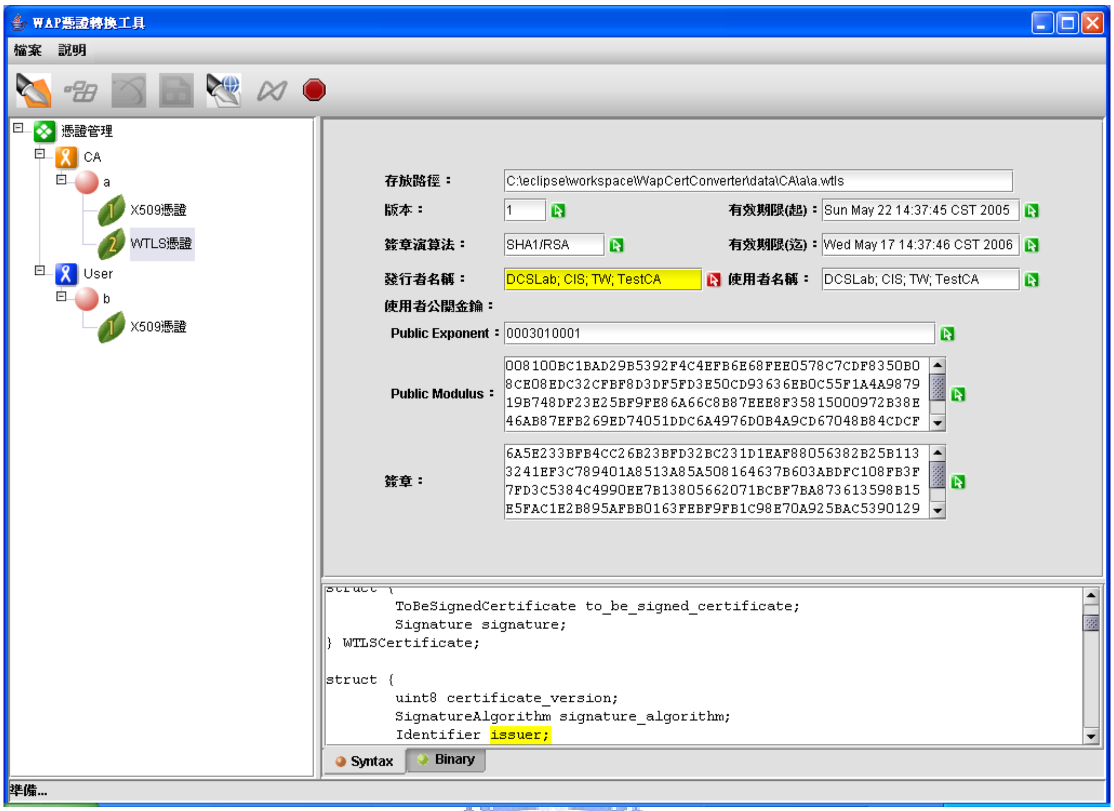


存放路徑：	C:\eclipse\workspace\WapCertConverter\data\User\b\b.cert		
版本：	3	有效期限(起)：	Sun May 22 14:40:38 CST 2005
憑證序號：	77	有效期限(迄)：	Wed May 17 14:40:38 CST 2006
簽章演算法：	SHA1/RSA	使用者名稱：	User,OU=DCSLab,O=CIS,C=TW
使用者公開金鑰：		發行者名稱：	O=CIS,OU=DCSLab,CN=TestCA
Public Exponent：	010001		
Public Modulus：	008775DCFF07DC461A4F51194FE9C49B9136C142090F1B11 416B80F781C629C37FE383D27A12D75D8DD2C00D5DAF2E3A 09ACE08EB801F417F15BB7CE7AE31E7A9026E784E2AAAF3B 28002C8297D2E156FE5752E8CD90BD3FB2AF2D1CD926C29B		
keyUsage：	digitalSignature, nonRepudiation, keyEncipherment		
SubjectKeyIdentifier：	04147F7AC0F7E753A6C9EC9FFC8D9D017B6657863A80		
AuthorityKeyIdentifier：	30168014E705342E5AF0DC5BBCFF9F622D681AD1FD2EBB19		
簽章：	81B14E19F30C5435746213 64D003560F9EE1C1B829635573 F48626FFAF21D688B9CC4B43E702643A1671D1E28ED9C597 AC26337EF0BCAD31C7DD483C75C61E9B70F8D534A27CC321		

```
Certificate  ::=  SEQUENCE  {
     tbsCertificate       TBSCertificate,
     signatureAlgorithm   AlgorithmIdentifier,
     signatureValue       BIT STRING  }

TBSCertificate  ::=  SEQUENCE  {
     version         [0]  EXPLICIT Version DEFAULT v1,
     serialNumber         CertificateSerialNumber,
     signature            AlgorithmIdentifier,
```

Figure 5-2: create a User X.509 certificate



存放路徑：C:\eclipse\workspace\WapCertConverter\data\CA\a\a.wtls

版本：1

有效期限(起)：Sun May 22 14:37:45 CST 2005

簽章演算法：SHA1/RSA

有效期限(迄)：Wed May 17 14:37:46 CST 2006

發行者名稱：DCSLab; CIS; TW; TestCA

使用者名稱：DCSLab; CIS; TW; TestCA

使用者公開金鑰：

Public Exponent：0003010001

Public Modulus：

```
008100BC1BAD29B5392F4C4EFB6E68FEE0578C7CDF8350B0
8CE08EDC32CFBF8D3DF5FD3E50CD93636EB0C55F1A4A9879
19B748DF23E25BF9FE86A66C8B87EEE8F35815000972B38E
46AB87EFB269ED74051DDC6A4976D0B4A9CD67048B84CDCF
```

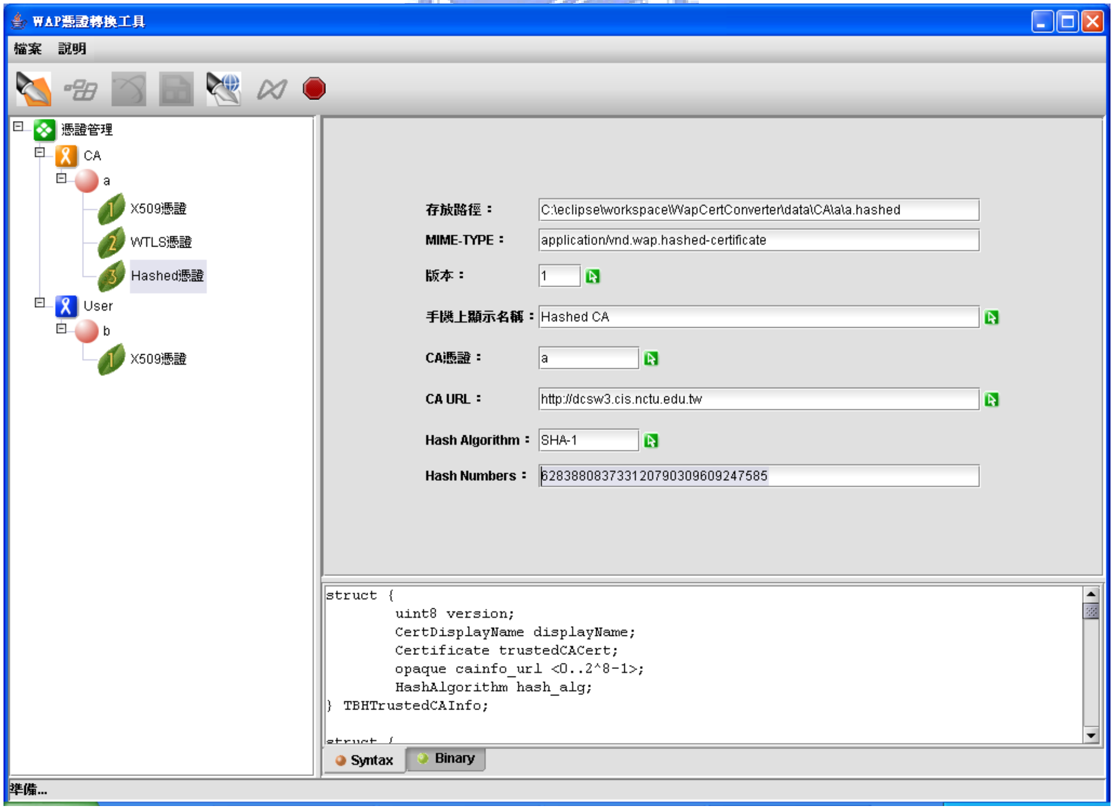
簽章：

```
6A5E233BFB4CC26B23BFD32BC231D1EAF88056382B25B113
3241EF3C789401A8513A85A508164637B603ABDFC108FB3F
7FD3C5384C4990EE7B13805662071BCBF7BA873613598B15
E5FAC1E2B895AFBB0163FEBF9FB1C98E70A925BAC5390129
```

```
        ToBeSignedCertificate to_be_signed_certificate;
        Signature signature;
} WTLSCertificate;

struct {
        uint8 certificate_version;
        SignatureAlgorithm signature_algorithm;
        Identifier issuer;
```

Figure 5-3: convert to WTLS certificate



存放路徑：C:\eclipse\workspace\WapCertConverter\data\CA\a\a.hashed

MIME-TYPE：application/vnd.wap.hashed-certificate

版本：1

手機上顯示名稱：Hashed CA

CA憑證：a

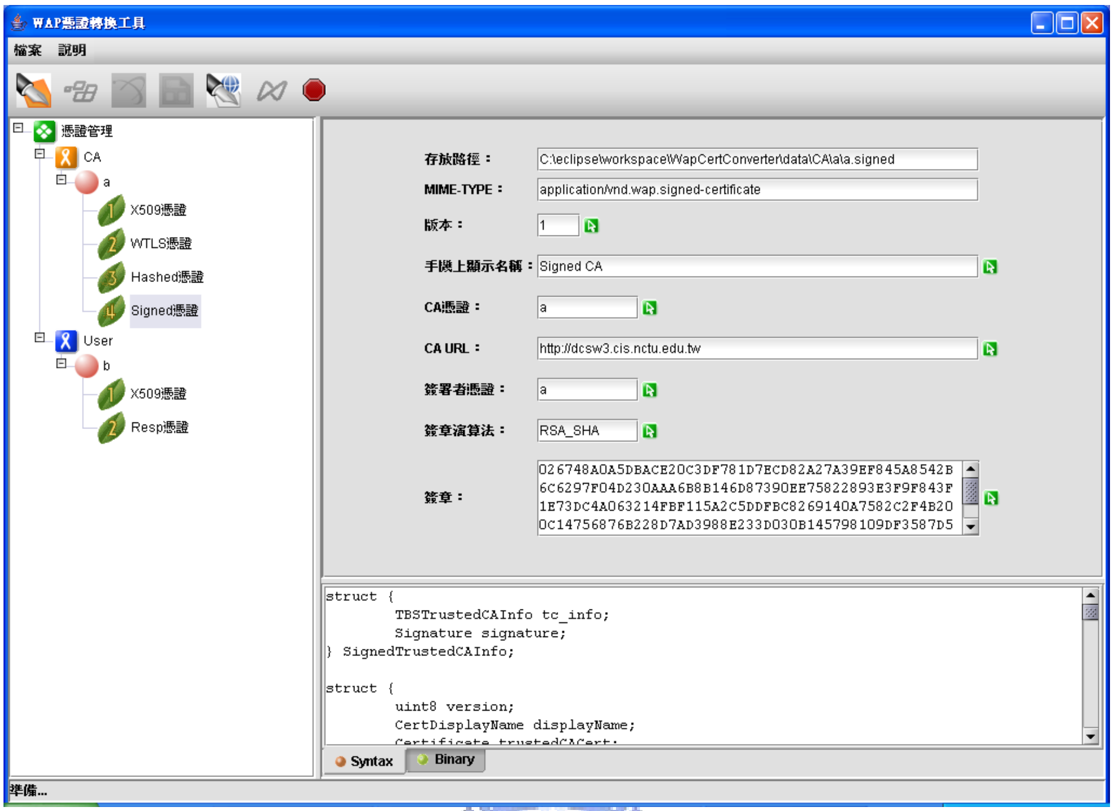
CA URL：http://dcsw3.cis.nctu.edu.tw

Hash Algorithm：SHA-1

Hash Numbers：628388083733120790309609247585

```
struct {
        uint8 version;
        CertDisplayName displayName;
        Certificate trustedCACert;
        opaque cainfo_url <0..2^8-1>;
        HashAlgorithm hash_alg;
} TBHTrustedCAInfo;
```

Figure 5-4: convert to Hashed certificate



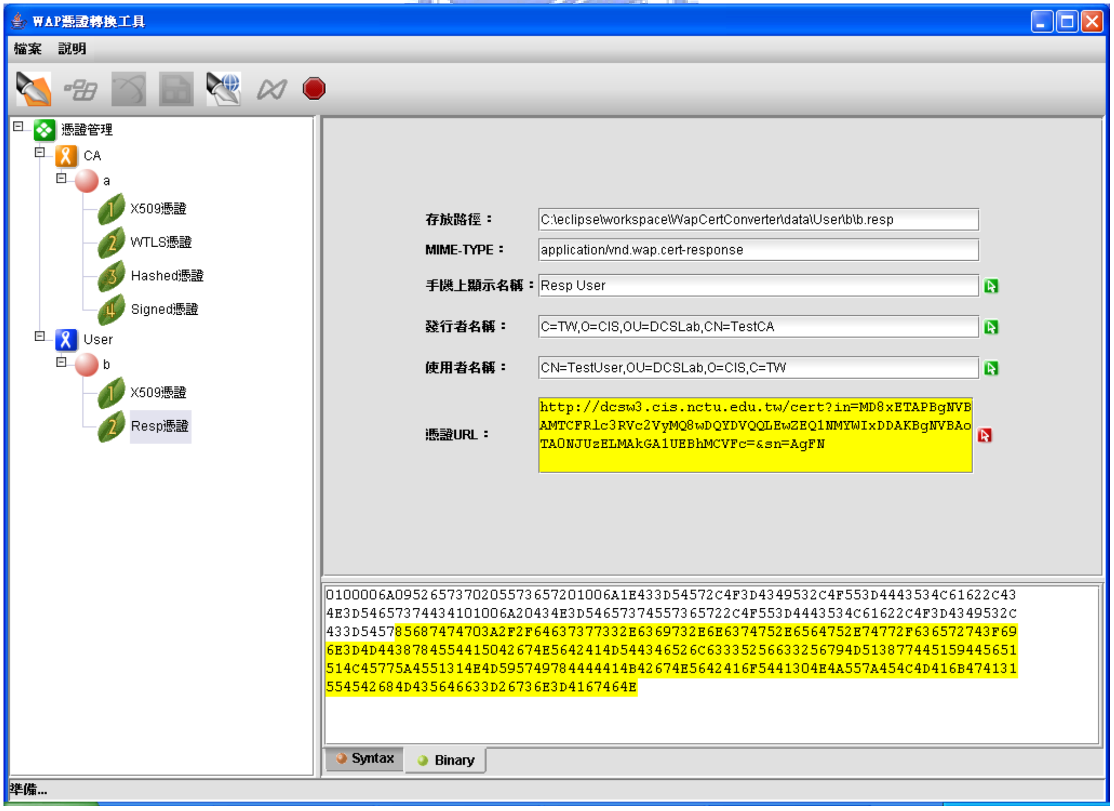
存放路徑：C:\eclipse\workspace\WapCertConverter\data\CA\a\a.signed
MIME-TYPE：application/vnd.wap.signed-certificate
版本：1
手機上顯示名稱：Signed CA
CA憑證：a
CA URL：http://dcsw3.cis.nctu.edu.tw
簽署者憑證：a
簽章演算法：RSA_SHA
簽章：

```
026748A0A5DBACE20C3DF781D7ECD82A27A39EF845A8542B
6C6297F04D230AAA6B8B146D87390EE75822893E3F9F843F
1E73DC4A063214FBF115A2C5DDFBC8269140A7582C2F4B20
0C14756876B228D7AD3988E233D030B145798109DF3587D5
```

```
struct {
        TBSTrustedCAInfo tc_info;
        Signature signature;
} SignedTrustedCAInfo;

struct {
        uint8 version;
        CertDisplayName displayName;
        Certificate trustedCACert;
```

Figure 5-5: convert to Signed certificate



存放路徑：C:\eclipse\workspace\WapCertConverter\data\User\b\b.resp
MIME-TYPE：application/vnd.wap.cert-response
手機上顯示名稱：Resp User
發行者名稱：C=TW,O=CIS,OU=DCSLab,CN=TestCA
使用者名稱：CN=TestUser,OU=DCSLab,O=CIS,C=TW
憑證URL：

```
http://dcsw3.cis.nctu.edu.tw/cert?in=MD8xETAPBgNVB
AMTCFRlc3RVc2VyMQ8wDQYDVQQLEwZEQ1NMYWIxDDAKBgNVBAo
TA0NJUzELMAkGA1UEBhMCVFc=&sn=AgFN
```

```
0100006A0952657370205573657201006A1E433D54572C4F3D4349532C4F553D4443534C61622C43
4E3D546573744341010006A20434E3D546573745573657 22C4F553D4443534C61622C4F3D4349532C
433D5457856874747 03A2F2F6463737 7332E6369732E6E6374752E6564752E74772F636572743F69
6E3D4D44387845544150426 74E5642414D5443465 26C63335256633256794D51387 7445159445651
514C45775A4551314E4D595749784444414B42674E5642416F5441304E4A557A454C4D416B474131
554542684D435646633D26736E3D4167464E
```

Figure 5-6: convert to Resp certificate

5.1.3. Evaluation

The six test items will create two private keys, six certificates, and a hash value of 30 digital numbers. Four of six certificates are the converted WAP certificates whose filename extensions are .wtls, .hashed, .signed and .resp. The hash value is stored in a file with extension .hashnumbers. From the view of certificate size, DER encoding of X.509 certificates really produces more bytes than the binary encoding of WTLS certificates.

5.2. Experiment 2: Import WAP Certificates into WIM

This experiment will check the correctness of those certificates created by Experiment 1 by importing them to the WIM within a mobile phone.

5.2.1. Environment Configuration

We use the Nokia Mobile Browser 4.0 (NMB 4.0) [19] as the simulated mobile phone. Within the NMB 4.0, there is software-based WIM called Nokia SoftID to store the imported certificates. We will import the certificates from locally and remotely. To locally import a certificate, we use the tool provided by Nokia SoftID. For remotely downloading certificates, we need the WAP gateway and the PKI Portal. We use the Nokia WAP Gateway Simulator 4.0 (NWGS 4.0) [20] and Tomcat 4.1 [22] to simulate the WAP gateway and the PKI Portal, respectively.

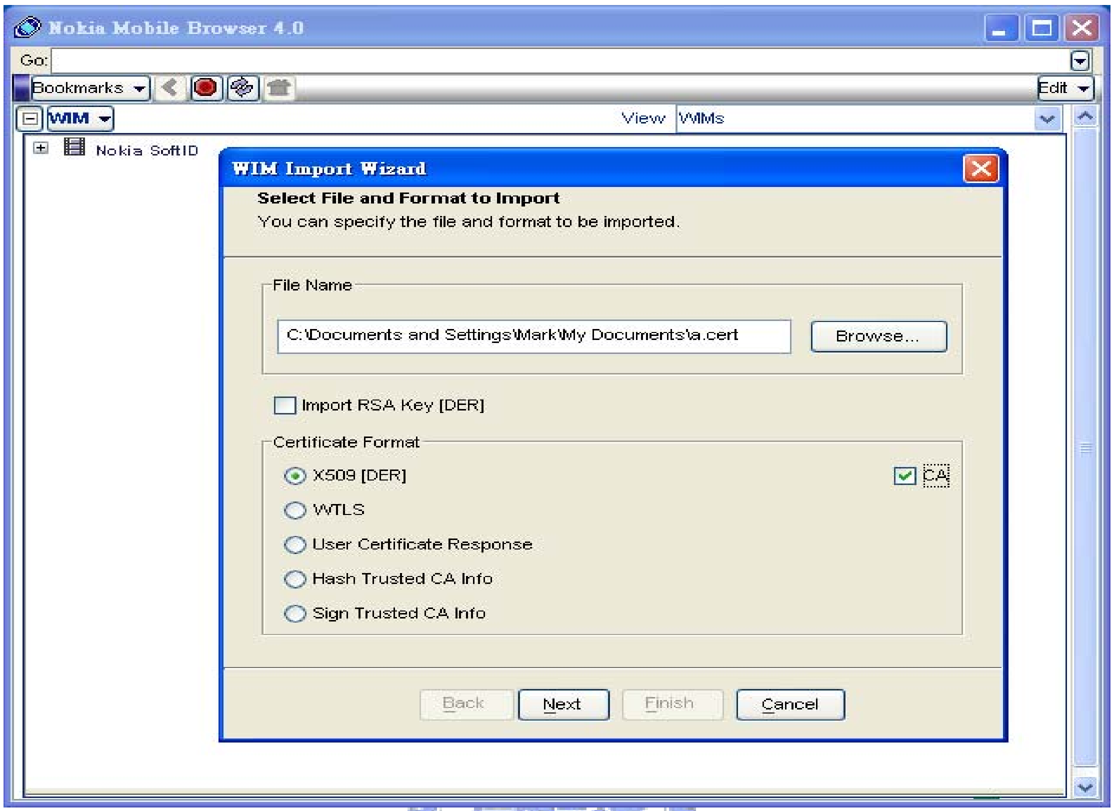
5.2.2.Experiment Result

The test items and results are shown in the following table.

Test Item	**Description**	**Result**
import a CA X.509 certificate	import the CA X.509 certificate locally to the WIM	see Figure 5-7
import a CA WTLS certificate	import the CA WTLS certificate locally to the WIM	see Figure 5-8
import a CA Hashed certificate	download the CA Hashed certificate remotely to the WIM	see Figure 5-9
import a CA Signed certificate	download the CA Signed certificate remotely to the WIM	see Figure 5-10
import a user X.509 certificate	import the user X.509 certificate locally to the WIM	see Figure 5-11
import a user Resp certificate	download the user X.509 certificate remotely to the WIM	see Figure 5-12

Table 5-2: Experiment 2 Result

Figure 5-7: import a CA X.509 certificate



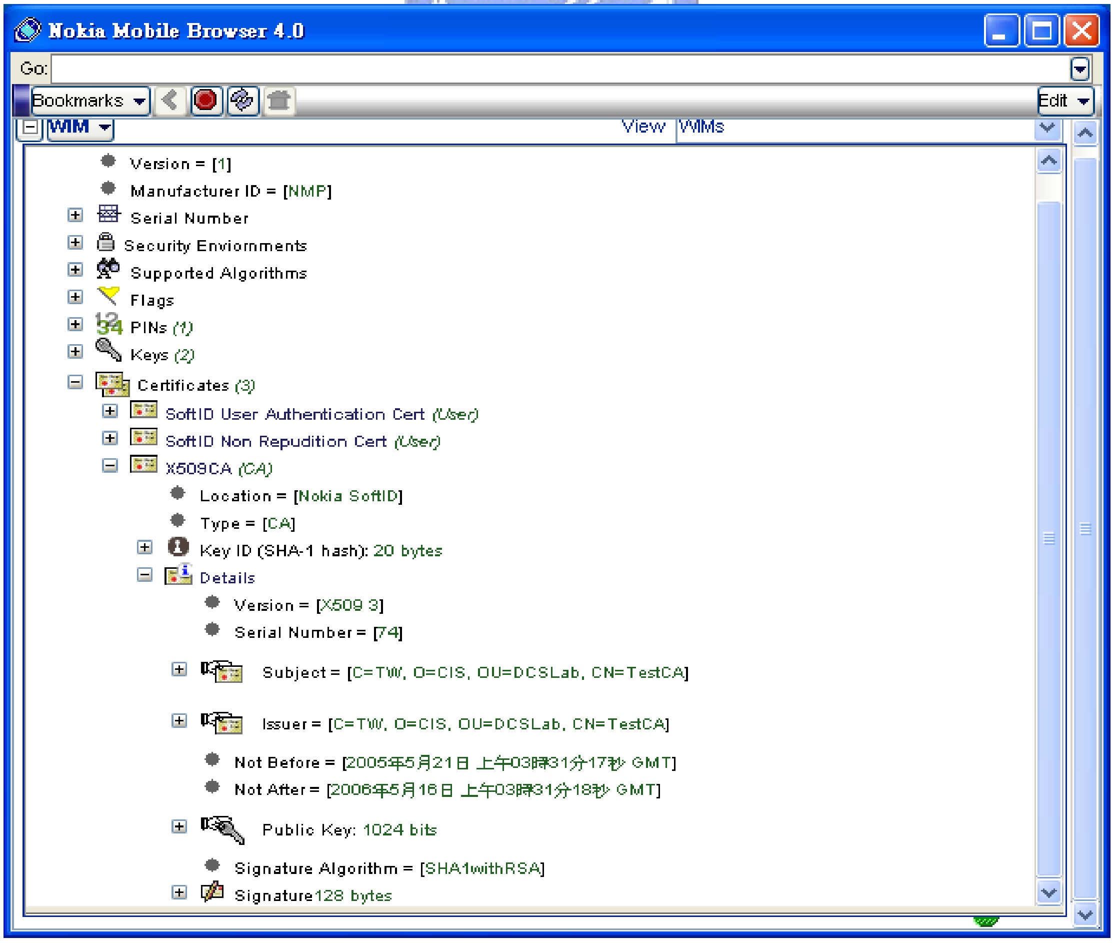
Select File and Format to Import
You can specify the file and format to be imported.

File Name
C:\Documents and Settings\Mark\My Documents\a.cert
Browse...

[ ] Import RSA Key [DER]

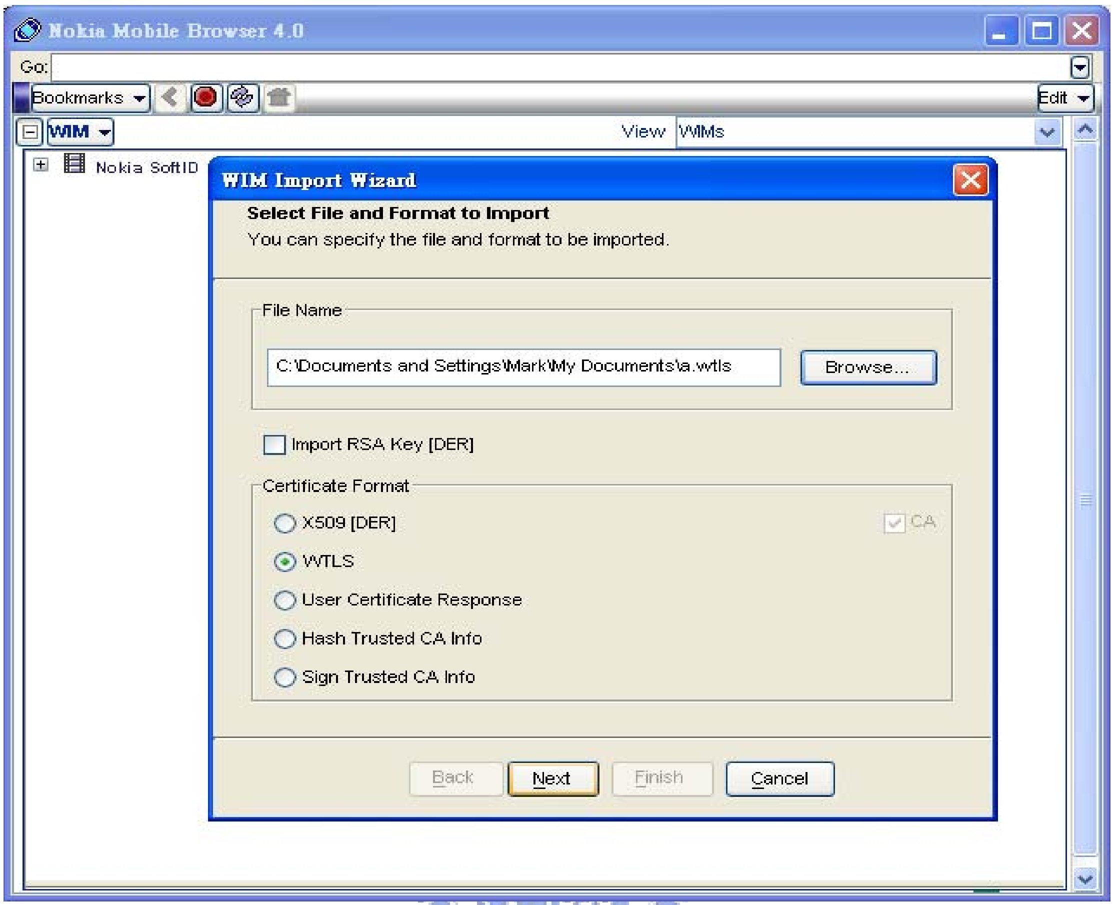
Certificate Format
(●) X509 [DER] [✓ CA]
( ) WTLS
( ) User Certificate Response
( ) Hash Trusted CA Info
( ) Sign Trusted CA Info

Back | Next | Finish | Cancel



```
Version = [1]
Manufacturer ID = [NMP]
Serial Number
Security Enviornments
Supported Algorithms
Flags
PINs (1)
Keys (2)
Certificates (3)
    SoftID User Authentication Cert (User)
    SoftID Non Repudition Cert (User)
    X509CA (CA)
        Location = [Nokia SoftID]
        Type = [CA]
        Key ID (SHA-1 hash): 20 bytes
        Details
            Version = [X509 3]
            Serial Number = [74]
            Subject = [C=TW, O=CIS, OU=DCSLab, CN=TestCA]
            Issuer = [C=TW, O=CIS, OU=DCSLab, CN=TestCA]
            Not Before = [2005年5月21日 上午03時31分17秒 GMT]
            Not After = [2006年5月16日 上午03時31分18秒 GMT]
            Public Key: 1024 bits
            Signature Algorithm = [SHA1withRSA]
            Signature128 bytes
```

Figure 5-8: import a CA WTLS certificate



WIM Import Wizard

Select File and Format to Import

You can specify the file and format to be imported.

File Name

C:\Documents and Settings\Mark\My Documents\a.wtls

Browse...

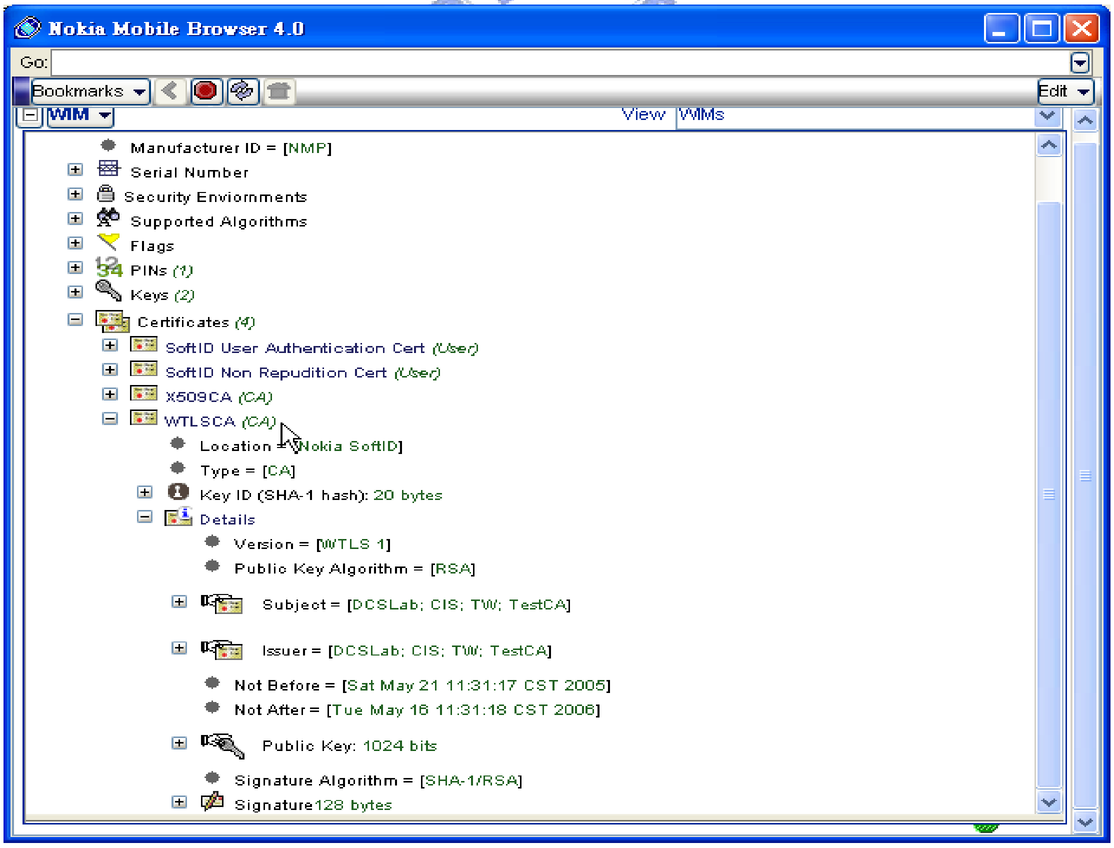
☐ Import RSA Key [DER]

Certificate Format

- ◯ X509 [DER]
- ◉ WTLS
- ◯ User Certificate Response
- ◯ Hash Trusted CA Info
- ◯ Sign Trusted CA Info

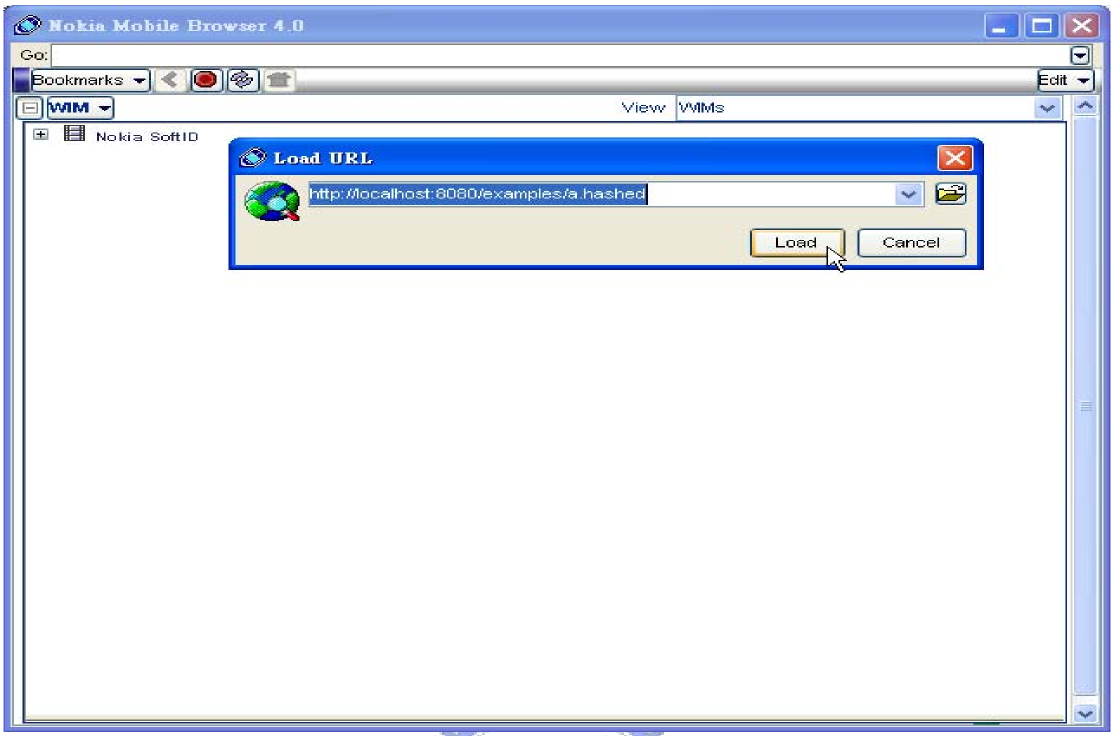
☑ CA

Back | Next | Finish | Cancel

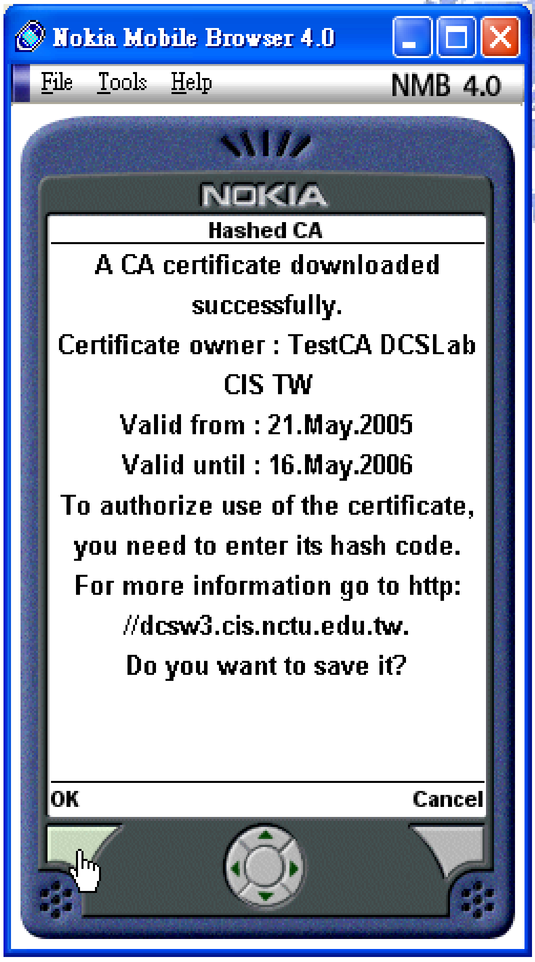


```
Manufacturer ID = [NMP]
Serial Number
Security Enviornments
Supported Algorithms
Flags
PINs (1)
Keys (2)
Certificates (4)
  SoftID User Authentication Cert (User)
  SoftID Non Repudition Cert (User)
  X509CA (CA)
  WTLSCA (CA)
    Location = [Nokia SoftID]
    Type = [CA]
    Key ID (SHA-1 hash): 20 bytes
    Details
      Version = [WTLS 1]
      Public Key Algorithm = [RSA]
      Subject = [DCSLab; CIS; TW; TestCA]
      Issuer = [DCSLab; CIS; TW; TestCA]
      Not Before = [Sat May 21 11:31:17 CST 2005]
      Not After = [Tue May 16 11:31:18 CST 2006]
      Public Key: 1024 bits
      Signature Algorithm = [SHA-1/RSA]
      Signature128 bytes
```

Figure 5-9: import a CA Hashed certificate

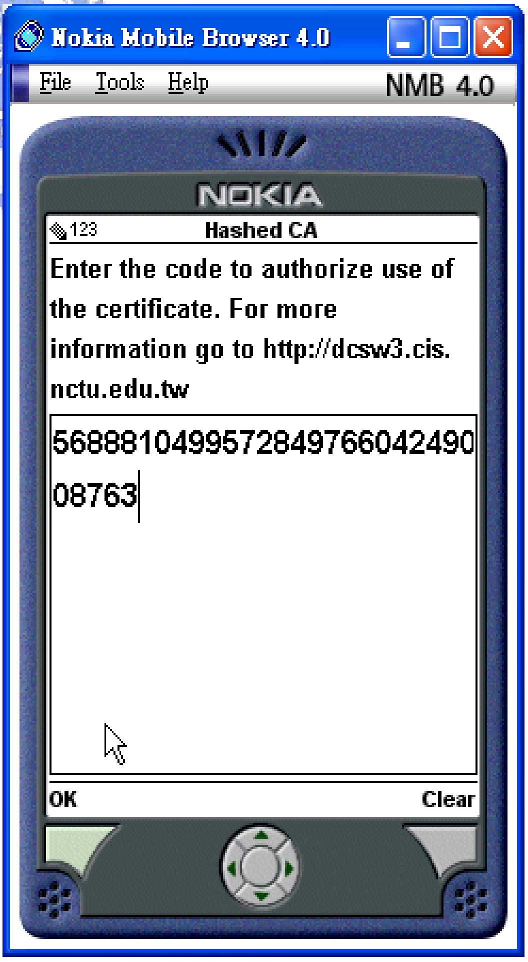


http://localhost:8080/examples/a.hashed



Hashed CA

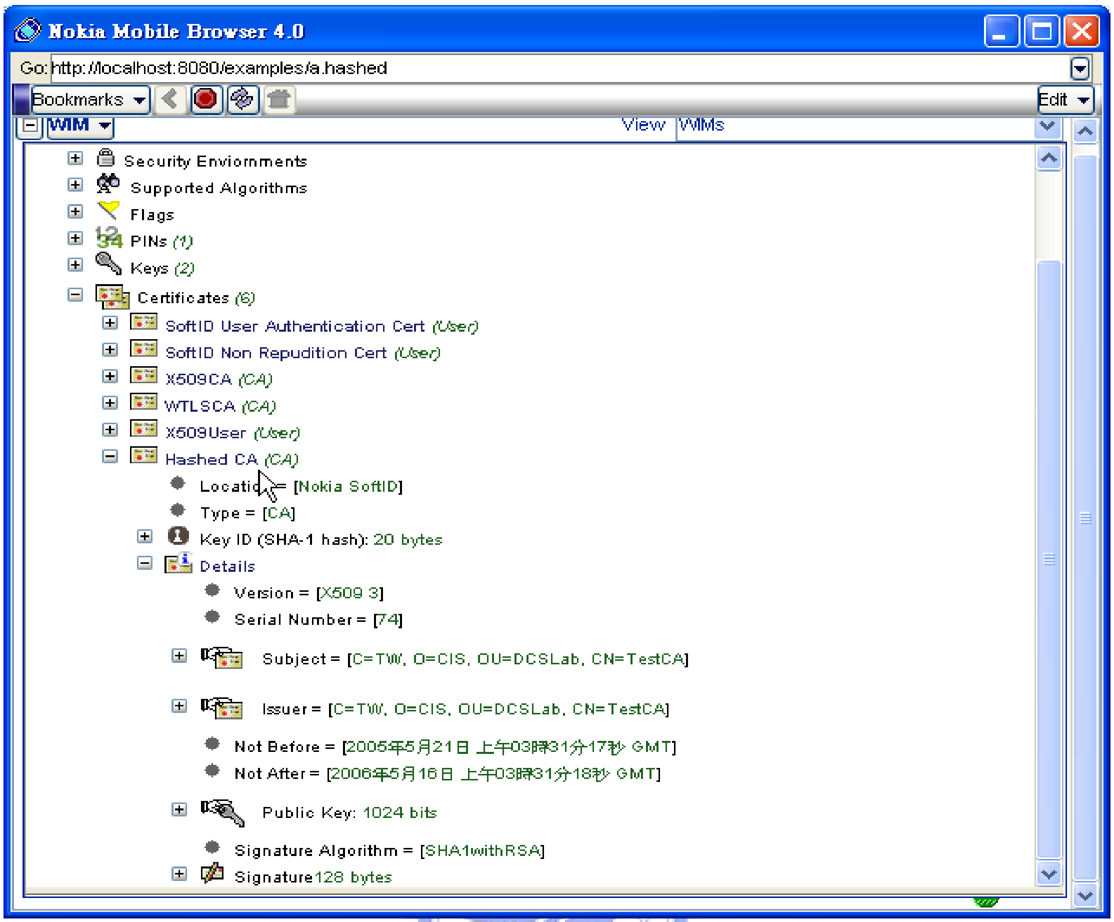
A CA certificate downloaded successfully.
Certificate owner : TestCA DCSLab CIS TW
Valid from : 21.May.2005
Valid until : 16.May.2006
To authorize use of the certificate, you need to enter its hash code.
For more information go to http://dcsw3.cis.nctu.edu.tw.
Do you want to save it?



Hashed CA

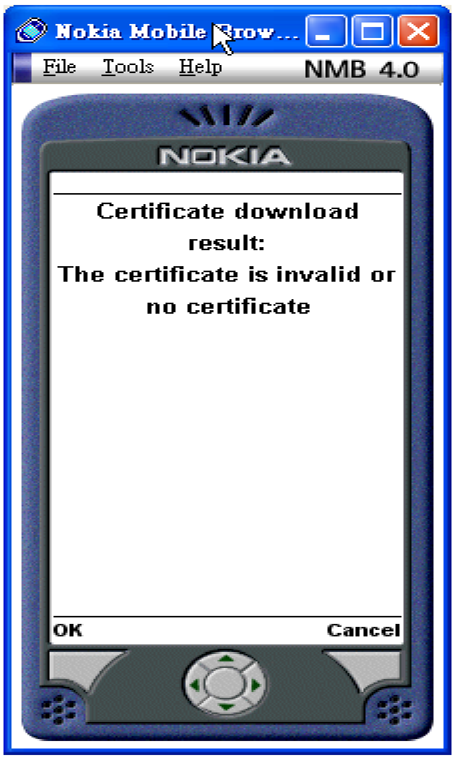
Enter the code to authorize use of the certificate. For more information go to http://dcsw3.cis.nctu.edu.tw

56888104995728497660424900 8763



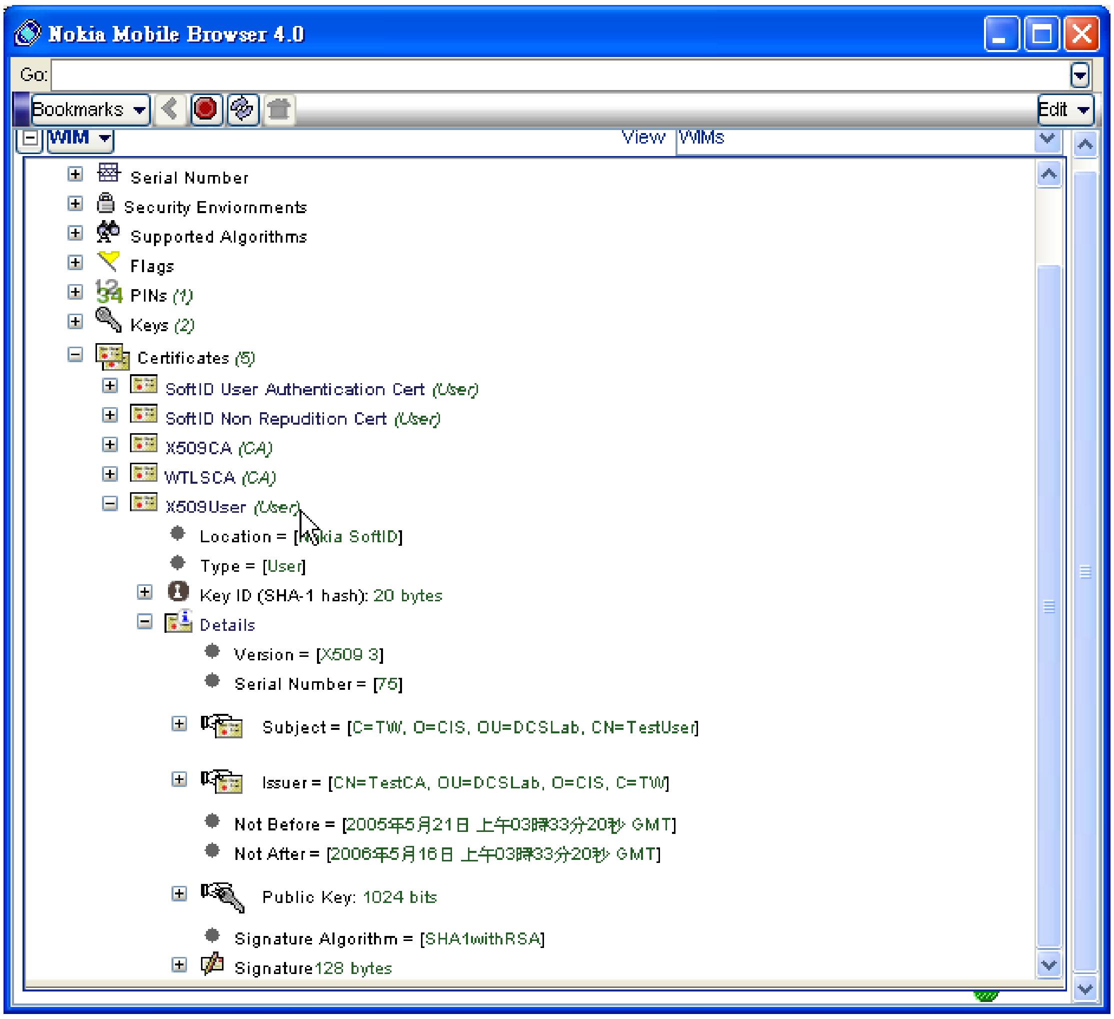
```
Security Enviornments
Supported Algorithms
Flags
PINs (1)
Keys (2)
Certificates (6)
    SoftID User Authentication Cert (User)
    SoftID Non Repudition Cert (User)
    X509CA (CA)
    WTLSCA (CA)
    X509User (User)
    Hashed CA (CA)
        Location = [Nokia SoftID]
        Type = [CA]
        Key ID (SHA-1 hash): 20 bytes
        Details
            Version = [X509 3]
            Serial Number = [74]
            Subject = [C=TW, O=CIS, OU=DCSLab, CN=TestCA]
            Issuer = [C=TW, O=CIS, OU=DCSLab, CN=TestCA]
            Not Before = [2005年5月21日 上午03時31分17秒 GMT]
            Not After = [2006年5月16日 上午03時31分18秒 GMT]
            Public Key: 1024 bits
            Signature Algorithm = [SHA1withRSA]
            Signature 128 bytes
```

Figure 5-10: import a CA Signed certificate



Certificate download result:
The certificate is invalid or no certificate

Figure 5-11: import a user X.509 certificate


Nokia Mobile Browser 4.0

Go:

Bookmarks

Edit

WIM

View WIMs

Serial Number

Security Enviornments

Supported Algorithms

Flags

PINs *(1)*

Keys *(2)*

Certificates *(5)*

SoftID User Authentication Cert *(User)*

SoftID Non Repudition Cert *(User)*

X509CA *(CA)*

WTLSCA *(CA)*

X509User *(User)*

Location = [Nokia SoftID]

Type = [User]

Key ID (SHA-1 hash): 20 bytes

Details

Version = [X509 3]

Serial Number = [75]

Subject = [C=TW, O=CIS, OU=DCSLab, CN=TestUser]

Issuer = [CN=TestCA, OU=DCSLab, O=CIS, C=TW]

Not Before = [2005年5月21日 上午03時33分20秒 GMT]

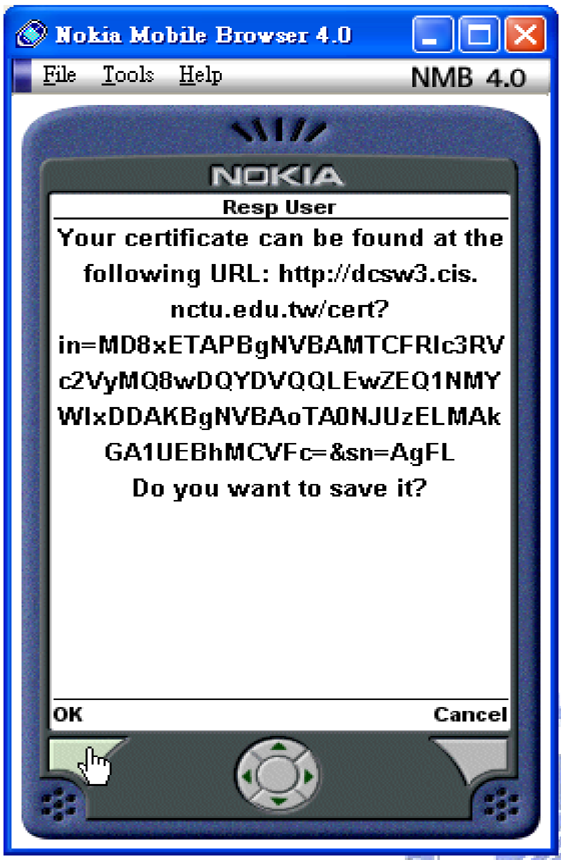
Not After = [2006年5月16日 上午03時33分20秒 GMT]

Public Key: 1024 bits

Signature Algorithm = [SHA1withRSA]

Signature128 bytes

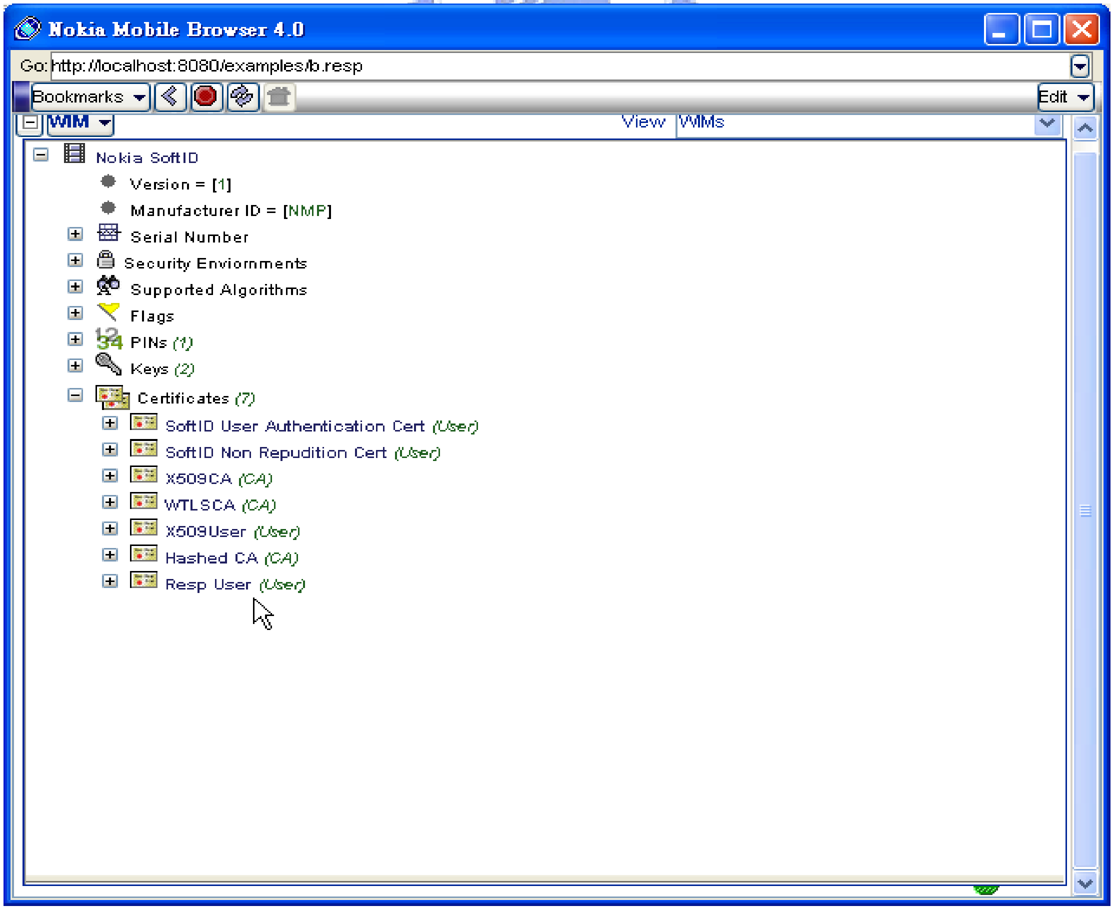
Figure 5-12: import a user Resp certificate



Resp User

Your certificate can be found at the following URL: http://dcsw3.cis.nctu.edu.tw/cert?in=MD8xETAPBgNVBAMTCFRlc3RVc2VyMQ8wDQYDVQQLEwZEQ1NMYWIxDDAKBgNVBAoTA0NJUzELMAkGA1UEBhMCVFc=&sn=AgFL
Do you want to save it?

OK Cancel



```
Nokia SoftID
    Version = [1]
    Manufacturer ID = [NMP]
    Serial Number
    Security Enviornments
    Supported Algorithms
    Flags
    PINs (1)
    Keys (2)
    Certificates (7)
        SoftID User Authentication Cert (User)
        SoftID Non Repudition Cert (User)
        X509CA (CA)
        WTLSCA (CA)
        X509User (User)
        Hashed CA (CA)
        Resp User (User)
```

5.2.3. Evaluation

All of the test items are done well except the one of importing CA Signed certificate. According to the WPKI specification, a CA Signed certificate will have both inner and outer signatures within it. The inner signature is the signature within the signerCert. The outer signature is the signature of TBSTrustedCAInfo structure signed by the signerCert. The client must trust the signerCert before using the certificate embedded in the TBSTrustedCAInfo structure. In our implementation, the signerCert and the CA certificate to be downloaded are the same because the CA is the Root CA. One possible reason to explain this failure is the NMB 4.0 can not trust our CA as a Root CA.

5.3. Experiment 3: A Simple M-Commerce Application with Crypto.signText

This experiment shows a simple m-commerce application to provide non-repudiation service via Crypto.signText method with the user certificate created by our toolkit.

5.3.1. Environment Configuration

The simple m-commerce application is modified from the sample provided by the Nokia Mobile Internet Toolkit (NMIT) 4.1 [21]. We use the same environment configuration in Experiment 2. The only difference is that the role of Tomcat server is changed to be a web fast-food store. The scenario of the simple m-commerce application is as follows. The user has ordered a pizza and a cup of coffee from the web fast-food store. The store sent the order back to the user for requesting confirmation. After make sure the order recorded in the web fast-food store is correct, the user signs the order by calling Crypto.signText() with his signature key stored in the WIM and then submits the order with its signature to the web fast-food store to accomplish the transaction.

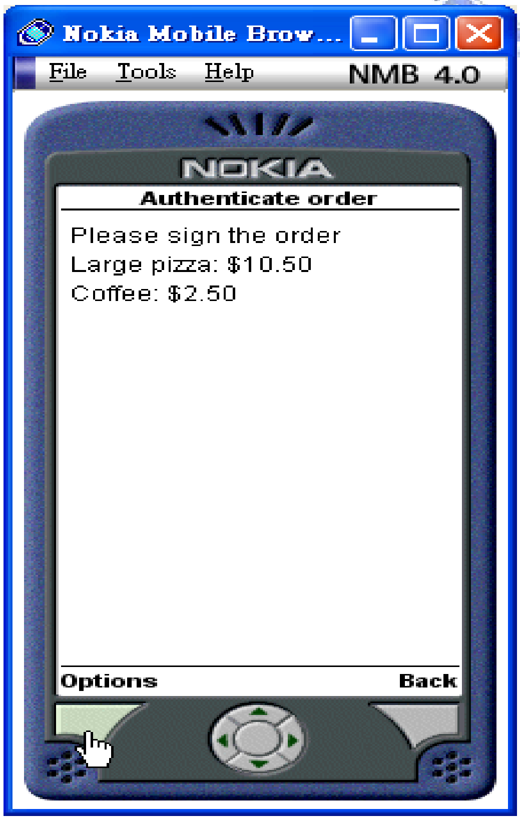
5.3.2.Experiment Result

The test items and result are shown in the following table.

Test Item	**Description**	**Result**
download the order for confirmation	download the order recorded in the web fast-food store	see Figure 5-13
sign the order and submit	confirm the order by signing it and send back to web fast-food store	see Figure 5-14

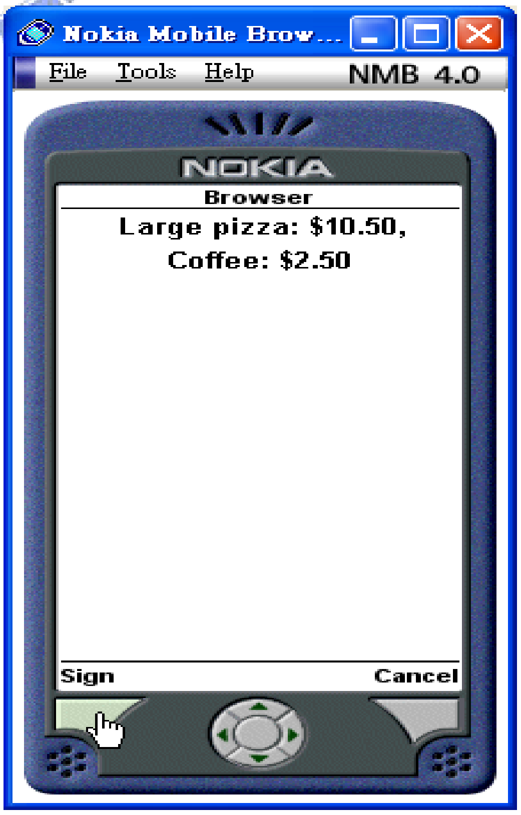
Table 5-3: Experiment 3 Result

Figure 5-13: download the order for confirmation



Authenticate order

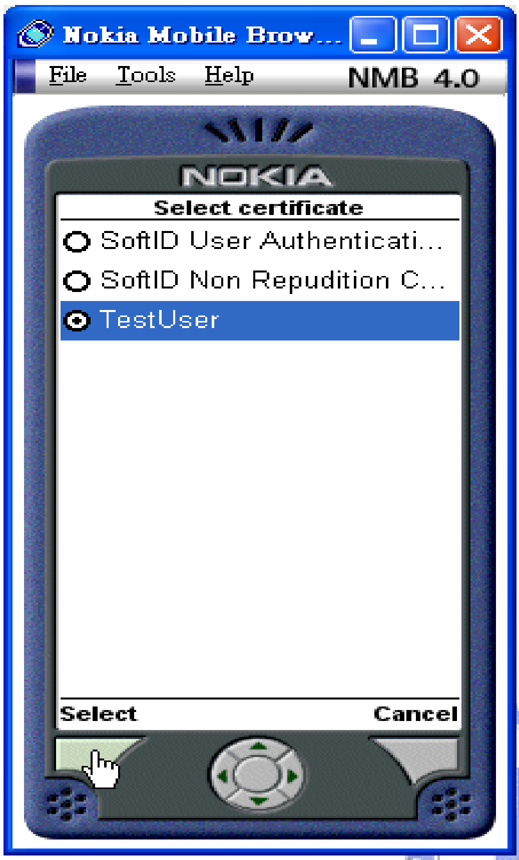
Please sign the order
Large pizza: $10.50
Coffee: $2.50



Browser

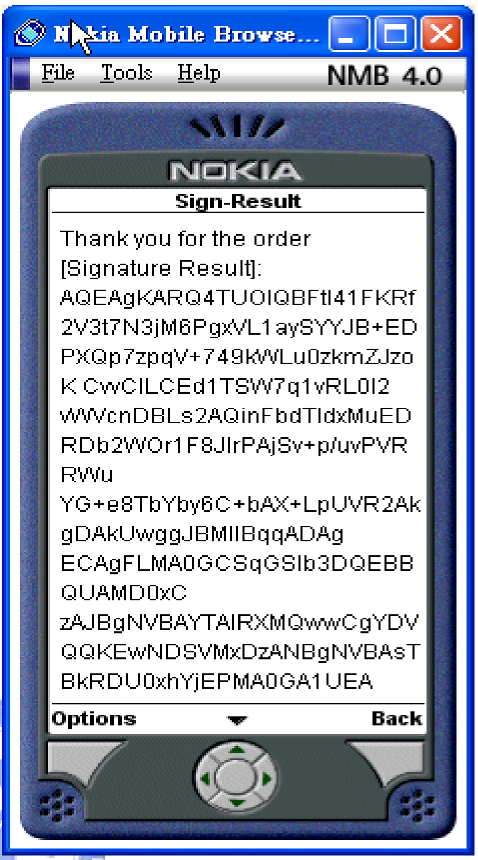
Large pizza: $10.50,
Coffee: $2.50

Figure 5-14: sign the order and submit



Select certificate

- ○ SoftID User Authenticati…
- ○ SoftID Non Repudition C…
- ◉ TestUser



Sign-Result

```
Thank you for the order
[Signature Result]:
AQEAgKARQ4TUOIQBFtl41FKRf
2V3t7N3jM6PgxVL1aySYYJB+ED
PXQp7zpqV+749kWLu0zkmZJzo
K CwCILCEd1TSW7q1vRL0I2
WWcnDBLs2AQinFbdTIdxMuED
RDb2WOr1F8JlrPAjSv+p/uvPVR
RWu
YG+e8TbYby6C+bAX+LpUVR2Ak
gDAkUwggJBMIIBqqADAg
ECAgFLMA0GCSqGSIb3DQEBB
QUAMD0xC
zAJBgNVBAYTAIRXMQwwCgYDV
QQKEwNDSVMxDzANBgNVBAsT
BkRDU0xhYjEPMA0GA1UEA
```

5.3.3.Evaluation

This experiment successfully illustrates how to use a private key and corresponding user certificate stored in the WIM with the Crypto.signText method to provide the non-repudiation service in a typical m-commerce application.

Chapter 6

Conclusions and Future Works

Conclusions

To promote more m-commerce applications developed on mobile devices, security undoubtedly plays the key role among them. To bring the public-key cryptography into wireless networks for mobile security, WAP has devoted itself for setting up several standards for it.

In this paper, we try to solve a problem related to the public-key cryptography in the wireless world. The problem is how to use existing Internet X.509 certificates on mobile devices. To figure it out, we consider three dimensions of the problem domain. The first dimension is the compatibility over WAP versions. The second one is the limitations in the wireless environment and mobile devices. The last one is the current situation about PKI in the Internet. The solution we propose to address those issues is the design of WAP Certificate Converter Toolkit.

The main merits of our developed toolkit can be concluded as follows:

- It can produce certificates compatible over all of WAP versions.
- The existing Internet CAs can easily transform their issued X.509 certificates into WAP certificates.
- Each field of the certificate produced by it can be seen in variable views.
- It provides two interfaces – Library API and GUI – for programmers and operators respectively.

Future Works

With our toolkit, it is easy to build the skeleton of WPKI. However, to completely build up the WPKI to provide full support of mobile security for m-commerce applications, there are several things to be done. Some of them are concerning about the extension of our toolkit; the others are with respect to the implementations of related WPKI facilities.

The future development of our toolkit can go to three directions. First is the support of ECDSA. Even though RSA is the widely used signature algorithm in the Internet, the advantages of ECDSA, compared to RSA, show it is suitable to be applied in limited computing environments. Another reason to add ECDSA support is it is another signature algorithm specified by WAP. The second is the design of decoder for SignedContent structure generated by Crypto.signText function. Before the web server verifies the signature signed by the user, it need to first decode the structure to retrieve the signature and related information about the public key.

As to the related facilities to build a complete WPKI, storage for certificates saving, either a database or a LDAP directory, a PKI Portal to check and audit the identity of users, web interface to do all the management, and a real e-commerce application are considered important issues to be further solved.

Reference

[1] WAP Forum (2000), “Wireless Application Protocol White Paper”, June 2000. URL: http://www.wapforum.org

[2] WAP Forum (2002), “Wireless Application Protocol WAP 2.0 Technical White Paper”, January 2002. URL: http://www.wapforum.org

[3] WAP Forum (2001), “WAP Architecture”, WAP-210-WAPArch-20010712-a, 12 July 2001. URL: http://www.wapforum.org

[4] WAP Forum (2001), “Wireless Transport Layer Security”, WAP-261-WTLS-20010406-a, 6 April 2001. URL: http://www.wapforum.org

[5] WAP Forum (2001), “WPKI”, WAP-217-WPKI-20010424-a, 24 April 2001. URL: http://www.wapforum.org

[6] WAP Forum (2001), “Wireless Identity Module”, WAP-260-WIM-20010712-a, 12 July 2001. URL: http://www.wapforum.org

[7] WAP Forum (2001), “WMLScript Crypto Library”, WAP-161-WMLScriptCrypto-20010620-a, 20 June 2001. URL: http://www.wapforum.org

[8] WAP Forum (2001), “WAP Certificate and CRL Profiles”, WAP-211-WAPCert-20010522-a, 22 May 2001. URL: http://www.wapforum.org

[9] G. Radhamani, K. Ramasamy, “Security Issues in WAP WTLS Protocol”, IEEE 2002 International Conference on Communications, Circuits and Systems and West Sino Expositions, Volume 1, Number 29, July 2002, 483-487.

[10] Thanh V. Do, “WAP Security: WTLS”, 2001. Available from http://ece.gmu.edu/courses/ECE636/project/reports/TDo.pdf

[11] RFC2459, "Internet X.509 Public Key Infrastructure Certificate and CRL Profile", January 1999.

[12] RFC3174,"US Secure Hash Algorithm 1 (SHA1)", September 2001.

[13] Vladimir Silva, “Manage X.509 certificates in your grid with Java Certificate Services”, 2003. Available from http://www-106.ibm.com/developerworks/grid/library/gr-jsc/?ca=dgr-lnxw06ManageX.509

[14] Chris Melnick, “How to translate into base64 and back”, 2004. Available from http://www.aardwulf.com/tutor/base64/index.asp

[15] R. L. Rivest, A. Shamir, and L. Adleman, “A Method for Obtaining Digital Signatures and Public-Key Cryptosystems”, Communications of the ACM, Volume 21, Issue 2, February 1978, 120-126.

[16] Don B. Johnson and Alfred J. Menezes, “Elliptic Curve DSA (ECDSA): An Enhanced DSA”, 1999. Available from http://www.certicom.com

[17] Bouncy Castle Crypto, http://www.bouncycastle.org

[18] Cryptix JCE, http://www.cryptix.org

[19] Nokia, Nokia Mobile Browser 4.0 (NMB 4.0), http://www.forum.nokia.com

[20] Nokia, Nokia WAP Gateway Simulator 4.0 (NWGS 4.0), http://www.forum.nokia.com

[21] Nokia, Nokia Mobile Internet Toolkit (NMIT) 4.1, http://www.forum.nokia.com

[22] Apache Jakarta Tomcat 4.1, http://jakarta.apache.org/tomcat

[23] Andrew Nash, Bill Duane, Derek Brink and Celia Joseph, *PKI: implementing and managing E-security*, McGraw-Hill, 2001.

[24] Li Gong, Gary Ellison and Mary Dageforde, *Inside Java 2 Platform Security: Architecture, API Design and Implementation*, second version, Sun, 2003

[25] Rich Helton, Johennie Helton 著，楊松諺/上官飛鳳 譯，*Java Security 全方*

位解決方案，碁峰，2002.

[26] Alfred Menezes, Paul van Oorschot, Scott Vanstone, *Handbook of Applied Cryptography*, CRC, 1997

[27] Charlie Kaufman, Radia Perlman, Mike Speciner, *Network Security*, second version, Prentice Hall, 2002.

[28] Eric Rescorla, *SSL and TLS*, Addison Wesley, 2001.

[29] Olivier Dubuisson, translated from French by Philippe Fouquart, *ASN.1 - Communication Between Heterogeneous Systems*, , Morgan Kaufmann, 2001.