Security and Public Key Cryptography on BREW Mobile Platform

Samsu Sempena 13507088[1]
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jl. Ganesha 10 Bandung 40132, Indonesia
[1]if17088@students.if.itb.ac.id

***Abstract*— Mobile users are continually increasing and mobile environments covers wider area of applications from contents download, e-shopping, e-government, and many more. Unfortunately, in the real world there are many malware (code designed to in some way attack the device, the network, or the end user) infiltrate in end user application. So, we need to secure mobile platforms.**

There are many mobile platforms out there, but this paper will focus to analyze the security issue in BREW Mobile Platform (BREW MP) as one of the largest mobile platform among Blackberry, iPhone, Windows Mobile, Android, and many more. This paper will begin with introduction about public key infrastructure in mobile networks and BREW Platform. After that this paper will cover some security functions provided by BREW MP such as to limit code privileges, to implement cryptography algorithm provided by BREW MP, and code authorization with RSA and some variants of SHA. Finally, this paper also provides analysis of strong and weak points of security on BREW MP if compared with other platforms.

***Index Terms*—BREW, Security, PKI, RSA, SHA**

I. INTRODUCTION

I.1 Public Key Cryptography

Until 1970s, there was only symmetric key cryptography that use the same key to decrypt and encrypt a message. But, one more important problem in cryptography, how to send the secret key to the receiver? Sending the secret key via public channel is really not safe. Then the idea of asymmetric key cryptography was emerged in 1976. The first paper published in IEEE about this asymmetric key cryptography was written by Diffie-Hellman (an engineer from Stanford University) with title "New Directions in Cryptography".

Later, asymmetric key cryptography is also called public key cryptography. In this scheme, both of sender and receiver will have a pair of key, public key to encrypt the message and private key to decrypt the message.

The idea behind public key cryptography is based to the facts that computation for encrypting and decrypting message is easy to be implemented and nearly impossible to derive private key (d) from public key (e).

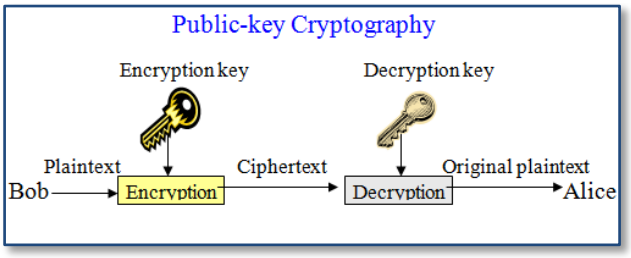

Figure 1. Public Key Cryptography Scheme

With the scheme of public key cryptography, we could gain some main benefits:

1. We need not have to send a secret key, we only need to keep our private key safely
2. Can be used to reassure the transmission of symmetric key
3. We need not to change the pair of our public and private key for a long period since its nearly impossible by computation for deriving our private key from our public key.

But, there are still some lack in public key algorithm, there are:

1. Basicly,the process of encryption and decryption will be slower than symmetric key cryptography because we use operation power of big integer in public key cryptography
2. Size of ciphertext can be bigger from plaintext (two until four times the size of plaintext)
3. Size of public/private key generally bigger than symmetric key
4. Can't give any information about sender's information, because our public key is known by everyone.
5. Although is really impossible to derive private key to decrypt a message, but still there is no guarantee that public key cryptography is really safe because it really depends on difficulties to solve arithmetic problem in generate the key.

I.2 RSA

RSA algorithm is one of the most popular public key algorithms. RSA algorithm was created by 3 researchers in MIT (Massachussets Institute of Technology) in 1976,

they are Ron (R)ivest, Adi(S)hamir, and Leonard (A)dleman.

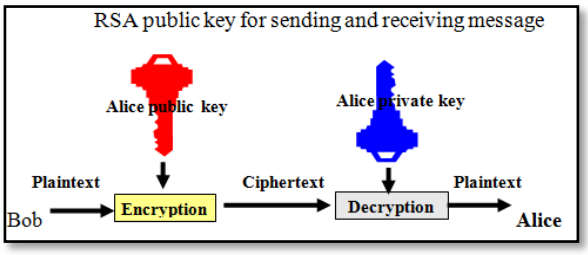

Figure 2. RSA public key encryption and decryption

Parameters used in RSA algorithm are

No	Parameters	Privilege
1	p and q (prime number)	Secret
2	n = p . q	Not secret
3	Φ(n) = (p-1).(q-1)	Secret
4	e (public key)	Not secret
5	d (private key)	Secret
6	m (plaintext)	Secret
7	c (ciphertext)	Not secret

Essentially, the public key is the product of two randomly selected large prime numbers 'p' and 'q', and the secret key is the two primes themselves. The algorithm encrypts data using the product, and decrypts it with the two primes, and vice versa. A mathematical description of the encryption and decryption expressions is shown below:

Encryption: $c=m^e \pmod n$
Decryption: $m=c^d \pmod n$

RSA algorithm works with these steps :

1. Choose prime numbers p and q
2. Find their product n = pq
3. Calculate phi(n) = (p-1).(q-1)
4. Select an integer "e" in which the gcd(e,phi(n)) = 1
5. Calculate d such that e*d = 1 mod (phi(n))
6. The public key is (e,n)
7. The private key is (d,n)
8. Plaintext can be any number m, where m < n, and neither p nor q divides m
9. The ciphertext is $C = M^e \pmod n$
10. The plaintext is $C^d = M^{ed} \pmod n$

The security of the RSA cryptosystem depends on the difficulty of factoring n. It is currently difficult to obtain the private key'd' from the public key (n, e). However if one could factor n into p and q, then one could obtain the private key'd'. If a method is discovered for factoring arbitrary integers quickly, then any RSA private key could be discovered and the system would become insecure.

Factoring n: The fastest known factoring algorithm developed by *Pollard* is the ***General Number Field Sieve***, which has running time for factoring a large number of size n, of order

$$O\left(\exp\left(\left(\frac{64}{9}\log n\right)^{\frac{1}{3}}(\log\log n)^{\frac{2}{3}}\right)\right)$$

The method relies upon the observation that if integers x and y are such that $x \neq y \pmod n$ and $x^2 = y^2 \pmod n$ then gcd(x − y, n) and gcd(x+y, n) are non-trivial factors of n.

The following table gives the number of operations needed to factor n with GNFS method, and the time required if each operation uses one microsecond, for various lengths of the number n (in decimal digits)

Digits	Number of Operation	Time
100	9.6× *108*	16 minutes
200	3.3 × *1012*	38 days
300	1.3 × *1015*	41 years
400	1.7 × *1017*	5313 years
500	1.1 × *1019*	3.5 × *105* years
1024	1.3 × *1026*	4.2 × *1012* years
2048	1.5 × *1035*	4.9 × *1021* years

From the table given above we can see that with the fastest know factoring algorithm, it still need a lot amount of time to factoring n. If no new method or approach to solve this problem faster, then RSA will always be safe.

I.3 BREW Mobile Platform

BREW (Binary Runtime Environment for Wireless) is an application development platform created by Qualcomm, originally for CDMA mobile phones, but GSM is now also supported. BREW is first debuted in September 2001.

The main advantage of BREW platforms is that the application can easily port between all Qualcomm devices. BREW acts between the application and the wireless device on-chip operating system in order to allow programmers to develop applications without needing to code for system interface or understand wireless application.

Developers can develop an application on BREW mobile platform with C or C++ language. Java is also

supported if the handset has a Java Virtual Machine available.

II. PUBLIC KEY CRYPTOGRAPHY IN BREW

Nowadays, mobile devices are very common. That's why we need to ensure the security for information transaction over mobile devices. Cryptography is one approach to ensure security, especially public key cryptography.

BREW as one of large mobile platform also has provided its developers with some cryptography algorithms API in "AEESecurity.h", "AEERSA.h",and many others.

The algorithms are categorized by its interface.

ICipher interface:

- Block3DES, BlockDES
- BlockAES128, BlockAES192, BlockAES256
- StreamCipher (CBC,CFB,CTR,OFB)
- StreamARC4

IPubKey interface:

- RSA
- ECC

IPubKeyUtil interface:

- SHA1, SHA256, SHA384, SHA512

This paper will only implement RSA algorithm through IRSA interface. IRSA interface provides access to the RSA public key cryptographic algorithm and the basic underlying modular exponentiation . It also provides some of the padding and encoding commonly used with the RSA algorithm. Last it is modestly useful for managing memory in which keys are stored. But, RSA implementation in BREW of course have some lack because it run on very limited resource of memory, so here are a few notes on what is absent. There is no key generation because it will be extremely slow in software, but might be acceptable on a DSP. Also there is no way to retrieve a key once stored. Some of the modern padding types, OEAP and PSS in particular, are not available. There is also no way to store a full private/public key pair to enable faster private key operations using the Chinese Remainder Theorem.

There are some methods will be used, such as IRSA_Decrypt() and IRSA_Encrypt(). Below is the prototype of function that will be used in next section.

```
int IRSA_Init
(
   IRSA* pIRSA,
   const byte* pchModulus,
   int cbModulus,
   const byte* pchExponent,
   int cbExponent
)
```

Parameters

pIRSA	: Pointer to IRSA interface object
pchModulus	: [in] Pointer to modulus
cbModulus	: [in] Modulus size in bytes
pchExponent	: [in] Pointer to public or private exponent
chExponent	: [in] Public or Private exponent size in bytes

Return Value

SUCCESS: RSA key initialization successful.

AEE_CRYPT_INVALID_KEY: Key NULL or zero length

ENOMEMORY: Storage for the key could not be allocated

```
void IRSA_Decrypt
(
   IRSA* pIRSA,
   const byte* pbIn,
   int cbIn,
   byte* pbOut,
   int *pcbOut,
   int ePadType,
   uint32* pdwResult,
   AEECallback *pCB
)
```

```
Void IRSA_Encrypt
(
   IRSA* pIRSA,
   const byte* pbIn,
   int cbIn,
   byte* pbOut,
   int *pcbOut,
   int ePadType,
   uint32* pdwResult,
   AEECallback *pCB
)
```

Parameters

pIRSA	: Pointer to the IRSA interface
pbIn	: [in] pointer to buffer of data to decrypt
cbIn	: [in] length of input buffer
pbOut	: [out] pointer to buffer to store encrpypted data, caller allocated
pcbOut	: [in/out] length of output buffer on input, length of data on output
ePadType	: [in] type of padding to use
pdwResult	: [out] result code
pCB	: [in] completion callback

Below is the snippet code for using this API to encrypting a message with RSA algorithm and decrypting it back.

1. Include statement

   We must list all of header library used in the program, because program can only run with the listed library. In this section, we also need to add include to file MIF (Module Information File) and also BRH (BREW Resource Header) if we use external resource in our BREW application.

```
/*=============================================================
INCLUDES AND VARIABLE DEFINITIONS
=============================================================*/
#include "AEEModGen.h"    // Module interface definitions
#include "AEEAppGen.h"    // Applet interface definitions
#include "AEEStdLib.h"    // standard library
#include "AEEText.h"      // library for text control
#include "AEERSA.h"       // library for RSA class

#include "brewapp.bid"    // link ke brewapp.mif (informasi program)
#include "brewapp.brh"    // link ke brewapp.brh (header resource)
```

2. Applet structure

We should declare all the control we need in this applet structure. Every attribute on application will be referenced as static since only one application can run. AEEApplet, AEEDeviceInfo is a must for every application. In this section, I have added some more attribute, there are Menu control, text control, and also RSA interface attribute.

```
typedef struct _brewapp {
    AEEApplet       a ;
    AEEDeviceInfo   DeviceInfo;
    AECHAR          UniCodeString[50];
    IMenuCtl        *pIMenu;
    ITextCtl        *pPlaintext;
    ITextCtl        *pCiphertext;
    IRSA            *pRSA;
} brewapp;
```

3. Initiation program

In this part we create every instance we need. I have added a menu control, two text control,and an instance of RSA interface.

```
boolean brewapp_InitAppData(brewapp* pMe)
{
    byte *pMod = (byte *) MALLOC(8 * sizeof(byte));
    pMod[0] = 1; pMod[1] = 1; pMod[2] = 1; pMod[3] = 1;
    pMod[4] = 1; pMod[5] = 1; pMod[6] = 1; pMod[7] = 1;

    pMe->DeviceInfo.wStructSize = sizeof(pMe->DeviceInfo);
    ISHELL_GetDeviceInfo(pMe->a.m_pIShell,&pMe->DeviceInfo);

    //menambahkan menu control
    if (ISHELL_CreateInstance(pMe->a.m_pIShell, AEECLSID_SOFTKEYCTL,
        (void **)&pMe->pIMenu) != SUCCESS) {
        pMe->pIMenu = NULL;
        return (FALSE);
    }
    //menambahkan text control
    if (ISHELL_CreateInstance(pMe->a.m_pIShell, AEECLSID_TEXTCTL,
        (void **)&pMe->pPlaintext) != SUCCESS) {
        pMe->pPlaintext = NULL;
        return (FALSE);
    }
    if (ISHELL_CreateInstance(pMe->a.m_pIShell, AEECLSID_TEXTCTL,
        (void **)&pMe->pCiphertext) != SUCCESS) {
        pMe->pCiphertext = NULL;
        return (FALSE);
    }
    //menambahkan kelas RSA
    if (ISHELL_CreateInstance(pMe->a.m_pIShell, AEECLSID_RSA,
        (void **)&pMe->pRSA) != SUCCESS) {
        pMe->pRSA = NULL;
        return (FALSE);
    }
    else {
        IRSA_Init(pMe->pRSA,pMod,8,pMod,8);
    }
    return TRUE;
}
```

4. Free

Mobile platform have limited resource, that's why every single byte of memory is so precious. So,free the memory binded to every control is a must for developer.

```
void brewapp_FreeAppData(brewapp* pMe)
{
    //1. menu
    if ( pMe->pIMenu != NULL ) {
        IMENUCTL_Release(pMe->pIMenu);
        pMe->pIMenu = NULL;
    }
    //2. textEncrypt
    if (pMe->pPlaintext != NULL ) {
        ITEXTCTL_Release(pMe->pPlaintext);
        pMe->pPlaintext = NULL;
    }
    //3. textDecrypt
    if (pMe->pCiphertext != NULL ) {
        ITEXTCTL_Release(pMe->pCiphertext);
        pMe->pCiphertext = NULL;
    }
    //4. RSA
    if (pMe->pRSA != NULL ) {
        IRSA_Release(pMe->pRSA);
        pMe->pRSA = NULL;
    }
}
```

5. Encrypt

This method is the implementation RSA algorithm to encrypt a message. In this case we get the text in the plaintext and show the encrypted text in the cipher text

```
void Encrypt(brewapp *pMe) {
    //we call IRSA_Encrypt function
    IRSA_Encrypt(pMe->pRSA,
        ITEXTCTL_GetTextPtr(pMe->pPlaintext),250,
        ITEXTCTL_GetTextPtr(pMe->pCiphertext),250);
}
```

6. Decrypt

This method is the implementation RSA algorithm to encrypt a message

```
void Decrypt(brewapp *pMe) {
    //we call IRSA_Decrypt function
    IRSA_Decrypt(pMe->pRSA,
        ITEXTCTL_GetTextPtr(pMe->pPlaintext),250,
        ITEXTCTL_GetTextPtr(pMe->pCiphertext),250);
}
```

Actually there were some important steps to get our BREW application run :

1. Specify MIF from MIF editor
2. Specify resource file with Resource Editor
3. Display the menu
4. Setting some development environment for BREW application

But, I only placed some part of the implementation code to keep focus only in the implementation RSA in BREW.

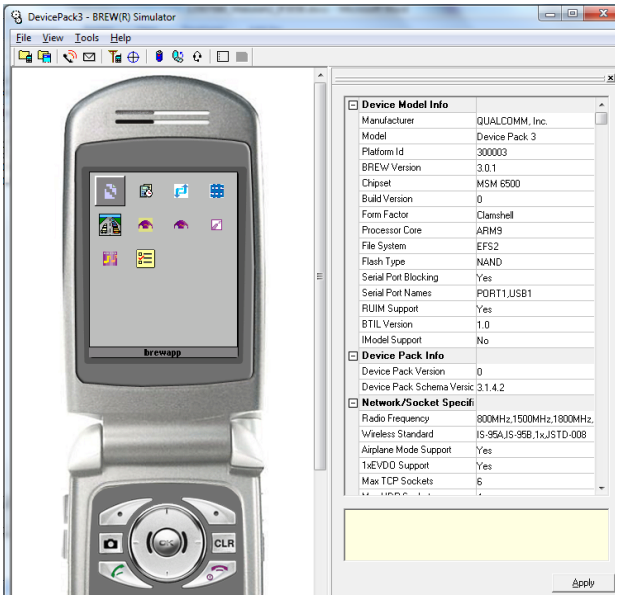

Figure 3. BREW Simulator

Figure 4. Program interface

III. CODE PRIVILEGE LIMITATION

BREW provide the ability to limit the privilege of an application to the telephone functionality. Privilege of an application can be set from the MIF (Module Information File). Some privileges in BREW are:

1. Access file
2. Access network
3. Web access
4. TAPI (Telephone API)
5. Position location
6. Access to ringer directory
7. Write access to shared directory
8. Access to sector information
9. Access to address book
10. Download*
11. All (system)*

Some privilige marked with asterisk (*) which means these kinds of privileges are not supplied for common application because selecting these privilege levels may cause the application to fail True BREW Testing which is the requirement for an application to be placed in QIS (Qualcomm Internet Services) where end users can download their application. So, a developer must grant special permission to use some of BREW privilege, so it is hoped can ensure better security.

IV. CODE AUTHORIZATION

Code authorization is achieved by implements digital signing of dynamic module on BREW-enabled devices. There are two types of keys and certificates necessary:

1. Root keys and certificates
2. Signing keys and certificates

Digital signing in BREW is based upon public key (or asymmetric) cryptography. Public key cryptography use public and private key that with adequate key lengths, one cannot deduce the private key from knowledge of its associated public key, a signature and the digital object which was signed. This is the basis of non-repudiation.

BREW Key Generation use standards X509 certificate hierarchy, formats, and algorithms. It use RSA key algorithm, modulus up to 4096 bits but for reasonable performance the maximum practical keys for today's (2009) hardware are modulus 2048 bits, public exponent 3.

BREW supports some certificate signing algorithms:

1. RSA with SHA1 and PKCS1 padding
2. RSA with SHA256 and PKCS1 padding
3. RSA with SHA384 and PKCS1 padding
4. RSA with SHA512 and PKCS1 padding

The process of digital signing in BREW will through some steps :

1. Certificate Authority provides device manufacturer with root certificate
2. Device manufacturer immutably configures root certificate into device image
3. Certificate authority issues signing certificate(s) to signing authority
4. Developers submit unsigned code to signing authorities
5. Signing authority issue digital signatures for code which meet the criteria set forth in their signing policy

Those steps are shown in diagram below,

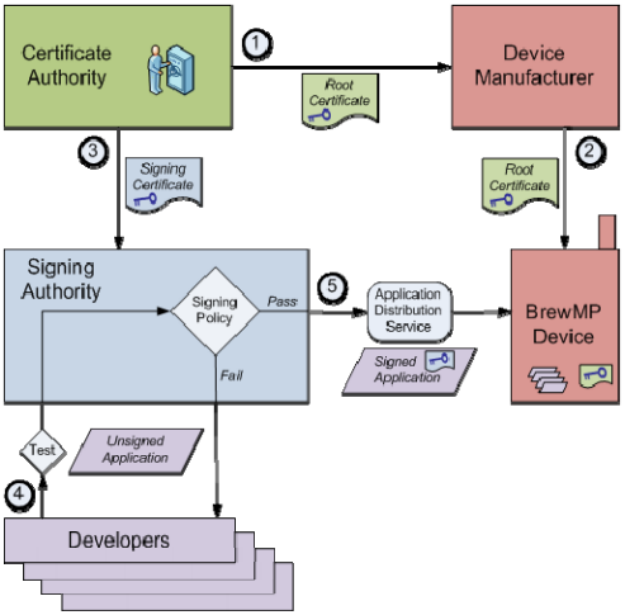

Figure 5. Digital Signing Ecosystem

Below is an example of certification for BREW application :

Field	Value
Version	CA_DEFINED
Serial Number	CA_DEFINED
Signature Algorithm	sha256WithRSAEncryption
Issuer	o=CA_NAME cn=CA_NAME Signing root
Validity	notBefore = GENERATION_DATE notAfter = 20_YEARS_LATER
Subject	o=CA_NAME cn=CA_NAME Code Signing root
Public Key Info	Algorithm = RSA Modulus = 2048 bits Exponent = 3(F0)
Extensions : basic constraint **OID {id-ce 19}** **Criticality : TRUE**	cA = TRUE pathLenConstraint = 2

V. CONCLUSION

After the explanation above, we have seen that the need of cryptographic for securing our information have been so important for mobile devices. BREW mobile platform have provided some security and cryptograhic feature, such as :

1. Cryptographic library (symmetric and asymmetric key cryptographic)
2. Code privilege limitation
3. Code authoritation

BREW also has changed their regulation since 2006 that every developers need to pay a great amount of money to get access to digital sign, so only serious developers that want to bring their application to the market will sign their application. Each application also need to pass TBT (TRUE BREW Testing) that ensure that the application doesn't contain security risk in it.

But, there is still weakness point in this system, for example RSA algorithm used in mobile devices is more simple than in general use because of its limited memory so basicly it will reduce the robustness of the cryptography algorithm.

REFERENCES

[1] https://BREWmobileplatform.qualcomm.com/devnet/prod/resources/devEx/library/techguides/CSK/Code_Auth_thru_Digital_Signing/Code_Auth_thru_Digital_Signing.pdf. Code Authorization on BREW MP through Digital Signing. Accessed on 28th April 2010.

[2] http://BREW.qualcomm.com/BREW/en/developer/faq/business_faq.htm. Accessed on 17th May 2010.

[3] BREW API Reference. Qualcomm, 2009.

[4] http://garsia.math.yorku.ca/~zabrocki/math5020f05/RSA.doc. RSA Cryptography. Accessed on 16th Mei 2010.

[5] http://itslab.csce.kyushu-u.ac.jp/iwap04/invited_tanaka.pdf. Current topics on Mobile PKI. Accessed on 28th April 2010.

[6] Munir,Rinaldi. Diktat Kriptografi. Bandung,2006.

PERNYATAAN

Dengan ini saya menyatakan bahwa makalah yang saya tulis ini adalah tulisan saya sendiri, bukan saduran, atau terjemahan dari makalah orang lain, dan bukan plagiasi.

Bandung, 17 Mei 2010

Samsu Sempena
NIM. 13507088