2002 International Computer Symposium (ICS2002)
Workshop on Cryptology and Information Security

Title : Java Card Cryptography Design

Author : Mr. Tseng Shau-Yin,
Affiliation: Industrial Technology Research Institute
Computer & Communications Research laboratories
Address: X300 Bldg. 51, 195-11, Sec. 4, Chung Hsing Rd., Chutung, Hsinchu, 310, Taiwan.
Tel: 03-591-5670 Fax: 03-582-0234
e-mail: tseng@itri.org.tw

Ms. Lin Hui-Ching
Affiliation: Industrial Technology Research Institute
Computer & Communications Research laboratories
Address: X300 Bldg. 51, 195-11, Sec. 4, Chung Hsing Rd., Chutung, Hsinchu, 310, Taiwan.
Tel: 03-591-5635 Fax: 03-582-0234
e-mail: **shirleylin@itri.org.tw**

Contact : Ms. Lin Hui-Ching

Keywords : Java card, Java Card security API, Visa Open Platform security API

Abstract :

In smart card application, the most import issue is security. Open platform is a common specification in this issue. But, how do security and cryptography mechanism work in Java card applet, is not wire known. In this paper, we'll have an integral description about the Visa's Open Platform APIs relationship and how to implement these APIs. Meanwhile, we'll have an overall introduction about how to apply these APIs. These subjects are rarely revealed.

Java Card Cryptography Design

1. Introduction

In recent time, Smart card has become more and more popular, and security issues are critical for the success of smart-card application. Java Card, a kind of smart card, provides cryptography mechanisms and enforces firewalls to protect applications and maintain data and operation security. Visa's Open Platform (OP or VOP) defines the secure procedure of smart card application.

Our Jixco Java Card based Java Card 2.1.1 platform and contains VOP 2.0.1 cryptographic functionality. Therefore, both the Javacard Runtime Environment (JCRE) and the OP and VOP standards contribute to the security features of the Jixco card.

Basically, all the cryptographic functionality are defined by the Java card 2.1 specifications, and VOP only defines the cryptographic functionality that required and usable by an application developer. In the other word, while not an extensible interface within the Java Card 2.1 specifications, VOP does not mandate the presence of the shareable interface.

In this paper, we do not explain cryptographic functions that do not defined in both Java Card 2.1 and VOP 2.0.1.

Our Jixco Java Card Cryptography follows VOP's specification to implements Java Card API2.1 functions. We will describe those algorithms in the following

sections.

2. Jixco Java Card Cryptographic Overview

Jixco cryptographic structure can separate into two parts: one is javacard.security package, another is javacardx.crypto package. For a VOP card the full set of classes shall be present and the presence of the interfaces depends on the features of the card.

2.1 javacard.security package

The package javacard.security defines a set of class and interface for the Java Card security framework. Jixco provides following algorithm.

(1) CryptoException class

(2) DESKey interface

(3) Key interface

(4) KeyBuilder class

- Length_DES
- Length_DES3_2Key
- Length_DES3_3Key
- Type_DES
- Length_RSA_1024
- Type RSA_Crt_Private
- Type RSA_Private

- Type RSA_Public

(5) MessageDigest class

- ALG_SHA

(6) PrivateKey interface

(7) PublicKey interface

(8) RSAPrivateKey interface

(9) RSAPrivateCrtKey interface

(10) RSAPublicKey interface

(11) RandomDara class

(12) Signature class

- Alg_DES_MAC8_ISO9797_M1
- Alg_DES_MAC8_ISO9797_M2
- Alg_DES_MAC8_Nopad
- Mode_SIGN
- MODE_VERIFY
- Alg_RSA_SHA_ISO9796
- Alg_RSA_SHA_PCKS1

2.2 javacardx.crypto package

This package is an extension package, which contains security classes and

interfaces for export-controlled functionality. Jixco provides following algorithm.

(1) Cipher Class

- Alg_DES_CBC_ISO9797_M1
- Alg_DES_CBC_ISO9797_M2
- Alg_DES_CBC_Nopad
- Alg_DES_ECB_ISO9797_M1
- Alg_DES_ECB_ISO9797_M2
- Alg_DES_ECB_Nopad
- Mode_Decrypt
- Mode_Encrypt
- Alg_RSA_PCKS1
- Alg_RSA_ISO9796
- Alg_RSA_Nopad

2.3 Jixco Implements architecture

To implement these functions, we separate these functions into few modules by coprocessor algorithm, padding way, data encryption modes, and key generators.

(1) coprocessor algorithm: this group contains RSA and DES algorithm. DES algorithm also includes 3DES function. Since RSA coprocessor spends much times to make numeric calculate, we implement it by hardware to increase

the carculation rate.

(2) padding way: there are five padding ways required by VOP: ISO9797_M1, ISO9797_M2, ISO9796, PCKS1, and Nopad. Some of these algorithms used in javacard.security, the others in javacardx.crypto. Therefore we design a ccl.impl.Padding class to implement all padding way.

(3) data encryption modes: VOP's specific only requests CBC and ECB encryption mode, hence we include these two mode in ccl.impl.Mode class

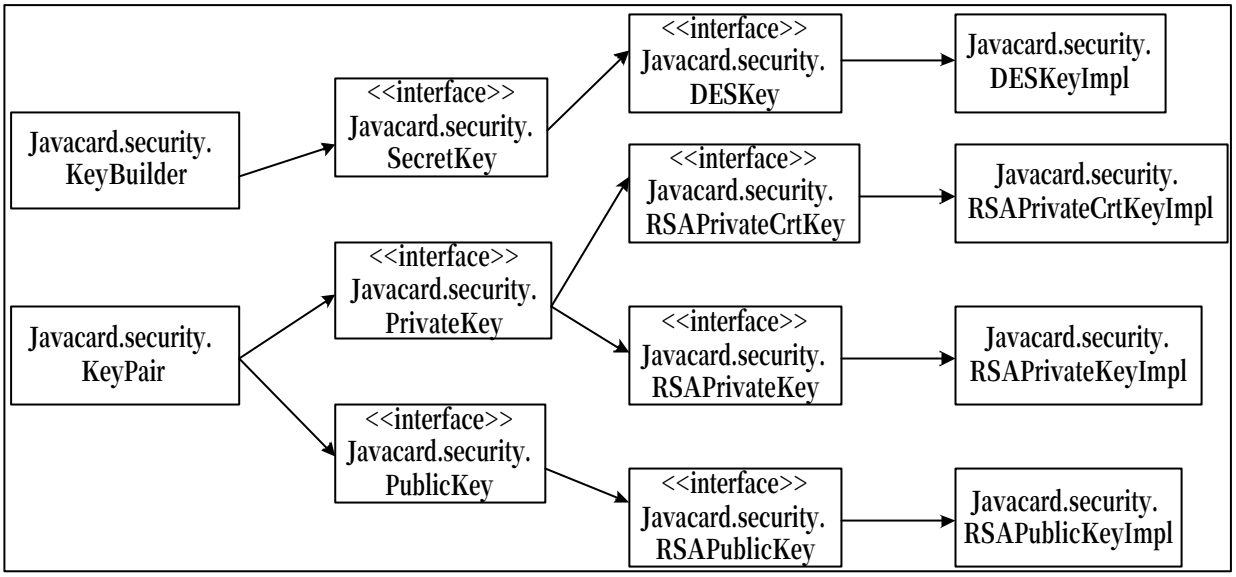
(4) key generators: all DES key and RSA key are defined at interface type, Herewith, we have classes to implement these interface one to one.

Not all of above four modules can be directly invoked by application developer, some modules only invoked by other modules. Only those modules that defined by JavaCard API 2.1 can be directly invoked. In the following sections, we'll describe the relation between each module.

3. Jixco Java Card Cryptographic Functionality

3.1 Key Module

Following fig is the calling graph of key generator.



Since all the key modules are defined as interface, the key builder will return the object that we implement.

Here we have a sample code:

```
Public static Key buildKey( byte keyType,
                            short keyLength,
                            boolean keyEncryption ) throws CryptoException
{
        Key theKey=null;
         switch ( keyType ) {
                  case TYPE_DES:
                        theKey=new DESKeyImpl();  // return the implement object
                        break;
        }
}
```

The application developer invokes buildKey(…) method to build a DES key (so as RSA key), then the developer will get a DESKeyImpl object. But developer does not know the DESKeyImpl type, he needs to cast this object to DESKey type (DESKey is an interface). Following is a sample code for application

developer reference.

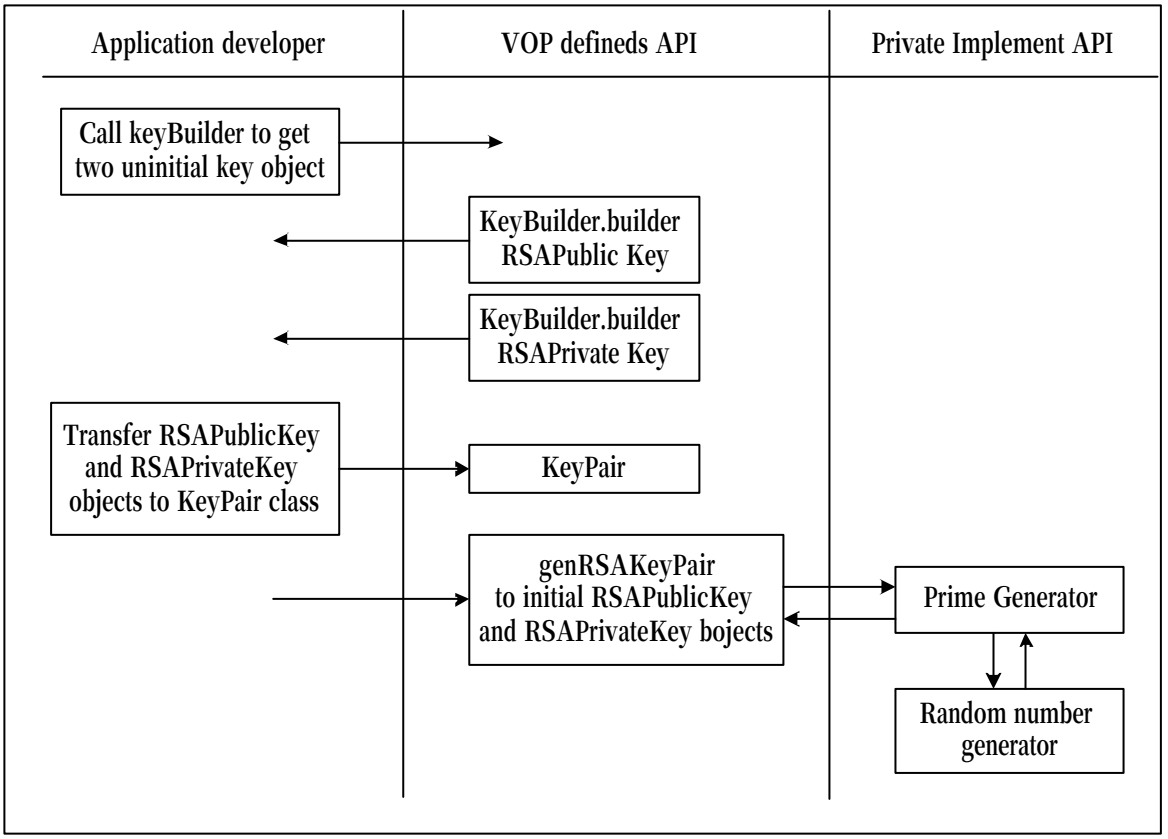
```
DESKey appleyKey;

appletKey = (DESKey) KeyBuilder . buildKey( KeyBuilder.TYPE_DES,
                KeyBuilder.LENGTH_DES , false);

appletKey . setKey ( keyVal , (byte) 0 ) ;

/* keyVal is a byte array, that contain the initial value of the key, which may come
from terminal or generate by card itself depended by application developer */
```

RSA key is more complicated than DES key. To generate RSA key needs prime generator that needs random number generator to support. But the first step is to get a RSAPublicKey object(or RSAPrivateKey object), then initial this key object with your initial value. The initial value is come from random number generator with serials of prime testing ; these process handle by ccl.impl.primegenerator. The calling graphs as follow:



RSA key is a pair key, therefor, we call KeyBuilder class to get the key object individually. Then call KeyPair class to put these two key objects. But, those two keys are not initialized. After that invoking genKeyPair(…) method to set the key value into key objects. Follows we have some sample code.

```
RSAPublicKey rsaPublicKey;
RSAPrivateKey rsaPrivateKey;

rsaPublicKey = (RSAPublicKey) KeyBuilder.buildKey
                        (KeyBuilder.TYPE_RSA_PUBLIC ,
                        KeyBuilder.LENGTH_RSA_1024 , false);
rsaPrivateKey = (RSAPrivateKey) KeyBuilder.buildKey
                        (KeyBuilder.TYPE_RSA_PRIVATE ,
                        KeyBuilder.LENGTH_RSA_1024 , false);

KeyPair keyPair= new KeyPair ( rsaPublicKey, rsaPrivateKey);
keyPair.genKeyPair();
```

At above sample code, keyPair.genKeyPair() method will initial the key. Then we can use the public and private key. Like follows code.

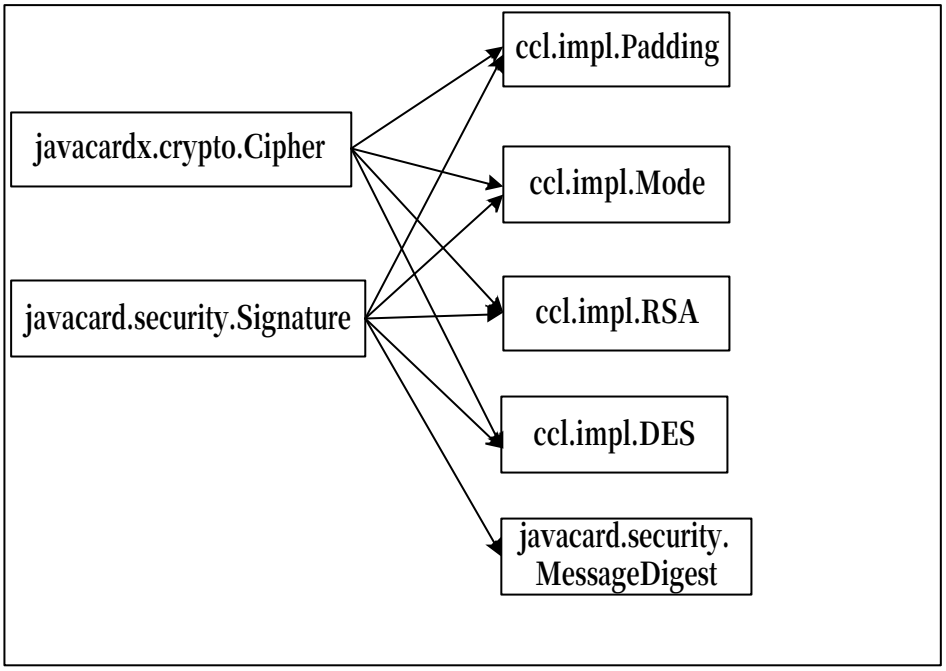
```
rsaPrivateKey=(RSAPrivateKey) keyPair.getPrivate();
rsaPublicKey=( RSAPublicKey) keyPair.getPublic();
```

3.2 Signature and Cipher

The method Signature(…) and Cipher(…) lets the smart card generate a crypt signal.

As our previous desiccation, we can get a key pair, and use it to sign data. When sign data, there are two algorithms: RSA and DES, hash function, and padding ways. On Jixco design, we breakup few classes to implement these functions. Following fig is the calling graph of Signature and Cipher function.

The classes (ccl.impl.*) are invoked by up-layer functions.



In our design if the signature algorithm is ALG_DES_MAC8_NoPad , we call ccl.impl.Mode.Mac8(…), ccl.impl.Padding.NoPad(…), and ccl.impl.DES(…) in our implementation. Herewith, the application developer does not know the exiting of ccl.impl.* class, and apply those classes.

4. Conclusion

In Jixco cryptography design, all functions are extended from Java Card API2.1, and followed VOP2.0.1 requests, which can full, conjoin into Java Card API2.1. Expect RSA kernel algorithm which build in Jixco's hardware, others need to converter into Java card bytecode (.jca) and download with JCRE.

Our Jixco cryptography function do not has an API which is out of VOP's specification, and all API definitions can be find in Java Card API2.1, hence it's very

easy to apply by every application developer.

Reference

[1] Uwe Hansmann, Martin S. Nicklous, Thomas Schack, Frank Seliger, “Smart Card Application Development Using Java”, Springer.

[2] Zhiqun Chen, “Java Card ™ Technology for smart cards: Architecture and Programmer’s Guide”, First Edition, Addison-Wesley, June 2000.

[3] “ GemXPresso 211PK Card Reference Manual”, GEMPLUS 2000.

[4] “Open Platform , Card Java Card Applet Compliance”, Global Platform, April 2000.

[5] “Java Card ™ 2.1.1 Application Programming Interface” Sun Microsystems, Inc., May 2000.