ESES/*j*-Crypto and its application

Joo-Young Lee[*], *Ju-Han Kim*[*], *Jung-Chan Na*[**], *and Seung-Won Sohn*[***]

EC Information Security Team, ETRI[*]
Active Protocol Research Team, ETRI[**]
Network Security Department, ETRI[***]

ABSTRACT

Recently e-businesses have been spreading and many existing firms have been changing to take advantages of e-commerce model. However, most e-commerce site visitors are reluctant to shop online due to security concerns. Therefore trust and control of security are imperative for participating in e-business and critical to its success. In order to provide security and cryptographic services such as digital signature, encryption and message digest to web application developers, the Java platform provides two types of APIs(Application Programming Interfaces): the Java Cryptography Architecture(JCA) and the Java Cryptography Extension(JCE). In this paper, we'll introduce ESES/*j*-Crypto that is developed as a subsystem of ESES(ETRI Secure E-commerce Service) system for supporting secure electronic commerce. ESES/*j*-Crypto is compatible with the JCA and the JCE 1.2 and provides a cryptographic basis to ESES/Signature and ESES/Cipher that are another subsystems of ESES. It consists of two software components: one is implementation of the JCE specification that defines and supports cryptographic services and the other is cryptographic service provider, so called the CSP. The ESES CSP includes not only generally used cryptographic algorithms, such as DSA, RSA, DES, and SHA1, but also algorithms adopted as domestic standard, such as SEED, KCDSA. It can be associated with other packages well and we can add new algorithms to it easily. This paper will explain how ESES/*j*-Crypto processes to offer security services and present a simple application that works for a cipher for an XML document. Finally we'll conclude our talk by presenting further works.

1. INTRODUCTION

Recently e-businesses are spreading and many existing firms are changing to take advantages of e-commerce model. However, according to the American Internet User Survey by Cyber Dialogue, most e-commerce site visitors are reluctant to shop online due to security concerns[1][2]. Therefore trust and control of security are imperative for participating in e-business and critical to its success. Customers must be reasonably sure that private information, such as credit card numbers and medical data, reaches their destination, in most cases, the e-business's web server, without the possibility of a third party intercepting the message. Customers and merchants both must be confident that the person on the other side of the network is whom they claim to be. This is known as authentication. Finally, the e-business must guard against "hackers" stealing information from internal networks or preventing legitimate users from accessing online services, which is known as denial of service.

In order to provide security and cryptographic services such as digital signature, encryption and message digest to web application developers, the Java platform provides two types of APIs(Application Programming Interfaces). One is the Java Cryptography Architecture(JCA) and the other is Java Cryptography Extension(JCE). In this paper, we'll introduce ESES/*j*-Crypto that is developed as a subsystem of ESES(ETRI Secure E-commerce Service) system for supporting secure electronic commerce. ESES/*j*-Crypto is compatible with the JCA and the JCE 1.2 and provides a cryptographic basis to ESES/Signature and ESES/Cipher that are another subsystems of ESES. This paper will explain how ESES/*j*-Crypto processes to offer security services and present a simple application that works for a cipher for an XML document. Finally we'll conclude our talk by presenting further works.

2. CRYPTOGRAPHIC API AND SPI

As we described in previous section, the Java platform provides two APIs for dealing with security and cryptographic services. The first is the JCA and provides a framework for basic security functions such as a certificate, a digital signature, and a message digest[3]. The second is the JCE and extends the JCA to provide an encryption, a key agreement protocol, a key generation and a message authentication code[4]. Fig. 1 shows Java security package. We can show that the JCA encompasses classes included in the JDK1.2 core as well as extensions from the JCE[5].

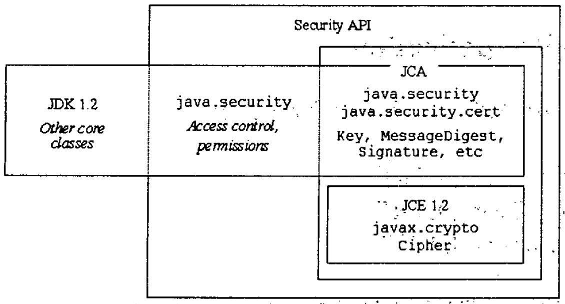


Fig 1. Java security package.

The JCA and the JCE have been designed around following principles: implementation independence, implementation interoperability, algorithm independence, and algorithm extensibility[3][4]. Implementation independence and algorithm independence intend to let users of the API utilize cryptographic concepts without concern for codes or algorithms being used to implement these concepts. Implementation interoperability means that various implementations can work with each other, and algorithm extensibility means that new algorithms that fit in one of the supported engine classes can easily be added.

The JCA and the JCE introduce the notion of a Cryptographic Service Provider(CSP)[6]. This term refers to a package or a set of packages that supplies a concrete implementation of a subset of the cryptography aspects of the JDK security API.

Sun Microsystems has come up with a design that separates APIs from SPIs(Service Provider Interfaces). Here, an API, sometimes called an engine class, defines a cryptographic service in an abstract fashion without a concrete implementation, so it provides an interface to the functionality of a specific type of cryptographic service independent of a particular cryptographic algorithm. On the other side, an SPI is an interface for service providers who want to offer cryptographic algorithms and security services and it consists of abstract classes. The concrete classes implemented by service providers must inherit from abstract classes of the SPI and are connected with the API classes through these SPI classes. This architecture gives an advantage to application programmers and service providers both. An application programmer who needs to call a method, i.e., init(), wants as many API selections which have same functionality but various formats as possible. However, the service provider has difficulty in satisfying the programmer's desire. The architecture of the separated API from the SPI can reduce this gap. Each API class has its corresponding SPI class. For example, according to the JCE specification, a Cipher class corresponds to a CipherSpi class. The Cipher class has six doFinal() API methods which encrypt or decrypt data in a single-part operation, or finish a multiple-part operation, and the CipherSpi class has only two engineDoFinal() methods that are called by doFinal() method of the Cipher class. Therefore the application programmer can choose one among the six API methods, on the other side, the service provider needs to provide the two SPI methods to satisfy user's desire.

Fig. 2 depicts how the API is associated with the SPI with a cipher example. An application programmer wants to implement a cipher function using a Cipher class. First, he/she has to request an instance of the class by calling a getInstance() method of the Cipher class. This method makes a security manager, which is also a class, search a provider list already registered, find a cryptographic algorithm he/she wants, and return its instance object. Next he/she calls a init() to initialize the cipher instance, it brings about calling a engineInit() method of the CipherSpi class. This engineInit() method invokes a new method, here, initialize() of DES class, which extends it and plays a role of initializing the cryptographic algorithm according to provider's implementation. A update() and a doFinal() methods of the Cipher class are processed like the init() method.

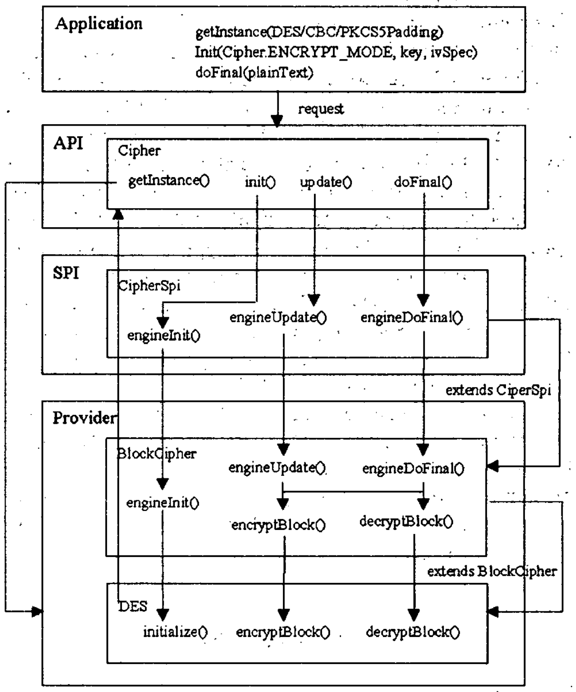


Fig. 2 How the API is associated with the SPI with a cipher example.

3. THE DESIGN OF ESES/*j*-Crypto

3.1. Components of ESES/*j*-Crypto

ESES/*j*-Crypto includes two software components: one is implementation of the JCE specification that defines and supports cryptographic services and the other is cryptographic service provider, so called the CSP. We implemented the JCE specification because the US government regards some cryptographic algorithms as a kind of weapon and prohibits them from exporting outside the US and Canada (Currently, JCE 1.2.1 beta version is exportable as a binary format with a jurisdiction policy file, which contains maximum allowable cryptography strength defined by the US law). Although several CSPs have been provided, we need to develop our package, which includes not only generally used algorithms, but also domestic standard algorithms. It can be associated with other CSPs well and we are able to add new algorithms to it easily.

Fig.3 shows components of ESES/*j*-Crypto and their relationship. ESES/*j*-Crypto consists of a cipher function block, a key generation block, a key & key spec block, a parameter spec block, and an ESES CSP that is concrete implementations of cryptographic algorithms and materials. Here, a random number generation block, which is a JCA component, is excluded. We only include it into diagram to explain the relation of other components more clearly.

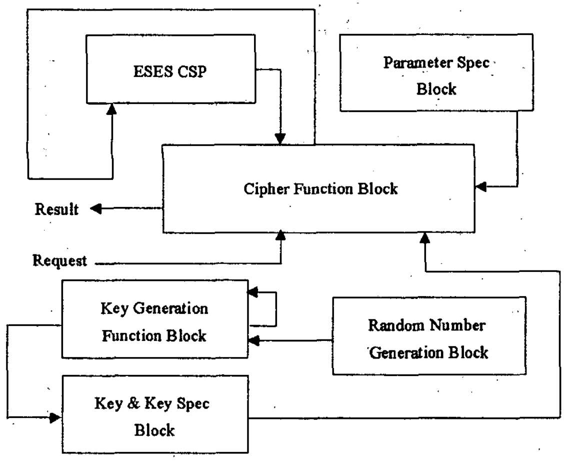


Fig. 3 Components of ESES/*j*-Crypto and their relation.

The cipher function block consists of classes that provide the functionality of encrypting or decrypting some specified data, such as a Cipher class, a CipherSpi class, a CipherInputStream class, a CipherOutputStream class, and a SealedObject class. The key generation function block provides a way to use a symmetric key, sometimes called secret key. The key can be created using a KeyGenerator class and a KeyGeneratorSpi class or be translated from byte array data, which is stored on disk or is transmitted over a network, using a SecretKeyFactory class and a SecretKeyFactorySpi class. The generated key is returned as an object format of a Key interface or a Key specification class included in the key & key spec block. The parameter spec block holds some materials and information used in cryptographic algorithms. In the addition, there are a KeyAgreement class that used to execute a key agreement (key exchange) protocol among two or more parties and a MAC class that used to compute the message authentication code of some specified data.

We have implemented a provider named ESES used by applications to refer to our CSP. The ESES CSP supplies concrete implementations of cryptographic services, such as a signature, a massage digest, a key pair generation, a cipher, a key agreement, defined in the JCA and the JCE 1.2. We have implemented them as the subclasses of the appropriate SPI classes. It contains symmetric and asymmetric cipher algorithms, hash algorithms, digital signature algorithms, and message authentication code algorithms. The supported security services and the cryptographic algorithms by the ESES/*j*-Crypto are listed in Table 1. We have selected and implemented algorithms, which used in the XML signature subsystem and the XML cipher subsystem of the ESES.

Table. 1 The supported security services and cryptographic algorithms by the ESES/*j*-Crypto.

Security Services	Cryptographic Algorithms
Message Digest	MD5, SHA1, HAS160
Block Cipher	DES, DESede, SEED
Cipher (Asymmetric)	RSA, ElGamal
Signature	DSA, ElGamal Signature, KCDSA
MAC	HMACwithMD5, HMACwithSHA1

3.2. The processing flow of ESES/*j*-Crypto

In this section, we'll explain how ESES/*j*-Crypto to be processed internally when an application programmer uses a specific cryptogrphic service. In order to do this, we assume a programmer who wants to encrypt some messages.

It is the first step to do that he/she decides a provider and an algorihtm he/she wants to use and gets an instance of the Cipher class by using a getInstance() method which returns a specified instance. Next, he/she needs to initialize the instance by calling an init() method of the Cipher class as passing prepared parameters such as an operation mode, an appropriate key and an initialization vector. To get a symmetric key used in the cipher, The KeyGenerator class is instanciated like the Cipher class and initialized using a secret random number. After that, the instance of the KeyGenerator class generates a symmetric key and returns it. Finally, he/she calls a doFinal() method, which has plain messages as a parameter and encrypts them. Fig. 4 depicts the processing flow of the previously described scenario. We can see the way each component is associated with others to provide a cryptographic service.

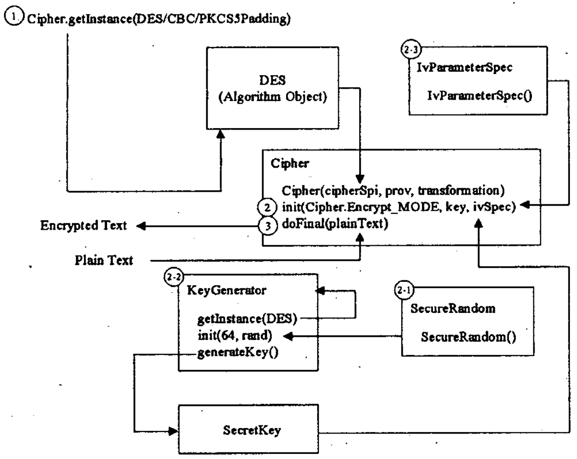
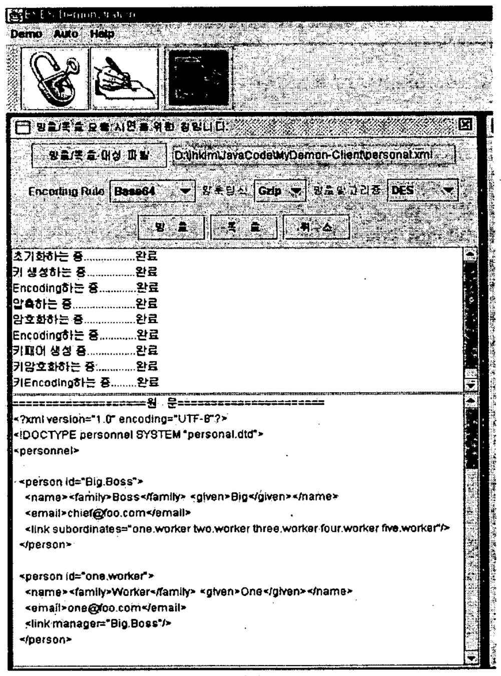


Fig. 4 The processing flow of a Cipher example using ESES/*j*-Crypto.

4. APPLICATION – A CIPHER FOR AN XML DOCUMENT

In order to how and where to use ESES/*j*-Crypto, we have developed a simple application, a cipher for an XML document. It consists of a client system and a server system. The client gets a plain XML document that has some secure information, and it generates a new secret key. The plain message is encoded and encrypted using the secret key, and then the secret key is also encrypted using the receiver's public key. Finally the encrypted message and the encrypted key are encoded with some encoding scheme such as a base64 algorithm and transferred over the network. On the server side, the server receives the encrypted message and key, and then it accomplishes the decrypting procedure, which decrypts the secret key using its private key and the ciphered message using the secret key to be decrypted right before.

Fig. 5 shows the encryption and the decryption procedures of an XML document using ESES/*j*-Crypto. The client system is shown in Fig.5 (a), a user can select an XML file to encrypt, an encoding mode, a compression style and a cryptographic algorithm.



D:\jhkim\JavaCode\MyDemon-Client\personal.xml

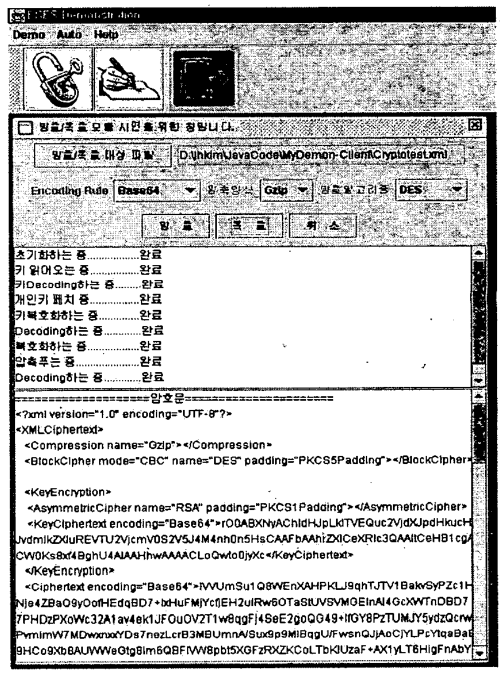
Encoding Rule Base64 압축방식 Gzip 암호알고리즘 DES

```
초기화하는 중................완료
키 생성하는 중...............완료
Encoding하는 중............완료
압축하는 중....................완료
암호화하는 중................완료
Encoding하는 중............완료
키페어 생성 중...............완료
키암호화하는 중.............완료
키Encoding하는 중.........완료
====================원  문======================
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE personnel SYSTEM "personal.dtd">
<personnel>

 <person id="Big.Boss">
  <name><family>Boss</family> <given>Big</given></name>
  <email>chief@foo.com</email>
  <link subordinates="one.worker two.worker three.worker four.worker five.worker"/>
 </person>

 <person id="one.worker">
  <name><family>Worker</family> <given>One</given></name>
  <email>one@foo.com</email>
  <link manager="Big.Boss"/>
 </person>
```

(a)



D:\jhkim\JavaCode\MyDemon-Client\Cryptotest.xml

Encoding Rule Base64 압축방식 Gzip 암호알고리즘 DES

```
초기화하는 중................완료
키 읽어오는 중...............완료
키Decoding하는 중.........완료
개인키 패치 중...............완료
키복호화하는 중.............완료
Decoding하는 중............완료
복호화하는 중................완료
압축푸는 중....................완료
Decoding하는 중............완료
====================암호문======================
<?xml version="1.0" encoding="UTF-8"?>
<XMLCiphertext>
  <Compression name="Gzip"></Compression>
  <BlockCipher mode="CBC" name="DES" padding="PKCS5Padding"></BlockCipher>

  <KeyEncryption>
   <AsymmetricCipher name="RSA" padding="PKCS1Padding"></AsymmetricCipher>
   <KeyCiphertext encoding="Base64">rO0ABXNyAChldHJpLkITVEQuc2VjdXJpdHkucH
JvdmlkZXIuREVTU2VjcmV0S2V5J4M4nh0n5HsCAAFbAAhrZXICeXRlc3QAAltCeHB1cgA
CW0Ks8xf4BghU4AIAAHhwAAAACLoQwto0jyXc</KeyCiphertext>
  </KeyEncryption>
  <Ciphertext encoding="Base64">IVVUmSu1Q8WEnXAHPKLJ9qhTJTV1BakvSyPZc1H
Nje4ZBaO9yOofHEdqBD7+bHuFMjYcfjEH2ulRw6OTaStUVSVMGEInAI4GcXWTnDBD7
7PHDzPXoWc32A1av4ek1JFOuOV2T1w8qgFj4SeE2goQG49+IfGY8PzTUMJY5ydzQcrW
PvmlmW7MDwxnxYDs7nezLcrB3MBUmnA/Sux9p9MIBqgU/FwsnQJjAoCjYLPcYtqaBaE
9HCo9XbBAUWWeGtg8lm6QBFIVW8pbt5XGFzRXZKCoLTbKIUzaF+AX1yLT6HIgFnAbY
```

(b)

Fig. 5 An example of the encryption and the decryption procedure for an XML document using ESES/*j*-Crypto. (a) The encryption procedure in the client side, (b) the decryption procedure in the server side.

When he/she clicks an encryption button, the encryption procedure is shown in the upper window and an original message and its encrypted message are appeared in the lower window. The server system shown in Fig. 5 (b) presents the decryption procedure in the upper window and the ciphertext and its plaintext in the lower window.

5. CONCLUDING REMARKS

In this paper, we have briefly introduced the JCA and JCE and their design principles. It allows programmers to implement some kinds of securities in their applications easily. We have developed ESES/*j*-Crypto system, which is as a subsystem of ESES. It is compatible with the JCA and the JCE and provides a cryptographic basis to ESES/Signature and ESES/Cipher. It includes not only generally used cryptographic algorithms, such as DSA, RSA, DES, SHA1, and HMAC, but also algorithms adopted as domestic standard such as SEED and KCDSA. It can be associated with other packages well and we are able to add new algorithms to it easily.

However, we have something to do for providing the improved CSP. First, we need to add some elliptic curve algorithms, such like ECDSA and ECKCDSA, which have been reported as providing the stronger cryptography than the existing algorithms with less number of key bits. And we are going to include Rijndael algorithm which has selected by NIST as the proposed AES(Advanced Encryption Standard) into our package.

We'll go on taking some elaborate tests to guarantee and validate our implementations and algorithms. In order to do this, we may use several test suits such as the cryptographic module validation program by NIST.

Finally, we can consider converting our packages as the Enterprise Java Beans(EJB) components. The EJB architecture allows enterprise application developers to create new applications or extend existing IT systems in shorter time to accommodate changes in the business models[7]. With transforming our packages into EJB components, we expect that these developers can apply security factor to their programs easily.

6. REFERENCES

[1] M. Mooney and T. Pozil, "American internet user survey," http://www.cyberdialogue.com, 1998.

[2] K. Mabley, "Privacy vs. personalization: A delicate balance," whitepaper, http://www.cyberdialogue.com , 1999.

[3] Sun, "Java™ Cryptography Architecture API Specification and Reference", Oct, 1999.

[4] Sun, "Java™ Cryptography Extension 1.2 API Specification and Reference", Mar, 1999.

[5] S. Oaks, *Java Security*, O'Reilly, May, 1998.

[6] J. Knudsen, *Java Cryptography*, O'Reilly, May, 1998.

[7] H. Jubin and J. Friedrichs, *Enterprise JavaBeans by Example*, Prentice Hall PTR, 1999.