Annales UMCS Informatica AI 2 (2004) 379-389

Annales UMCS
Informatica
Lublin-Polonia
Sectio AI

http://www.annales.umcs.lublin.pl/

Evaluation of possibilities of java cryptography architecture and java mail libraries usage to encrypt e-mail messages

Piotr Kopniak*

Institute of Computer Science, Faculty of Electrical Engineering and Information Technology, Lublin University of Technology, Nadbystrzycka 36B, 20-618 Lublin, Poland

Abstract

The following paper presents the possibilities of applying Java language to encrypt e-mail messages. The introduction includes a short outline about symmetric and asymmetric cryptography and how to build a safe electronic letter is discussed later. Finally, we would like to consider the possibility of implementing the discussed Java model of the safe message creation.

1. Introduction

The most frequently used medium of electronic data interchange is e-mail. A message, in a form of an electronic letter, is repeatedly passed among numerous computers on its way from a sender to a receiver. It may also happen that a message gets into the wrong hands, for instance of a mail server administrator because of the wrong address. How should we protect ourselves from the secret data disclosure? To make this transfer more secure we may pack the message into "an electronic envelope" by taking advantage of cryptography.

Over the last few years we have been researching a wide range of possibilities that Java language may possess and, therefore, we have decided to check and evaluate the ability of applying Java Cryptography Architecture and Java Mail libraries to the protection of electronic mail.

2. Data protection

The major task of present cryptography is to protect the data stored or transported among various computer systems. Because of numerous forms of attack on the message that is being sent, one can distinguish the following subjects of protection [1]:

* *E-mail address:* copy@pluton.pol.lublin.pl

1. *Confidentiality* – information should be secret and unreadable for unauthorised people. In this case cryptography has a tool in the form of cipher algorithms (ciphers). Ciphering enables the mathematical transformation of a visible message into an unreadable string of bytes, the reading of which is only possible when a decryption process has been conducted. Scrambling and de-scrambling transformations make use of additional sets of parameters, called *keys*, that are characteristic of a specific transformation [2-4].
2. *Integrity* – it means that the data that are sent are not altered in any way. Modification of an accidentally chosen part of encrypted information can make us unable to decrypt it. Cryptography can offer us a solution called *shortcut* or *hash function* that provides the integrity of data [5]. The shortcut function calculates a special unique number called *message digest* on the basis of the input data. MD5 and SHA-1 are the most common hash functions nowadays.
3. *Authenticity* – it means that the sent data indisputably comes from a sender who claims to be their author. Cryptography enables us to prove identity by means of *certificates* or *digital signatures*. A certificate is a confirmation of both: subject identity and his public key carried out and digitally signed by one of Certificate Authorities (abbr. CA) such as VeriSign [11], Thawte [12], Entrust [13]. Although there are numerous examples of certificates used in Internet, the most common one is X.509 Standard on which Internet Engineering Task Force (IETF PKIX) group is working [14].

   A digital signature is a combination of encrypting with a shortcut function. It enables us to verify the integrity and the authenticity of a message at the same time.

3. Cryptographic architecture in java

To evaluate the possibilities of applying Java to data protection one needs to get to know the cryptographic architecture that has been included in the standard version of the language since JDK 1.1 version. The cryptographic architecture of the Java language environment is based on the separation of a project from its implementation. The structure of the Java Cryptography Architecture (JCA) defines the whole process of creating cryptographic classes, which is something like a design pattern. Conceptual, abstract classes that describe the architecture core are included in the following packages: java.security and javax.crypto. The implementation that complements JCA requires the creation of our own inheritance classes from conceptual ones and it is later added taking advantage of the conception of cryptographic providers. Java is distributed with two default providers: SUN and SunJCE (which were previously distributed as an external library in Java Cryptography Extension and became an integral part of JDK 1.4).

It is possible to create one's own provider and then to add it to Java environment. Such attitude allows for an easy extension with new security mechanisms, for instance one's own implementations of existing or new encrypting algorithms. More about Java security architecture can be found in the language distribution documentation [6,7] as well as in paper [8].
SUN and SunJCE default providers offer us the implementation of various cryptographic algorithms (see Table 1) [6,7].

Table 1. Algorithms implemented by SUN and SunJCE providers (JDK 1.4)

The name of the algorithm	Assignment
Blowfish, DES, DESede, PBEwithMD5andDES, PBEwithMD5andDESede	Symmetric cipher
MD2, MD5, SHA-1, HMAC-MD5, HMAC-SHA1	Hash algorithm
Diffle-Hellman	Key agreement algorithm
DSA, SHA1withDSA MD2withRSA, MD5withRSA, SHA1withRSA	Digital signature

Because of American export restrictions, Java environment contains the implementation of only those algorithms that are either given access to export or those that are free of charge. That is why the implementation of such algorithms as RSA or IDEA is covered in the paper. The solution (that would make it likely to offer endless cryptographic possibilities of Java) uses an additional provider, for instance one of free packages: Cryptix JCE [15], Bouncy Castle Crypto API [16] or ISNetworks JCE Provider [17]. All of them implement similar cipher algorithm sets. We have chosen Criptix JCE for tests. This library contains the implementation of all the algorithms that are included in Sun libraries and numerous additional ones (see Table 2).

Table 2. Additional algorithms implemented in Cryptix JCE (17 Feb 2002 version)

The name of the algorithm	Assignment
CAST5, IDEA, MARS, RC2, RC4, RC6, Rijndael, Serpent, SKIPJACK, Square, Twofish	Symmetric cipher
MD4, RIPEMD-128, RIPEMD-160, SHA-0, SHA-256/384/512, Tiger	Hash function
HMAC-MD2, HMAC-MD4, HMAC-RIPEMD-128, HMAC-RIPEMD-160, HMAC-SHA-0, HMAC-Tiger	Keyed hash function
RSAES-OAEP, RSA/PKCS#1, ElGamal/PKCS#1	Asymmetric function
RawDSA, RSASSA-PKCS1, RSASSA-PSS	Digital signature

Java Cryptography Architecture is based on abstract classes as well as static factory method [9]. Such construction allows for using recently added algorithms and provides confidentiality, integrity and authenticity. The

successive steps of creating a message that will meet all these requirements are described below.

4. Building a protected message

The right assessment of the possibilities of data protection when using Java language can be carried out only when we put its cryptographic mechanisms into practice, i.e. when building a secure message. When a secret message is to be sent by an unprotected canal such as Internet, it should be built in a special way that provides confidentiality, integrity and authenticity. There are two very different and conflicting email encryption standards for creating secure e-mail messages: S/MIME and OpenPGP [18]. Both of them are based on multipart/signed and multipart/encrypted MIME encapsulations [19] but their message formats are quite different. However, they use similar message building algorithms and almost the same body parts generally. We describe body construction method not compatible to the standards but built on such algorithm (with blocks used in the two standards) and more distinctly showing JCA abilities (Fig. 1).

Firstly, we need a written message; in this case the contents of an e-mail letter that will be called *plain message*. Next, the message should be signed with a digital signature, i.e. we need to generate a message shortcut (or digest) and scramble it with one's own private key. In a digital signature the SHA-1 shortcut function (message digest) and RSA cipher, as it happens in the example given, will enable us to verify the message as well as the authenticity of its sender.

Java language environment gives us the possibility of keys and certificates management with the help of key stores (objects of `java.security.KeyStore` class) which are collected in the form of a file (a default file - `.keystore`). When we want to sign a message, the system needs to read the private sender's key from the key store (by means of a password from `ourPassword` parameter), which initialises the sign function. Then, using the `update()` method, the plain text is added to the signing object and the `sign()` method conducts the signing process:

```
KeyStore ks =
KeyStore.getInstance(KeyStore.getDefaultType());
ks.load(new   FileInputStream("c:/windows/.keystore"),new
  String("StoreStore").toCharArray());
byte[] plain_byte = plain_txt.getBytes();
PrivateKey privateKey =
(PrivateKey)ks.getKey(ourKeyAlias,ourPassword);
Signature sign = Signature.getInstance("SHA1withRSA");
sign.initSign(privateKey);
sign.update(plain_byte);
byte[] plainSign = sign.sign();
```

The next step is scrambling the contents of the message with a symmetric cipher (in this case DESede) because the speed of symmetric cryptosystems is much faster than that used for scrambling with an asymmetric cryptosystem (e.g. RSA) [2,5]. The measurements of encrypting times that we have conducted also seem to prove that fact (see Fig. 2 and Table 3).

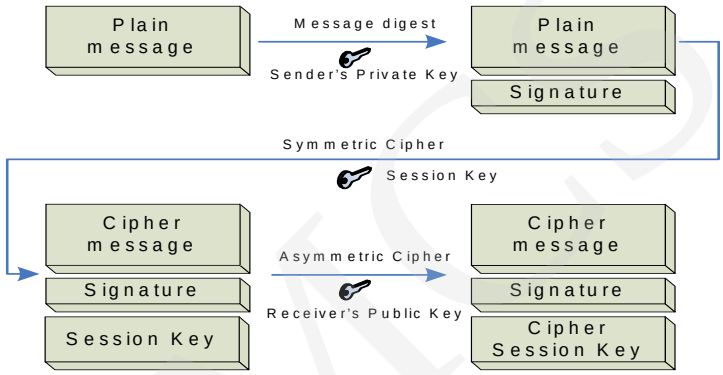

Fig. 1. Secure message building process

Block ciphers, for instance DES, scramble a given portion of information called a *block*. In the case of the DES algorithm a block consists of 64 bits [11]. Consecutive data blocks can be joined together in various ways. The rules of joining the two consecutive blocks. More about that can be found in works [1,5]. Because there is no previous block for the first one that is scrambled, we need to generate a random block which is called *initialisation vector* (*IV* in short).

Encrypting of a message starts with generating a key with the help of an object of `KeyGenerator` class and a `generateKey()` method. Next, with the use of a factory method an instance of a cipher is created in CBC mode and with PKCS#5 *padding scheme* [20]. This padding scheme specifies how to fill the remainder of the given plaintext block to the block quantity that is required by the cipher. The cipher is initialised with the encrypt mode (unlike in the decrypt mode) together with the `sessionKey` and finally the proper scrambling or de-scrambling method (`doFinal()`) is executed. The code looks like the following:

```
KeyGenerator kg = KeyGenerator.getInstance("DESede");
kg.init(new SecureRandom());
Key sessionKey = kg.generateKey();
Cipher cph =
Cipher.getInstance("DESede/CBC/PKCS5Padding");
cph.init(Cipher.ENCRYPT_MODE,sessionKey);
      byte[] cipher_byte = cph.doFinal(plain_byte);
      byte[] iv = cph.getIV();
```

In order to improve the safety, we need to use a certain scrambling key, called *session key,* that has to be changed for every new message so it needs to be added to the message and sent with it every time. Due to this the key must be protected by scrambling so as not to be used for decrypting by an intruder.

The process of scrambling of a key (which is a much smaller set of data than a message itself) can be executed by means of asymmetric RSA cipher that comes from Cryptix JCE library. An asymmetric cipher allows the receiver to de-scramble a session key without any problems only when his public key was used for scrambling. Only then can we be sure that the right addressee will read the message.

The encrypting process begins with downloading a receiver's certificate that is stored in a sender's key store. Next, the instance of the cipher is created. After initialising the cipher by means of a certificate, the proper encrypting method (`doFinal()`) is executed, which looks like that:

```
java.security.cert.Certificate cert =
ks.getCertificate(cert_alias);
cph = Cipher.getInstance("RSA/ECB/PKCS#1");
cph.init(Cipher.ENCRYPT_MODE, cert);
cipherKey=cph.doFinal(sessionKey.getEncoded());
```

Additionally, one can attach the name of the sender to the message and this will help the receiver to search for the right certificate and public key in his own key store:

```
java.security.cert.Certificate cert =
ks.getCertificate(ourKeyAlias);
String ourDistName =
((X509Certificate)cert).getSubjectDN().toString();
```

The combination of all the parts of an encrypted message into one byte array `cipher_byte` looks like that:

```
ByteArrayOutputStream       byteOutput       =       new
ByteArrayOutputStream();
DataOutputStream dataOutput = new
DataOutputStream(byteOutput);
dataOutput.writeUTF(ourDistName);
dataOutput.writeInt(iv.length);
dataOutput.write(iv);
dataOutput.writeInt(cipherKey.length);
dataOutput.write(cipherKey);
dataOutput.writeInt(plainSign.length);
dataOutput.write(plainSign);
```

```
dataOutput.writeInt(cipher_byte.length);
dataOutput.write(cipher_byte);
byte[] cipher_byte = byteOutput.toByteArray();
```

A message, that has been prepared in such a way, can finally be sent. How can Java Mail library be used for that will be described in the later section of the paper.

The implementation of the message decrypting method is based on executing the same set of activities when encrypting but in the reversed order.

Apart from that, after getting a message, one can also verify the authenticity of a sender. The shortcut function is executed on the de-scrambled message. At the same time the message digest that was sent together with a message is decrypted using a sender's public key. Both figures of message digest are compared and only if they are identical it means that the sender is in fact the person he claims to be:

```
X509Certificate xCert =
(X509Certificate)kstore.getCertificate(tmpAlias);
PublicKey theirPublic = xCert.getPublicKey();
Signature sign = Signature.getInstance("SHA1withRSA");
sign.initVerify(theirPublic);
sign.update(plain_byte);
boolean signOK = sign.verify(plainSign);
```

As it was shown above, Java Cryptography Architecture seems to be the right solution for data protection due to the fact that Java possesses all the necessary tools that provide the confidentiality, integrity and authenticity for information exchange.

5. Java mail library

JavaMail API consists of a set of classes that are used for developing applications that transport electronic messages via Internet. This library enables us to create client or server side programs that are independent of the system platform, an Internet provider or communication protocol. JavaMail is distributed as an optional package of the Java Standard Edition (J2SE) and it is an internal part of Java Enterprise Edition (J2EE).

In the discussed Java library there are implemented the following e-mail standards [21,22]:

1. SMTP (Simple Mail Transfer Protocol) that is used for e-mail messages sending,
2. POP (Post Office Protocol) used for downloading e-mails from a mail server,

3. IMAP (Internet Message Access Protocol) – an advanced protocol for downloading e-mails,
4. MIME (Multipurpose Internet Mail Extension) that allows us to specify the type of the contents of the data that are sent.

6. Sending e-mails

An e-mail message consists of essential parts: a header (in which one can distinguish, among other things, the information about the sender, the receiver's address and the subject of a letter) and the main body of a letter. Sending a message using Java Mail Library is based on constructing a `Message` object, adding some content to it by means of `setText()` method and, finally, sending it using the static `send()` method of `Transport` class. Creating a message demands constructing a session object that contains the address of SMTP server:

```
Properties props = System.getProperties();
props.put("mail.smtp.host", host);
Session session = Session.getDefaultInstance(props,
null);
MimeMessage message = new MimeMessage(session);
message.setFrom(new InternetAddress(from));
message.addRecipient(Message.RecipientType.TO,new
InternetAddress(to));
message.setSubject("Cipher Mail");
message.setText(new String(cipher_byte));
Transport.send(message);
```

7. Receiving e-mails

When receiving a message, we need to connect to a specified e-mail box on POP or IMAP server and download a letter. If we want to use Java Mail library in this operation, we need to create a Store object that will manage the connections and mail box folders. The Store object is equipped with the `connect()` method (that makes it possible to get connected to the server) and the `getFolder()` method (that returns the mail box folder, the default one is INBOX). After opening the folder one can read all the messages that are there. The following example shows the operation of getting e-mail messages and displaying all of them on a system console:

```
Properties prop = new Properties();
Session session = Session.getDefaultInstance(prop, null);
Store store = session.getStore("pop3");
store.connect(host, username, password);
Folder folder = store.getFolder("INBOX");
```

```
folder.open(Folder.READ_ONLY);
Message message[] = folder.getMessages();
for (int i=0, n=message.length; i<n; i++) {
  System.out.println(i + ": " + message[i].getFrom()[0]
  + "\t" + message[i].getSubject() + ”\n” +
  message[i].writeTo(System.out);
)
folder.close(false);
store.close();
```

8. Conclusions and future works

The research that we carried out was to evaluate the cryptographic possibilities of Java language. The analysis of its cryptographic architecture as well as developing a sample mail application allowing for exchanging secure messages proved that:

- Java is a language of outstanding cryptographic functionality. The standard edition of the language is equipped with cryptographic mechanisms that satisfy the requirements of data protection while extending the language environment by means of extra cryptographic algorithms is quite easy due to a well-thought-out architecture of cryptographic providers.
- In Java environment we can distinguish an advanced mechanism of key and certificate management that is based on the key stores which is physically represented by password protected files. There are also special command line tools, for instance: `keytool` that is used for creating and managing of keys and certificates, generating the keys, import, export, creating one's own certificates etc.
- Java Mail library is a convenient and easy-to-use set of classes that allow us to build an e-mail client application which uses basic communication protocols as well as password authentication.

While developing an e-mail encrypting application we have done some research on the encrypting times with the use of symmetric and asymmetric algorithms as well as on the time that is needed to carry out an effective brutal attack on the DES cipher.

Table 3. Encryption times (ms) for different algorithms

Size (byte)	7 000	14 000	100 000	1 000 000
DES (56 bit)	0	0	50	300
DESede (168 bit)	0	0	120	800
RSA (512 bit)	150	200	1 100	40 000
RSA (1024 bit)	110	250	1 500	85 000

The efficiency tests of such encrypting algorithms as DES, DESede, and RSA carried out for the differentiated lengths of messages proved that the encrypting time grows together with the key length and it is about ten times longer for the asymmetric ciphers than for symmetric ones.

It confirms that the choice of the symmetric algorithm for encrypting longer data (e.g. a message) is the right solution, just as deciding on an asymmetric algorithm when encrypting considerably shorter figures (like a key). Long encryption times for RSA algorithms are observable especially with a considerable amount of scrambled data. In the case of the text that is of about 1MB size and using a 1024 bites key the scrambling lasts as long as 85 seconds.

The study of the time needed for the brutal breaking of the DES cipher (by searching the key space), and supposing that the right solution would be found after searching half of the possible keys (36 028 797 018 963 968), proves that the time needed to complete the task by means of a one-thread Java application on a 1GHz processor machine would take 3 238 506 years. The results turned out too long in comparison to the solution times published in the Internet [23]. It is due to the reason that Java is an interpreted language that cannot be used for cryptoanalytic purposes. The best results are achieved thanks to hardware implementation [2].

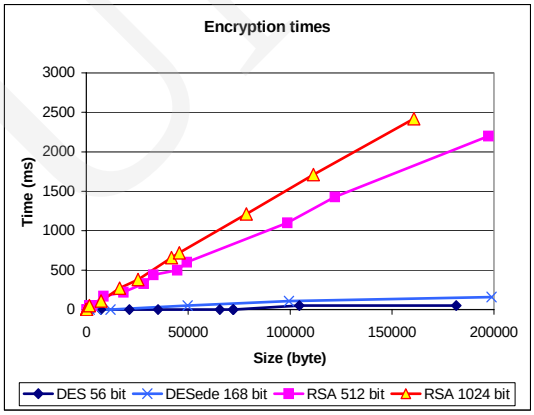

Fig. 2. Symmetric and asymmetric ciphers encryption times

In the research that will follow we will conduct the studies of the abilities of using Java implementation standards S/MIME in Bouncy Castle Crypto API, OpenPGP in Criptix as well as the Free cryptographic extension to JavaMail that currently undergoes some tests, that is JavaMail-Crypto API [24].

References

[1] Robling Denning D. E., *Kryptografia i ochrona danych*, WNT, Warszawa, (1992) (1993), in Polish.
[2] Levy S., *Rewolucja w kryptografii*, WNT, Warszawa, (2002), in Polish.

[3] Koblitz N., *Algebraiczne aspekty kryptografii*, WNT, Warszawa, (2000), in Polish.
[4] Koblitz N., *Wykład z teorii liczb i kryptografii*, WNT, Warszawa, (1995), in Polish.
[5] Knudsen J. B., *Java Cryptography*, O'Reilly, (1998).
[6] *Java Cryptography Architecture API Specification & Reference*, Sun Microsystems, Inc., 4 August 2002.
[7] *Java Cryptography Extension (JCE) Reference Guide for the Java 2 SDK, Standard Edition, v 1.4*, Sun Microsystems, Inc, 10 Jan 2002.
[8] Matusiewicz M., *Mechanizmy ochrony danych w języku Java*, Materiały VII Krajowej Konferencji Zastosowań Kryptografii ENIGMA'2003, Warszawa, 545, in Polish.
[9] Eckel B., *Thinking in Java*, Helion, Gliwice, (2001).
[10] Loudon K., *Algorytmy w C*, Helion, Gliwice, (2003), in Polish.
[11] http://www.verisign.com/ - VeriSign
[12] http://www.thawte.com/ - Thawte
[13] http://www.entrust.com/ - Entrust
[14] http://www.ietf.org/ – The Internet Engineering Task Force
[15] http://www.cryptix.org/ – The Cryptix Foundation Limited
[16] http://www.bouncycastle.org/ - Legion of the Bouncy Castle
[17] http://www.isnetworks.com/ - ISNetworks
[18] http://www.imc.org/smime-pgpmime.html - S/MIME and OpenPGP
[19] http://www.ietf.org/rfc/rfc1847 – RFC 1847: Security Multiparts for MIME
[20] http://www.rsasecurity.com/rsalabs/pkcs/ - RSA Security PKCS Workshop
[21] http://java.sun.com/products/javamail/ - JavaMail homepage, Sun Microsystems, Inc.
[22] http://www.imc.org/ – Internet Mail Consortium
[23] http://www.bezpieczenstwoit.pl/Kryptografia.html – Bezpieczeństwo IT
[24] http://javamail-crypto.sourceforge.net/ - JavaMail-Crypto API Homepage