Laille David
Advanced Operating Systems

Homework 2 - AES encryption and RSA digital signature

Instructions:

Write a method that encrypts and signs a file and writes a result to a file. The encryption should combine symmetric cypher (e.g. AES) and public-key encryption (RSA). RSA-SHA1 should be used for digital signature:

1. Generate random key k
2. c1 = AES-CBC(file_contents, k)
3. c2 = RSA(k, public_key_of_recepient)
4. c4 = RSA-SIGN(SHA1(m), secret_key_of_sender)

Also write a method that reverses the above procedure, decrypting the file and verifying the signature.

Reference:
AES/RSA: https://docs.oracle.com/javase/7/docs/api/javax/crypto/Cipher.html
SHA1: https://docs.oracle.com/javase/7/docs/api/java/security/MessageDigest.html

The homework will be realised with java. The program will be divided into three main class:

GenerateRSAKeys.java
CipheringProcess.java
DecipheringProcess.java

Below, we can find a copy of the code. The code is also joined to this report.

Code:

GenerateRSAKeys.java

```
package EncryptAndSign;

import java.io.FileNotFoundException;
import java.io.FileOutputStream;
import java.io.IOException;
import java.security.KeyPair;
import java.security.KeyPairGenerator;
import java.security.NoSuchAlgorithmException;

//Generate a pair of RSA private and public keys.
//These keys are saved in files with the extensions ".private" and ".public"
//They are saved as binary data

public class GenerateRSAKeys {

	//How to use : java CipheringProcess args[0] args[1]
	//args[0] : name of the file where the private RSA key will be saved
	//args[1] : name of the file where the public RSA key will be saved

	public static void main(String[] args)
		throws FileNotFoundException, IOException,
		NoSuchAlgorithmException
	{

		//Generating the RSA pair of keys
		//System.out.println("1.Test generation of RSA pair of key");

		KeyPairGenerator kpg = KeyPairGenerator.getInstance("RSA");
		kpg.initialize(2048);
		KeyPair kp = kpg.generateKeyPair();
		System.out.println("Generation of the pair of keys done.");

		//Saving the pair of key
			//System.out.println("2.Test backup of RSA pair of key");

		String fileBase = args[0];
		try (FileOutputStream out = new FileOutputStream(fileBase +
".private")) {
			out.write(kp.getPrivate().getEncoded());
		}

		try (FileOutputStream out = new FileOutputStream(fileBase +
".public")) {
			out.write(kp.getPublic().getEncoded());
		}

		System.out.println("Private key saved in " + fileBase + ".private");
		System.out.println("Public key saved in " + fileBase + ".public");
	}
}
```

CipheringProcess.java

```
package EncryptAndSign;

import java.io.FileInputStream;
import java.io.FileOutputStream;
import java.io.IOException;
import java.io.InputStream;
import java.io.OutputStream;
import java.nio.file.Files;
import java.nio.file.Paths;
import java.security.InvalidAlgorithmParameterException;
import java.security.InvalidKeyException;
import java.security.KeyFactory;
import java.security.NoSuchAlgorithmException;
import java.security.PrivateKey;
import java.security.PublicKey;
import java.security.SecureRandom;
import java.security.Signature;
import java.security.SignatureException;
import java.security.spec.InvalidKeySpecException;
import java.security.spec.PKCS8EncodedKeySpec;
import java.security.spec.X509EncodedKeySpec;

import javax.crypto.BadPaddingException;
import javax.crypto.Cipher;
import javax.crypto.IllegalBlockSizeException;
import javax.crypto.KeyGenerator;
import javax.crypto.NoSuchPaddingException;
import javax.crypto.SecretKey;
import javax.crypto.spec.IvParameterSpec;

//Class CipheringProcess
//This process take place in computer A
//This computer A will send the encrypted message to a computer B
//A and B own each one a pair of RSA private and public keys
//The AES secret key will be generated randomly and associated with a
initialization vector
//The AES key will be encrypted with B's public key
//The AES encrypted key will be saved in a file with extension ".enc"
//The signature will be done with A's private key

public class CipheringProcess {

	//Method signFile : allow the signature of a file
	//Parameters : Cipher object, Signature object, InputStream, OutputStream
	//Cipher : used to encrypt the file
	//Signature : used to sign the message
	//InputStream : to read the message from this file
	//OutputStream : to write the output

	static private void signFile(Cipher ci,Signature sign,InputStream
in,OutputStream out)
			throws javax.crypto.IllegalBlockSizeException,
				javax.crypto.BadPaddingException,
				java.security.SignatureException,
				java.io.IOException
```

```
            {
                byte[] ibuf = new byte[1024];
                int len;
                while ((len = in.read(ibuf)) != -1) {
                    sign.update(ibuf, 0, len);
                    byte[] obuf = ci.update(ibuf, 0, len);
                    if ( obuf != null ) out.write(obuf);
                }
                byte[] obuf = ci.doFinal();
                if ( obuf != null ) out.write(obuf);
            }

    //How to use class CipheringProcess: java CipheringProcess args[0] args[1]
args[2]
    //args[0] : name of the file containing the A's private key
    //args[1] : name of the file containing the B's public key
    //args[2] : name of the file we want to cipher with AES

    public static void main(String[] args)
                throws      NoSuchAlgorithmException, InvalidKeySpecException,
                            IllegalBlockSizeException, BadPaddingException,
                            IOException, InvalidKeyException,
                            NoSuchPaddingException,
                            InvalidAlgorithmParameterException,
                            SignatureException
            {

            //Load A's private key and B's public keys
            //System.out.println("1.Test load of the public and private keys");

            PrivateKey pvt = null;
            String pvtKeyFile = args[0];
            {
            byte[] bytes = Files.readAllBytes(Paths.get(pvtKeyFile));
            PKCS8EncodedKeySpec ks = new PKCS8EncodedKeySpec(bytes);
            KeyFactory kf = KeyFactory.getInstance("RSA");
            pvt = kf.generatePrivate(ks);
            }
            System.out.println("Private key read from " + pvtKeyFile);

            PublicKey pub = null;
            String pubKeyFile = args[1];
            {
            byte[] bytes = Files.readAllBytes(Paths.get(pubKeyFile));
            X509EncodedKeySpec ks = new X509EncodedKeySpec(bytes);
            KeyFactory kf = KeyFactory.getInstance("RSA");
            pub = kf.generatePublic(ks);
            }
            System.out.println("Public key read from " + pubKeyFile);


            //Generation of the AES secret key
            //System.out.println("2.Test generation of AES key");

            KeyGenerator kgen = KeyGenerator.getInstance("AES");
            kgen.init(128);
            SecretKey skey = kgen.generateKey();

            System.out.println("Generation of AES key done.");
```

```
            //Initialization vector
            //System.out.println("3.Test initialization vector");

            byte[] iv = new byte[128/8];
            new SecureRandom().nextBytes(iv);
            IvParameterSpec ivspec = new IvParameterSpec(iv);

            System.out.println("Generation of initialization vector done.");

            //Encryption of the AES key with RSA
            //System.out.println("4.Test encryption of the AES key");

            String inputFile = args[2];
            FileOutputStream out = new FileOutputStream(inputFile + ".enc");
            {
            Cipher cipher = Cipher.getInstance("RSA/ECB/PKCS1Padding");

            cipher.init(Cipher.ENCRYPT_MODE, pub); // Encrypt using B's public
key
            byte[] b = cipher.doFinal(skey.getEncoded());
            out.write(b);
            }

            out.write(iv);

            System.out.println("Encryption of the AES key done with" +
pubKeyFile);

            //Encryption of the message with AES secret key and signature
            //System.out.println("5.Test encryption of the message");

            Signature sign = Signature.getInstance("SHA256withRSA");
            sign.initSign(pvt); // Sign using A's private key

            Cipher ci = Cipher.getInstance("AES/CBC/PKCS5Padding");
            ci.init(Cipher.ENCRYPT_MODE, skey, ivspec);

            try (FileInputStream in = new FileInputStream(inputFile)) {
                signFile(ci, sign, in, out);
            }
            byte[] s = sign.sign();
            out.write(s);
            out.close();

            System.out.println("Encryption of the file" + inputFile + " done.");
            System.out.println("Signature of the file " + inputFile + " done with
" + pvtKeyFile);

            //System.out.println("6.Test program go to the end");
      }

}
```

DecipheringProcess.java

```
package EncryptAndSign;

import java.io.File;
import java.io.FileInputStream;
import java.io.FileNotFoundException;
import java.io.FileOutputStream;
import java.io.IOException;
import java.io.InputStream;
import java.io.OutputStream;
import java.nio.file.Files;
import java.nio.file.Paths;
import java.security.InvalidAlgorithmParameterException;
import java.security.InvalidKeyException;
import java.security.KeyFactory;
import java.security.NoSuchAlgorithmException;
import java.security.PrivateKey;
import java.security.PublicKey;
import java.security.Signature;
import java.security.SignatureException;
import java.security.spec.InvalidKeySpecException;
import java.security.spec.PKCS8EncodedKeySpec;
import java.security.spec.X509EncodedKeySpec;

import javax.crypto.BadPaddingException;
import javax.crypto.Cipher;
import javax.crypto.IllegalBlockSizeException;
import javax.crypto.NoSuchPaddingException;
import javax.crypto.spec.IvParameterSpec;
import javax.crypto.spec.SecretKeySpec;

//Class DecipheringProcess
//This process take place in computer B
//This computer B receive the encrypted message from computer A
//A and B own each one a pair of RSA private and public keys
//AES secret key is first decrypted with B's private key
//Then, the message is decrypted with the AES key
//Finally, the signature is decrypted with A's public key
//The signature will be compared with the initial signature (present in the file)

public class DecipheringProcess {

	//Method authFile : verifies the signature of the decrypted message
	//Parameters : Cipher object, Signature object, InputStream, OutputStream
	//Cipher : used to decrypt the file
	//Signature : used to sign the file; the signed file will be compared to the
original



	static private void authFile(Cipher ci,Signature ver,InputStream
in,OutputStream out,long dataLen)
			throws javax.crypto.IllegalBlockSizeException,
				javax.crypto.BadPaddingException,
				java.security.SignatureException,
				java.io.IOException
```

```
            {
                byte[] ibuf = new byte[1024];
                while (dataLen > 0) {
                    int max = (int)(dataLen > ibuf.length ? ibuf.length :
dataLen);
                    int len = in.read(ibuf, 0, max);
                    if ( len < 0 ) throw new java.io.IOException("Insufficient
data");
                    dataLen -= len;
                    byte[] obuf = ci.update(ibuf, 0, len);
                    if ( obuf != null ) {
                        out.write(obuf);
                        ver.update(obuf);
                    }
                }
                byte[] obuf = ci.doFinal();
                if ( obuf != null ) {
                    out.write(obuf);
                    ver.update(obuf);
                }
            }

      //How to use class DecipheringProcess: java DeipheringProcess args[0]
args[1] args[2]
      //args[0] : name of the file containing the A's public key
      //args[1] : name of the file containing the B's private key
      //args[2] : name of the file we want to decipher with AES

      public static void main(String[] args)
                  throws FileNotFoundException,
                        IOException, InvalidKeySpecException,
                        NoSuchAlgorithmException, IllegalBlockSizeException,
                        BadPaddingException, NoSuchPaddingException,
                        InvalidKeyException, InvalidAlgorithmParameterException,
                        SignatureException
            {

            //Load A's public key and B's private key
            //System.out.println("1.Test load of the public and private keys");

            PrivateKey pvt = null;
            String pvtKeyFile = args[0];
            {
                  byte[] bytes = Files.readAllBytes(Paths.get(pvtKeyFile));
                  PKCS8EncodedKeySpec ks = new PKCS8EncodedKeySpec(bytes);
                  KeyFactory kf = KeyFactory.getInstance("RSA");
                  pvt = kf.generatePrivate(ks);
            }
            System.out.println("Private key read from " + pvtKeyFile);

            PublicKey pub = null;
            String pubKeyFile = args[1];
            {
                byte[] bytes = Files.readAllBytes(Paths.get(pubKeyFile));
                X509EncodedKeySpec ks = new X509EncodedKeySpec(bytes);
                KeyFactory kf = KeyFactory.getInstance("RSA");
                pub = kf.generatePublic(ks);
            }
            System.out.println("Public key read from " + pubKeyFile);
```

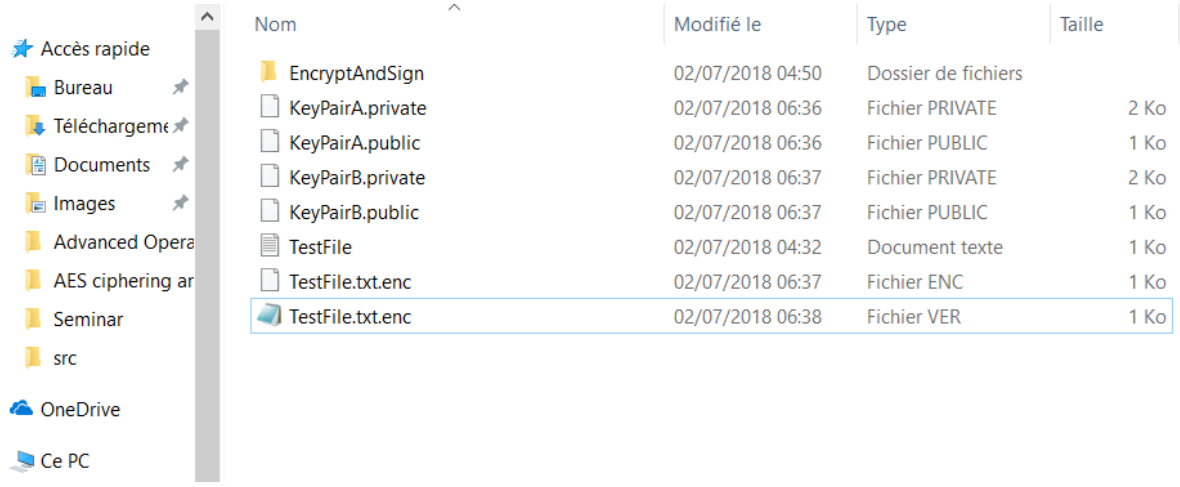
```
            //Decrypting the AES key
            //System.out.println("2.Test decryption of AES key");

            String inputFile = args[2];

            //Computing the length of the encrypted file
            long dataLen = new File(inputFile ).length()
                       - 256       // AES Key
                       - 16        // IV
                       - 256;      // Signature

            FileInputStream in = new FileInputStream(inputFile);
            SecretKeySpec skey = null;
            {
            Cipher cipher = Cipher.getInstance("RSA/ECB/PKCS1Padding");
            cipher.init(Cipher.DECRYPT_MODE, pvt); // B's private key here
            byte[] b = new byte[256];
            in.read(b);
            byte[] keyb = cipher.doFinal(b);
            skey = new SecretKeySpec(keyb, "AES");
            }

            System.out.println("Decryption of the key done with" + pvtKeyFile);

            //Loading the initialization vector
            //System.out.println("3.Test load of initialization vector");

            byte[] iv = new byte[128/8];
            in.read(iv);
            IvParameterSpec ivspec = new IvParameterSpec(iv);

            System.out.println("Initialization vector read.");

            //Decrypting the message and verifying the signature
            //System.out.println("4.Test decryption and signature");

            Signature ver = Signature.getInstance("SHA256withRSA");
            ver.initVerify(pub); // Using B's public key
            Cipher ci = Cipher.getInstance("AES/CBC/PKCS5Padding");
            ci.init(Cipher.DECRYPT_MODE, skey, ivspec);
            try (FileOutputStream out = new FileOutputStream(inputFile+".ver")){
                authFile(ci, ver, in, out, dataLen);
            }

            System.out.println("Decryption of the file " + inputFile + " done.");
            System.out.println("Signature of the file " + inputFile + "checked");

            byte[] s = new byte[256];
            int len = in.read(s);
            if ( ! ver.verify(s) ) {
                  System.out.println("Signature not valid: ");
            }

            System.out.println("The decrypted message is located in the file " +
inputFile + ".ver");;

      }
}
```

Compilation and execution:

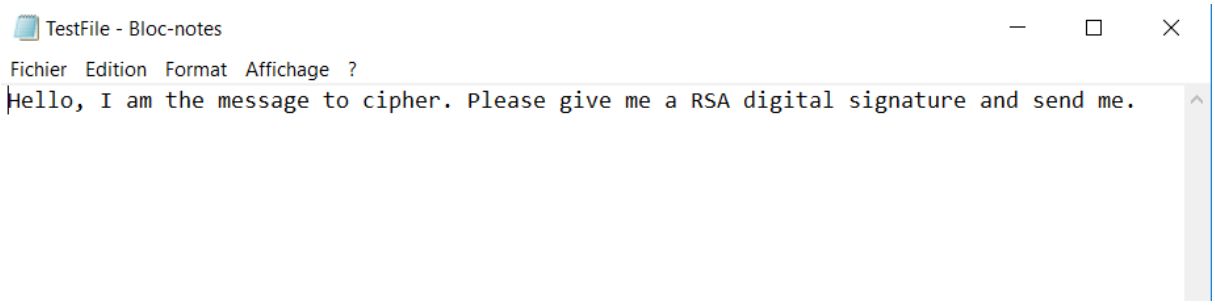
```
C:\Users\Laille David\eclipse-workspace\Homework2\src>cd EncryptAndSign

C:\Users\Laille David\eclipse-workspace\Homework2\src\EncryptAndSign>javac GenerateRSAKeys.java

C:\Users\Laille David\eclipse-workspace\Homework2\src\EncryptAndSign>javac CipheringProcess.java

C:\Users\Laille David\eclipse-workspace\Homework2\src\EncryptAndSign>javac DecipheringProcess.java

C:\Users\Laille David\eclipse-workspace\Homework2\src\EncryptAndSign>cd..

C:\Users\Laille David\eclipse-workspace\Homework2\src>java EncryptAndSign/GenerateRSAKeys "KeyPairA"
Generation of the pair of keys done.
Private key saved in KeyPairA.private
Public key saved in KeyPairA.public

C:\Users\Laille David\eclipse-workspace\Homework2\src>java EncryptAndSign/GenerateRSAKeys "KeyPairB"
Generation of the pair of keys done.
Private key saved in KeyPairB.private
Public key saved in KeyPairB.public

C:\Users\Laille David\eclipse-workspace\Homework2\src>java EncryptAndSign/CipheringProcess "KeyPairA.pri
vate" "KeyPairB.public" "TestFile.txt"
Private key read from KeyPairA.private
Public key read from KeyPairB.public
Generation of AES key done.
Generation of initialization vector done.
Encryption of the AES key done withKeyPairB.public
Encryption of the fileTestFile.txt done.
Signature of the file TestFile.txt done with KeyPairA.private

C:\Users\Laille David\eclipse-workspace\Homework2\src>java EncryptAndSign/DecipheringProcess "KeyPairB.p
rivate" "KeyPairA.public" "TestFile.txt.enc"
Private key read from KeyPairB.private
Public key read from KeyPairA.public
Decryption of the AES secret key done withKeyPairB.private
Initialization vector read.
Decryption of the file TestFile.txt.enc done.
Signature of the file TestFile.txt.enc checked
The decrypted message is located in the file TestFile.txt.enc.ver

C:\Users\Laille David\eclipse-workspace\Homework2\src>
```

Results:

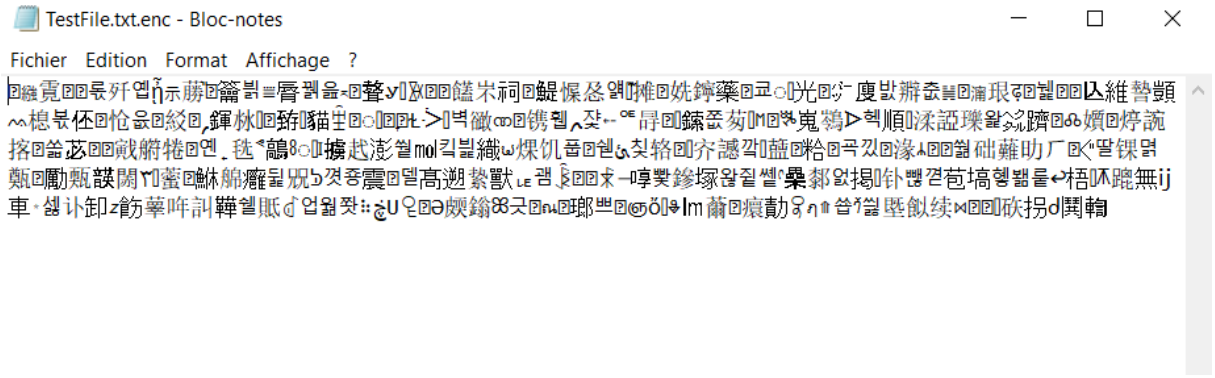
Folder src



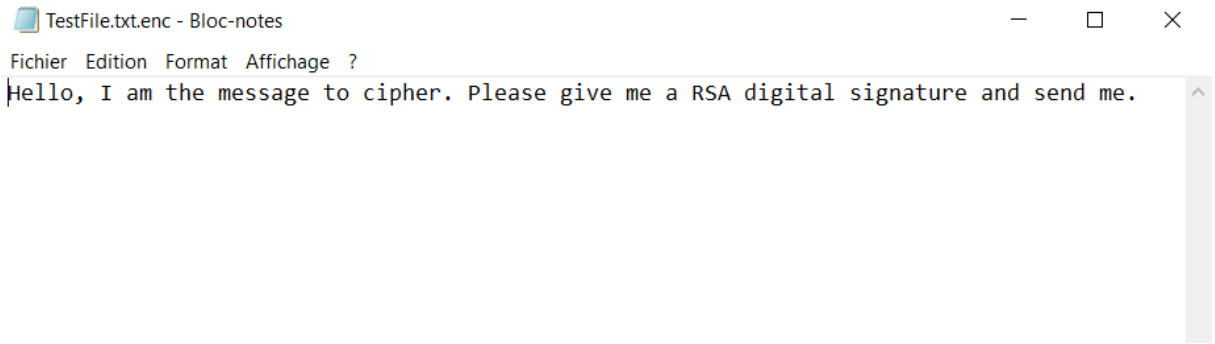
File TestFile.txt



File TestFile.txt.enc



File TestFile.txt.enc.ver



As we can see, the encrypted file is impossible to read and the decrypted file is exactly the same than the initial one.