package ro.ase.ism.sap;

import java.io.BufferedInputStream;

import java.io.BufferedOutputStream;

import java.io.File;

import java.io.FileInputStream;

import java.io.FileNotFoundException;

import java.io.FileOutputStream;

import java.io.IOException;

import java.rmi.UnexpectedException;

import java.security.InvalidKeyException;

import java.security.Key;

import java.security.KeyStore;

import java.security.KeyStoreException;

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.UnrecoverableKeyException;

import java.security.cert.CertificateException;

import java.security.cert.CertificateFactory;

import java.security.cert.X509Certificate;

import javax.crypto.BadPaddingException;

import javax.crypto.Cipher;

import javax.crypto.IllegalBlockSizeException;

import javax.crypto.KeyGenerator;

import javax.crypto.NoSuchPaddingException;

import javax.crypto.spec.SecretKeySpec;

public class assignment3 {

public static String getHex(byte[] value) {

String output = "";

for(byte byteValue : value) {

output += String.format("%02x", byteValue);

}

return output;

}

public static PublicKey getCertificatePublicKey(String cerFileName) throws CertificateException, IOException {

File cerFile = new File(cerFileName);

if(!cerFile.exists()) {

System.out.println("Certificate not found!");

throw new FileNotFoundException();

}

FileInputStream fis = new FileInputStream(cerFile);

CertificateFactory certFactory = CertificateFactory.getInstance("X.509");

X509Certificate certificate = (X509Certificate) certFactory.generateCertificate(fis);

fis.close();

return certificate.getPublicKey();

}

public static KeyStore getKeyStore(String keyStoreFile, String keyStorePass, String type) throws KeyStoreException, NoSuchAlgorithmException, CertificateException, IOException {

File ksFile = new File(keyStoreFile);

if(!ksFile.exists()) {

System.out.println("File not found!");

throw new FileNotFoundException();

}

FileInputStream fis = new FileInputStream(ksFile);

KeyStore ks = KeyStore.getInstance(type);

ks.load(fis, keyStorePass.toCharArray());

fis.close();

return ks;

}

public static PrivateKey getPrivateKey(KeyStore ks, String entryAlias,String keyPass) throws UnexpectedException, UnrecoverableKeyException, KeyStoreException, NoSuchAlgorithmException {

if(ks==null) {

System.out.println("No key Store available!");

throw new UnexpectedException("No keystore!");

}

if(ks.containsAlias(entryAlias)) {

return (PrivateKey) ks.getKey(entryAlias,keyPass.toCharArray());

}else {

return null;

}

}

public static void encryptECB(

String inputFile, byte [] key, String algorithm, String cipherFile) throws IOException, NoSuchAlgorithmException, NoSuchPaddingException, InvalidKeyException, IllegalBlockSizeException, BadPaddingException {

File input = new File(inputFile);

if(!input.exists()) {

throw new FileNotFoundException();

}

FileInputStream fis = new FileInputStream(input);

BufferedInputStream bis = new BufferedInputStream(fis);

File output = new File(cipherFile);

if(!output.exists()) {

output.createNewFile();

}

FileOutputStream fos = new FileOutputStream(output);

BufferedOutputStream bos = new BufferedOutputStream(fos);

Cipher cipher = Cipher.getInstance(algorithm + "/ECB/PKCS5Padding");

SecretKeySpec keySpec = new SecretKeySpec(key, algorithm);

cipher.init(Cipher.ENCRYPT\_MODE, keySpec);

byte[] buffer = new byte[cipher.getBlockSize()];

int noBytes = 0;

byte[] cipherBuffer;

while(noBytes != -1) {

noBytes = bis.read(buffer);

if(noBytes != -1) {

cipherBuffer = cipher.update(buffer, 0, noBytes);

bos.write(cipherBuffer);

}

}

cipherBuffer = cipher.doFinal();

bos.write(cipherBuffer);

fis.close();

bos.close();

}

public static void decryptECB(

String cipherFile, byte [] key, String algorithm, String plainFile) throws IOException, NoSuchAlgorithmException, NoSuchPaddingException, InvalidKeyException, IllegalBlockSizeException, BadPaddingException {

File input = new File(cipherFile);

if(!input.exists()) {

throw new FileNotFoundException();

}

FileInputStream fis = new FileInputStream(input);

BufferedInputStream bis = new BufferedInputStream(fis);

File output = new File(plainFile);

if(!output.exists()) {

output.createNewFile();

}

FileOutputStream fos = new FileOutputStream(output);

BufferedOutputStream bos = new BufferedOutputStream(fos);

Cipher cipher = Cipher.getInstance(algorithm + "/ECB/PKCS5Padding");

cipher.init(Cipher.DECRYPT\_MODE, new SecretKeySpec(key, algorithm));

byte[] inputBuffer = new byte[cipher.getBlockSize()];

byte[] outputBuffer;

int noBytes = 0;

while(noBytes != -1) {

noBytes = bis.read(inputBuffer);

if(noBytes != -1) {

outputBuffer = cipher.update(inputBuffer,0,noBytes);

bos.write(outputBuffer);

}

}

outputBuffer = cipher.doFinal();

bos.write(outputBuffer);

bos.close();

fis.close();

}

public static byte[] encryptRSA(Key key, byte[] input) throws NoSuchAlgorithmException, NoSuchPaddingException, InvalidKeyException, IllegalBlockSizeException, BadPaddingException {

Cipher cipher = Cipher.getInstance("RSA/ECB/PKCS1Padding");

cipher.init(Cipher.ENCRYPT\_MODE, key);

return cipher.doFinal(input);

}

public static byte[] decryptRSA(Key key, byte[] input) throws NoSuchAlgorithmException, NoSuchPaddingException, InvalidKeyException, IllegalBlockSizeException, BadPaddingException {

Cipher cipher = Cipher.getInstance("RSA/ECB/PKCS1Padding");

cipher.init(Cipher.DECRYPT\_MODE, key);

return cipher.doFinal(input);

}

public static boolean ValidSignature(String fileName, PublicKey pubKey, String signatureFile) throws InvalidKeyException, NoSuchAlgorithmException, IOException, SignatureException {

Signature signature = Signature.getInstance("SHA512withRSA");

signature.initVerify(pubKey);;

File file = new File(fileName);

if(!file.exists()) {

System.out.println("File is missing!");

throw new FileNotFoundException();

}

FileInputStream fis = new FileInputStream(file);

File fileSign = new File(signatureFile);

if(!file.exists()) {

System.out.println("File is missing!");

throw new FileNotFoundException();

}

FileInputStream fisSign= new FileInputStream(fileSign);

//let's suppose the file is small and we can read it at once:

byte[] fileContent = fis.readAllBytes();

byte[] digitalSignature = fisSign.readAllBytes();

signature.update(fileContent);

fis.close();

fisSign.close();

return signature.verify(digitalSignature);

}

public static byte[] getRandomSessionKey(String algorithm, int noBytes) throws NoSuchAlgorithmException {

KeyGenerator keyGenerator = KeyGenerator.getInstance(algorithm);

keyGenerator.init(noBytes);

return keyGenerator.generateKey().getEncoded();

}

public static byte[] getRandomBytes(int noBytes, byte[] seed) throws NoSuchAlgorithmException {

SecureRandom secureRandom = SecureRandom.getInstance("SHA1PRNG");

if(seed != null) {

secureRandom.setSeed(seed);

}

byte[] randomBytes = new byte[noBytes];

secureRandom.nextBytes(randomBytes);

return randomBytes;

}

public static void storeData(String fileName, byte[] values) throws IOException {

File file = new File(fileName);

if(!file.exists()) {

file.createNewFile();

}

//DataOutputStream = used for primitive data types String, int etc

FileOutputStream fos = new FileOutputStream(file);

fos.write(values);

fos.close();

}

public static void main(String[] args) throws CertificateException, IOException, InvalidKeyException,

NoSuchAlgorithmException, SignatureException, NoSuchPaddingException, IllegalBlockSizeException,

BadPaddingException, KeyStoreException, UnrecoverableKeyException {

System.out.println("Proffesor's Public key is: ");

PublicKey ProfPubKey = getCertificatePublicKey("SimplePGP\_ISM.cer");

System.out.println("File number 1 result is: ");

System.out.println(ValidSignature("SAPExamSubject1.txt", ProfPubKey, "SAPExamSubject1.signature") + "\n");

System.out.println("File number 2 result is: ");

System.out.println(ValidSignature("SAPExamSubject2.txt", ProfPubKey, "SAPExamSubject2.signature")+ "\n");

System.out.println("File number 3 result is: ");

System.out.println(ValidSignature("SAPExamSubject3.txt", ProfPubKey, "SAPExamSubject3.signature")+ "\n");

byte[] AESRandomSessionKey = getRandomBytes(16, null);

System.out.println("AES session key: ");

System.out.println(getHex(AESRandomSessionKey));

PublicKey myPubKey = getCertificatePublicKey("ISMCertificateX509Assignment3.cer");

KeyStore ks = getKeyStore("ismkeystore.ks", "passks", "pkcs12");

PrivateKey myPriv = getPrivateKey(ks, "assignment3", "passks");

//System.out.println("My Private key: " + getHex(myPriv.getEncoded()));

byte[] encAESKey = encryptRSA(ProfPubKey, AESRandomSessionKey);

System.out.println("Encrypted AES key " + getHex(encAESKey) + "\n");

storeData("aes\_key.sec", encAESKey);

encryptECB("secret.txt", AESRandomSessionKey, "AES", "secret.enc");

File file = new File("secret.enc");

if(!file.exists()) {

System.out.println("The file does not exist!");

}

FileInputStream fis = new FileInputStream(file);

byte[] values = fis.readAllBytes();

fis.close();

byte[] encMessage = encryptRSA(myPriv, values);

System.out.println("Encrypted Message " + getHex(encMessage) + "\n");

storeData("response.sec", encMessage);

//checks for decryption side:

System.out.println("Just checks from here: \n");

byte[] decMessage = decryptRSA(myPubKey, encMessage);

System.out.println("Decrypted Message " + getHex(decMessage) + "\n");

storeData("decMessage.txt", decMessage);

System.out.println("Original enc message: " + getHex(values));

decryptECB("decMessage.txt", AESRandomSessionKey, "AES", "decryptedOriginalMessage.txt");

}

}