package ro.ism.sap.test;

import java.io.IOException;

import java.security.InvalidAlgorithmParameterException;

import java.security.InvalidKeyException;

import java.security.KeyStoreException;

import java.security.NoSuchAlgorithmException;

import java.security.PrivateKey;

import java.security.PublicKey;

import java.security.Security;

import java.security.UnrecoverableKeyException;

import java.security.cert.CertificateException;

import java.security.cert.CertificateFactory;

import javax.crypto.BadPaddingException;

import javax.crypto.IllegalBlockSizeException;

import javax.crypto.NoSuchPaddingException;

import javax.crypto.ShortBufferException;

import javax.xml.bind.DatatypeConverter;

import org.bouncycastle.jce.provider.BouncyCastleProvider;

public class Test {

public static void main(String[] args) throws NoSuchAlgorithmException, IOException, KeyStoreException, CertificateException, UnrecoverableKeyException, InvalidKeyException, NoSuchPaddingException, InvalidAlgorithmParameterException, ShortBufferException, IllegalBlockSizeException, BadPaddingException {

//Security.addProvider(new BouncyCastleProvider());

byte[] hash = Hash.getHash("mesaj.txt");

System.out.println("Hash file : "+ DatatypeConverter.printHexBinary(hash));

final String ksPass = "passks" ;

Cert.getStoreContent("ismkeystore.ks", ksPass.toCharArray());

PublicKey publicKey = Cert.getPublicKey("ismkeystore.ks", ksPass.toCharArray(), "ismkey1");

System.out.println(">>> PUB Key : "+ publicKey.toString());

PrivateKey privateKey = Cert.getPrivateKey("ismkeystore.ks", ksPass.toCharArray(), "ismkey1", "passism1".toCharArray());

System.out.println(">>> PRIV Key : "+ DatatypeConverter.printHexBinary(privateKey.getEncoded()));

// AES

byte[] pass = "passwordpassword".getBytes();

AESECB.encrypt("mesaj.txt", "mesaj.enc", pass);

System.out.println("Message encrypted!");

AESECB.decrypt("mesaj.enc", "mesaj.dec", pass);

System.out.println("Message decrypted!");

byte[] passDes = "password".getBytes();

DES.encrypt("mesaj.txt", "mesajDES.enc", passDes);

DES.decrypt("mesaj.enc", "mesajDES.dec", passDes);

}

}

public class Hash {

public static byte[] getHash(String fileIn, String alg) throws IOException, NoSuchAlgorithmException {

FileInputStream fis = new FileInputStream(fileIn);

MessageDigest md = MessageDigest.getInstance(alg);

byte[] buffer = new byte[8];

int noReadBytes = 0;

while((noReadBytes=fis.read(buffer))!=-1) {

md.update(buffer, 0, noReadBytes);

}

byte[] hash = md.digest();

fis.close();

return hash;

}

}

public class DES {

public static void encrypt(String fileIn, String fileOut, byte[] key) throws

FileInputStream fis = new FileInputStream(fileIn);

FileOutputStream fos = new FileOutputStream(fileOut);

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

SecretKey secret = new SecretKeySpec(key,"DES");

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

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

byte[] encryptedBuffer = null;

int noReadBytes = 0;

while((noReadBytes=fis.read(buffer))!=-1) {

encryptedBuffer = new byte[cipher.getOutputSize(noReadBytes)];

int noBytes = cipher.update(buffer, 0, noReadBytes, encryptedBuffer, 0);

fos.write(encryptedBuffer, 0, noBytes);

}

int noBytes = cipher.doFinal(encryptedBuffer, 0);

fos.write(encryptedBuffer, 0, noBytes);

fis.close();

fos.close();

}

public static void decrypt(String fileIn, String fileOut, byte[] key) throws

FileInputStream fis = new FileInputStream(fileIn);

FileOutputStream fos = new FileOutputStream(fileOut);

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

SecretKey secret = new SecretKeySpec(key, "DES");

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

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

byte[] decryptedBuffer = null;

int noReadBytes = 0;

while((noReadBytes=fis.read(buffer))!=-1) {

decryptedBuffer = new byte[cipher.getOutputSize(noReadBytes)];

int noBytes = cipher.update(buffer, 0, noReadBytes, decryptedBuffer, 0);

fos.write(decryptedBuffer, 0, noBytes);

}

int noBytes = cipher.doFinal(decryptedBuffer, 0);

fos.write(decryptedBuffer, 0, noBytes);

fis.close();

fos.close();

}

}

public class Cert {

public static void getStoreContent(String ksfile, char[] key) throws KeyStoreException, NoSuchAlgorithmException, CertificateException, IOException {

File file = new File(ksfile);

if(file.exists()) {

FileInputStream fis = new FileInputStream(file);

KeyStore ks = KeyStore.getInstance("JKS");

ks.load(fis, key);

Enumeration<String> keyAlias = ks.aliases();

while(keyAlias.hasMoreElements()) {

String alias = keyAlias.nextElement();

if(ks.isCertificateEntry(alias)) {

System.out.println(alias + " is a Certificate!");

}

if(ks.isKeyEntry(alias)) {

System.out.println(alias + " is a Key!");

}

}

}

}

public static PublicKey getCertKey(String certificate) throws CertificateException, IOException {

FileInputStream fis = new FileInputStream(new File(certificate));

BufferedInputStream bis = new BufferedInputStream(fis);

CertificateFactory cf = CertificateFactory.getInstance("X509");

PublicKey pk = cf.generateCertificate(bis).getPublicKey();

bis.close();

return pk;

}

public static PublicKey getPublicKey(String ksfile, char[] key, String alias ) throws KeyStoreException, NoSuchAlgorithmException, CertificateException, IOException {

FileInputStream fis = new FileInputStream(ksfile);

BufferedInputStream bis = new BufferedInputStream(fis);

KeyStore ks = KeyStore.getInstance("JKS");

ks.load(bis, key);

PublicKey pk = ks.getCertificate(alias).getPublicKey();

bis.close();

return pk;

}

public static PrivateKey getPrivateKey(String ksfile, char[] kspass, String alias, char[] pass) throws NoSuchAlgorithmException, CertificateException, IOException, KeyStoreException, UnrecoverableKeyException {

FileInputStream fis = new FileInputStream(ksfile);

BufferedInputStream bis = new BufferedInputStream(fis);

KeyStore ks = KeyStore.getInstance("JKS");

ks.load(bis, kspass);

PrivateKey privKey = (PrivateKey)ks.getKey(alias, pass);

return privKey;

}

}

public class AESECB {

public static void encrypt(String fileIn, String fileOut, byte[] key) throws

FileInputStream fis = new FileInputStream(fileIn);

FileOutputStream fos = new FileOutputStream(fileOut);

byte[] IV = new byte[key.length];

Random rand = new Random();

for(int i = 0; i < IV.length; i++) {

IV[i] = (byte)rand.nextInt(255);

}

fos.write(IV, 0, IV.length);

Cipher cipher = Cipher.getInstance("AES/CBC/PKCS5Padding");

SecretKey secret = new SecretKeySpec(key, "AES");

IvParameterSpec ivSpec = new IvParameterSpec(IV);

cipher.init(Cipher.ENCRYPT\_MODE, secret, ivSpec);

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

byte[] encBuffer = null;

int noReadBytes = 0;

while((noReadBytes = fis.read(buffer)) != -1) {

encBuffer = new byte[cipher.getOutputSize(noReadBytes)];

int noBytes = cipher.update(buffer, 0, noReadBytes, encBuffer, 0);

fos.write(encBuffer, 0, noBytes);

}

int noBytes = cipher.doFinal(encBuffer, 0);

fos.write(encBuffer, 0, noBytes);

fis.close();

fos.close();

}

public static void decrypt(String fileIn, String fileOut, byte[] key) throws

FileInputStream fis = new FileInputStream(fileIn);

FileOutputStream fos = new FileOutputStream(fileOut);

byte[] IV = new byte[key.length];

fis.read(IV, 0, IV.length);

Cipher cipher = Cipher.getInstance("AES/CBC/PKCS5Padding");

SecretKey secret = new SecretKeySpec(key, "AES");

IvParameterSpec ivSpec = new IvParameterSpec(IV);

cipher.init(Cipher.DECRYPT\_MODE, secret, ivSpec);

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

int noReadBytes = 0;

byte[] decrBuffer = null;

while((noReadBytes = fis.read(buffer)) != -1) {

decrBuffer = new byte[cipher.getOutputSize(noReadBytes)];

int noBytes = cipher.update(buffer, 0, noReadBytes, decrBuffer, 0);

fos.write(decrBuffer, 0, noBytes);

}

int noBytes = cipher.doFinal(decrBuffer, 0);

fos.write(decrBuffer, 0, noBytes);

fis.close();

fos.close();

}

}

RSA

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

Cipher cipher = Cipher.getInstance("RSA/None/NoPadding");

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

return cipher.doFinal(input);

}

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

Cipher cipher = Cipher.getInstance("RSA/None/NoPadding");

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

return cipher.doFinal(input);

}

=====================================================

//#include <stdio.h>

//#include <malloc.h>

//#include <openssl/md5.h>

//

//#define ADD\_STRING\_LENGTH 16

//#define MESSAGE\_CHUNK 128

//

//int main(int argc, char\*\*argv)

//{

// if(argc == 2) {

//

// // 1. it uses a file as input message

// FILE\* f = NULL;

// errno\_t err;

// MD5\_CTX ctx; // defined in md5.h

//

// unsigned char finalDigest[MD5\_DIGEST\_LENGTH]; // MD5\_DIGEST\_LENGTH defined in md5.h

// MD5\_Init(&ctx); // function called from openssl library binaries

//

// unsigned char\* fileBuffer = NULL;

//

// err = fopen\_s(&f, argv[1], "rb");

// if(err == 0) {

// fseek(f, 0, SEEK\_END);

// int fileLen = ftell(f);

// fseek(f, 0, SEEK\_SET);

//

// fileBuffer = (unsigned char\*)malloc(fileLen); // heap mem allocation for the fle content

// fread(fileBuffer, fileLen, 1, f); // read the file content and move it to the allocated heap memory

// unsigned char\* tmpBuffer = fileBuffer;

//

// while (fileLen > 0) {

// if (fileLen > MESSAGE\_CHUNK) {

// MD5\_Update(&ctx, tmpBuffer, MESSAGE\_CHUNK); // MD5\_Update - one single iteration for A, B, C, D update

// }

// else {

// MD5\_Update(&ctx, tmpBuffer, fileLen); // MD%\_Update - called for the latest input message block (less 16 butes as length)

// }

// fileLen -= MESSAGE\_CHUNK; // update for the remaining input message

// tmpBuffer += MESSAGE\_CHUNK; // update the remaining input message

// }

//

// MD5\_Final(finalDigest, &ctx); // final transformation and moving the A, B, C and D blocks from CTX structure to my md5 buffer

//

// // printing the result at console

// int count = 0;

// printf("\nMD using an input file = ");

// for( int i = 0; i < MD5\_DIGEST\_LENGTH; i++) {

// printf( "%2X", finalDigest[i] );

// printf( " " );

// }

//

// // 2. easier md5 generation for inputs with 16 bytes lengths

// unsigned char addChars[ADD\_STRING\_LENGTH];

// for (int i = 0; i < ADD\_STRING\_LENGTH; i++)

// addChars[i] = i + 15;

// printf("\n%s\n", addChars);

// MD5\_Transform(&ctx, addChars);

// MD5\_Final(finalDigest, &ctx);

//

// count = 0;

// printf("\nMD using local buffer (addChars) = "); // CTX structure is not re-initialized

// for( int i=0; i<MD5\_DIGEST\_LENGTH; i++) {

// printf( "%2X", finalDigest[i] );

// printf( " " );

// }

//

// printf("\n");

//

// // 3. one call md5 generation using addChars as input message

// unsigned char \*result = 0;

// result = MD5(addChars, ADD\_STRING\_LENGTH, finalDigest);

//

// count = 0;

// printf("\nMD using local buffer (from finalDigest)= "); // CTX structure is not re-initialized

// for( int i=0; i<MD5\_DIGEST\_LENGTH; i++) {

// printf( "%2X", finalDigest[i] );

// printf( " " );

// }

//

// count = 0;

// printf("\nMD using local buffer (from result) = "); // CTX structure is not re-initialized

// for( int i=0; i<MD5\_DIGEST\_LENGTH; i++) {

// printf( "%2X", result[i] );

// printf( " " );

// }

// printf("\n");

//

// fclose(f);

// }

//

// }

// else {

// printf("\n Usage Mode: ProgMainMD5.exe fSrc.txt");

// return 1;

// }

//

// return 0;

//}