Security

Assignment 10, Wednesday 3 December, 2008

Goals: After successfully completing this exercise you should have a working grasp of the implementation of digital signatures and encrypting in *Java Cryptography Architecture (JCA)*.

Deadline: Friday December 12 or earlier. You have to send to **O-L-H-A** in the body of your e-mail: Java file "testjeMyname.java" and the "output text file of your program". **Attention: Flavio is leaving on vacations on 11/12. He will not be reachable by e-mail or any other way!!!**

Hints on study:

- Find
  `http://java.sun.com/javase/6/docs/technotes/guides/security/crypto/CryptoSpec.html`.

  Bart has given you examples on the section *the Message Digest Class*. Now read *the Signature Class* and *the Cipher Class*.

- Look at the java code below. It is slightly modified Bart's code. What is different?
  - To use symmetric cryptography (such as DES algorithm), we add
    `import javax.crypto.*;;`
  - we have added routines that convert a byte array into a hexadecimal string, print out a hexadecimal string, print out byte array as a bit string, print out byte array as an array of characters;
  - comments on binary presentations of integers in Java;
  - an example using the Signature Class.

- Experiment with the program: use different input strings to hash, encrypt and sign. Use different hash-function algorithms. Use different key length. Pay attention: in some cases you may chose the length of the key which is not good for a chosen encryption algorithm. Then you will get the exception that tells you which length you are allowed to use.

Tasks:

1. Extend the program where you see the comment "Your code!" and send to Olha. Use only hexadecimal string printing to print out a cypher text or hashes. Basically, you have to do two things:
   - Similarly to our example of a signature with "MD5withRSA", sign the message "I'm very happy with the midterm exam!" with "DSA" algorithm, print out the hexadecimal presentation of the signature and verify the signature.
   - Write code to experiment with encryption/decryption using *Cipher Class*. Look at the section "Managing Algorithm Parameters" for examples. Experiment with your code. Send us the same text "I'm very happy with the midterm exam!", encrypted using "AES". Print out the encryption in the hexadecimal representation. Decrypt and print out the result of the decryption as a string.
2. (Optional) Feel free to optimise, to fix bugs, do self-study and have fun with Java Secure!

```
import java.math.*;
import java.security.*;
import java.security.interfaces.*;
import javax.crypto.*;
```

```
public class testje {

    static SecureRandom random = new SecureRandom();
    static private int RSAlength = 1024;


    /**
     * Converts a byte array into capitalized hexadecimal text.
     * @param text The byte array to convert.
     * @return Capitalized hexadecimal text representation of
     *         <code>text</code>.
     */
    public static String bytesToHexString(byte[] text) {
       return bytesToHexString(text,0,text.length);
    }

    /**
     * Converts part of a byte array into capitalized hexadecimal text.
     * Conversion starts at index <code>offset</code> until (excluding)
     * index <code>offset+length</code>.
     * @param text The byte array to convert.
     * @param offset Where to start.
     * @param length How many bytes to convert.
     *
     * @return Capitalized hexadecimal text representation of
     *         <code>text</code>.
     */

    private static String bytesToHexString(byte[] text,
                                           int offset,
                                           int length) {
       String result = "";
       for (int i=0; i<length; i++) {
          int b = (text[offset+i] & 0xFF);
          if (b < 0x10) {
             result += "0"+Integer.toHexString(b);
          } else {
             result += Integer.toHexString(b);
          }
         if((i+1)%25 == 0) result += "\n";
         else result +=" ";

       }
       return result.toUpperCase();
    }



    /**
     * PAS OP: since we use integers to present hash strings, we should now the following.
     * Contrary to C/C++, all integers  (i.e. the types "int", "short", "byte", "long"
     * and the class "BigInteger") in Java are SIGNED,
     * that is the leftmost bit in the bit presentation of an integer is a sign bit.
```

```
     * Once it is "1", the bit presentation is treated as a negative number
     * in its two's-complement presentation (google, wiki for more detail).
    */

//  Printing a byte as an array of bits
    public static void printByte(byte b){

     int tmp;
      if (b<0) {
      System.out.print("1");

      }
      else {System.out.print("0");
           }

      tmp = (127 & b); // turn the oldest bit of b to 0

      int buf[]= new int[7];
      for (int i=0; i<7; i++) {
     buf[i]= tmp % 2;
     tmp = tmp / 2;
     };
     for (int i=0; i<7; i++) {
     System.out.print(buf[6-i]);
     };


    }

    //Printing bytes[] one-by-one, as arrays of bits
    public static void printBytes(byte[] arg){
     int count;
     for (count=0; count < arg.length; count++)
        {
        printByte(arg[count]);
        System.out.print(" ");
        if ((count +1) % 8 == 0){System.out.println();}
        };
        if ((count) % 8 != 0) {System.out.println(); }
    }

//  Printing bytes[] as char[]
    public static void printBytesAsChars(byte[] arg){
     int count;
     for (count=0; count < arg.length; count++)
        {
        System.out.print((char) arg[count]);
        };
        System.out.println();
    }


    public static void main (String[] args) throws Exception {
```

```
    String s = "Here must be your test string";

     System.out.println("==== Hash test");
// Alternatives "MD5" or "SHA1"
MessageDigest mdHash = MessageDigest.getInstance( "MD5" );

// Feed the Byte Array containing the string to the hash
mdHash.update(s.getBytes());
// Compute the hash
byte[] raw = mdHash.digest();



/**
 * We use the class "BigInteger" to present byte-arrays in a nice form.
 * Look documentation on the constructor BigInteger(byte[] raw):
 * it treats "raw" as a two's-complement presentation.
 * The oldest bit is considered as a sign bit. This is undesirable
 * when we want to see the hash "b" as a binary presentation of some
 * integer. We add a zero-byte in front of "raw", so the constructor
 * BigInteger "thinks" that it is a presentation of a nonnegative number.
 */


byte[] raw0 = new byte[raw.length+1];
raw0[0]=0;
for (int count=1; count < raw0.length; count++)
    {
    raw0[count]=raw[count-1];
    };

System.out.println("Hash value of the string \"" + s
           + "\" in the bit-representation is\n");
    printBytes(raw);
    System.out.println("\nwith " + raw.length + " bytes length.");
    System.out.println("===============");

    System.out.println("Hash value of the string \"" +
                     s + "\" in the hexadecimal representation is\n");
    System.out.println(bytesToHexString(raw));
    System.out.println("===============");

    System.out.println("Hash value of the string \"" + s +
              "\" as a nonegative big integer is\n" +
     new BigInteger(raw0));
    System.out.println("Compare: hash value of the string \"" + s +
       "\" as a possibly big integer is\n" +
     new BigInteger(raw));
    System.out.println(" ");
    System.out.println("===============");

KeyPairGenerator RSAkeyGen = KeyPairGenerator.getInstance("RSA");
RSAkeyGen.initialize(RSAlength);
KeyPair RSAkeypair = RSAkeyGen.generateKeyPair();
```

```
RSAPublicKey RSApubkey = (RSAPublicKey)RSAkeypair.getPublic();
BigInteger RSAmod = RSApubkey.getModulus();
  BigInteger E = RSApubkey.getPublicExponent();
  RSAPrivateCrtKey RSAprivkey = (RSAPrivateCrtKey)RSAkeypair.getPrivate();
  BigInteger P = RSAprivkey.getPrimeP();
  BigInteger Q = RSAprivkey.getPrimeQ();
  BigInteger EulerMod =
    P.subtract(BigInteger.ONE).multiply(Q.subtract(BigInteger.ONE));
  BigInteger D = RSAprivkey.getPrivateExponent();
System.out.println("==== RSA test ====== \n" +
   "RSA key check works: " +
   RSAmod.equals(P.multiply(Q)));
System.out.format("Public key modulus (N) %d and exponent (E) %d\n",
  RSAmod, E);
System.out.println("Private key exponent (D): " + D);
System.out.println("Product of E and D is: " +
   E.multiply(D).mod(EulerMod));
  //
  BigInteger message = new BigInteger(RSAlength-1, random);
System.out.println("===\nEncryption by hand test of message : " +
   message);
  BigInteger enc = message.modPow(E, RSAmod);
  // System.out.println("Encrypted: " + enc);
  BigInteger dec = enc.modPow(D, RSAmod);
  System.out.println("Encryption-Decryption test " +
     ((message.equals(dec) ? "succeeded" : "failed")));

  ////////////////////////////////////////////


  System.out.println("============");
  System.out.println("Testing Signatures");

  //Signature with MD5withRSA
  Signature MySignMD5 = Signature.getInstance("MD5withRSA");

  //Signing
  MySignMD5.initSign(RSAprivkey);
  MySignMD5.update(s.getBytes());
  byte[] SignatureBytesMD5 = MySignMD5.sign();

  System.out.println("==== Signed message MD5=====:");
  printBytes(SignatureBytesMD5);
  System.out.println("\nwith " + SignatureBytesMD5.length + " bytes length.");
  System.out.println("In the hexadecimal representation:\n" + bytesToHexString(SignatureBytesMD5)

  //Verifying

  MySignMD5.initVerify(RSApubkey);
  MySignMD5.update(s.getBytes());
  if (MySignMD5.verify(SignatureBytesMD5))
  {System.out.println("Verify: the MD5 Signature is OK");}
  else {System.out.println("Verify: the MD5 Signature is NOT OK");};
  System.out.println(" ");
```

```
  Signature MySignDSA = Signature.getInstance("DSA");

  // DSA Key Pair Generation
        //Your Code!!



  //Signing
        //Your Code!!


  //Verifying
  //Your Code!!



  // Symmetric encrypting/decrypting

        String algorithm = "AES"; // Experiment with the other cyphers
  int length = 128;

//Your Code! Look at Cypher class in the Reference Guide.
        //Examples are in the section ‘‘Managing Algorithm Parameters’’.


    }


}
```