HACETTEPE UNIVERSITY DEPARTMENT OF COMPUTER ENGINEERING

November 18, 2019

Group Name/Surname	: Mehmet Taha USTA & Burcu ÖZTAŞ
Numbers	: 21527472 - 21483435
Course	: BBM-465 Information Security Lab.
Experiment	: Assignment 2
Subject	: Asymmetric Cryptography, Hashing& Digital Signatures
Due Date	: 18/11/2019 - 23:59
Advisor	: Dr. Ahmet Selman BOZKIR
Main Program	: Java

1 Problem

At this project, It is required which subject is Asymmetric Cryptography,Md5 and Digital Signatures. Although symmetric ciphers have strong features, the mediator between the two sources is extremely vulnerable to the attacks of the source. Asymmetric encryption provides the solution to this problem. As is stated, in this experiment, we are expected to develop of a licensing framework by utilizing the methods of asymmetric cryptography, MD5 and digital signatures. The following lines state the requirements of out assignment.

2 Method & Solution of the Problem

The program starts from main.java First, the client creates. After creating the client, it checks the existence of the license.txt file with the check function. The check() function returns a boolean value. Collects credentials if the file exists. The information is encrypted with the RSA algorithm using the encrypt() function. The encrypted information is processed by the Md5 hash algorithm with the Md5() function. With the result from the Md5 hash algorithm, the license file is verified. Validation is performed with the verification() function. The verification() function returns a boolean value. If validation fails, the process() function is called and the license file is rebuilt. If there is no file, the process() function is called. The process() function collects credentials. The collected credentials are encrypted with the Rsa algorithm using the Encrypt function and public key. The encrypted data is then sent to the Md5 () function for verification. The result in the Md5() function is stored for later use. The encrypted data is sent to the Licence Manager with the send() function. The data sent is decrypted with the decrypted() function and the private key in the Licence Manager.
The decoded data is hashed by the Md5 algorithm with the Md5() function. Encrypted data is processed with the SHA256withRSA algorithm and private key in the sign() function. The result of the Sign function is sent to the client. The signature verification() function from the client is validated with hashed Md5 and public key, and the boolean value is returned. If the Boolean value is True, the signature is written to the file. If it is False, it gives an error.

3 Socket Programming

The program starts from Socket_server.java. Then socket_client.java is executed. After creating the client, it checks the existence of the license.txt file with the check function. The check() function returns a boolean value. Collects credentials if the file exists. The information is encrypted with the RSA algorithm using the encrypt() function. The encrypted information is processed by the Md5 hash algorithm with the Md5() function. With the result from the Md5 hash algorithm, the license file is verified. Validation is performed with the verification() function. The verification() function returns a boolean value. If validation fails, the process() function is called and the license file is rebuilt. If there is no file, the process() function is called. The process() function collects credentials. The collected credentials are encrypted with the Rsa algorithm using the Encrypt function and public key. The encrypted data is then sent to the Md5 () function for verification. The result in the Md5() function is stored for later use. The encrypted data is sent to the Server with the DataOutputStream object. The data sent is decrypted with the decrypted() function and the private key in the Server. The decoded data is hashed by the Md5 algorithm with the Md5() function. Encrypted data is processed with the SHA256withRSA algorithm and private key in the sign() function. The result of the Sign function is sent to the client. The signature verification() function from the client is validated with hashed Md5 and public key, and the boolean value is returned. If the Boolean value is True, the signature is written to the file. If it is False, it gives an error.

4 Socket_client.java

```
// A Java program for a ClientSide
import java.net.*;
import java.nio.file.Files;
import java.nio.file.Paths;
import java.security.KeyFactory;
import java.security.MessageDigest;
import java.security.PublicKey;
import java.security.Signature;
import java.security.spec.X509EncodedKeySpec;
import java.util.ArrayList;
import java.util.Base64;
import java.util.Scanner;

import javax.crypto.Cipher;



import java.io.*;

public class Socket_client {
   // initialize socket and input output streams

   private Socket socket = null;
   private DataInputStream dinput = null;
   private DataOutputStream dout = null;
   // initialize assignment variables
   private String username, serial_number;
   private static String mac_adress, disk_serial_number, motherboard_id;

   // keys
   private static PublicKey public_key;
   private static byte[] enc_data, signature;
   public byte[] hashed_text;

   // constructor to put ip address and port
   public Socket_client(String address, int port) {
      // establish a connection
      try {
         socket = new Socket(address, port);
         this.username = System.getProperty("user.name");
         this.serial_number = this.serial_number = readfile().get(0);;
         this.mac_adress = get_mac_adress();
         this.disk_serial_number = get_serial_number("diskdrive");
         this.motherboard_id = get_serial_number("baseboard");
         this.public_key = get_public_key_from_file();

      } catch (Exception e) {
         e.printStackTrace();
      }

   }
   public static ArrayList<String> readfile() {
      ArrayList<String> line = null;
      try {
         line = new ArrayList<String>();
         // the file to be opened for reading
         FileInputStream fis = new FileInputStream("user_serial.txt");
         Scanner sc = new Scanner(fis); // file to be scanned
```

```
      // returns true if there is another line to read
      while (sc.hasNextLine()) {
         //System.out.println(sc.nextLine()); // returns the line that was skipped
         line.add(sc.nextLine());
      }
      sc.close(); // closes the scanner
   } catch (IOException e) {
      e.printStackTrace();
   }
   return line ;
}
/*this function returns disk_serial_number or motherboard_id by parameter*/
private static String get_serial_number(String input) {
   StringBuilder sb = null;
   try {
      // diskdrive or baseboard
      String sc = "cmd /c" + "wmic " + input + " get serialnumber";

      Process p = Runtime.getRuntime().exec(sc);
      p.waitFor();

      BufferedReader reader = new BufferedReader(new InputStreamReader(p.getInputStream()));

      String line;
      sb = new StringBuilder();

      while ((line = reader.readLine()) != null) {
         sb.append(line);
      }

   } catch (Exception e) {
      e.printStackTrace();
   }
   return sb.substring(sb.toString().lastIndexOf("r") + 1).trim();
}
/*returns the mac address of the machine if there is an active internet connection*/
private static String get_mac_adress() {
   InetAddress ip;
   StringBuilder sb = null;
   try {
      ip = InetAddress.getLocalHost();

      NetworkInterface network = NetworkInterface.getByInetAddress(ip);

      byte[] mac = network.getHardwareAddress();
      if(mac == null) {
         mac = "Standart".getBytes();
      }
      sb = new StringBuilder();
      for (int i = 0; i < mac.length; i++) {
         sb.append(String.format("%02X%s", mac[i], (i < mac.length - 1) ? "-" : ""));
      }

   } catch (Exception e) {

      e.printStackTrace();

   }
   return sb.toString();
}
/*this function reads the public key from the file and stores it in memory*/
private static PublicKey get_public_key_from_file() {
```

```
    PublicKey return_key = null;
    try {
        byte[] keyBytes = Files.readAllBytes(Paths.get("public.key"));

        X509EncodedKeySpec spec = new X509EncodedKeySpec(keyBytes);
        KeyFactory kf = KeyFactory.getInstance("RSA");
        return_key = kf.generatePublic(spec);

    } catch (Exception e) {
        e.printStackTrace();
    }
    return return_key;
}

// this function encrypts data using public key and rsa algorithm
private static byte[] encrypt(String plainText, PublicKey publicKey) {
    byte[] cipherText = null;

    try {
        Cipher encryptCipher = Cipher.getInstance("RSA");
        encryptCipher.init(Cipher.ENCRYPT_MODE, publicKey);

        cipherText = encryptCipher.doFinal(plainText.getBytes("UTF8"));
    } catch (Exception e) {
        e.printStackTrace();
    }
    return cipherText;
}
/*this function hashes data by md5 algorithm*/
private static byte[] md5(String data) {
    byte[] resultByte = null;
    try {
        MessageDigest messageDigest = MessageDigest.getInstance("MD5");
        messageDigest.reset();
        messageDigest.update(data.getBytes("UTF8"));
        resultByte = messageDigest.digest();
    } catch (Exception e) {
        e.printStackTrace();
    }

    return resultByte;
}
/*this function verify signature using public key and SHA256withRSA algorithm*/
private static boolean verification(byte[] hashed_text) {
    boolean bool = false;
    try {
        Signature sign_verify = Signature.getInstance("SHA256withRSA");
        sign_verify.initVerify(public_key);
        sign_verify.update(hashed_text);
        bool = sign_verify.verify(signature);
    } catch (Exception e) {
        e.printStackTrace();
    }
    return bool;
}

// Method which write the bytes into a file
private static void writeByte(byte[] bytes) {
    try {
        // Initialize a pointer in file using OutputStream
        OutputStream os = new FileOutputStream(new File("license.txt"));
```

```
      // Starts writing the bytes in it
      os.write(bytes);

      // Close the file
      os.close();
   }

   catch (Exception e) {
      System.out.println("Exception: " + e);
   }
}

/**
 * This method uses java.io.FileInputStream to read file content into a byte
 * array
 *
 * @param file
 * @return
 */
private static byte[] readFileToByteArray(File file) {
   FileInputStream fis = null;
   // Creating a byte array using the length of the file
   // file.length returns long which is cast to int
   byte[] bArray = new byte[(int) file.length()];
   try {
      fis = new FileInputStream(file);
      fis.read(bArray);
      fis.close();

   } catch (IOException ioExp) {
      ioExp.printStackTrace();
   }
   return bArray;
}
//helper function for file exist
public static boolean check() {
   File temp = new File("license.txt");
   return temp.exists();
}
/*the assignment process begins*/
private static void process(Socket_client client1) {
   System.out.println("Client started...");
   System.out.println("My Mac: " + client1.mac_adress);
   System.out.println("My DiskID: " + client1.disk_serial_number);
   System.out.println("My Motherboard ID: " + client1.motherboard_id);
   System.out.println("LicenseManager service started...");


   // "user-serialid-hw spec info"
   String plain_text = client1.username + "$" + client1.serial_number + "$" +
       client1.mac_adress + "$"+ client1.disk_serial_number + "$" + client1.motherboard_id;
   System.out.println("Client -- Raw Licence Text: " + plain_text);

   client1.enc_data = client1.encrypt(plain_text, client1.public_key);
   System.out.println("Client -- Encryted License Text: " +
       Base64.getEncoder().encodeToString(client1.enc_data));

   client1.hashed_text = client1.md5(plain_text);
   System.out.println("Client -- MD5fied Plain License Text: " +
       Base64.getEncoder().encodeToString(client1.hashed_text));

   System.out.println("Server -- Server is being requested...");
```

```
    /*sending data*/
    try {
       client1.dout = new DataOutputStream(client1.socket.getOutputStream());
       client1.dout.writeInt(client1.enc_data.length);
       client1.dout.write(client1.enc_data);
    } catch (IOException e) {
       e.printStackTrace();
    }
    // for synchronization
    try {
       Thread.sleep(1000);
    } catch (InterruptedException e) {
       e.printStackTrace();
    }
    try {
       /*accept data*/
       client1.dinput = new DataInputStream(client1.socket.getInputStream());
       int length = client1.dinput.readInt();              // read length of incoming message
       if(length>0) {
           client1.signature = new byte[length];
           client1.dinput.readFully(client1.signature, 0, client1.signature.length); // read
               the message
       }
    } catch (IOException e) {
       // TODO Auto-generated catch block
       e.printStackTrace();
    }

    if (client1.verification(client1.hashed_text)) {
       // store licence in license.txt file
       client1.writeByte(client1.signature);
       System.out.println("Client -- Succeed. The License file content is secured and signed
           by the server");

    } else {
       System.out.println("Client -- verification failed");
    }
}

public static void main(String args[]) {
    Socket_client client1 = new Socket_client("127.0.0.1", 5000);
    // check license file if existed
    if (!client1.check()) {
       process(client1);
    } else {
       // System.out.println("license file created, checking");
       // "user-serialid-hw spec info"
       String plain_text = client1.username + "$" + client1.serial_number + "$" +
           client1.mac_adress + "$" + client1.disk_serial_number + "$" +
           client1.motherboard_id;
       client1.enc_data = client1.encrypt(plain_text, client1.public_key);
       client1.hashed_text = client1.md5(plain_text);
       client1.signature = client1.readFileToByteArray(new File("license.txt"));
       if (client1.verification(client1.hashed_text)) {
          System.out.println("Succeed. The license is correct.");
       } else {
          System.out.println("The license file has been broken!!");
          // re-execute the licensing process again to obtain a valid digital signature
          process(client1);
       }

    }
```

```
      // close connection
      try {
         if(client1.dinput != null) {
            client1.dinput.close();
         }
         if(client1.dout != null) {
            client1.dout.close();
         }
         if(client1.socket != null) {
            client1.socket.close();
         }
      } catch (Exception e) {
         e.printStackTrace();
      }

   }
}
```

5 Socket_server.java

```
// A Java program for a Serverside
import java.net.*;
import java.nio.file.Files;
import java.nio.file.Paths;
import java.security.KeyFactory;
import java.security.MessageDigest;
import java.security.PrivateKey;
import java.security.PublicKey;
import java.security.Signature;
import java.security.spec.PKCS8EncodedKeySpec;
import java.security.spec.X509EncodedKeySpec;
import java.util.Base64;

import javax.crypto.Cipher;

import java.io.*;

public class Socket_server {

// constructor with port

   // initialize socket and input stream
   private Socket socket = null;
   private ServerSocket server = null;
   private DataInputStream dIn = null;
   private DataOutputStream dOut = null;

   // initialize variable for assignment2
   private static PrivateKey privateKey;
   public PublicKey publicKey;
   public static byte[] enc_Data;
   public static String decrypted_text;
   public static byte[] hashed_text;
   public static byte[] signature;

   public Socket_server(int port) {
      // starts server and waits for a connection
      try {
         server = new ServerSocket(port);
         this.publicKey = get_public_key_from_file();
         this.privateKey = get_private_key_from_file();
         System.out.println("LicenseManager server started");
         System.out.println("Waiting for a client ...");
         socket = server.accept();
         // takes input from the client socket
         dIn = new DataInputStream(socket.getInputStream());

         try {
            Thread.sleep(1000);
         } catch (InterruptedException e) {
            e.printStackTrace();
         }
         /*InputStreamReader blocks EOFexception*/
         InputStreamReader reader = new InputStreamReader(dIn);
         if(reader.ready()) {
            /*accept data*/
            int length = dIn.readInt(); // read length of incoming message
            if (length > 0) {
```

```
            enc_Data = new byte[length];
            dIn.readFully(enc_Data, 0, enc_Data.length); // read the message
         }
         System.out.println("Server -- Incoming Encryted Text: " +
             Base64.getEncoder().encodeToString(enc_Data));

         Socket_server.decrypted_text = Socket_server.decrypt(Socket_server.enc_Data);
         System.out.println("Server -- Decrypted Text: " + Socket_server.decrypted_text);

         Socket_server.hashed_text = Socket_server.md5(Socket_server.decrypted_text);
         System.out.println("Server -- MD5fied Plain License Text: " +
             Base64.getEncoder().encodeToString(Socket_server.hashed_text));

         Socket_server.signature = Socket_server.sign(Socket_server.hashed_text);
         System.out.println("Server -- Digital Signature: " +
             Base64.getEncoder().encodeToString(Socket_server.signature));

         /*sending data*/
         dOut = new DataOutputStream(socket.getOutputStream());
         dOut.writeInt(Socket_server.signature.length); // write length of the message
         dOut.write(Socket_server.signature);       // write the message


      }

      // close connection
      if(dOut != null) {
         dOut.close();
      }
      if(reader != null) {
         reader.close();
      }
      if(socket != null) {
         socket.close();
      }
      if(dIn != null) {
         dIn.close();
      }

   } catch (IOException i) {
      System.out.println(i);
   }
}
// this function decrypts data using private key and rsa algorithm
public static String decrypt(byte[] buffer) {
   try {
      Cipher decryptCipher = Cipher.getInstance("RSA");
      decryptCipher.init(Cipher.DECRYPT_MODE, Socket_server.privateKey);
      byte[] result = decryptCipher.doFinal(buffer);
      return new String(result, "UTF8");
   } catch (Exception e) {
      e.printStackTrace();
   }
   return null;
}
/*this function hashes data by md5 algorithm*/
public static byte[] md5(String data) {
   byte[] resultByte = null;
   try {
      MessageDigest messageDigest = MessageDigest.getInstance("MD5");
      messageDigest.reset();
      messageDigest.update(data.getBytes("UTF8"));
```

```
            resultByte = messageDigest.digest();
        } catch (Exception e) {
            e.printStackTrace();
        }

        return resultByte;
    }
    /*this function signs data using private key and SHA256withRSA algorithm*/
    public static byte[] sign(byte[] hashed_by_md5) {
        byte[] hashed = null;
        try {
            Signature privateSignature = Signature.getInstance("SHA256withRSA");
            privateSignature.initSign(privateKey);
            privateSignature.update(hashed_by_md5);
            hashed = privateSignature.sign();

        } catch (Exception e) {
            e.printStackTrace();

        }
        return hashed;
    }
    /*this function reads the private key from the file and stores it in memory*/
    private static PrivateKey get_private_key_from_file() {
        PrivateKey return_private = null;
        try {
            byte[] keyBytes = Files.readAllBytes(Paths.get("private.key"));

            PKCS8EncodedKeySpec spec = new PKCS8EncodedKeySpec(keyBytes);
            KeyFactory kf = KeyFactory.getInstance("RSA");
            return_private = kf.generatePrivate(spec);
        } catch (Exception e) {
            e.printStackTrace();
        }
        return return_private;

    }
    /*this function reads the public key from the file and stores it in memory*/
    public static PublicKey get_public_key_from_file() {
        PublicKey return_key = null;
        try {
            byte[] keyBytes = Files.readAllBytes(Paths.get("public.key"));

            X509EncodedKeySpec spec = new X509EncodedKeySpec(keyBytes);
            KeyFactory kf = KeyFactory.getInstance("RSA");
            return_key = kf.generatePublic(spec);

        } catch (Exception e) {
            e.printStackTrace();
        }
        return return_key;
    }

    public static void main(String args[]) {
        Socket_server server = new Socket_server(5000);
    }

}
```