КІТ-м120Б
Богомаз Олексій

Лабораторна робота №7

Тема: Створення ліцензійного ключа.

Мета роботи: Дослідити і порівняти існуючі механізми створення і перевірки валідності ліцензійних ключів.

Завдання:

Механізм генерації ліцензійоного ключа №1.

- створити пару ключів (приватний та публічний)
- створити додаток, що генерує ліцензійний ключ, що має інформацію о кінцевому корисувачі. У кінці цього ЛК повина бути ЕЦП - геш сума, для калькуляції якої використовувся приватний ключ
- створити додаток, що читає ліцензійний ключ та виводить його дані на екран (у тому числі, чи валідний "підпис", використовуючі публічний ключ)

Механізм генерації ліцензійоного ключа №2. Створити клієнт-серверний додаток. Задача веб-сервера : отримати строку-ключ та повернути відполвідь, чи валідний цей ключ (перелік ключів зберігається на сервері, метод зберігання не має значення). Задача кліента: Спитати у користувача ключ, отримати відповідь, на базі якої вивести, чи має змогу користувач далі користатися даним ПЗ
Варіант: 7;

Виконання роботи.

Для виконання роботи використаємо - Java.

Код програм :

GenerateKeys.java генерація пари приватний-публічний ключ

```
import java.io.File;
import java.io.FileOutputStream;
import java.io.IOException;
import java.security.KeyPair;
import java.security.KeyPairGenerator;
import java.security.NoSuchAlgorithmException;
import java.security.NoSuchProviderException;
import java.security.PrivateKey;
import java.security.PublicKey;

public class GenerateKeys {

    private KeyPairGenerator keyGen;
    private KeyPair pair;
    private PrivateKey privateKey;
    private PublicKey publicKey;

    public GenerateKeys(int keylength) throws NoSuchAlgorithmException,
NoSuchProviderException {
        this.keyGen = KeyPairGenerator.getInstance("RSA");
        this.keyGen.initialize(keylength);
    }
```

```
    public void createKeys() {
        this.pair = this.keyGen.generateKeyPair();
        this.privateKey = pair.getPrivate();
        this.publicKey = pair.getPublic();
    }

    public PrivateKey getPrivateKey() {
        return this.privateKey;
    }

    public PublicKey getPublicKey() {
        return this.publicKey;
    }

    public void writeToFile(String path, byte[]key) throws IOException {
        File f = new File(path);
        f.getParentFile().mkdirs();

        FileOutputStream fos = new FileOutputStream(f);
        fos.write(key);
        fos.flush();
        fos.close();
    }

    public static void main(String[]args) {
        GenerateKeys gk;
        try {
            gk = new GenerateKeys(512);
            gk.createKeys();
            gk.writeToFile("KeyPair/publicKey", gk.getPublicKey().getEncoded());
            gk.writeToFile("KeyPair/privateKey",
gk.getPrivateKey().getEncoded());
        } catch (NoSuchAlgorithmException | NoSuchProviderException e) {
            System.err.println(e.getMessage());
        } catch (IOException e) {
            System.err.println(e.getMessage());
        }

    }

}
```

KeyGen.java генерує ліцензійний ключ

```
import java.security.MessageDigest;
import java.security.NoSuchAlgorithmException;
import java.util.Base64;
import java.io.File;
import java.io.FileInputStream;
import java.io.FileOutputStream;
import java.io.IOException;
import java.io.UnsupportedEncodingException;
import java.nio.file.Files;
import java.security.GeneralSecurityException;
import java.security.InvalidKeyException;
import java.security.KeyFactory;
import java.security.NoSuchAlgorithmException;
import java.security.PrivateKey;
import java.security.PublicKey;
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;
```

```
public class KeyGen {
    private Cipher cipher;

    public KeyGen() throws NoSuchAlgorithmException, NoSuchPaddingException {
        this.cipher = Cipher.getInstance("RSA");
    }


//https://docs.oracle.com/javase/8/docs/api/java/security/spec/PKCS8EncodedKeySpec.html
    public PrivateKey getPrivate(String filename) throws Exception {
        byte[]keyBytes = Files.readAllBytes(new File(filename).toPath());
        PKCS8EncodedKeySpec spec = new PKCS8EncodedKeySpec(keyBytes);
        KeyFactory kf = KeyFactory.getInstance("RSA");
        return kf.generatePrivate(spec);
    }


//https://docs.oracle.com/javase/8/docs/api/java/security/spec/X509EncodedKeySpec.html
    public PublicKey getPublic(String filename) throws Exception {
        byte[]keyBytes = Files.readAllBytes(new File(filename).toPath());
        X509EncodedKeySpec spec = new X509EncodedKeySpec(keyBytes);
        KeyFactory kf = KeyFactory.getInstance("RSA");
        return kf.generatePublic(spec);
    }

    private void writeToFile(File output, byte[]toWrite)
            throws IllegalBlockSizeException, BadPaddingException, IOException {
        FileOutputStream fos = new FileOutputStream(output);
        fos.write(toWrite);
        fos.flush();
        fos.close();
    }

    public String encryptText(String msg, PrivateKey key)
            throws NoSuchAlgorithmException, NoSuchPaddingException,
            UnsupportedEncodingException, IllegalBlockSizeException,
            BadPaddingException, InvalidKeyException {
        this.cipher.init(Cipher.ENCRYPT_MODE, key);
        return new
String(Base64.getEncoder().encode(cipher.doFinal(msg.getBytes("UTF-8"))));
    }

    public String decryptText(String msg, PublicKey key)
            throws InvalidKeyException, UnsupportedEncodingException,
            IllegalBlockSizeException, BadPaddingException {
        this.cipher.init(Cipher.DECRYPT_MODE, key);
        return new String(cipher.doFinal(Base64.getDecoder().decode(msg)), "UTF-
8");
    }

    public byte[]getFileInBytes(File f) throws IOException {
        FileInputStream fis = new FileInputStream(f);
        byte[]fbytes = new byte[(int) f.length()];
        fis.read(fbytes);
        fis.close();
        return fbytes;
    }

    public static void main(String[]args) throws Exception {
        KeyGen ac = new KeyGen();
        PrivateKey privateKey = ac.getPrivate("KeyPair/privateKey");
        PublicKey publicKey = ac.getPublic("KeyPair/publicKey");

        String msg = "Kovalenko";
```

```
        String encrypted__msg = ac.encryptText(msg, privateKey);
        String decrypted__msg = ac.decryptText(encrypted__msg, publicKey);
        System.out.println("Original Message: " + msg +
                "\nEncrypted Message: " + encrypted__msg
                + "\nDecrypted Message: " + decrypted__msg);

        ac.writeToFile(new File("KeyPair/text__encrypted.txt"),
encrypted__msg.getBytes("UTF-8"));
    }
}
```

Licence.java перевіряє власника ключа

```
import javax.crypto.BadPaddingException;
import javax.crypto.Cipher;
import javax.crypto.IllegalBlockSizeException;
import javax.crypto.NoSuchPaddingException;
import java.io.*;
import java.nio.file.Files;
import java.security.*;
import java.security.spec.PKCS8EncodedKeySpec;
import java.security.spec.X509EncodedKeySpec;
import java.util.Base64;
import java.util.Formatter;


public class Licence {
    private Cipher cipher;

    public Licence() throws NoSuchAlgorithmException, NoSuchPaddingException {
        this.cipher = Cipher.getInstance("RSA");
    }


//https://docs.oracle.com/javase/8/docs/api/java/security/spec/PKCS8EncodedKeySp
ec.html
    public PrivateKey getPrivate(String filename) throws Exception {
        byte[]keyBytes = Files.readAllBytes(new File(filename).toPath());
        PKCS8EncodedKeySpec spec = new PKCS8EncodedKeySpec(keyBytes);
        KeyFactory kf = KeyFactory.getInstance("RSA");
        return kf.generatePrivate(spec);
    }


//https://docs.oracle.com/javase/8/docs/api/java/security/spec/X509EncodedKeySpe
c.html
    public PublicKey getPublic(String filename) throws Exception {
        byte[]keyBytes = Files.readAllBytes(new File(filename).toPath());
        X509EncodedKeySpec spec = new X509EncodedKeySpec(keyBytes);
        KeyFactory kf = KeyFactory.getInstance("RSA");
        return kf.generatePublic(spec);
    }

    private void writeToFile(File output, byte[]toWrite)
            throws IllegalBlockSizeException, BadPaddingException, IOException {
        FileOutputStream fos = new FileOutputStream(output);
        fos.write(toWrite);
        fos.flush();
        fos.close();
    }

    public String decryptText(String msg, PublicKey key)
            throws InvalidKeyException, UnsupportedEncodingException,
            IllegalBlockSizeException, BadPaddingException {
        this.cipher.init(Cipher.DECRYPT_MODE, key);
```

```
        return new String(cipher.doFinal(Base64.getDecoder().decode(msg)), "UTF-
8");
    }

    public byte[]getFileInBytes(File f) throws IOException {
        FileInputStream fis = new FileInputStream(f);
        byte[]fbytes = new byte[(int) f.length()];
        fis.read(fbytes);
        fis.close();
        return fbytes;
    }

    public static void main(String[]args) throws Exception {
        Licence ac = new Licence();
        PublicKey publicKey = ac.getPublic("KeyPair/publicKey");

        String msg = new String(ac.getFileInBytes(new
File("KeyPair/text__encrypted.txt")));
        String decrypted__msg = ac.decryptText(msg, publicKey);
        System.out.println("Key: " + msg +
                "\nName: " + decrypted__msg);
        if(decrypted__msg.equals("Bohomaz")) {
            System.out.println("Key is valid");
        }

    }

}
```

TCPServer.java

```
import java.io.*;
import java.net.*;
class TCPServer {
    public static void main(String argv[]) throws Exception
    {
        String trueKey =
"DbXVepMJ+d/Zj5DDR2qA85Q5HoJojEr0yehitd8wMEqb+FI9BX5jDAYGHUXCl2R9hnZxiFnHagb+Nia
7JC00dg==";
        String clientSentence;
        String capitalizedSentence;
        ServerSocket welcomeSocket = new ServerSocket (3345);
        while (true) {
            Socket connectionSocket = welcomeSocket.accept();
            BufferedReader inFromClient = new BufferedReader(new
InputStreamReader(connectionSocket.getInputStream()));
            DataOutputStream outToClient = new
DataOutputStream(connectionSocket.getOutputStream());
            clientSentence = inFromClient.readLine();
            if(clientSentence.equals(trueKey)) {
                capitalizedSentence = "Key is valid.".toUpperCase() + '\n';
            } else {
                capitalizedSentence = "Key is bad.".toUpperCase() + '\n';
            }
            //capitalizedSentence = clientSentence.toUpperCase() + '\n';
            outToClient.writeBytes(capitalizedSentence);
        }
    }
}
```

TCPClient.java

```
import java.io.*;
import java.net.*;
import java.security.MessageDigest;
import java.security.NoSuchAlgorithmException;
import java.util.Formatter;

class TCPClient {
    public static void main(String argv[]) throws Exception
    {
        String sentence;
        String modifiedSentence;
        BufferedReader inFromUser = new BufferedReader(new
InputStreamReader(System.in));
        Socket clientSocket = new Socket("localhost", 3345);
        DataOutputStream outToServer = new
DataOutputStream(clientSocket.getOutputStream());
        BufferedReader inFromServer = new BufferedReader(new
InputStreamReader(clientSocket.getInputStream()));
        System.out.println("Print Key:");
//DbXVepMJ+d/Zj5DDR2qA85Q5HoJojEr0yehitd8wMEqb+FI9BX5jDAYGHUXCl2R9hnZxiFnHagb+Ni
a7JC00dg==
        sentence = inFromUser.readLine();
        outToServer.writeBytes(sentence + '\n');
        modifiedSentence = inFromServer.readLine();
        System.out.println( "FROM SERVER: " + modifiedSentence);
        clientSocket.close();
    }

    private static String encryptPassword(String password)
    {
        String sha1 = "";
        try
        {
            MessageDigest crypt = MessageDigest.getInstance("SHA-1");
            crypt.reset();
            crypt.update(password.getBytes("UTF-8"));
            sha1 = byteToHex(crypt.digest());
        }
        catch(NoSuchAlgorithmException e)
        {
            e.printStackTrace();
        }
        catch(UnsupportedEncodingException e)
        {
            e.printStackTrace();
        }
        return sha1;
    }

    private static String byteToHex(final byte[] hash)
    {
        Formatter formatter = new Formatter();
        for (byte b : hash)
        {
            formatter.format("%02x", b);
        }
        String result = formatter.toString();
        formatter.close();
        return result;
    }
}
```

Результат:

Генерація ліцензійного ключа (рис. 1):

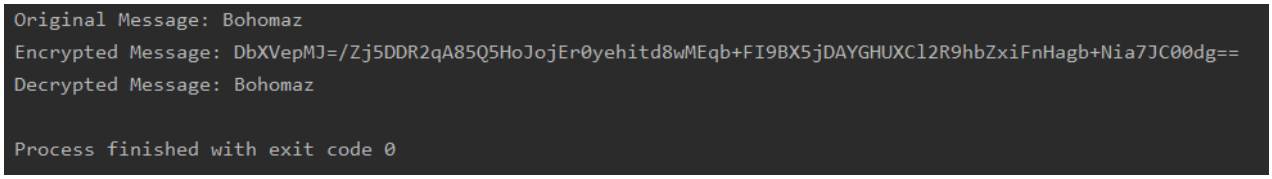

Рисунок 1

Перевіряемо(рис. 2):

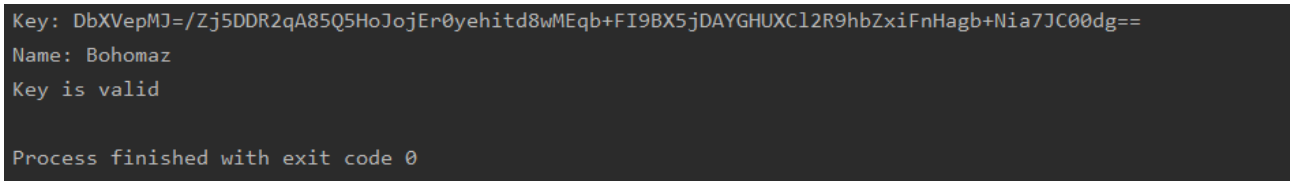

Рисунок 2

Кліент(рис. 3):

Рисунок 3

Висновок: у ході лабораторної роботи розроблено програму для формування та перевірки ліцензійних підписів з використанням мови Java.