Лабораториска вежба 1

Доверливост и интегритет на информации

Рок на изработка: 03.04.2020

Во оваа лабораториска вежба ќе создадеме систем кој ќе ни овозможи **доверливост** и **интегритет** на информации што ги праќаме преку интернет. За таа цел, ќе треба да имплементираме функции за **енкрипција** и **декрипција** на информации, како и функции за пресметување на **хеш сума** на информации.
При имплементација, ќе го користиме Java програмскиот јазик, со што ќе се запознаеме со библиотеките кои ни ги нуди, поточно Java Cryptography API[1].

Пред да почнеме со имплементација на самиот систем, ќе ги разгледаме опциите што ни ги нуди Java за потребите на нашиот систем.

Иницијализација на проект

За решавање на оваа лабораториска вежба, потребно е да креираме нов проект со помош на Maven. Maven ќе ни помогне да ги симне сите надворешни зависности кои ќе ни требаат.
Во зависност од развојната околина што ќе ја користите (обично IntelliJ или Eclipse) начинот на креирање знае да дивергира. Доколку користите IntelliJ, тогаш при креирање на нов проект, селектирате лево Maven како тип на проект, со сите default-ни опции.

Откако ќе го имате новокреираниот проект, треба да ја додадете следната зависност во pom.xml фајлот:

```
<dependencies>
    <dependency>
        <groupId>org.bouncycastle</groupId>
        <artifactId>bcprov-jdk15on</artifactId>
        <version>1.64</version>
    </dependency>
</dependencies>
```

[1] За време на креирање на оваа лабораториска вежба, беа користени следните ресурси: http://tutorials.jenkov.com/java-cryptography/index.html, https://howtodoinjava.com/security/

Java Cryptography API

Главни класи и интерфејси

Java Cryptography API-то е поделено низ следните Java пакети:

- `java.security`
- `java.security.cert`
- `java.security.spec`
- `java.security.interfaces`
- `javax.crypto`
- `javax.crypto.spec`
- `javax.crypto.interfaces`

Главните класи и интерфејси на тие пакети се:

- **`Provider`** `# нуди множество на имплементации`
- `SecureRandom`
- **`Cipher`** `# го дефинира шифрувачот (алгоритам/мод/падинг)`
- **`MessageDigest`** `# хеш сума на некој влез (без клуч)`
- `Signature`
- **`Mac`** `# хеш сума на некој влез (со клуч)`
- `AlgorithmParameters`
- `AlgorithmParameterGenerator`
- `KeyFactory`
- `SecretKeyFactory`
- `KeyPairGenerator`
- **`KeyGenerator`** `# ни овозможува да генерираме клуч од некоја фраза`
- `KeyAgreement`
- `KeyStore`
- `CertificateFactory`
- `CertPathBuilder`
- `CertPathValidator`
- `CertStore`

Останатите класи и интефјеси ќе ги разгледаме во наредните вежби, по потреба.

Провајдер

Најпрво нешто што треба да направиме е да дефинираме или вклучиме некој провајдер во Java. Провајдер претставува множество на имплементации кои што ќе може да ги користиме при

создавање на нашиот систем, за да немора да кодираме веќе добро дефинирани алгоритми. Java има и default-ен провајдер кој што може да се користи, но овде ќе користиме екстерен популарен провајдер наречен BouncyCastleProvider, кој што го вклучивме при иницијализација на проектот.

За да Java го вклучи провајдерот, мора експлицитно да напишеме дека сакаме тоа множество да е дел од Security-то на Java. Начинот на кој тоа може да го направиме е следен:

```
import org.bouncycastle.jce.provider.BouncyCastleProvider;
import java.security.Security;

public class ProviderExample {
    public static void main(String[] args) {

        // Го додава ова множество за користење
        Security.addProvider(new BouncyCastleProvider());

    }
}
```

Шифрувач

Cipher класата во Java го претставува било кој криптографски алгоритам за енкрипција и декрипција на информациите. Може да се специфицира алгоритамот за енкриптирање, режимот на работа и начинот на кој што ќе се врши падинг (пополување на празните места за последниот блок).

Пример, доколку сакаме да користиме AES, во ECB режим, со PKCS#5 падинг, тогаш треба да напишеме:

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

За да може да го користиме овој шифрувач, треба да го инцијализираме, односно да дефинираме дали ќе енкриптира или ќе декриптира, и кој клуч ќе го користи. Еден начин на кој што тоа може да го направиме е следниот:

```
// myKey е стринг што е ја содржи фразата за енкрипција
byte[] key              = myKey.getBytes("UTF-8");
MessageDigest sha       = MessageDigest.getInstance("SHA-1");
key                     = sha.digest(key);
key                     = Arrays.copyOf(key, 16);
SecretKeySpec secretKey = new SecretKeySpec(key, "AES");
cipher.init(Cipher.ENCRYPT_MODE, secretKey);
```

Доколку сакаме да енркиптираме податоци, треба да го повикаме методот `update()` или `doFinal()` дококлку е последниот блок. Пример:

```
byte[] plainText  = "abcdefghijklmnopqrstuvwxyz".getBytes("UTF-8");
byte[] cipherText = cipher.doFinal(plainText);
String output     = Base64.getEncoder().encodeToString(cipherText);
```

Последната линија ја додаваме за да енкриптираниот текст го прикажеме на убав начин на екран, бидејќи бајтите што се добиваат од самата енкрипција може да бидат карактери што не можат да бидат прикажани на екран.

MAC

Шифрувачот ни ја гарантира доверливоста на информациите, но за интегритетот ни треба нешто друго. Овде ќе користиме хеш функција која што ќе ни го обезбеди тоа.
Во Java имаме две слични имплементации кои што може да ни пресметаа хеш: `MessageDigest` и `Mac`. Разликата меѓу нив е што `MessageDigest` директно пресметува хеш на дадениот влез, додека пак Mac користи дополнителна безбедност, така што во влезот влегува и таен клуч. Начинот на генерирање на клучот е ист како во погорниот пример, и во следниот пример не е прикажан:

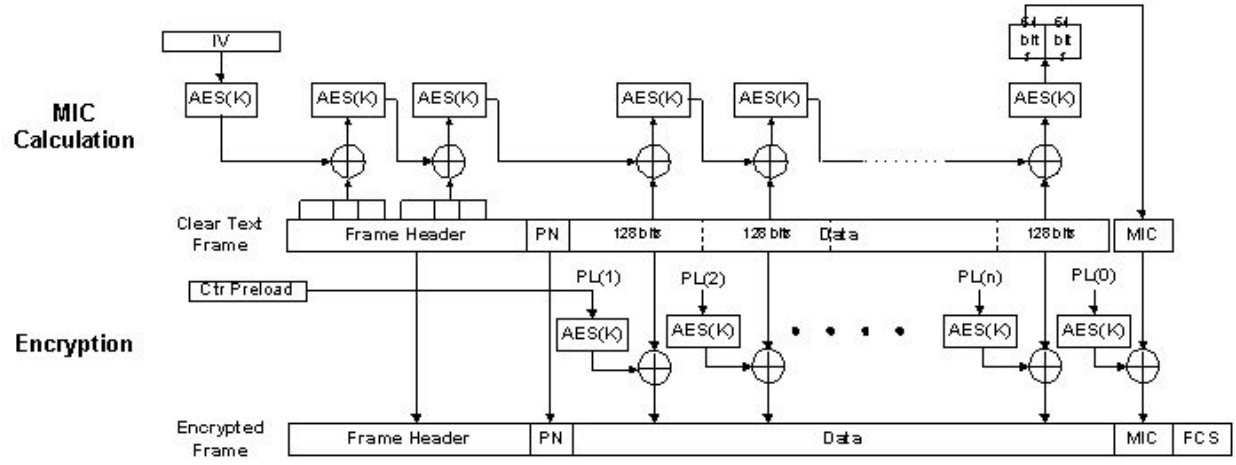
```
MessageDigest messageDigest = MessageDigest.getInstance("SHA-256");
byte[] data1 = "0123456789".getBytes("UTF-8");
byte[] digest = messageDigest.digest(data1);

Mac mac = Mac.getInstance("HmacSHA256");
mac.init(secretKey); // генериран на сличен начин како за шифрувачот
byte[] data = "abcdefghijklmnopqrstuvxyz".getBytes("UTF-8");
byte[] macBytes = mac.doFinal(data);
```

Задача

Ваша задача во оваа лабораториска вежба да го симулирате CCMP[2] протоколот кој што се користи во безжичните мрежи. Тој е користен во WPA2[3] стандардот кој што е најчесто користениот стандард во домашните безжични мрежи.

Во продолжение, една шема на како функционира:



Бидејќи овој стандард бара да имплементираме примање и праќање на рамки (податочни единици на второ ниво), ќе дефинираме малку поедноставен пристап. Имено, за секој дел од рамката ќе имаме посебна променлива, и тоа:

```
// Следните информации ја дефинираат ClearTextFrame класата
// Овие две информации го претставуваат Frame Header-от
byte[] sourceMAC; // Овде под MAC се мисли на media access control address
byte[] destinationMAC;
// Го содржи целиот payload на рамката
byte[] data;
```

[2] https://en.wikipedia.org/wiki/CCMP_(cryptography)
[3] https://en.wikipedia.org/wiki/Wi-Fi_Protected_Access

Барања

(1) Од вас се бара да го имплементирате енкриптирањето на рамката, односно да се симулира праќање на рамката:

```
// Враќа енкриптирана рамка што ги содржи податоците како на сликата
// Игнорирајте ги PN и FCS
EncryptedFrame encryptFrame(ClearTextFrame frame, String key);
```

Притоа, **рачно поделете ги податоците** да бидат пратени по 128 бита на шифрувачот и на Mac (не цел payload наеднаш до шифрувачот), како и **`EncryptedFrame`** и **`ClearTextFrame`** треба да ги искуцате сами, а ги содржат податоците како на сликата. Се користи истиот клуч при енкрипција и пресметка на MAC, а енкриптираните податоци треба да бидат енкодирани во Base64 форматот кога ќе се принтаат на екран.

(2) Дополнително, треба да ја напишете и функцијата за декрипција на самата рамка и верификација на интегритетот на рамката, односно да се симулира примање на рамката:

```
ClearTextFrame decryptFrame(EncryptedFrame frame, String key);
```

Доколку рамката не може да се енкриптира, декриптира, или доколку е нарушен интегритетот, фрлете исклучок од типот: **`IllegalStateException`**.

По успешно енкриптирање или декриптирање, испечатете го резултатот на екран во следниот формат:

```
Source MAC: <source-mac>
Destination MAC: <destination-mac>
Payload: <payload> // во Base64 доколку е енкриптиран
MIC: <mic> // доколку е дел од типот на рамка (Base64)
```

ЗАБЕЛЕШКА: Внимавајте на редоследот во кој ги извршувате операциите, параметрите што ги праќате и пазете на кои битови треба да се процесираат.

Како решение на оваа лабораториска задача, прикачете .zip фајл што го содржи решението (.java), и објаснување на решението (.pdf)