Міністерство освіти і науки України
Національний технічний університет України
“Київський політехнічний інститут ім. Ігоря Сікорського”
Фізико-технічний інститут

Лабораторна робота № 5

з предмету «Криптографія»

«Вивчення криптосистеми RSA та алгоритму електронного підпису; ознайомлення з методами генерації параметрів для асиметричних криптосистем»

Виконали:

Студенти 3 курсу,

ФТІ,групи ФБ-74

Люшняк Катерина, Харченко Владислав

Мета та основні завдання роботи

Ознайомлення з тестами перевірки чисел на простоту і методами генерації ключів для асиметричної криптосистеми типу RSA; практичне ознайомлення з системою захисту інформації на основі криптосхеми RSA, організація з використанням цієї системи засекреченого зв'язку й електронного підпису, вивчення протоколу розсилання ключів.

Порядок і рекомендації щодо виконання роботи

1. Написати функцію пошуку випадкового простого числа з заданого інтервалу або заданої довжини, використовуючи датчик випадкових чисел та тести перевірки на простоту. В якості датчика випадкових чисел використовуйте вбудований генератор псевдовипадкових чисел вашої мови програмування. В якості тесту перевірки на простоту рекомендовано використовувати тест Міллера-Рабіна із попередніми пробними діленнями. Тести необхідно реалізовувати власноруч, використання готових реалізацій тестів не дозволяється.

2. За допомогою цієї функції згенерувати дві пари простих чисел p, q і 1 1 p , q довжини щонайменше 256 біт. При цьому пари чисел беруться так, щоб pq ≤ p1q1 ; p і q – прості числа для побудови ключів абонента A, 1 p і q1 – абонента B.

3. Написати функцію генерації ключових пар для RSA. Після генерування функція повинна повертати та/або зберігати секретний ключ (d, p,q) та відкритий ключ (n,e) . За допомогою цієї функції побудувати схеми RSA для абонентів A і B – тобто, створити та зберегти для подальшого використання відкриті ключі (e,n) , (,) 1 n1 e та секретні d і d1 .

4. Написати програму шифрування, розшифрування і створення повідомлення з цифровим підписом для абонентів A і B. Кожна з операцій (шифрування, розшифрування, створення цифрового підпису, перевірка цифрового підпису) повинна бути реалізована окремою процедурою, на вхід до якої повинні подаватись лише ті ключові дані, які необхідні для її виконання. За допомогою датчика випадкових чисел вибрати відкрите повідомлення M і знайти криптограму для абонентів A и B, перевірити правильність розшифрування. Скласти для A і B повідомлення з цифровим підписом і перевірити його.

5. За допомогою раніше написаних на попередніх етапах програм організувати роботу протоколу конфіденційного розсилання ключів з підтвердженням справжності по відкритому каналу за допомогою алгоритму RSA. Протоколи роботи кожного учасника (відправника та приймаючого) повинні бути реалізовані у вигляді окремих процедур, на вхід до яких повинні подаватись лише ті ключові дані, які необхідні для виконання. Перевірити роботу програм для випадково обраного ключа $0 < k < n$.

Хід роботи, опис труднощів, що виникали, та шляхів їх розв’язання:

Під час виконання лабораторної роботи виникло питання як працювати в великими числами.
У мові Java є спеціальний клас BigInteger, який саме для цього і розрахований. Новим було те, що всі операції з числами мають виконуватись за допомогою спеціальних функцій.
Труднощі з’являлись лише через неуважність, забути написати один символ може слугувати 2 годинам витраченого часу. Проте практика виявилась не складної та корисною.

Згенеровані ключі для A та B:

P: 433996551200722609709353132915286740 73

Candidate for P---> 144766448758228223333950573952581212371

Candidate for P---> 306627479826900836789993975410629700343

Candidate for P---> 226427305413222029363975842973545513187

Candidate for P---> 130296617592189210050085155826387464975

Q: 130580507916984957504958663382739097567

Candidate for Q---> 143707522497174989722846160632354772812

Candidate for Q---> 395545967936859324504424819143201372 4

Candidate for Q---> 168975390817481855334199703093007983409

Candidate for Q---> 574113922648401829840233466717845479 49

P1:533311009451546175915180121071036071 77

Candidate for P1---> 247689725361059793246563185202058854534

Candidate for P1---> 131000061568155992612847447269709958724

Candidate for P1---> 247105300632286325636089032022589729 44

Candidate for P1---> 154829800798944392196313804337735564384

Q: 295430518263560483288730000329718905483

Candidate for Q2---> 533311009451546175915180121071036071 77

Candidate for Q2---> 309284635464008053917412778385949813795

Candidate for Q2---> 239036938572944127514866033146059924 28

Candidate for Q2---> 278300671754863348665869270251987387626

User A Generate public key pair

n = 566714900900101262023036937784108777255011052111846537219694949960409028039 1

fN = 566714900900101262023036937784108777237613035808140815372105552292982250875 2

e = 196879660673346544962368553353216241305438683293008591074326639938561212286 3

User A generate private key

d = 500621192528789860936430797111033091652977811911871637221005916151897000507 1

User A create message and encrypt it

Message = 999128281

We encrypt the message

Encrypt message =
42065157619833283629469659887568788702683388288092474854322793438223928704 28

c decrypt = 999128281

If we verify it, we can see the same message

v = 999128281

User B Generate public key pair

n1 = 157556347917932890045790750387114390911564650847751475276493539276816234514 91

e1 = 144069948605752135930062099087255041153459537391707010173027965558535269334 69

User B generate private key

d1 = 261833431155412634412748522381347348134197783781429892517336640215921644 1205

User A send message to B the pair (k1 , S1)

[550097718302427587141623391861091511310461865251356380932463596603935154764 6,
104003853093171083807773332479481090253955324777164380894805628534528163619 58]

User B receive and send to A the pair (k, S)

[999128281,
211445401092352334596004609143903170154340420757230964978593413967216078685]

Код

```
package com.gmail.xapchenko2000;
import java.math.BigInteger;
import java.util.ArrayList;
import java.util.Random;

public class Main {
   public static void main(String[] args) {
      rsa(generateKeyPair());
   }

   public static boolean testMillerRabin(BigInteger p) {
      int k = 15;
      BigInteger s = howManyTimesDiv2(p);
      BigInteger t =
(p.subtract(BigInteger.valueOf(1)).divide(BigInteger.valueOf(2).pow(s.intValue())));
      BigInteger one = new BigInteger("1");
```

```
    for (int i = 0; i < k; i++) {
        BigInteger a = randBigInteger(p.subtract(BigInteger.valueOf(2)));
        BigInteger x = a.modPow(t, p);

        if (x.equals(one) || x.equals(p.subtract(BigInteger.valueOf(1)))) continue;

        for (int j = 0; j < s.intValue() - 1; j++) {

            x = x.modPow(BigInteger.valueOf(2), p);
            if (x.equals(one)) return false;
            if (x.equals(p.subtract(BigInteger.valueOf(1)))) break;

        }
        return false;
    }
    return true;
}

public static BigInteger pow(BigInteger base, BigInteger exponent) {
    BigInteger result = BigInteger.ONE;
    while (exponent.signum() > 0) {
        if (exponent.testBit(0)) result = result.multiply(base);
        base = base.multiply(base);
        exponent = exponent.shiftRight(1);
    }
    return result;
}

public static BigInteger randBigInteger(BigInteger upperLimit) {
    BigInteger randomNumber;
    BigInteger zero = new BigInteger("0");
    Random random = new Random();
    do {
        randomNumber = new BigInteger(upperLimit.bitLength(), random);
    } while (randomNumber.compareTo(upperLimit) >= 0);

    return randomNumber;
}

public static BigInteger howManyTimesDiv2(BigInteger p) {
    BigInteger counter = BigInteger.valueOf(0);
    BigInteger one = new BigInteger("1");
    while (!p.mod(BigInteger.valueOf(2)).equals(one)) {
        counter = counter.add(BigInteger.valueOf(1));
        p = p.divide(BigInteger.valueOf(2));
```

```
        }
        return counter;
    }
    public static ArrayList<BigInteger> generateKeyPair() {
        ArrayList<BigInteger> keyPair = new ArrayList<>();
        ArrayList<BigInteger> badReturn = new ArrayList<>();

        BigInteger upperLimit = BigInteger.valueOf(2).pow(128);
        for (int i = 0; i < 4; i++) {
            BigInteger randBigInteger;
            do {
                randBigInteger = randBigInteger(upperLimit);
                System.out.println("Candidate for key = " + i + " ---> " + randBigInteger);
            } while (!testMillerRabin(randBigInteger));
            keyPair.add(i, randBigInteger);
        }
        BigInteger p = keyPair.get(0);
        BigInteger q = keyPair.get(1);
        BigInteger p1 = keyPair.get(2);
        BigInteger q1 = keyPair.get(3);

        if (p.multiply(q).compareTo(p1.multiply(q1)) == -1) {
            return keyPair;
        } else {
            generateKeyPair();
        }
        return badReturn;
    }

    public static void rsa(ArrayList<BigInteger> listOfAandB) {
        //User A
        System.out.println("Key ->");
        listOfAandB.forEach(x -> System.out.println(x));
        System.out.println("============================");
        BigInteger n = listOfAandB.get(0).multiply(listOfAandB.get(1));
        BigInteger oyler =
(listOfAandB.get(0).subtract(BigInteger.valueOf(1))).multiply(listOfAandB.get(1).subtract(BigInteg
er.valueOf(1)));
        BigInteger e = findE(oyler);// open key (n, e)
        BigInteger d = e.modInverse(oyler);//A private key
        BigInteger m = new BigInteger("999128281");//Our message
        BigInteger sign = sign(m, d, n);//Sign A user

        BigInteger c = findC(m, e, n);
```

```
        BigInteger v = verify(sign, e, n);

        BigInteger cDecrypt = decrypt(c, d, n);
        //User B
        BigInteger n1 = listOfAandB.get(2).multiply(listOfAandB.get(3));
        BigInteger oyler1 =
(listOfAandB.get(2).subtract(BigInteger.valueOf(1))).multiply(listOfAandB.get(3).subtract(BigInteg
er.valueOf(1)));
        BigInteger e1 = findE(oyler1);// open key (n, e)
        BigInteger d1 = e1.modInverse(oyler1);//A private key

        //Send key
        ArrayList<BigInteger> send = sendKey(m, e1, n, n1, d);

        ArrayList<BigInteger> receive = receiveKey(send, d1, n1);

        Boolean isCorrect = userACheckTheMessageOfUserB(receive, e, n);
        System.out.println("User A Generate public key pair");
        System.out.println("n = " + n);
        System.out.println("fN = " + oyler);
        System.out.println("e = " + e);
        System.out.println("User A generate private key");
        System.out.println("d = " + d);
        System.out.println("User A create message and encrypt it");
        System.out.println("Message = " + m);
        System.out.println("We encrypt the message");
        System.out.println("Encrypt message = " + c);
        System.out.println("c decrypt = " + cDecrypt);
        System.out.println("If we verify it, we can see the same message");
        System.out.println("v = " + v);
        System.out.println("User B Generate public key pair");
        System.out.println("n1 = " + n1);
        System.out.println("e1 = " + e1);
        System.out.println("User B generate private key");
        System.out.println("d1 = " + d1 + "\n");
        System.out.println("User A send message to B the pair (k1 , S1)");
        System.out.println(sendKey(m, e1, n, n1, d).toString());
        System.out.println("User B receive and send to A the pair (k, S)");
        System.out.println(receiveKey(send, d1, n1).toString());
        System.out.println("If all ok you can see true -> " + isCorrect);

    }
    public static BigInteger findE(BigInteger oyler) {
        BigInteger e;
        for (int i = 0; i < 101; i++) {
```

```
            e = randBigInteger(oyler);
            if (e.gcd(oyler).compareTo(BigInteger.valueOf(1)) == 0) return e;
        }
        return BigInteger.valueOf(-1);
    }

    public static BigInteger findC(BigInteger m, BigInteger e, BigInteger n) {
        return m.modPow(e, n);
    }

    public static BigInteger decrypt(BigInteger c, BigInteger d, BigInteger n) {
        return c.modPow(d, n);
    }


    public static BigInteger sign(BigInteger m, BigInteger d, BigInteger n) {
        return m.modPow(d, n);
    }


    public static BigInteger verify(BigInteger sign, BigInteger e, BigInteger n) {
        return sign.modPow(e, n);
    }


    public static ArrayList<BigInteger> sendKey(BigInteger k, BigInteger e1, BigInteger n,
BigInteger n1, BigInteger d) {
        //A make message (k1, S1)
        ArrayList<BigInteger> userAMessage = new ArrayList<>();
        BigInteger s = k.modPow(d, n);
        BigInteger k1 = k.modPow(e1, n1);
        BigInteger s1 = s.modPow(e1, n1);
        userAMessage.add(0, k1);
        userAMessage.add(1, s1);
        return userAMessage;
    }
    public static ArrayList<BigInteger> receiveKey(ArrayList<BigInteger> sendKey, BigInteger d1,
BigInteger n1) {
        //B make message (k, S)
        ArrayList<BigInteger> userBMessage = new ArrayList<>();
        BigInteger k = sendKey.get(0).modPow(d1, n1);
        BigInteger s = sendKey.get(1).modPow(d1, n1);
        userBMessage.add(0, k);
        userBMessage.add(1, s);
        return userBMessage;
    }
```

```
    public static boolean userACheckTheMessageOfUserB(ArrayList<BigInteger> userBMessage,
BigInteger e, BigInteger n) {
        //k = s^e mod n
        BigInteger k = userBMessage.get(0);
        BigInteger s = userBMessage.get(1);

        return k.equals(s.modPow(e, n));
    }
}
```

Висновок

У ході комп’ютерного практикуму ми навчились будувати модель RSA для розсилання ключів по відкритих каналам. Набули навичок роботи з великим числами, та згадали всі ти знання, які здобули на попередніх семестрах.