Міністерство освіти і науки України Національний
технічний університет України «Київський політехнічний
інститут імені Ігоря Сікорського» Фізико-технічний
інститут

ЛАБОРАТОРНА РОБОТА №5

з **дисципліни**

«Криптографія»

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

Виконали:

студенти 3 курсу ФТІ

групи ФБ-74

Стурчак Максим та Харламова Катерина

Перевірили:

Чорний О.

Савчук М. М.

Завадська Л. О.

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.

Порядоквиконання 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 p_1 , q_1 довжини щонайменше 256 біт. При цьому пари чисел беруться так, щоб $pq \leq p_1 q_1$; p і q – прості числа для побудови ключів абонента A, p_1 і q_1 – 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) , (e_1, n_1) та секретні d і d_1 .

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 $0 < k < n$.

Кожна з наведених операцій повинна бути реалізована у вигляді окремої процедури, інтерфейс якої повинен приймати лише ті дані, які необхідні для її роботи; наприклад, функція Encrypt(), яка шифрує повідомлення для абонента, повинна приймати на вхід повідомлення та відкритий ключ адресата (і тільки його), повертаючи в якості результату шифротекст. Відповідно, програмний код повинен містити сім високорівневих процедур: GenerateKeyPair(), Encrypt(), Decrypt(), Sign(), Verify(), SendKey(), ReceiveKey().

Згенеровані прості числа

P

372864076361718389513458614317220192366470497101454611178061441332816612477

Q:

691109507872588008602990968072427687804995719904761344072882408198562397331

P1:

595940706172400692676324918006143105575250383629293810049792085071445662789

Q1:

900806767875757502545121561431267259501008089268806210599787566982700133587

Числа,що не підійшли

p = 227704275149453850303252736028324662384302434422092582207,

q = 92449187385677356941232870530835965270723694541781173 75

p1 = 33684207344847343107412011845357786997467143311538585 6083

q1 = 17345308327809206118428499240549633170457587727623696 9255

p=22269061880887548648321385559566979955864689259840959688315312221872727930669434412111586626002 1

q=429790945533885412200258781817972472433315497791520979306920812732 3

p1=208803679589454733795660873717103390943622859038646369650954189781579011974842313266908934220 55,

q1=894558490717789456077109774154202065232990167474393288166277338204101921057707013136368151827 5

E:

13895481676080028603115527090868775776029081450313550076416434363604323602388342122697279456392707044199994464736782640819611566056430668859881861217 9

E1:

50383801202841633498878659907060054695000630870934319718322494111790223870499941510726741550806941635589199670748150718726938518242599275662283064536 9

D:

62553484154229267604969005148242733466608538611399006272856460170607138620361720421731293793913327420305330552766384651482337825219711213934732233179

D1:

17345654865271307051974477295126187542217126176773982043699013072315459872825903668120538757809130371061034371390347885662346084165587288811738877249

Відкрите повідомлення:

10336150300275641097121421344742930428863903981306030912142915405254688369165991824572767110958131494644666422812168087319474696647543676797717594154

Шифроване повідомлення:

2348847893144728387154083227892038939508586040003976957201800115815159864749812298278253039815746063224197601209275501074494564844409423498884546984 59

Цифровий підпис:

1034258989295227064933443513653022562906241508038163991530407850468332683986968739225846015931963648989332590485733591715059459569832106144570799624 56

Повідомлення,яке відправили

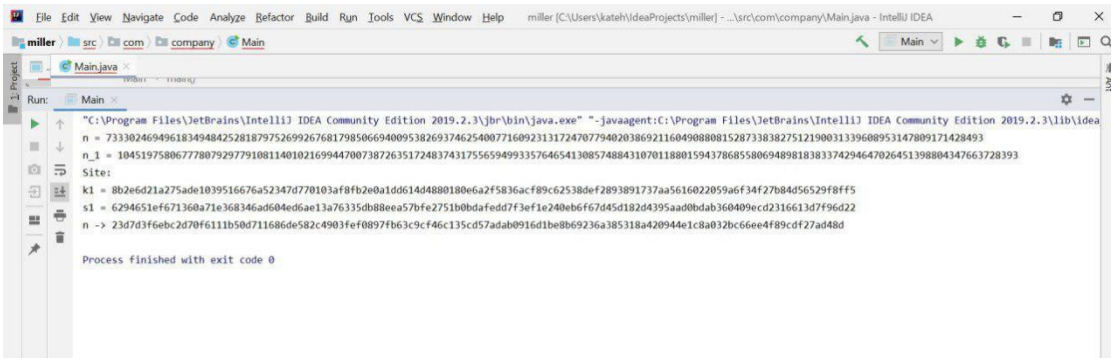
(5161179848669917121681301514349088780469317648006671919184845651612437359662476975539316447491647254252963569314226170017719433870566038639583906124 3,

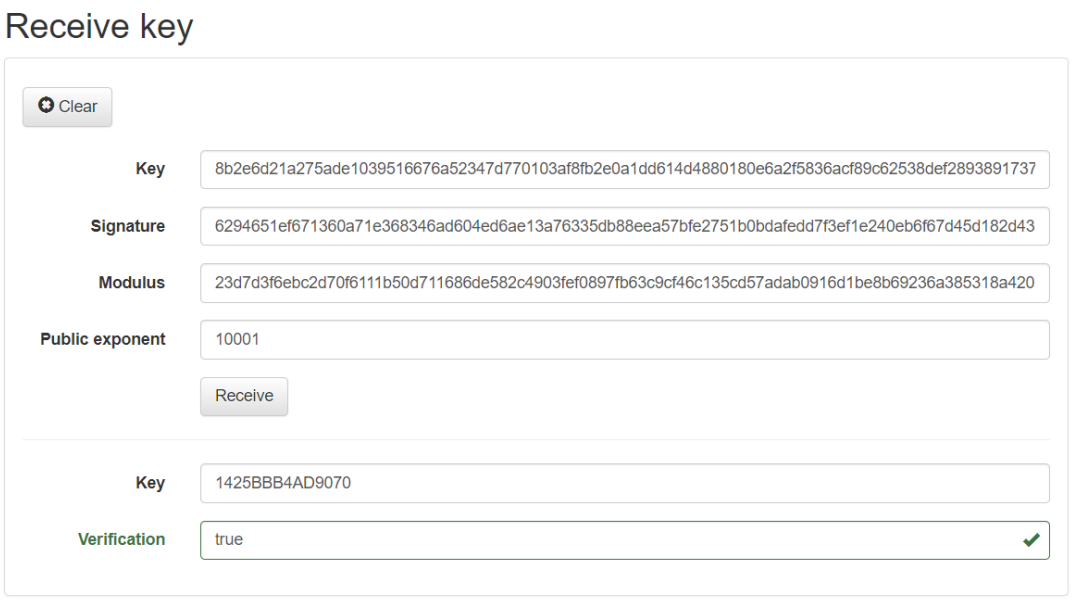
4289965552613248701695100072060391561566464567210893916382284157284025912607269897495160203541535080144577004550634499767195238515298243135094013258 69)

Повідомлення,яке отримали

(1033615030027564109712142134474293042886390398130603091214291540525468836916599182457276711095813149464466642281216808731947469664754367679771759415 4,

1034258989295227064933443513653022562906241508038163991530407850468332683986968739225846015931963648989332590485733591715059459569832106144570799624 56)





КОД

```
package com.company;

import java.math.BigInteger;
import java.util.*;
import java.util.stream.Collector;
import java.util.stream.Collectors;

import static java.math.BigInteger.*;

public class Main {
```

```
    public static void main(String[] args) {
        pq();
    }

    public static void pq() {
        BigInteger p_1 = number(rand());
        BigInteger q_1 = number(rand());
        BigInteger p1_1 = number(rand());
        BigInteger q1_1 = number(rand());

        BigInteger p = prov(p_1, q_1, p1_1, q1_1).get(0);
        BigInteger q = prov(p_1, q_1, p1_1, q1_1).get(1);
        BigInteger p1 = prov(p_1, q_1, p1_1, q1_1).get(2);
        BigInteger q1 = prov(p_1, q_1, p1_1, q1_1).get(3);


        System.out.println("Generate prime numbers : p =  " + p
+ ", q = " + q);
        System.out.println("Generate prime numbers : p1 =  " +
p1.toString(16) + ", q1 = " + q1.toString(16));
        System.out.println();

        BigInteger fN = getFN(p, q);
        BigInteger fN1 = getFN(p1, q1);

        BigInteger e = getE(fN);
        System.out.println("e = " + e.toString(16));
        BigInteger e1 = getE(fN1);
        System.out.println("e1 = " + e1.toString(16));

        BigInteger n = getN(p, q);
        System.out.println("n = " + n.toString(16));
        BigInteger n1 = getN(p1, q1);
        System.out.println("n1 = " + n1.toString(16));

        BigInteger d = getD(e, fN);
        System.out.println("d = " + d.toString(16));
        BigInteger d1 = getD(e1, fN1);
        System.out.println("d1 = " + d1.toString(16));

        getKeyPair(fN, n, e, p, q);
        getKeyPair(fN1, n1, e1, p1, q1);

        BigInteger m = getRandMes(n);//відкрите повідомлення m
```

```
        BigInteger m1 = getRandMes(n1);

        System.out.println();
        System.out.println("A's message for B  = " +
m.toString(16));
        System.out.println("A's message for B  = " +
m1.toString(16));
        System.out.println();


        BigInteger c = encrypt(m, e, n);//Зашифрування відкритого
повідомлення
        BigInteger c1 = encrypt(m1, e1, n1);


        System.out.println("encrypted message = " +
c.toString(16));
        System.out.println("encrypted message = " +
c1.toString(16));
        System.out.println();

        BigInteger m_1 = decrypt(c, d, n);//Розшифрування
криптограми
        BigInteger m1_1 = decrypt(c1, d1, n1);

        System.out.println("decrypted message = " +
m_1.toString(16));
        System.out.println("decrypted message = " +
m1_1.toString(16));
        System.out.println();

        BigInteger s = sign(m_1, d, n);//Цифровий підпис у системі
RSA.
        BigInteger s1 = sign(m1_1, d1, n1);

        System.out.println("sing = " + s.toString(16));
        System.out.println("sing = " + s1.toString(16));
        System.out.println();

        messPair(s, m_1);
        messPair(s1, m1_1);

        BigInteger mes = varificateSign(s, e, n);
        BigInteger mes1 = varificateSign(s1, e1, n1);

        System.out.println();
```

```
        System.out.println("verificated sign message = " +
mes.toString(16));
        System.out.println("verificated sign message = " +
mes1.toString(16));
        System.out.println();
        verify(receive(send(m, s, n, n1, e1, d), d1, n1), e, n);

    }


    public static ArrayList<BigInteger> send(BigInteger k,
BigInteger s, BigInteger n, BigInteger n1, BigInteger e1,
BigInteger d) {
        s = sign(k, d, n);
        BigInteger s1 = sign(s, e1, n1);
        BigInteger k1 = sign(k, e1, n1);

        ArrayList send = new ArrayList();
        send.add(s);
        send.add(0, s1);
        send.add(1, k1);

        System.out.println("A send message to B (" +
k1.toString(16) + "," + s1.toString(16) + ")");

        return send;

    }

    public static ArrayList<BigInteger>
receive(ArrayList<BigInteger> send, BigInteger d1, BigInteger n1)
{

        ArrayList<BigInteger> receive = new ArrayList<>();

        BigInteger k = send.get(1).modPow(d1, n1);
        BigInteger s = send.get(0).modPow(d1, n1);

        receive.add(0, k);
        receive.add(1, s);

        System.out.println("B receive message from A  (" +
k.toString(16) + "," + s.toString(16) + ")");

        return receive;
    }
```

```
    public static Boolean verify(ArrayList<BigInteger> receive,
BigInteger e, BigInteger n) {


        BigInteger s = receive.get(1);
        BigInteger k = receive.get(0);
        BigInteger prov = s.modPow(e, n);


        if (k.compareTo(prov) == 0) {
            System.out.println("identity verification is
passed");
            return true;
        } else if (k.compareTo(prov) != 0) {
            System.out.println("identity verification isn't
passed");
            return false;
        }
        return true;
    }


    public static BigInteger getN(BigInteger p, BigInteger q) {

        BigInteger n = p.multiply(q);
        return n;
    }

    public static BigInteger getFN(BigInteger p, BigInteger q) {

        BigInteger fN =
(p.subtract(valueOf(1))).multiply(q.subtract(valueOf(1)));
        return fN;
    }

    public static ArrayList getKeyPair(BigInteger fN, BigInteger
n, BigInteger e, BigInteger p, BigInteger q) {

        BigInteger d = getD(e, fN);

        ArrayList openKeyPair = new ArrayList();
        openKeyPair.add(n);
        openKeyPair.add(e);
```

```
        System.out.println("open key pair (" + n.toString(16) +
"," + e.toString(16) + ")");

        ArrayList closeKeyPair = new ArrayList();
        closeKeyPair.add(p);
        closeKeyPair.add(q);
        closeKeyPair.add(d);

        System.out.println("close key pair (" + d.toString(16) +
"," + p.toString(16) + " ," + q.toString(16) + ")");

        return openKeyPair;
    }

    public static ArrayList messPair(BigInteger s, BigInteger m)
{

        ArrayList messPair = new ArrayList();
        messPair.add(s);
        messPair.add(m);
        System.out.println("electronic signature (" +
s.toString(16) + "," + m.toString(16) + ")");
        return messPair;
    }

    public static BigInteger getD(BigInteger e, BigInteger fN) {
        BigInteger d = e.modInverse(fN);
        return d;
    }

    public static BigInteger getRandMes(BigInteger n) {

        System.out.println("n " + n);
        BigInteger maxLimit = (n.subtract(valueOf(1)));
        BigInteger minLimit = new BigInteger("2");

        BigInteger bigInteger = maxLimit.subtract(minLimit);
        Random randNum = new Random();
        int len = maxLimit.bitLength();
        BigInteger mes = new BigInteger(len, randNum);

        if (mes.compareTo(minLimit) < 0)
            mes = mes.add(minLimit);

        if (mes.compareTo(bigInteger) >= 0)
            mes = mes.mod(bigInteger).add(minLimit);
```

```
        return mes;
    }

    public static BigInteger encrypt(BigInteger m, BigInteger e,
BigInteger n) {
        BigInteger c = m.modPow(e, n);
        System.out.println();
        return c;
    }

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


    public static BigInteger sign(BigInteger M, BigInteger d,
BigInteger n) {
        BigInteger s = M.modPow(d, n);//
        return s;
    }

    public static BigInteger varificateSign(BigInteger s,
BigInteger e, BigInteger n) {
        BigInteger m = s.modPow(e, n);//
        return m;
    }

    public static BigInteger getRand(BigInteger fN) {
        BigInteger maxLimit = (fN.subtract(valueOf(1)));
        BigInteger minLimit = new BigInteger("0");

        BigInteger bigInteger = maxLimit.subtract(minLimit);
        Random randNum = new Random();
        int len = maxLimit.bitLength();
        BigInteger res = new BigInteger(len, randNum);

        if (res.compareTo(minLimit) < 0)
            res = res.add(minLimit);

        if (res.compareTo(bigInteger) >= 0)
            res = res.mod(bigInteger).add(minLimit);

        BigInteger e = verificateGcd(res, fN);
```

```
        return e;
    }

    public static BigInteger verificateGcd(BigInteger randE,
BigInteger fN) {
        BigInteger tempGcd = randE.gcd(fN);
        if (tempGcd.compareTo(valueOf(1)) == 0) {
            return randE;
        } else if (tempGcd.compareTo(valueOf(1)) != 0) {
            randE = getRand(fN);
        }
        return randE;
    }


    public static BigInteger getE(BigInteger fN) {
        BigInteger e = getRand(fN);
        return e;
    }

    public static BigInteger rand() {
        BigInteger maxLimit = new
BigInteger("10000000000000000000000000000000000000000000000000
00000000000000000000000000");
        BigInteger minLimit = new
BigInteger("50000000000000000000000000000000");

        BigInteger bigInteger = maxLimit.subtract(minLimit);
        Random randNum = new Random();
        int len = maxLimit.bitLength();
        BigInteger mes = new BigInteger(len, randNum);

        if (mes.compareTo(minLimit) < 0)
            mes = mes.add(minLimit);

        if (mes.compareTo(bigInteger) >= 0)
            mes = mes.mod(bigInteger).add(minLimit);


        return mes;
    }

    public static BigInteger number(BigInteger mes) {
        if (millerRabin(mes) == true) {
            return mes;
```

```
        } else if (millerRabin(mes) == false) {
            do mes = rand();
            while (millerRabin(mes) == false);
        }
        return mes;

    }


    public static BigInteger randN(BigInteger n) {
        BigInteger maxLimit = n.subtract(valueOf(2));
        BigInteger minLimit = new BigInteger("2");

        BigInteger bigInteger = maxLimit.subtract(minLimit);
        Random randNum = new Random();
        int len = maxLimit.bitLength();
        BigInteger mes = new BigInteger(len, randNum);

        if (mes.compareTo(minLimit) < 0)
            mes = mes.add(minLimit);

        if (mes.compareTo(bigInteger) >= 0)
            mes = mes.mod(bigInteger).add(minLimit);


        return mes;
    }

    private static Boolean millerRabin(BigInteger n) {
        if (n.compareTo(valueOf(1)) == -1) {
            return false;
        }
        if (n.compareTo(valueOf(3)) == -1) {
            return false;
        }
        if (n.mod(valueOf(2)).equals(valueOf(0))) {
            return false;
        } else {
            int k = 15;
            for (int j = 0; j < k; j++) {
                BigInteger a = randN(n);
                if (a.compareTo(n) == -1) {
                    int s = 0;
                    BigInteger d = n.subtract(valueOf(1));
                    while (d.mod(valueOf(2)).equals(valueOf(0)))
{
```

```
                        s++;
                        d = d.divide(valueOf(2));

                    }
                    BigInteger x = a.modPow(d, n);

                    if (x.equals(valueOf(1)) ||
x.equals(n.subtract(valueOf(1))))
                        continue;

                    for (int i = 1; i < s; i++) {

                        x = (x.multiply(x)).mod(n);

                        if (x.equals(valueOf(1))) {
                            return false;
                        }

                        if (x.equals(n.subtract(valueOf(1))))
                            break;

                    }

                    if (x.compareTo(n.subtract(valueOf(1))) != 0)
{
                        return false;
                    }
                }
            }
            return true;
        }
    }


    public static ArrayList<BigInteger> prov(BigInteger p,
BigInteger q, BigInteger p1, BigInteger q1) {
        BigInteger pq = p.multiply(q);
        BigInteger p1q1 = p1.multiply(q1);


        ArrayList val = new ArrayList();
        ArrayList val1 = new ArrayList();

        if (p1q1.compareTo(pq) == 1) {
            val.add(0, p);
            val.add(1, q);
```

```
            val.add(2, p1);
            val.add(3, q1);
        } else {
            do {
                p = number(rand());
                q = number(rand());
                p1 = number(rand());
                q1 = number(rand());

                val1.add(0, p);
                val1.add(1, q);
                val1.add(2, p1);
                val1.add(3, q1);

                return val1;

            } while (p1q1.compareTo(pq) != 1);

        }
        return val;
    }

}
```