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

Постолюк Діана та Хацько Микита

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.

Порядок 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 p_1 , q_1 довжини щонайменше 256 біт. При цьому пари чисел беруться так, щоб $pq \neq p_1q_1$; 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 ізнайти 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 $0 \neq k \neq n$.

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: 22770427514945385030325273602832466238430243442209258220 7
Q: 92449187385677356941232870530835965270723694541781173 75
P1: 33684207344847343107412011845357786997467143311538585608 3
Q1: 17345308327809206118428499240549633170457587727623696925 5

Числа,що не підійшли

p=22269061880887548648321385559566979955864689259840959688315312221877279306694344121115866260021

q=42979094553388541220025878181797247243331549779152097930692081273 23

p1=20880367958945473379566087371710339094362285903864636965095418978157901197484231326690893422055,

q1=89455849071778945607710977415420206523299016747439328816627733820410192105770701313636815182 75

E:

13895481676080028603115527090868775776029081450313550076416434363604323602388342122697279456392707044199994464736782640819611566056430668859881861217 9

E1:

50383801202841633498878659907060054695000630870934319718322494111790223870499941510726741550806941635589199670748150718726938518242599275662283064536 9

D:

62553484154229267604969005148242733466608538611399006272856460170607138620361720421731293793913327420305330552766384651482337825219711213934732233179

D1:

17345654865271307051974477295126187542217126176773982043699013072315459872825903668120538757809130371061034371390347885662346084165587288811738877249

Відкрите повідомлення:

25024758296460960386859474539861832793557740178126907334218234151812660688954213619779720066547329154

Шифроване повідомлення:

62507944771354433553619032758757822936509729488852923457670974178432311858637

Цифровий підпис:

542103105132789022852383565745561405052706949213783281294431042518359631129345718
661087184171755678919007546852111623790807271734211914190200998880601780 5

Повідомлення,яке відправили

625079447713544335536190327587578229365097294888529234576709741784323118586 37

Повідомлення,яке отримали

encryptedMessage:
250247582964609603868594745398618327935577401781269073342182341518126606889542
136197720066547329154
decryptedMessage:
625079447713544335536190327587578229365097294888529234576709741784323118586 37

Код програми:

```
package com.company;
import javafx.util.Pair;
import java.math.BigInteger;
import java.util.Objects;
import java.util.Random;
import static java.math.BigInteger.*;

public class Main {

    private static final BigInteger MIN_BOUND = new
BigInteger("35742549198872617291353508656626642567");
    private static final BigInteger MAX_BOUND = new
BigInteger("357425491988726172913535086566266425677868009 2345");

    // e = 2^16 + 1
    private static final BigInteger E = TWO.pow(16).add(ONE);

    public static void main(String[] args) {
        Pair<BigInteger, BigInteger> pq1 = generateSimpleNumberPair();
        BigInteger p1 = pq1.getKey();
        BigInteger q1 = pq1.getValue();
        // n1 = p1 * q1
        BigInteger n1 = pq1.getKey().multiply(pq1.getValue());
```

```
        System.out.println("p1=" + p1);
        System.out.println("q1=" + q1);
        System.out.println("n1=" + n1);

        Pair<BigInteger, BigInteger> pq2 = generateSimpleNumberPair();
        BigInteger p2 = pq2.getKey();
        BigInteger q2 = pq2.getValue();
        // n2 = p2 * q2
        BigInteger n2 = pq2.getKey().multiply(pq2.getValue());
        // n1 < n2
        while (n1.compareTo(n2) < 0) {
            pq2 = generateSimpleNumberPair();
            p2 = pq2.getKey();
            q2 = pq2.getValue();
            // n2 = p2 * q2
            n2 = p2.multiply(q2);
        }
        System.out.println("p2=" + p2);
        System.out.println("q2=" + q2);
        System.out.println("n2=" + n2);

        BigInteger d1 = getD(p1, q1);
        System.out.println("A parameters");
        System.out.println("d = " + d1);
        System.out.println("n = " + n1);
        BigInteger d2 = getD(p2, q2);
        System.out.println("B parameters");
        System.out.println("d = " + d2);
        System.out.println("n = " + n2);


        BigInteger message = new
BigInteger("8A323E34209BA7068FD7C190C132122F859ECA9F9C6057AADBCE86D58C2395CD",
16);

        BigInteger encryptedMessage = encrypt(message, n1);
        BigInteger decryptedMessage = decrypt(encryptedMessage, d1, n1);
        System.out.println("Message: " + message);
        System.out.println("encryptedMessage: " + encryptedMessage);
        System.out.println("decryptedMessage: " + decryptedMessage);


        Pair<BigInteger, BigInteger> notification = sendKey(d1, n1, n2, E, message);
        BigInteger k1 = notification.getKey();
        BigInteger s1 = notification.getValue();
        System.out.println("k1=" + k1);
        System.out.println("s1=" + s1);
```

```
        System.out.println(receiveKey(n1, d2, n2, k1, s1));

        // Public key server
        BigInteger modulus = new
BigInteger("BDA2A6415BE9AD90013121CA74A5431E1ECE0C2A38E8894B22904CE612DA397B4B
78894E3ED46398E74F037C0BFF52968F4C19D0D6CCB824D79BFE86BD3DC231", 16);
        BigInteger exp = new BigInteger("10001", 16);
        BigInteger key = new
BigInteger("381A125EA7520651FD231565441969916A667DE71108CC0BDCECA8B3C08E3D093F61
09890B63927B74E6", 16);

        Pair<BigInteger, BigInteger> sendKeyPair = sendKey(d1, n1, modulus, exp, key);
        System.out.println("Key: " + sendKeyPair.getKey());
        System.out.println("Sign: " + sendKeyPair.getValue());
    }

    private static Pair<BigInteger, BigInteger> generateSimpleNumberPair() {
        BigInteger coreP, coreQ, q; // p = 2*i*coreP + 1, q = 2*i*coreQ + 1; i = 1, 2...
        coreP = generatePrime();
        coreQ = generatePrime();
        while (Objects.equals(coreP, coreQ)) {
            coreQ = generatePrime();
        }
        int i = 1;
        BigInteger p;
        do {
            // p = 2 * i * coreP + 1;
            p = TWO.multiply(valueOf(i)).multiply(coreP).add(ONE);
            i++;
        } while (!isPrime(p, 10));
        i = 1;
        do {
            // q = 2 * i * coreQ + 1;
            q = TWO.multiply(valueOf(i)).multiply(coreQ).add(ONE);
            i++;
        } while (!isPrime(q, 10));
        return new Pair<>(p, q);
    }

    private static BigInteger generatePrime() {
        BigInteger number;
        do {
            // number = (gen() % MAX_BOUND + MIN_BOUND) % MAX_BOUND;
            number = gen().remainder(MAX_BOUND).add(MIN_BOUND).remainder(MAX_BOUND);
        } while (!isPrime(number, 15));
        return number;
    }
```

```
private static BigInteger gen() {
   return BigInteger.probablePrime(256, new Random());
}

private static boolean isPrime(BigInteger numberToCheck, int rounds) {
   // numberToCheckCopy = numberToCheck - 1
   BigInteger numberToCheckCopy = numberToCheck.subtract(ONE);
   int s = 0;
   // numberToCheck % 2 == 0
   while (numberToCheckCopy.remainder(TWO).equals(ZERO)) {
      s++;
      // numberToCheckCopy = numberToCheckCopy / 2
      numberToCheckCopy = numberToCheckCopy.divide(TWO);
   }
   BigInteger d = numberToCheckCopy; // d is odd

   BigInteger temp, X;
   for (int i = 0; i < rounds; i++) {
      // temp = (1 + gen()) % numberToCheck - 2;
      temp = ONE.add(gen()).remainder(numberToCheck).subtract(TWO);
      // X = (temp ^ d) mod numberToCheck
      X = temp.modPow(d, numberToCheck);
      // (numberToCheck == 1 || numberToCheck == (numberToCheck - 1))
      if (X.equals(ONE) || X.equals(numberToCheck.subtract(ONE))) {
         continue;
      }
      for (int j = 0; j < s - 1; j++) {
         // X = (X ^ 2) mod numberToCheck
         X = X.modPow(TWO, numberToCheck);
         // (X == 1)
         if (X.equals(ONE)) {
            return false;
         // (X == numberToCheck - 1)
         } else if (X.equals(numberToCheck.subtract(ONE))) {
            break;
         }
      }
      return false;
   }
   return true;
}

private static BigInteger getD(BigInteger p, BigInteger q) {
   // Euler function; fi(n) = (p-1)*(q-1)
   BigInteger fi = p.subtract(ONE).multiply( q.subtract(ONE) );
   System.out.println("gcd(e,fi)=" + E.gcd(fi));
```

```
        // d = e^(-1) mod fi
        BigInteger d = E.modInverse(fi);
        System.out.println("e*d mod fi = " + E.multiply(d).remainder(fi));
        return d;
    }

    private static BigInteger encrypt(BigInteger message,
                                      BigInteger n) {
        // C = M^e mod n
        return message.modPow(E, n);
    }

    private static BigInteger decrypt(BigInteger cipherText,
                                      BigInteger d,
                                      BigInteger n) {
        // M = C^d mod n
        return cipherText.modPow(d, n);
    }

    private static Pair<BigInteger, BigInteger> sendKey(BigInteger d1,
                                                        BigInteger n1,
                                                        BigInteger n2,
                                                        BigInteger e,
                                                        BigInteger message) {
        // s = k(message)^d1 mod n1;
        BigInteger s = message.modPow(d1, n1);
        // k1 = k(message)^e mod n1;
        BigInteger k1 = message.modPow(e, n2);
        // s1 = s^e mod n1;
        BigInteger s1 = s.modPow(e, n2);
        return new Pair<>(k1, s1);
    }

    private static boolean receiveKey(BigInteger n1,
                                      BigInteger d2,
                                      BigInteger n2,
                                      BigInteger k1,
                                      BigInteger s1) {
        // message = k1^d2 mod n2
        BigInteger message = k1.modPow(d2, n2);
        // s = s1^d2 mod n2
        BigInteger s = s1.modPow(d2, n2);
        // message == s^e mod n1
        return message.equals(s.modPow(Main.E, n1));
    }
}
```