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»

Виконав:

студенти 3 курсу ФТІ

групи ФБ-74 і ФБ-73

Сизов Ігор

Божко Анастасія

Перевірили:

Чорний О.

Савчук М. М.

Завадська Л. О.

Мета

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 q p, і 1 1, qp довжини щонайменше 256 біт. При цьому пари чисел беруться так, щоб 1 1qppq ; p і q – прості числа для побудови ключів абонента A, 1 p і 1 q – абонента B. 3. Написати функцію генерації ключових пар для RSA. Після генерування функція повинна повертати та/або зберігати секретний ключ) ,,(q pd та відкритий ключ) ,(e n . За допомогою цієї функції побудувати схеми RSA для абонентів A і B – тобто, створити та зберегти для подальшого використання відкриті ключі) ,(n e ,) ,(1 1 ne та секретні d і 1 d . 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 n k 0 .

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().

Значення

p = 1B2E69ADC0379

q = 1BAC474ABAECF

n = 2F02FE622959F1AE5E8160CD7

fi(n) = 2F02FE622959BAD3AD88E5A90

e = 10001

d = 29F226DD98FD99E8031F71A31

Abonent B

p = 0B5062A23AF0F

q = 2319C204E318F

n = 18D20E8F1F08F25679502A861

fi(n) = 18D20E8F1F08C3EC54A90C7C4

e = 10001

d = 11279DC86DBAB12F7694207D

(n>n1) Generating new keys...

Abonent A

p = 1E5EE0A5043C1

q = 0B5E6B932AAA7

n = 1594740B135E36A2BBC0C5CE7

fi(n) = 1594740B135E0CE56F8896E80

e = 10001

d = 0FFE784761325AC28DF8A1481

S1= 46524032708FF054BD7E1BAC

k1= 09D53710D28989107FC3157D5

ReceiveKey TRUE

Висновок

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

Код

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Numerics;
using System.Security.Cryptography;



namespace CryptoLab5
{
    class Abonent:Program
    {
        private BigInteger p, q, d;
        public BigInteger e, n;
        public string name;
        public Abonent()
        {
            p = 0;q = 0;d = 0;e = 0;n = 0;
        }
        public Abonent(string a)
        {
            name = a;
        }
        public void GenerateKeyPair()
        {
            p = SimpleChislo();
            q = SimpleChislo();
            n = BigInteger.Multiply(p, q);
            BigInteger fin = BigInteger.Multiply(BigInteger.Subtract(p, (BigInteger)1),
BigInteger.Subtract(q, (BigInteger)1));
            e = (BigInteger)(Math.Pow(2, 16) + 1);
            if (BigInteger.GreatestCommonDivisor(fin,e)!= 1)
            {
                Console.WriteLine("Please reload programm,gcd(fin,e)>1");
                return;
            }

            d = MultiplicativeInverse(e, fin);
            Console.WriteLine(name);
            Console.WriteLine("");
            Console.WriteLine($"p = {p.ToString("X")}");
            Console.WriteLine($"q = {q.ToString("X")}");
            Console.WriteLine($"n = {n.ToString("X")}");
            Console.WriteLine($"fi(n) = {fin.ToString("X")}");
            Console.WriteLine($"e = {e.ToString("X")}");
            Console.WriteLine($"d = {d.ToString("X")}");
            Console.WriteLine($"_____________________________________________");

        }

        public static BigInteger Encrypt(BigInteger massage, BigInteger e,BigInteger n)
        {
```

```
            BigInteger answ=BigInteger.ModPow(massage, e, n);
            Console.WriteLine(answ);
            Console.WriteLine(answ.ToString("X"));

            return answ;
        }
        public static BigInteger Decrypt(BigInteger massage, BigInteger d,BigInteger n)
        {
            BigInteger answ = BigInteger.ModPow(massage, d, n);
            return answ;
        }
        public  BigInteger Sign(int massage,BigInteger d, BigInteger n)
        {
            BigInteger answ = BigInteger.ModPow(massage, d, n);

            return answ;

        }
        public  void Verify(int massage,BigInteger s,BigInteger n, BigInteger e)
        {
            if (massage == BigInteger.ModPow(s, e, n)) Console.WriteLine("Verifed");
            else Console.WriteLine("Not verified");
        }

        public BigInteger SendKey(int key,BigInteger e1, BigInteger n1,ref BigInteger
Value4returnS1)
        {
            while (n1 < n)
            {
                Console.WriteLine("Generating new keys...");
                GenerateKeyPair();
            }
            BigInteger k1 = BigInteger.ModPow(key, e1, n1);
            BigInteger S = BigInteger.ModPow(key, d, n);
            Value4returnS1 = BigInteger.ModPow(S, e1, n1);
            return k1;
        }
        public void ReceiveKey(BigInteger k1,BigInteger S1,BigInteger eA, BigInteger nA)
        {
            BigInteger k = BigInteger.ModPow(k1, d, n);
            BigInteger S = BigInteger.ModPow(S1, d, n);
            BigInteger k2 = BigInteger.ModPow(S, eA, nA);
            if (k == k2) Console.WriteLine("ReceiveKey TRUE");
            else Console.WriteLine("FALSE");
        }



    }
    class Program
    {
        static Random rnd = new Random();
        static int colvo = 15; //длинна генереруемого числа

        public static BigInteger MultiplicativeInverse(BigInteger e, BigInteger fi)
        {
            BigInteger result;
            BigInteger res1;
            int k = 1;
            while (true)
            {
                result = (1 + (k * fi)) / e;
                res1= (1 + (k * fi)) % e;
                if (res1 == 0) //integer
                {
                    return result;
                }
```

```
                else
                {
                    k++;
                }
            }
        }

        public static BigInteger mod(BigInteger a, BigInteger mod)
        {
            do
            {
                if (a.CompareTo((BigInteger)0) == -1) a = BigInteger.Add(a, mod);
                if (a.CompareTo(mod) == 1) a = BigInteger.Subtract(a, mod);


            } while (a.CompareTo((BigInteger)0) == -1 || a.CompareTo((BigInteger)mod) == 1);
//a<0 || a>mod
            return a;
        }

        static public bool ProstotaTest(BigInteger chislo, int cycles)
        {
            if (chislo == 2 || chislo == 3)
                return true;
            if (chislo < 2 || chislo % 2 == 0)
                return false;


            BigInteger t = chislo - 1;
            int counter = 0;
             while (t % 2 == 0)
            {
                t=t/2;
                counter++;
            }

            for (int i = 0; i < cycles; i++)
            {
                RNGCryptoServiceProvider rng = new RNGCryptoServiceProvider();

                byte[] _zz = new byte[chislo.ToByteArray().LongLength];

                BigInteger zz;

                do
                {
                    rng.GetBytes(_zz);
                    zz = new BigInteger(_zz);
                }
                while (zz < 2 || zz >= chislo - 2);
                BigInteger x = BigInteger.ModPow(zz, t, chislo);

                if (x == 1 || x == chislo - 1)
                    continue;

                for (int r = 1; r < counter; r++)
                {
                    x = BigInteger.ModPow(x, 2, chislo);
                    if (x == 1)
                        return false;
                    if (x == chislo - 1)
                        break;
                }

                if (x != chislo - 1)
                    return false;
            }
            return true;
        }
```

```
        static BigInteger BigGen(int length)
        {

            string a = "";

            a += rnd.Next(1, 10).ToString();
            for (int i = 0; i < length - 1; i++)
            {
                int value = rnd.Next(0, 10);
                a += value.ToString();
            }
            BigInteger b= BigInteger.Parse(a);
            return b;
        }

        public static BigInteger SimpleChislo()
        {

            BigInteger a;
            do
            {
                a = BigGen(colvo);
            } while (!(ProstotaTest(a, 20)));
            return a;
        }

        public static BigInteger To10(string input)
        {
            BigInteger result = 0;
            result = BigInteger.Parse(input, System.Globalization.NumberStyles.HexNumber);
            return result;
        }


        static void Main(string[] args)
        {

            string a =
"98925152457816432035483225126942970027349299303361039613088270768589518012321943681540958432803657018002802462447497822786287652462575082306857234224206 31";
            BigInteger n =BigInteger.Parse(a);
            BigInteger e = To10("10001");

            Abonent A = new Abonent("Abonent A");
            Abonent B = new Abonent("Abonent B");
            A.GenerateKeyPair();
            B.GenerateKeyPair();
            BigInteger S1 = 0;
            BigInteger k1 = A.SendKey(35, B.e, B.n, ref S1);

            Console.WriteLine($"S1= {S1.ToString("X")}");
            Console.WriteLine($"k1= {k1.ToString("X")}");

            B.ReceiveKey(k1, S1, A.e, A.n);

            Console.ReadKey();
        }
    }
}
```