|  |  |  |
| --- | --- | --- |
|  | | |
| Федеральное государственное бюджетное  образовательное учреждение высшего образования «Новосибирский государственный технический университет» | | |
|  | | |
| Кафедра прикладной математики | | |
| Практическое задание № 1 | | |
| по дисциплине «Численные методы» | | |
|  | | |
|  | | |
|  | Бригада | Пучков дмитрий |
| Группа ПМ-05 | Лепский егор |
| Вариант 10 |  |
|  |  |
|  |  |
|  |  |
| Преподаватель | Патрушев Илья Игоревич |
|  |  |
| Новосибирск, 2022 | | |

1. **Условие задачи**

Вариант 3 (18 баллов). Алгоритм RSA

Написать программы реализующие алгоритм RSA.

1. Генерация ключей
   1. Прочитать из консоли числа *p* и *q*.
   2. Проверить что p и q простые числа
   3. Вычислить *n* и λ(n), где λ функция Кармайкла.
   4. Сгенерировать открытую экспоненту *e* (1 < *e* < λ(n)) и проверить что она является взаимно простой с λ(n).
   5. Вычислить закрытую экспоненту *d*, вывести открытый ключ *(n, e)* и закрытый ключ *(n, d)*.
2. Шифрование текста
   1. Прочитать из консоли открытый ключ *(n, e)*.
   2. Прочитать сообщение M (число меньшее *n*), зашифровать его алгоритмом RSA и вывести результат.
3. Расшифровка текста
   1. Прочитать из консоли закрытый ключ *(n, d)*.
   2. Прочитать зашифрованное сообщение, расшифровать его алгоритмом RSA и вывести результат.

Для возведения в степень должен использоваться алгоритм быстрого возведения в степень. Программа должна корректно работать для значений *n* < 232.

1. **Код программы**

*RSA.cs*

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace lab2

{

class Rsa

{

public Dictionary<string, List<Int64>> GenKeys(Int64 p, Int64 q)

{

// step 1

if (!isPrime(p) || !isPrime(q))

throw new Exception("Number(s) is not prime");

var keys = new Dictionary<string, List<Int64>>();

//step 2

Int64 n = p \* q;

//step 3

Int64 lambda = Lcm(p - 1, q - 1);

//step 4, 5

Int64 e = 0;

Int64 d = 0;

bool flag = true;

Dictionary<string, Int64> eucklid\_ifno;

for (Int64 i = 3; i<lambda && flag; i+=2)

{

eucklid\_ifno = GcdEx(i, lambda);

if (isPrime(i) && eucklid\_ifno["gcd"] == 1)

{

e = i;

d = eucklid\_ifno["x"];

flag = false;

}

}

if (flag)

throw new Exception("Programm can't find e");

keys.Add("public", new List<Int64>(){n, e});

keys.Add("private", new List<Int64>() { n, d });

return keys;

}

public Int64 Lcm(Int64 a, Int64 b)

{

var tmp = GcdEx(a, b)["gcd"];

return (a \* b) / GcdEx(a, b)["gcd"];

}

public Dictionary<string, Int64> GcdEx(Int64 a, Int64 b)

{

var res = new Dictionary<string, Int64>();

if(a==0)

return new Dictionary<string, Int64>(){

{ "gcd", b },

{ "x", 0 },

{ "y", 1 }

};

var tmpDict = GcdEx(b % a, a);

res.Add("x", tmpDict["y"] - (b / a) \* tmpDict["x"]);

res.Add("y", tmpDict["x"]);

res.Add("gcd", tmpDict["gcd"]);

return res;

}

public static bool isPrime(Int64 a)

{

Int64 sqrta = (Int64)Math.Ceiling(Math.Sqrt(a));

for (Int64 i = 2; i <= sqrta; i++)

if (a % i == 0)

return false;

return true;

}

public Int64 Encode(Int64 M, List<Int64> pubkey)

{

return FastPowFunc(M, pubkey[1], pubkey[0]);

}

public Int64 Decode(Int64 C, List<Int64> privatekey)

{

return FastPowFunc(C, privatekey[1], privatekey[0]);

}

public static Int64 FastPowFunc(Int64 Number, Int64 Pow, Int64 Mod)

{

Int64 Result = 1;

Int64 Bit = Number % Mod;

while (Pow > 0)

{

if ((Pow & 1) == 1)

{

Result \*= Bit;

Result %= Mod;

}

Bit \*= Bit;

Bit %= Mod;

Pow >>= 1;

}

return Result;

}

}

}

Program.cs

using System;

using System.Collections.Generic;

namespace lab2

{

class Program

{

static void Help()

{

Console.WriteLine("Simple RSA usage:");

Console.WriteLine("gen - generate key pair");

Console.WriteLine("enc - encode number");

Console.WriteLine("dec - decode number");

Console.WriteLine("quit - exit from programm\n");

}

static void GenKeys(Rsa rsa)

{

Int64 p, q;

Console.WriteLine("Input p:");

p = Int64.Parse(Console.ReadLine());

Console.WriteLine("Input q:");

q = Int64.Parse(Console.ReadLine());

try

{

var keys = rsa.GenKeys(p, q);

Console.WriteLine("Public Key:");

Console.WriteLine($"n: {keys["public"][0]}");

Console.WriteLine($"e: {keys["public"][1]}");

Console.WriteLine("\nPrivate Key:");

Console.WriteLine($"n: {keys["private"][0]}");

Console.WriteLine($"d: {keys["private"][1]}");

}

catch (Exception err)

{

Console.WriteLine(err);

}

}

static void Encode(Rsa rsa)

{

Int64 n, e, M;

Console.WriteLine("Input Public key");

Console.WriteLine("Input n:");

n = Int64.Parse(Console.ReadLine());

Console.WriteLine("Input e:");

e = Int64.Parse(Console.ReadLine());

Console.WriteLine("Input number for encode:");

M = Int64.Parse(Console.ReadLine());

try

{

var C = rsa.Encode(M, new List<Int64>() { n, e });

Console.WriteLine("Encoded number:");

Console.WriteLine(C.ToString());

}

catch (Exception err)

{

Console.WriteLine(err);

}

}

static void Decode(Rsa rsa)

{

Int64 d, n, C;

Console.WriteLine("Input Private key");

Console.WriteLine("Input n:");

n = Int64.Parse(Console.ReadLine());

Console.WriteLine("Input d:");

d = Int64.Parse(Console.ReadLine());

Console.WriteLine("Input number for decode:");

C = Int64.Parse(Console.ReadLine());

try

{

var M = rsa.Decode(C, new List<Int64>() { n, d });

Console.WriteLine("Decoded number:");

Console.WriteLine(M.ToString());

}

catch (Exception err)

{

Console.WriteLine(err);

}

}

static void Main(string[] args)

{

Help();

string command = "";

var rsa = new Rsa();

while(command != "quit")

{

Console.Write(">");

command = Console.ReadLine();

switch(command)

{

case "gen": GenKeys(rsa); break;

case "enc": Encode(rsa); break;

case "dec": Decode(rsa); break;

case "help": Help(); break;

case "quit": break;

default: Console.WriteLine("Uncorrect command"); break;

}

}

}

}

}

1. **Генерация ключей**

>gen

Input p:

3557

Input q:

2579

Public Key:

n: 9173503

e: 3

Private Key:

n: 9173503

d: 1527895

1. **Шифрование**

>enc

Input Public key

Input n:

9173503

Input e:

3

Input number for encode:

123456

Encoded number:

8692685

1. **Расшифровка**

>dec

Input Private key

Input n:

9173503

Input d:

1527895

Input number for decode:

8692685

Decoded number:

123456