Учреждение образования

«БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ ИНФОРМАТИКИ И РАДИОЭЛЕКТРОНИКИ»

Кафедра информатики

Отчёт

по лабораторной работе №4

**Ассиметричная криптография. Алгоритм Эль-Гамаля**

Выполнил:

Студент гр. 853504

Пресный В.И,

Проверил:

Олисейчик М.М.

Минск 2021

**1. Краткие теоретические сведения**

Схема Эль-Гамаля (Elgamal) — криптосистема с открытым ключом, основанная на трудности вычисления дискретных логарифмов в конечном поле. Криптосистема включает в себя алгоритм шифрования и алгоритм цифровой подписи. Схема Эль-Гамаля лежит в основе бывших стандартов электронной цифровой подписи в США (DSA) и России (ГОСТ Р 34.10-94).

Схема была предложена Тахером Эль-Гамалем в 1985 году. Эль-Гамаль разработал один из вариантов алгоритма Диффи-Хеллмана. Он усовершенствовал систему Диффи-Хеллмана и получил два алгоритма, которые использовались для шифрования и для обеспечения аутентификации. В отличие от RSA алгоритм Эль-Гамаля не был запатентован и, поэтому, стал более дешевой альтернативой, так как не требовалась оплата взносов за лицензию. Считается, что алгоритм попадает под действие патента Диффи-Хеллмана.

**Алгоритм создания ключей**

1. Генерируется случайное простое число p.

2. Выбирается целое число g — первообразный корень p.

3. Выбирается случайное целое число x такое, что 1<x<p-1

4. Вычисляется y=g^x(mod p).

5.Открытым ключом является y, закрытым ключом — число x.

**Алгоритм шифрования:**

Сообщение M должно быть меньше числа p. Сообщение шифруется следующим образом:

1. Выбирается сессионный ключ —случайное целое число k такое, что 1<k<p-1.

2. Вычисляются числа a=g^k (mod p) и b=y^(k)\*M (mod p).

3. Пара чисел (a,b) является шифротекстом.

Нетрудно увидеть, что длина шифротекста в схеме Эль-Гамаля длиннее исходного сообщения M вдвое.

**Алгоритм расшифрования:**

Зная закрытый ключ x, исходное сообщение можно вычислить из шифротекста (a,b) по формуле:

M=b(a^x)^(-1) (mod p)

При этом нетрудно проверить, что

(a^x)^(-1) (mod p) =g^(-kx) (mod p)

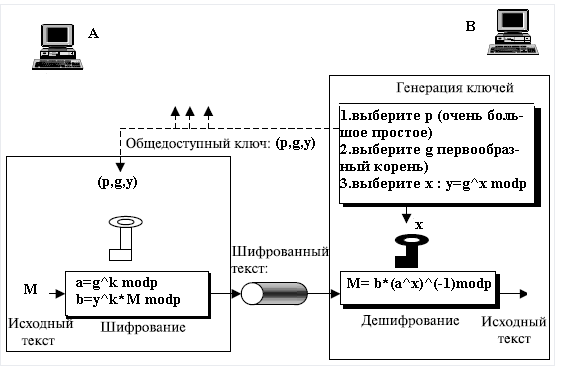
и поэтому

b(a^x)^(-1)=(y^k)M)g^(-xk) = (g^(xk)M)g^(-xk) = M (mod p).

Для практических вычислений больше подходит следующая формула:

M=b(a^x)^(-1)=ba^(p-1-x) (mod p)

**Схема алгоритма**



# Результаты выполнения

# 

# Выводы

# В результате выполнения лабораторной работы была получена реализация алгоритма Эль-Гамаля, а также алгоритм генерации ключей для данного алгоритма. Данный алгоритм относится к семейству асимметричных алгоритмов.

# ПРИЛОЖЕНИЕ 1. ИСХОДНЫЙ ТЕКСТ ПРОГРАММЫ

using System;

using System.Collections.Generic;

using System.Linq;

using System.Numerics;

using System.Text;

using System.Threading.Tasks;

namespace LR4\_MZI

{

public class ElGamal

{

public int X { get; set; }

private int Y { get; set; }

private int P { get; set; }

public int G { get; set; }

public ElGamal()

{

GenerateKeys();

}

public void GenerateKeys()

{

Random rand = new Random();

P = GetPrimeP();

G = GetPrimitiveRoot(P);

X = rand.Next(1, P - 1);

Y = (int)BigInteger.ModPow(G, X, P);

}

public (long, long[]) Encypt(byte[] data)

{

Random rand = new Random();

var k = rand.Next(1, P - 1);

long A;

long[] B = new long[data.Length];

A = (long)BigInteger.ModPow(G, k, P);

for (int i = 0;i<data.Length;i++)

{

B[i] = (long)(BigInteger.Pow(Y, k) \* data[i] % P);

}

return (A, B);

}

public byte[] Decrypt(long A, long[] B)

{

byte[] res = new byte[B.Length];

for(int i = 0;i<res.Length;i++)

{

res[i] = (byte)(B[i] \* BigInteger.Pow(A, P - 1 - X) % P);

}

return res;

}

private int GetPrimeP()

{

int max = 8000;

List<int> primes = new List<int>();

bool isPrime;

for (int j = 1000; j < max; j++)

{

isPrime = true;

for (int i = 2; i < j; i++)

{

if (j % i == 0)

{

isPrime = false;

break;

}

}

if (isPrime)

{

primes.Add(j);

}

}

Random rand = new Random();

return primes[rand.Next(primes.Count)];

}

public int GetPrimitiveRoot(int p)

{

List<int> arr = new List<int>();

int phi = p - 1, n = phi;

for (int i = 2; i \* i <= n; i++)

{

if (n % i == 0)

{

arr.Add(i);

while (n % i == 0)

{

n /= i;

}

}

}

if (n > 1)

{

arr.Add(n);

}

for (int g = 2; g <= p; g++)

{

bool ok = true;

for (int i = 0; i < arr.Count() && ok; i++)

{

ok &= BigInteger.ModPow(g, phi / arr[i], p) != 1;

}

if (ok)

{

return g;

}

}

throw new Exception("Cannot find primitive root");

}

}

}