Учреждение образования

«БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ ИНФОРМАТИКИ И РАДИОЭЛЕКТРОНИКИ»

Кафедра информатики

Отчёт

по лабораторной работе №3

**Ассиметричная криптография. RSA**

Выполнил:

Студент гр. 853504

Пресный В.И,

Проверил:

Олисейчик В.В.

Минск 2021

**1. Краткие теоретические сведения**

RSA (аббревиатура от фамилий Rivest, Shamir и Adleman) - криптографический алгоритм с открытым ключом, основывающийся на вычислительной сложности задачи факторизации больших целых чисел.

Криптосистема RSA стала первой системой, пригодной и для шифрования, и для цифровой подписи. Алгоритм используется в большом числе криптографических приложений, включая PGP, S/MIME, TLS/SSL, IPSEC/IKE и других.

Опубликованная в ноябре 1976 года статья Уитфилда Диффи и Мартина Хеллмана «Новые направления в криптографии» (англ. New Directions in Cryptography) перевернула представление о криптографических системах, заложив основы криптографии с открытым ключом. Разработанный впоследствии алгоритм Диффи — Хеллмана позволял двум сторонам получить общий секретный ключ, используя незащищенный канал связи. Однако этот алгоритм не решал проблему аутентификации. Без дополнительных средств пользователи не могли быть уверены, с кем именно они сгенерировали общий секретный ключ.

Изучив эту статью, трое учёных Рональд Ривест, Ади Шамир и Леонард Адлеман из Массачусетского технологического института (MIT) приступили к поискам математической функции, которая бы позволяла реализовать сформулированную Уитфилдом Диффи и Мартином Хеллманом модель криптографической системы с открытым ключом. После работы над более чем 40 возможными вариантами им удалось найти алгоритм, основанный на различии в том, насколько легко находить большие простые числа и насколько сложно раскладывать на множители произведение двух больших простых чисел, получивший впоследствии название RSA. Система была названа по первым буквам фамилий её создателей.

**Алгоритм создания открытого и секретного ключей**

RSA-ключи генерируются следующим образом:

1. Выбираются два различных случайных простых числа p и q заданного размера (например, 1024 бита каждое).

2. Вычисляется их произведение n = p\*q, которое называется модулем.

3/ Вычисляется значение функции Эйлера от числа n:

phi(n) = phi(pq) = (p-1)(q-1)

4.Выбирается целое число e (1<e<n), взаимно простое со значением функции phi(n)

Число e называется открытой экспонентой (англ. public exponent)

Обычно в качестве e берут простые числа, содержащие небольшое количество единичных бит в двоичной записи, например, простые из чисел Ферма: 17, 257 или 65537, так как в этом случае время, необходимое для шифрования с использованием быстрого возведения в степень будет меньше.

Слишком малые значения e, например 3, потенциально могут ослабить безопасность схемы RSA.[17]

5. Вычисляется число d, мультипликативно обратное к числу e по модулю phi(n)

Число d называется секретной экспонентой. Обычно оно вычисляется при помощи расширенного алгоритма Евклида.

6. Пара (e,n) публикуется в качестве открытого ключа RSA (англ. RSA public key).

7. Пара (d,n)играет роль закрытого ключа RSA (англ. RSA private key) и держится в секрете.

**Алгоритм шифрования:**

1.Взять открытый ключ (e,n)

2. Взять открытый текст m

3. Зашифровать сообщение с использованием открытого ключа:

c=E(m)=m^e (mod n)

**Алгоритм расшифрования:**

1. Принять зашифрованное сообщение c

2. Взять свой закрытый ключ (d,n)

3.Применить закрытый ключ для расшифрования сообщения:

m=D(c)=c^d(mod n)

**Схема алгоритма**



# Результаты выполнения

# 

# Выводы

# В результате выполнения лабораторной работы была получена реализация алгоритма RSA, 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 RSA

{

public class RSA

{

public RSA()

{

}

public long E { get; set; }

private long D { get; set; }

private long N { get; set; }

public long GCD(long a, long b)

{

long temp;

while(b != 0)

{

temp = b;

b = a % b;

a = temp;

}

return a;

}

public long[] ByteToLong(byte[] vs)

{

var res = new long[vs.Length / 8];

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

{

res[i] = BitConverter.ToInt64(vs, i \* 8);

}

return res;

}

public byte[] EncryptMy(byte[] data, int p, int q)

{

if(IsTheNumberSimple(p)==false || IsTheNumberSimple(q) == false)

{

throw new ArgumentException("Numbers should be prime!");

}

long n = p \* q;

long f = (p - 1) \* (q - 1);

long d = Calculate\_d(f);

long e = Calculate\_e(d, f);

byte[] res = new byte[data.Length \* 8];

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

{

BigInteger t = BigInteger.ModPow(data[i], (int)e, n);

BitConverter.GetBytes((long)t).CopyTo(res, i \* 8);

}

return res;

}

public byte[] Decrypt(byte[] data, int p, int q)

{

var arr = ByteToLong(data);

var res = new byte[arr.Length];

long n = p \* q;

long f = (p - 1) \* (q - 1);

long d = Calculate\_d(f);

long e = Calculate\_e(d, f);

for (long i = 0; i < arr.Length; i++)

{

BigInteger t = BigInteger.ModPow(arr[i], (int)d, n);

res[i] = (byte)t;

}

return res;

}

public long Calculate\_d(long f)

{

long d = 2;

while(true)

{

if (GCD(d, f) == 1)

return d;

d++;

}

}

private long Calculate\_e(long d, long f)

{

long e = 7;

while (true)

{

if ((e \* d) % f == 1)

break;

else

e++;

}

return e;

}

private bool IsTheNumberSimple(long n)

{

if (n < 2)

return false;

if (n == 2)

return true;

for (long i = 2; i < n; i++)

if (n % i == 0)

return false;

return true;

}

}

}