Федеральное государственное автономное

образовательное учреждение

высшего профессионального образования

«СИБИРСКИЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ»

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

институт

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

кафедра

**ОТЧЕТ ПО ЛАБОРАТОРНОЙ РАБОТЕ**

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

тема

\_\_\_\_\_\_\_\_\_\_\_\_Алгоритм шифрования RSA\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Преподаватель \_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_

подпись, дата инициалы, фамилия

Студент \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_

номер группы, зачетной книжки подпись, дата инициалы, фамилия

Оглавление

[Задание 4](#_Toc416260442)

[Описание алгоритма 5](#_Toc416260443)

[Получение открытого и закрытого ключей 5](#_Toc416260444)

[Шифрование 5](#_Toc416260445)

[Расшифровка 5](#_Toc416260446)

[Блок-схема алгоритма 6](#_Toc416260447)

[Листинг программы, реализующей алгоритмы 7](#_Toc416260448)

[Класс PartitionManager 7](#_Toc416260449)

[Класс RSACrypto 9](#_Toc416260450)

[Контрольные примеры 13](#_Toc416260451)

[Пример №1 13](#_Toc416260452)

[Пример №2 14](#_Toc416260453)

[Пример №3 15](#_Toc416260454)

[Пример №4 16](#_Toc416260455)

[Пример №5 17](#_Toc416260456)

[Заключение 18](#_Toc416260457)

# Задание

1. Составить в виде блок-схемы алгоритм шифрования/дешифрования RSA, со следующими особенностями:

* объём исходного текста – любой (в разумных пределах);
* исходный текст может состоять из русских и английских букв, цифр, а также знаков препинания;
* исходный текст находится в кодировке ASCII;
* N состоит из 38 десятичных знаков.
* Числа P и Q выбираются случайным образом, так, что , где P и Q – простые числа.
* исходный текст разбивается на K блоков, где K выбирается исходя из значения модуля N

1. Убедиться в правильности составления алгоритмов, а затем на языке C# составить программу, которая реализует данный алгоритм.
2. На ряде контрольных примеров (не менее 10) открытого текста проверить правильность работы алгоритмов шифрования и дешифрования (в качестве контрольного примера понимается текстовый файл в кодировке ASCII).
3. Оценить криптостойкость моего варианта алгоритма RSA, а также сделать оценку производительности, разработанной программы.

# Описание алгоритма

Получение открытого и закрытого ключей

* 1. Мы задаем два различных простых числа .
  2. Положим
  3. Вычисляем
  4. Случайным образом выбираем число и взаимно простое с
  5. Вычисляем e, такое что

Шифрование

* 1. Текст ***M*** разбивается на на блоки, так что размер каждого блока **.**
  2. Зашифровываем каждый блок:
  3. Зашифрованный передается.

Расшифровка

* 1. Расшифровываем каждый полученный блок зашифрованного текста

**:**

* 1. Блоки объединяются в сообщение.
  2. - исходный текст

# Блок-схема алгоритма



# Листинг программы, реализующей алгоритмы

## Класс PartitionManager

using System;

using System.Collections.Generic;

using System.Text;

namespace RSA\_Project

{

//Класс работающий с кодировками и разбиением текста

public static class PartitionManager

{

public static String PartitionToString(List<byte[]> p)

{

//Переобразует разбиение в строку

String result = "";

foreach (var b in p)

{

result += GetString(b);

}

return result;

}

public static List<byte[]> StringToPartition(String s, long length)

{

/\*Разбиваем строку на блоки

\* s - входная строка, length - длина блока в байтах

\*/

var result = new List<byte[]>();

byte[] bytes = GetBytes(s);

var block = new List<byte>();

int count = 0;

foreach (byte x in bytes)

{

count++;

if (count < length)

{

block.Add(x);

}

else

{

result.Add(block.ToArray());

block.Clear();

block.Add(x);

count = 1;

}

}

if (block.Count > 0) result.Add(block.ToArray());

return result;

}

public static byte[] GetBytes(string str)

{

/\*

\* Препобразует строку кодировки windows-1251 в байт-массив

\*

\*/

return Encoding.Convert(Encoding.Unicode,

Encoding.GetEncoding("windows-1251"),

Encoding.Unicode.GetBytes(str));

}

public static string GetString(byte[] bytes)

{

/\*

\* Преобразует байт-массив в строку в кодировке windows-1251

\* \*/

byte[] unicodeBytes = Encoding.Convert(

Encoding.GetEncoding("windows-1251"), Encoding.Unicode, bytes);

return Encoding.Unicode.GetString(unicodeBytes);

}

}

}

## Класс RSACrypto

using System;

using System.Numerics;

using System.Security.Cryptography;

namespace RSA\_Project

{ /\*

\* Алгоритм шифрования с открытм ключом

\*/

public class RsaCrypto

{

private static readonly Random Rnd = new Random((int) DateTime.Now.Ticks);

private static readonly RandomNumberGenerator Rng = RandomNumberGenerator.Create();

#region Методы для генерации случайного ключа

private static BigInteger Random(int bytes)

{

if (bytes == 0) return new BigInteger(0);

var buffer = new byte[bytes];

Rng.GetBytes(buffer);

buffer[bytes - 1] = (byte) ((buffer[bytes - 1] & 127) | 64); // старший байт

return new BigInteger(buffer);

}

private static bool IsPrimary(BigInteger x)

{

if (x < 2) return false; // отбрасываем отрицательные и единицу

int len = x.ToByteArray().Length;

int tests = 2\*len + 100; // Чем больше тестов тем меньше вероятность ошибиться

BigInteger y = x - 1;

for (int i = 0; i < tests; i++)

{

BigInteger a = (Random((int) (len\*Rnd.NextDouble()))%y) + 1; // берём ненулевое

// проверяем выполнение малой теоремы Ферма

// если простое то a^(x-1)==1 mod x

if (!(BigInteger.ModPow(a, y, x) - 1).IsZero) return false;

}

// признаём число простым

// хотя можем продолжать ошибаться

// и в случае ошибки стойкость рушится

return true;

}

private static BigInteger GeneratePrimary(int bytes)

{

BigInteger x = Random(bytes) | 1; // Простые являются нечётными

while (!IsPrimary(x)) x += 2; // Движемся вперёд пока не встретим простое

return x;

}

public void GeneratePairs(int decimals)

{

// 10^x==2^y

// x=y\*log10(2)

var bits = (int) Math.Ceiling(decimals/Math.Log10(2));

int bits1 = bits/2;

int bits2 = bits - bits1;

int bytes1 = (bits1 + 7)/8;

int bytes2 = (bits2 + 7)/8;

\_p = GeneratePrimary(bytes1);

\_q = GeneratePrimary(bytes2);

\_n = \_p\*\_q;

BigInteger eulierFunction = (\_p - 1)\*(\_q - 1);

\_e = \_primeNumbers[Rnd.Next(0, \_primeNumbers.Length)];

\_d = GeneratePrivateKey(eulierFunction);

}

#endregion

private readonly int[] \_primeNumbers = {17, 257, 65537}; //Простые числа для открытого ключа(взаимо простые с N)

private BigInteger \_d,

//private key

\_e; //public key

private BigInteger \_n; //p,q,n

private BigInteger \_p; //p,q,n

private BigInteger \_q; //p,q,n

public RsaCrypto(int decimals)

{

GeneratePairs(decimals);

}

public RsaCrypto(BigInteger p, BigInteger q)

{

/\*

\* \_p,\_q два простых числа которые инициализируют класс

\*/

\_p = p;

\_q = q;

\_n = \_p\*\_q; //Вычисляем число n как произведение целых чисел

BigInteger eF = (\_p - 1)\*(\_q - 1); //Считаем функцию эйлера

\_e = \_primeNumbers[Rnd.Next(0, \_primeNumbers.Length)]; //Находим взаимо-простое с ним

\_d = GeneratePrivateKey(eF);

}

public BigInteger N

{

get { return \_n; }

}

public BigInteger PrivateKey

{

get { return \_d; }

}

public BigInteger PublicKey

{

get { return \_e; }

}

public long NbyteLength

{

//Длина числа байт-масссива числа n

get { return \_n.ToByteArray().Length; }

}

public byte[] EncryptMessage(byte[] message)

{

/\*Шифруем сообщение открытым ключом

\* message - сообщение (в виде байт массива)

\*/

return BigInteger.ModPow(new BigInteger(message), \_d, \_n).ToByteArray();

}

public byte[] DecryptMessage(byte[] message)

{

//Расшифруем cообщение открытм ключом

return BigInteger.ModPow(new BigInteger(message), \_e, \_n).ToByteArray();

}

private BigInteger GeneratePrivateKey(BigInteger E)

{

//Вычисляем закрытый ключ, находя обратный по модулю элемент кольца

BigInteger x, y;

BigInteger g = NOD(\_e, E, out x, out y);

return (x%E + E)%E;

}

private BigInteger NOD(BigInteger a, BigInteger b, out BigInteger x, out BigInteger y)

{

/\*Расширеный Алгоритм Евклида

\* Решает диофантово уравнение

\* \*/

if (a == 0)

{

x = 0;

y = 1;

return b;

}

BigInteger x1, y1;

BigInteger d = NOD(b%a, a, out x1, out y1);

x = y1 - (b/a)\*x1;

y = x1;

return d;

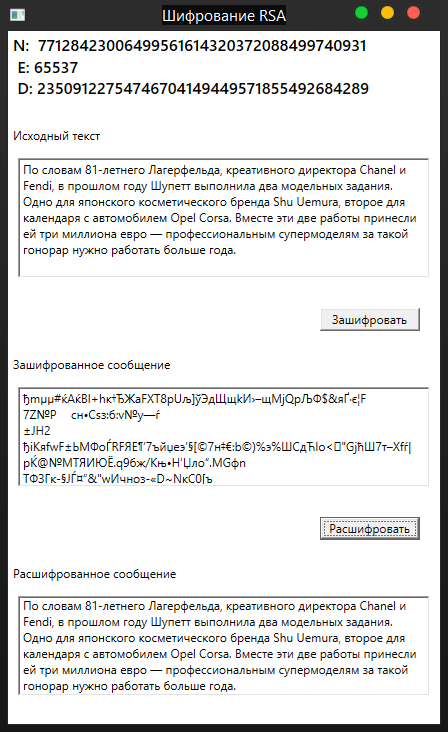
}

}

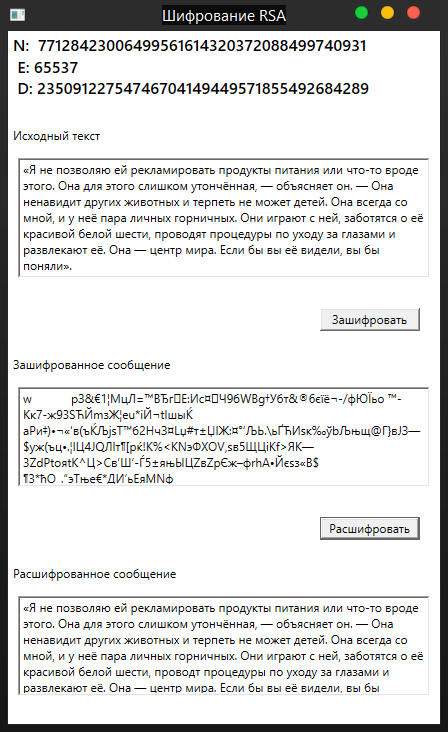
}

# Контрольные примеры

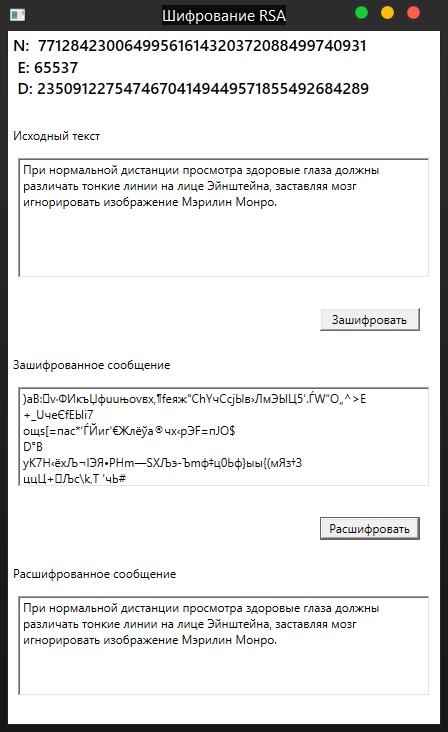
Пример №1



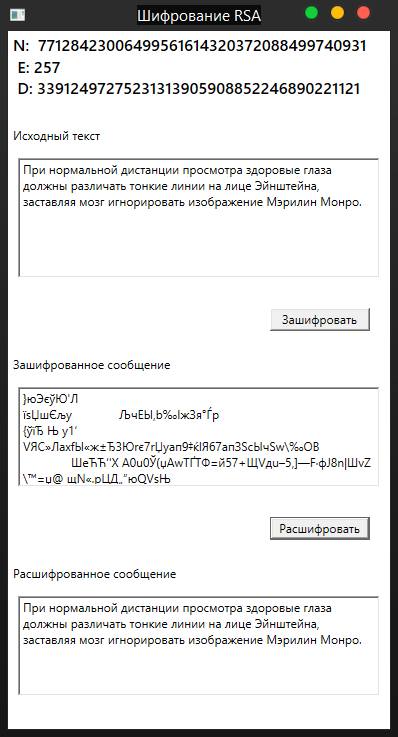
Пример №2



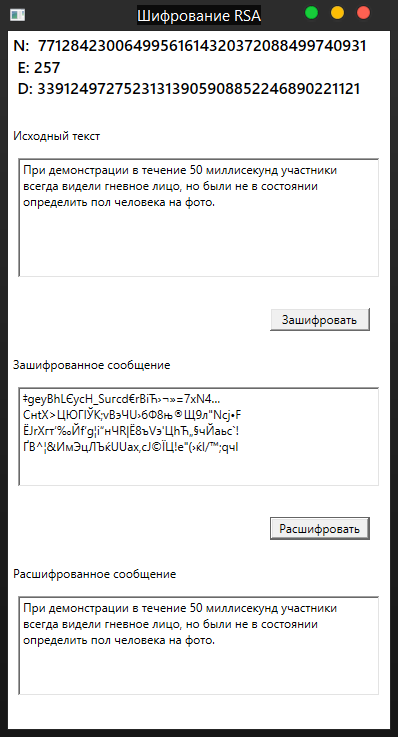
Пример №3



Пример №4



Пример №5



# Заключение

1. Полученные навыки:

* Изучены вспомогательные алгоритмы, такие как:
  + Обобщенный алгоритм Евклида
  + Вычисление функции Эйлера
  + Нахождение обратного по модулю элемента
* Длинная арифметика в .NET(BigInteger)
* Для создания графического интерфейса использовалась технология WPF
* Базовые знания теории чисел такие как:
  + Кольцо целых чисел
  + Функция Эйлера
  + Теорема Ферма
  + НОД
  + Сравнение по модулю

1. Алгоритм RSA хорошо себя показал, как действительно надежный алгоритм шифрования. Для его дешифровки без знаний больших простых чисел, лежащих в основе его построения, потребуются очень много времени, даже я бы сказал бесконечно много. Сложность дешифрования связана с длинами простых чисел. RSA – действительно криптостойкий алгоритм.