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 3](#_Toc417024399)

[Описание алгоритма 4](#_Toc417024400)

[Получение открытого и закрытого ключей 4](#_Toc417024401)

[Шифрование 4](#_Toc417024402)

[Расшифровка 4](#_Toc417024403)

[Блок-схема алгоритма 5](#_Toc417024404)

[Листинг программы, реализующей алгоритмы 6](#_Toc417024405)

[Класс PartitionManager 6](#_Toc417024406)

[Класс Cryptosystem 7](#_Toc417024407)

[Контрольные примеры 10](#_Toc417024408)

[Пример №1 10](#_Toc417024409)

[Пример №2 12](#_Toc417024410)

[Пример №3 13](#_Toc417024411)

[Пример №4 14](#_Toc417024412)

[Пример №5 16](#_Toc417024413)

[Тестирование производительности 17](#_Toc417024414)

[Заключение 22](#_Toc417024415)

# 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 44 десятичных знака.
* 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. Генерируем простые числа .
  2. Положим
  3. Вычисляем
  4. Случайным образом выбираем число и взаимно простое сможно взять простое
  5. Вычисляем ***e***, такое что

Шифрование

* 1. Исходный текст ***M*** разбивается на на блоки : **.**
  2. Шифруем каждый блок:
  3. Зашифрованный текст передается.

Расшифровка

* 1. Дешифруем каждый полученный блок зашифрованного текста по формуле :

**:**

* 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.Linq;

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 (byte[] 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

\*/

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

byte[] bytes = GetBytes(s);

List<byte> 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

\* \*/

var unicodeBytes = Encoding.Convert(

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

return Encoding.Unicode.GetString(unicodeBytes);

}

}

}

## Класс Cryptosystem

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;

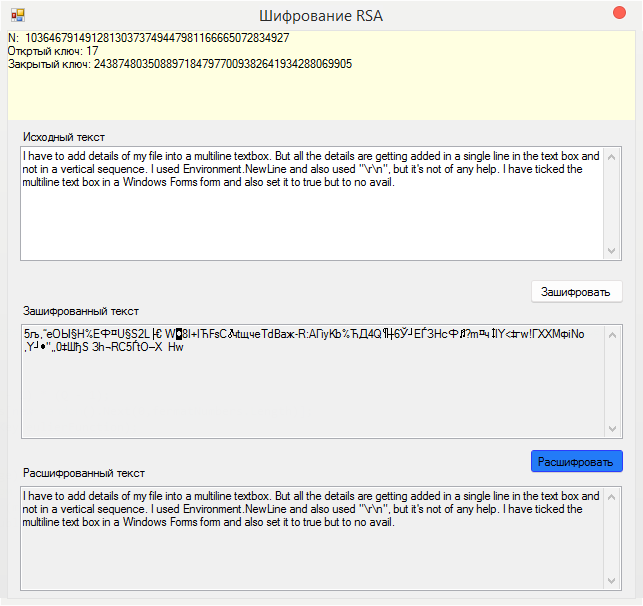
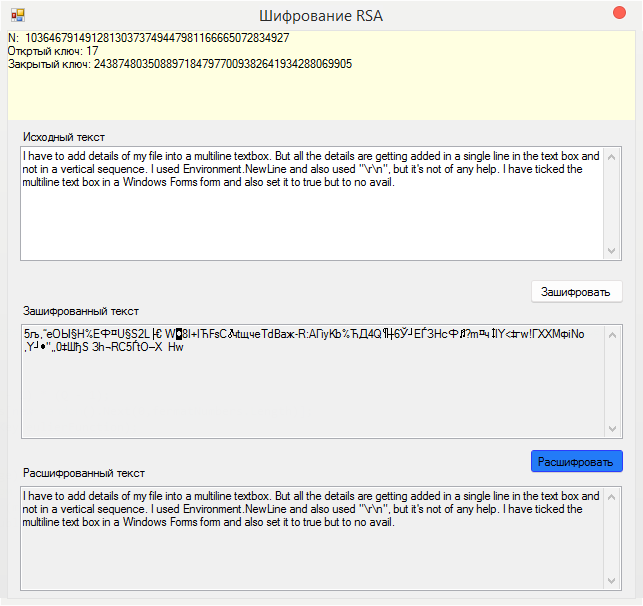
}

}

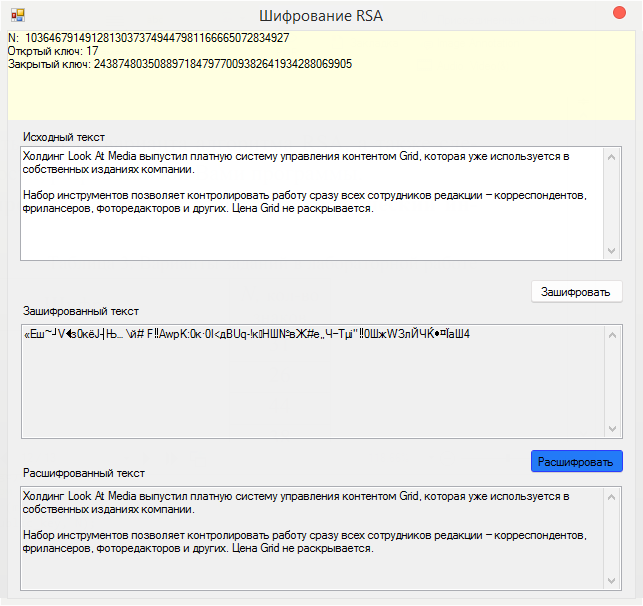
}

# Контрольные примеры

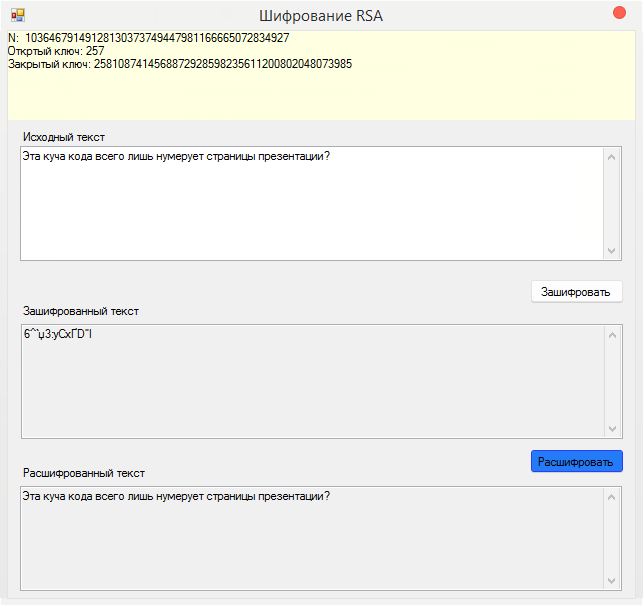
Пример №1



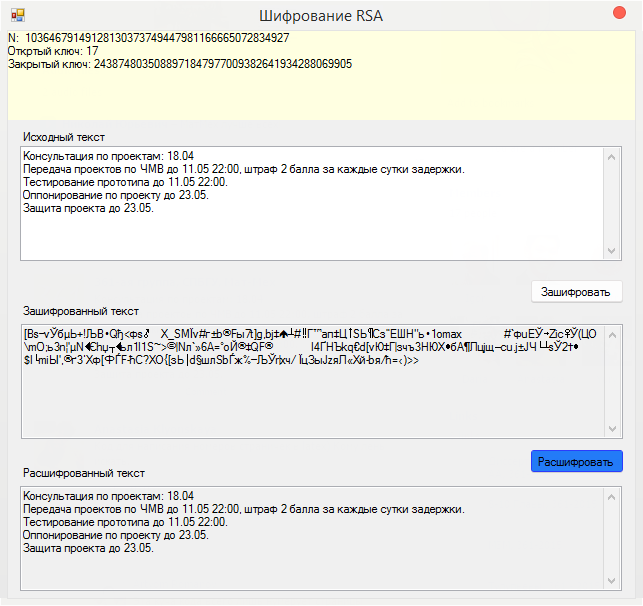
Пример №2



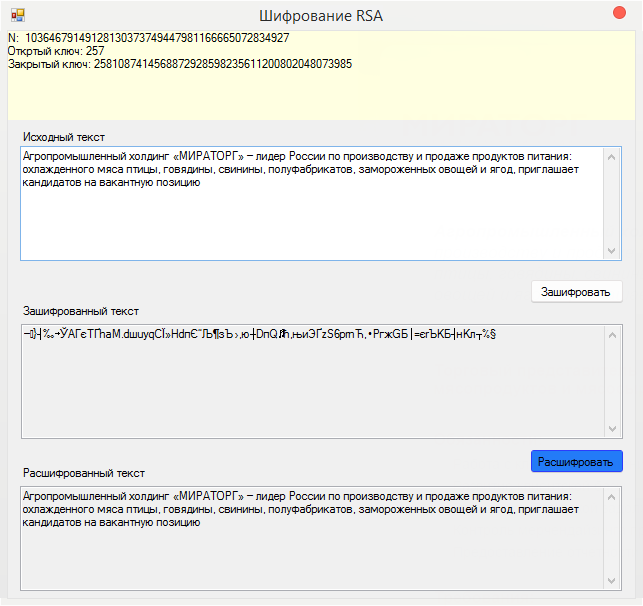
Пример №3



Пример №4

****

Пример №5



# Тестирование производительности

Windows 8.1 Профессиональная

Процессор: Intel Core Duo E8600 @ 3.33GHz 2.67GHz

Установленная память: 8,00 ГБ

Тип системы: 64-разрядная операционная система, процессор x64

Время измеряется в миллисекундах (1s == 1000ms)

N – количество десятичных разрядов

TestMethod1 – измерение среднего времени генерации ключа

TestMethod2 – измерение среднего времени генерации одного простого числа

TestMethod3 – измерение среднего времени одной итерации проверки числа на простоту

using System;

using System.Numerics;

using Microsoft.VisualStudio.TestTools.UnitTesting;

namespace RSACryptosystemProject.UnitTest

{

[TestClass]

public class UnitTest1

{

[TestMethod]

public void TestMethod1()

{

const int count = 20;

Console.WriteLine(@"# N GeneratePairs");

for (int decimals = 20; decimals < 50; decimals++)

{

var rsa = new Cryptosystem(decimals);

DateTime t = DateTime.Now;

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

rsa.GeneratePairs(decimals);

var ts = new TimeSpan(DateTime.Now.Ticks - t.Ticks);

Console.WriteLine(@"{0} {1}", decimals, ts.TotalMilliseconds/count);

}

}

[TestMethod]

public void TestMethod2()

{

const int count = 20;

Console.WriteLine(@"# N GeneratePrimary");

for (int decimals = 20; decimals < 50; decimals++)

{

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

int bytes = (bits + 7)/8;

DateTime t = DateTime.Now;

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

Cryptosystem.GeneratePrimary(bytes);

var ts = new TimeSpan(DateTime.Now.Ticks - t.Ticks);

Console.WriteLine(@"{0} {1}", decimals, ts.TotalMilliseconds/count);

}

}

[TestMethod]

public void TestMethod3()

{

const int count = 20;

Console.WriteLine(@"# N IsPrimary");

for (int decimals = 20; decimals < 50; decimals++)

{

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

int bytes = (bits + 7)/8;

DateTime t = DateTime.Now;

int total = 0;

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

{

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

total += Cryptosystem.NumberOfTests(x);

Cryptosystem.IsPrimary(x);

}

var ts = new TimeSpan(DateTime.Now.Ticks - t.Ticks);

Console.WriteLine(@"{0} {1}", decimals, ts.TotalMilliseconds/total);

}

}

}

}

**# N GeneratePairs**

**20 4.45027**

**21 4.500275**

**22 4.25029**

**23 4.350265**

**24 4.750315**

**25 6.050365**

**26 6.05035**

**27 6.20037**

**28 6.55039**

**29 6.550405**

**30 7.750485**

**31 7.650445**

**32 7.15041**

**33 7.50041**

**34 8.00043**

**35 8.65051**

**36 8.650515**

**37 9.250525**

**38 9.75058**

**39 14.85087**

**40 15.150855**

**41 14.900875**

**42 14.3508**

**43 14.55084**

**44 17.250975**

**45 17.75108**

**46 17.801005**

**47 17.401**

**48 16.600975**

**49 19.501245**

**# N GeneratePrimary**

**20 7.650435**

**21 7.400405**

**22 8.750515**

**23 8.750485**

**24 8.250495**

**25 9.450535**

**26 10.45062**

**27 12.600745**

**28 11.400665**

**29 17.400985**

**30 16.80097**

**31 18.251055**

**32 19.201105**

**33 20.901235**

**34 21.40125**

**35 22.801285**

**36 23.451365**

**37 26.20152**

**38 37.35216**

**39 36.40211**

**40 34.30199**

**41 39.802315**

**42 36.30209**

**43 39.602285**

**44 39.602305**

**45 41.452395**

**46 47.85275**

**47 42.452475**

**48 50.202915**

**49 72.354165**

**# N IsPrimary**

**20 0.00423775423728814**

**21 0.000423474576271187**

**22 0.000416708333333333**

**23 0.000416666666666667**

**24 0.00375020833333333**

**25 0.00409856557377049**

**26 0.00409868852459016**

**27 0.00403221774193548**

**28 0.000806733870967742**

**29 0.0011906746031746**

**30 0.00754003968253968**

**31 0.00119043650793651**

**32 0.000781171875**

**33 0.01328203125**

**34 0.00115411538461538**

**35 0.000769269230769231**

**36 0.00769257692307692**

**37 0.000757613636363636**

**38 0.00113643939393939**

**39 0.00149272388059702**

**40 0.0115678731343284**

**41 0.00147066176470588**

**42 0.00183830882352941**

**43 0.0113976470588235**

**44 0.0134067028985507**

**45 0.00144923913043478**

**46 0.00178578571428571**

**47 0.012858**

**48 0.0139292857142857**

**49 0.017606514084507**

# 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 Windows Forms
* Базовые знания алгебры такие как:
  + Целые числа
  + Кольца
  + Теорема Ферма

1. Алгоритм RSA продемонстрировал себя с лучшей стороны, он очень сложен для взлома. Так как задача разложения огромных чисел на простые множители очень трудоемкая и не существует эффективных алгоритма их поиска. Сложность алгоритма растет с увеличением значения произведения заданных нами простых чисел.