Федеральное государственное автономное

образовательное учреждение

высшего профессионального образования

«СИБИРСКИЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ»

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

институт

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

кафедра

**ОТЧЕТ ПО ЛАБОРАТОРНОЙ РАБОТЕ**

Алгоритм шифрования RSA

тема

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

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

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

подпись, дата инициалы, фамилия

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

номер группы, зачетной книжки подпись, дата инициалы, фамилия

Красноярск 2015

**Задания**

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. Оцените криптостойкость моего варианта алгоритма RSA, а также сделать оценку производительности, разработанной программы.

**Описание алгоритма**

1. Генерация открытого и закрытого ключей
   1. Выбираются два различных случайных простых числа заданного размера.
   2. Вычисляется их произведение
   3. Вычисляется значение *функции Эйлера*
   4. Случайным образом выбирается число и взаимно простое с
   5. Вычисляется ***e***, такое что - обратный элемент в кольце целых чисел
2. Шифрование
   1. Сообщение ***M*** разбивается на на ***K***, блоков : так что размер каждого блока **.**
   2. Каждый из блоков шифруется по формуле
   3. Зашифрованное сообщениепередается.
3. Расшифровка
   1. Для каждого блока из полученного  вычисляется

* 1. Блоки объединяются в сообщение.
  2. - исходное сообщение

**Блок-схема алгоритма**

**Листинг программы**

public class RSA

{

private int[] fermatNumbers = new int[] { 17, 257, 65537 };//Числа ферма

private byte lengthN;//длина числа n

private BigInteger N, P, Q;//P,Q -простые числа. N - их произведение

private BigInteger public\_key, private\_key;//откртый и закрытый ключи

public RSA(byte lengthN)

{//Инициализация класса

this.lengthN = lengthN;

KeyValuePair<BigInteger,BigInteger> pair = generatePrimesPair(lengthN%2 - lengthN%10);//Генерирует пару простых чисел

this.P = pair.Key;

this.Q = pair.Value;

this.N = P \* Q;

BigInteger eulierFunction = (P - 1) \* (Q - 1);

this.public\_key = fermatNumbers[new Random().Next(0,fermatNumbers.Length-1)];

this.private\_key = calcPrivateKey(eulierFunction);

}

public byte[] Encrypt(byte[] message)

{//Шифруем сообщение открытым ключом

BigInteger val = new BigInteger(message);

val = BigInteger.ModPow(val, public\_key, N);

return val.ToByteArray();

}

public long ModuleLength

{//Длина модуля в байтах

get

{

long l = N.ToByteArray().Length;

return l;

}

}

public byte[] Decrypt(byte[] message)

{//Расшифруем cообщение открытм ключом

BigInteger val = new BigInteger(message);

val = BigInteger.ModPow(val, private\_key, N);

return val.ToByteArray();

}

public RSA(BigInteger P, BigInteger Q)

{

this.P = P;

this.Q = Q;

this.N = P \* Q;

BigInteger eulierFunction = (P - 1) \* (Q - 1);

this.public\_key = 11;

this.private\_key = calcPrivateKey(eulierFunction);

}

private BigInteger calcPrivateKey(BigInteger E)

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

BigInteger x,y;

BigInteger g = GCD(public\_key,E ,out x,out y);

return (x % E + E) % E;

}

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

{//Расширеный Алгоритм Евклида

if (a.IsZero)//Находит НОД чисел A и B, и коэфициенты x,y уравнения Ax +By = НОД(A,B)

{

x = 0;

y = 1;

return b;

}

BigInteger x1, y1;

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

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

y = x1;

return d;

}

private KeyValuePair<BigInteger,BigInteger> generatePrimesPair(int length)

{//Генерирует пару простых чисел, таких что их произведение имеет ровно lengthN знаков

List<BigInteger> resList = new List<BigInteger>();

BigInteger p = BigInteger.One;

int curLength = 1;

while (curLength<length)

{

curLength++;

p \*= 10;

}

BigInteger lowLimit = p;

BigInteger upLimit = p\*10;

//Генерируем число в заданном диапазоне

RandomNumberGenerator rng = RandomNumberGenerator.Create();

byte[] bytes = new byte[p.ToByteArray().LongLength];

while (p<=lowLimit || p>upLimit)

{

rng.GetBytes(bytes);

p = new BigInteger(bytes);

}

//Ищем ближайшее простое

while (!IsProbablePrime(p, 40))

{

p += 1;

}

// Генерируем число в диапазоне 10^lengthN - 10^(lengthN+1)

while (curLength < lengthN)

{

curLength++;

lowLimit \*= 10;

}

upLimit = lowLimit \*10;

bytes = new byte[lowLimit.ToByteArray().LongLength];

BigInteger q;

do

{

rng.GetBytes(bytes);

q = new BigInteger(bytes);

}

while (q <= lowLimit || q >= upLimit);

//делим с остатком на найденное просто число

q = q / p;

while (!IsProbablePrime(q, 40))

{

q += 1;

}

//ищем ближайшее простое к нему

return new KeyValuePair<BigInteger,BigInteger>(p,q);//Возвращаем пару найденных чисел

}

private bool IsProbablePrime(BigInteger source, int certainty)

{//Вероятностный тест Миллера-Рабина для определения простоты числа

if (source == 2 || source == 3)

return true;

if (source < 2 || source % 2 == 0)

return false;

BigInteger d = source - 1;

int s = 0;

while (d % 2 == 0)

{

d /= 2;

s += 1;

}

RandomNumberGenerator rng = RandomNumberGenerator.Create();

byte[] bytes = new byte[source.ToByteArray().LongLength];

BigInteger a;

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

{

do

{

rng.GetBytes(bytes);

a = new BigInteger(bytes);

}

while (a < 2 || a >= source - 2);

BigInteger x = BigInteger.ModPow(a, d, source);

if (x == 1 || x == source - 1)

continue;

for (int r = 1; r < s; r++)

{

x = BigInteger.ModPow(x, 2, source);

if (x == 1)

return false;

if (x == source - 1)

break;

}

if (x != source - 1)

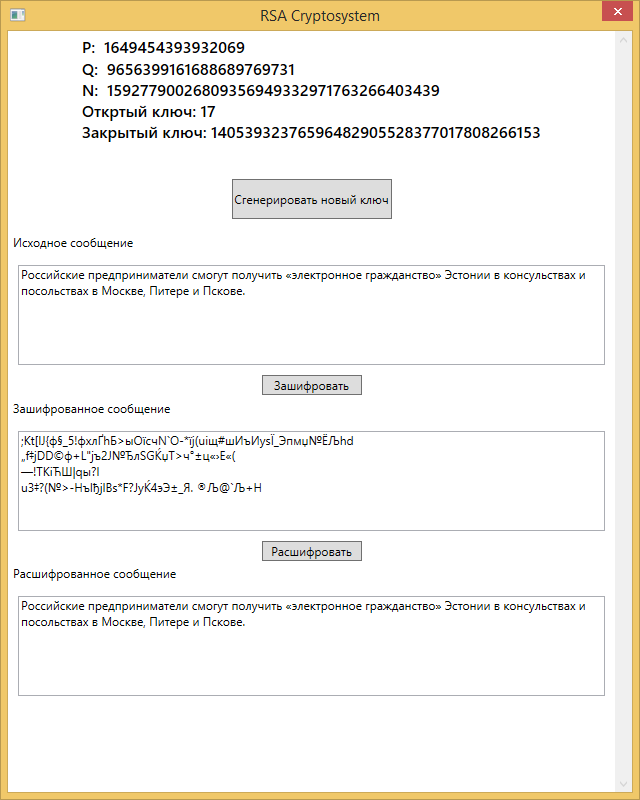
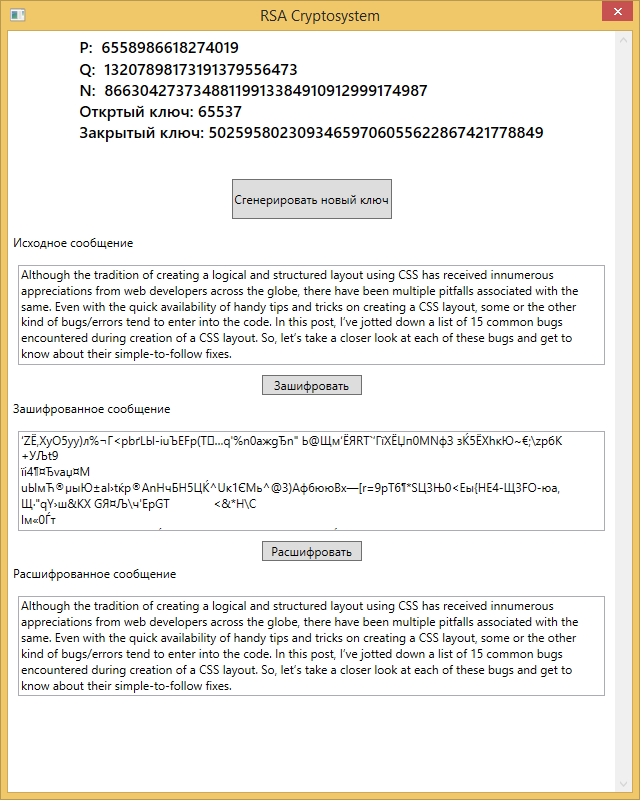
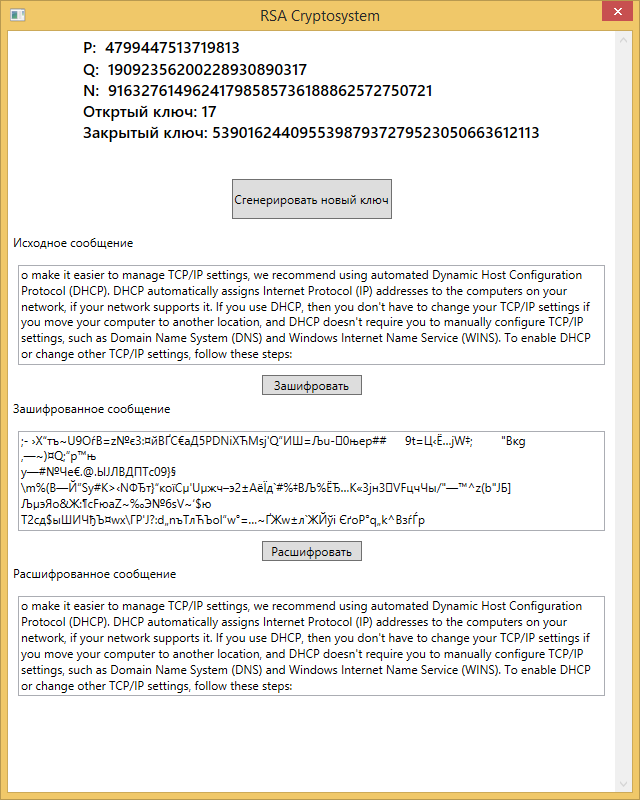
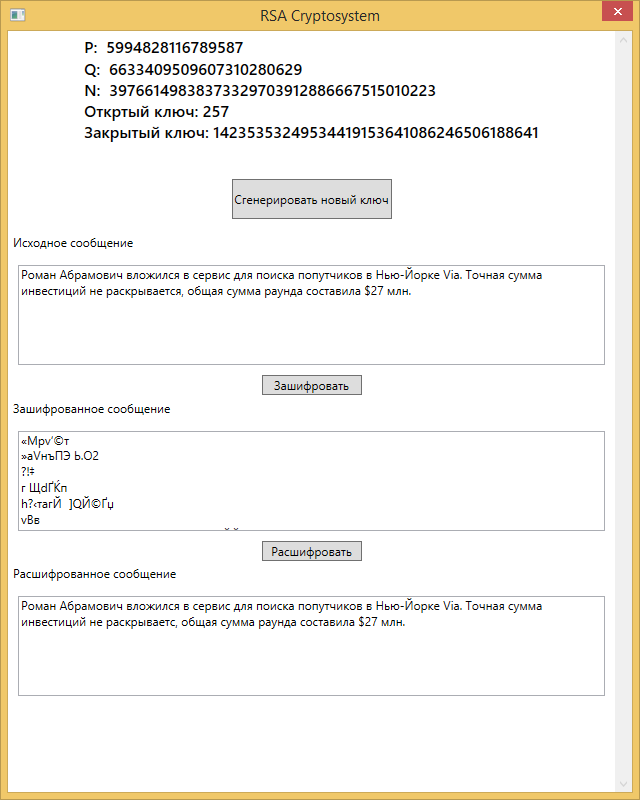
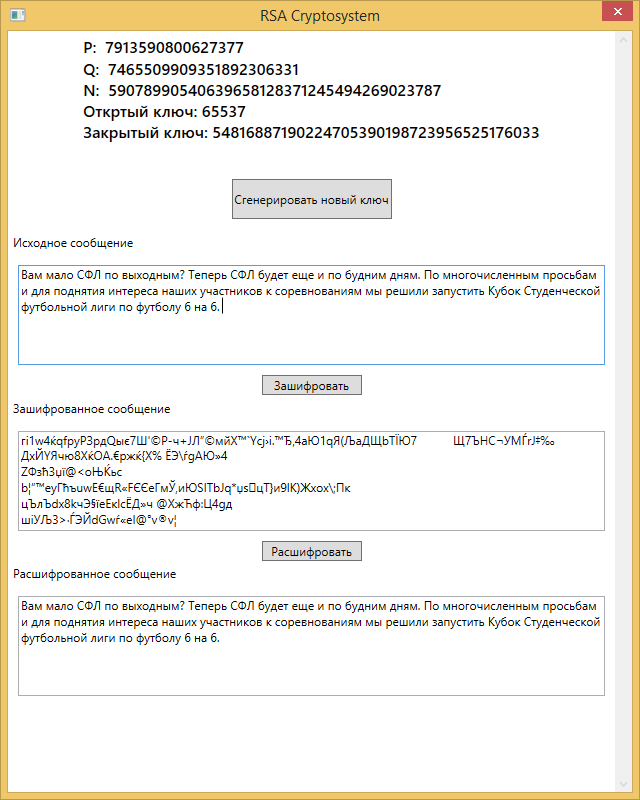
return false;

}

return true;

}

}

**Примеры работы программы: **

**Заключения и выводы:**

1. В ходе работы были изучены следующие дополнительные навыки:

* Обобщеный алгоритм Евклида
* Вероятностный тест на простоту Миллера-Рабина
* Работ с большими числами в .NET
* Реализация оконного интерфейса с помощью технологии WPF
* Базовые знания теории чисел такие как:
  + Кольцо целых чисел
  + Функция Эйлера
  + Теорема Ферма
  + НОД
  + Сравнение по модулю

1. Алгоритм RSA является криптостойким и это свойство зависит напрямую от длины модуля N. Чем длиннее число тем быстрее его факторизовать.