Министерство образования Республики Беларусь
Учреждение образования «Белорусский государственный университет информатики и радиоэлектроники»

Факультет компьютерных систем и сетей

кафедра Информатики

Дисциплина: Программирование

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА
к курсовой работе
на тему

Шифрование данных методом RSA

Студент: гр. 953501 Жамойдик Н.С.

Руководитель: ассистент кафедры информатики
Удовин И.А.

Минск 2020

Содержание

Введение

Криптогра́фия (от др.-греч. κρυπτός «скрытый» + γράφω «пишу») — наука о методах обеспечения конфиденциальности (невозможности прочтения информации посторонним), целостности данных (невозможности незаметного изменения информации), аутентификации (проверки подлинности авторства или иных свойств объекта).

Изначально криптография изучала методы шифрования информации — обратимого преобразования открытого (исходного) текста на основе секретного алгоритма или ключа в шифрованный текст (шифротекст). Традиционная криптография образует раздел симметричных криптосистем, в которых зашифровывание и расшифровывание проводится с использованием одного и того же секретного ключа. Помимо этого раздела современная криптография включает в себя асимметричные криптосистемы, системы электронной цифровой подписи (ЭЦП), хеш-функции, управление ключами, получение скрытой информации, квантовую криптографию.

Криптография не занимается защитой
от обмана, подкупа или шантажа законных абонентов, кражи ключей и других угроз информации, возникающих в защищённых системах передачи данных.

Криптография — одна из старейших наук, её история насчитывает несколько тысяч лет.

Для современной криптографии характерно использование открытых алгоритмов шифрования, предполагающих использование вычислительных

средств. Известно более десятка проверенных алгоритмов шифрования, которые при использовании ключа достаточной длины и корректной реализации алгоритма криптографически стойки. Распространённые алгоритмы:

- симметричные DES, AES, ГОСТ 28147-89, Camellia, Twofish, Blowfish, IDEA, RC4 и др.;
- асимметричные RSA и Elgamal (*Эль-Гамаль*);
- хеш-функций MD4, MD5, MD6, SHA-1, SHA-2, ГОСТ Р 34.11-2012 («Стрибог») .

Криптографические методы стали широко использоваться частными лицами в электронных коммерческих операциях, телекоммуникациях и многих других средах.

1. Анализ предметной области

1.1 История возникновения

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*)[4] 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.

В августе 1977 года в колонке «Математические игры» Мартина Гарднера в журнале Scientific American с разрешения Рональда Ривеста[5] появилось первое описание криптосистемы RSA[6]. Читателям также было предложено дешифровать английскую фразу, зашифрованную описанным алгоритмом:

9686	9613	7546	2206
1477	1409	2225	4355
8829	0575	9991	1245
7431	9874	6951	2093
0816	2982	2514	5708
3569	3147	6622	8839
8962	8013	3919	9055
1829	9451	5781	5154

В качестве открытых параметров системы были использованы числа n=1143816...6879541 (129 десятичных знаков, 425 бит, также известно как RSA-129) и e=9007. За расшифровку была обещана награда в 100 долларов США. По заявлению Ривеста, для факторизации числа потребовалось бы более 40 квадриллионов лет. Однако чуть более, чем через 15 лет, 3 сентября 1993 года было объявлено о запуске проекта распределённых вычислений с координацией через электронную почту по нахождению сомножителей числа RSA-129 и решению головоломки. На протяжении полугода более 600 добровольцев из 20 стран жертвовали процессорное время 1600 машин (три из которых были факс-машинами). В результате были найдены простые множители и расшифровано исходное сообщение, которое представляет собой фразу «THE MAGIC WORDS ARE SQUEAMISH OSSIFRAGE (англ.)» («Волшебные слова — это брезгливый ягнятник»)[8][9]. Полученную награду победители пожертвовали в фонд свободного программного обеспечения.

После публикации Мартина Гарднера полное описание новой криптосистемы любой желающий мог получить, выслав по почте запрос Рональду Ривесту с

приложенным конвертом с обратным адресом и марками на 35 центов. Полное описание новой криптосистемы было опубликовано в журнале «Communications of the ACM» в феврале 1978 года.

Заявка на патент была подана 14 декабря 1977 года, в качестве владельца был указан MIT. Патент 4405829 был выдан 20 сентября 1983 года, а 21 сентября 2000 года срок его действия истёк. Однако за пределами США у изобретателей патента на алгоритм не было, так как в большинстве стран его необходимо было получить до первой публикации.

1.2 Постановка задачи

Основной задачей курсовой работы является разработка приложения, способного зашифровать данные, расшифровать данные, создать цифровую подпись. В качестве среды разработки приложения была выбрана Microsoft Visual Studio. Язык программирования – C#.

2. Описание алгоритма

2.1Введение

Как и было сказано ранее, RSA является методом шифрования с открытым ключом. Это значит, что данные будут зашифровываться одним ключом (открытым), который может быть известен каждому, а расшифровываться данные будут другим ключом (закрытым), который будет известен только вам.

Криптографические системы с открытым ключом используют так называемые односторонние функции, которые обладают следующим свойством:

- если известно x, то $F(x)$ вычислить относительно просто;
- если известно $y = F(x)$, то для вычисления x нет простого (эффективного) пути.

Под односторонностью понимается не теоретическая однонаправленность, а практическая невозможность вычислить обратное значение, используя современные вычислительные средства, за обозримый интервал времени.

В основу криптографической системы с открытым ключом RSA положена сложность задачи факторизации произведения двух больших простых чисел. Для шифрования используется операция возведения в степень по модулю большого числа. Для дешифрования (обратной операции) за разумное время необходимо уметь вычислять функцию Эйлера от данного большого числа, для чего необходимо знать разложение числа на простые множители.

В криптографической системе с открытым ключом каждый участник располагает как открытым ключом, так и закрытым ключом. В криптографической системе RSA каждый ключ состоит из пары целых чисел.

Каждый участник создаёт свой открытый и закрытый ключ самостоятельно. Закрытый ключ каждый из них держит в секрете, а открытые ключи можно сообщать кому угодно или даже публиковать их. Открытый и закрытый ключи каждого участника обмена сообщениями в криптосистеме RSA образуют «согласованную пару» в том смысле, что они являются взаимно обратными, то есть:

“Для всяких допустимых пар открытого и закрытого ключей (p,s) существуют соответствующие функции шифрования $E_p(x)$ и расшифрования D_s(x) такие, что всякое сообщение m принадлежит $m \in M$, где M – множество допустимых сообщений, $m = D_s(E_p(x)) = E_p(D_s(x))$.”

2.2Алгоритм

Ключи шифрование генерируются следующим образом:

1) Выбираются 2 случайных простых числа p и q;
2) Вычисляется их произведение $n = p*q$, которое называется модулем;
3) Вычисляется значение функции Эйлера от числа n

$$\varphi(n) = (p-1) * (q-1)$$

4) Выбирается целое число e ($1<e<\varphi(n)$), взаимно простое со значением функции $\varphi(n)$
5) Вычисляется число d, мультипликативно обратное числу e по модулю $\varphi(n)$

$$d * e = 1\ mod(\varphi(n))$$

6) Пара (e, n) публикуется в качестве открытого ключа шифрования
7) Пара (d, n) является закрытым ключом шифрования и держится в секрете

3.3 Шифрование и дешифрование

Шифрование

- Берётся открытый ключ (e, n)
- Берётся открытый текст m
- Сообщение шифруется с использованием открытого ключа

$$c = E(m) = m^e \bmod n$$

Дешифрование

- Принимается зашифрованное сообщение c
- Берётся свой закрытый ключ (d, n)
- Сообщение дешифруется с использование закрытого ключа

$$m = D(c) = c^d \bmod n$$

Данная схема на практике не используется по причине того, что она не является практически надёжной. Действительно, односторонняя функция $E(m)$ является детерминированной — при одних и тех же значениях входных параметров выдаёт одинаковый результат. Это значит, что не выполняется необходимое условие практической надёжности шифра.

2.3Цифровая подпись

Система RSA может использоваться не только для шифрования, но и для цифровой подписи.

Алгоритм отправителя

- Взять открытый текст m
- Создать цифровую подпись s с помощью своего секретного ключа (d,n)

$$s = S_A(m) = m^d \ mod \ n$$

- Передать пару (m,s)

Алгоритм получателя

- Принять пару (m,s)
- Взять открытый ключ отправителя (e,n)
- Вычислить прообраз сообщения из подписи

$$m' = P_A(s) = s^e \ mod \ n$$

- Проверить подлинность подписи, сравнив m и m'

Поскольку цифровая подпись обеспечивает как аутентификацию автора сообщения, так и подтверждение целостности содержимого подписанного сообщения, она служит аналогом подписи, сделанной от руки в конце рукописного документа.

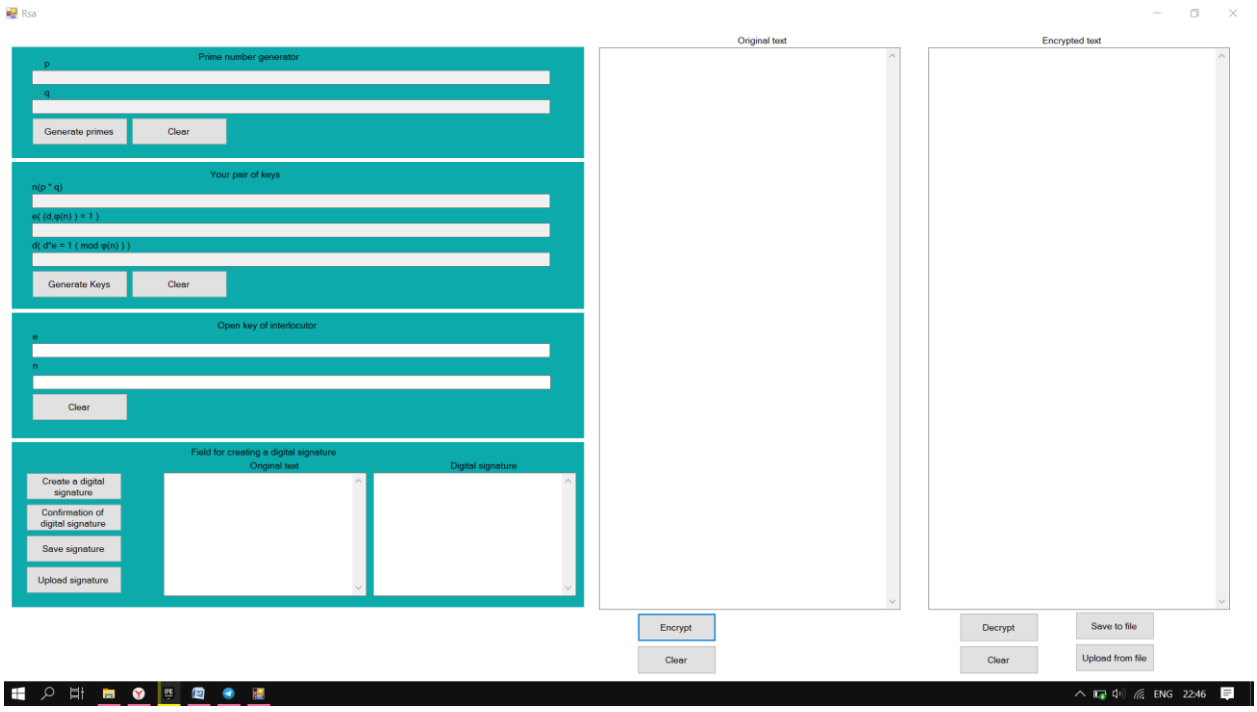
Важное свойство цифровой подписи заключается в том, что её может проверить каждый, кто имеет доступ к открытому ключу её автора. Один из участников обмена сообщениями после проверки подлинности цифровой подписи может передать подписанное сообщение ещё кому-то, кто тоже в состоянии проверить эту подпись.

Заметим, что подписанное сообщение m не зашифровано. Оно пересылается в исходном виде и его содержимое не защищено от нарушения конфиденциальности. Путём совместного применения представленных выше схем шифрования и цифровой подписи в системе RSA можно создавать сообщения, которые будут и зашифрованы, и содержать цифровую подпись. Для этого автор сначала должен добавить к сообщению свою цифровую подпись, а затем — зашифровать получившуюся в результате пару (состоящую из самого сообщения и подписи к нему) с помощью открытого ключа, принадлежащего получателю. Получатель расшифровывает полученное сообщение с помощью своего секретного ключа. Если проводить аналогию с пересылкой обычных бумажных документов, то этот процесс похож на то, как если бы автор документа поставил под ним свою печать, а затем положил его в бумажный конверт и запечатал, с тем чтобы конверт был распечатан только тем человеком, кому адресовано сообщение.

3.Методика работы с программой

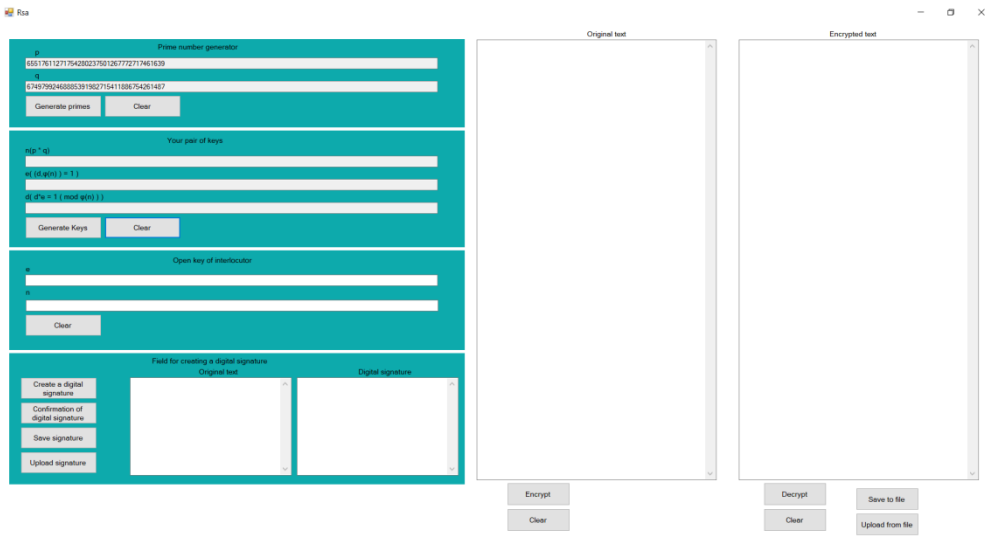
Интерфейс программы состоит из нескольких блоков:

- Поле генерации простых чисел
- Поле генерации собственных ключей
- Поле открытого ключа собеседника
- Поле создания и подтверждения цифровой подписи
- Поле текста предназначенного для шифрования
- Поле текста предназначенного для дешифрования

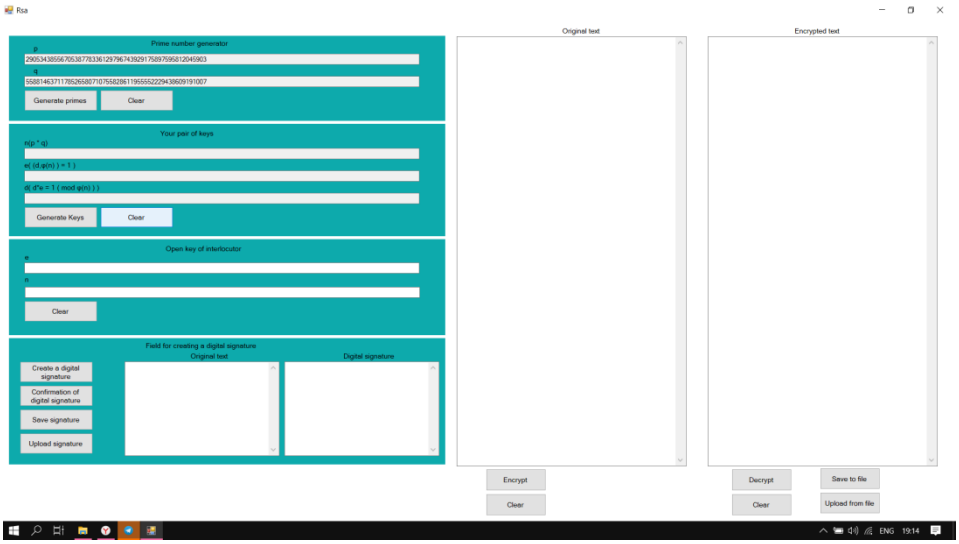


Изначально собеседники генерируют простые числа.

Собеседник А

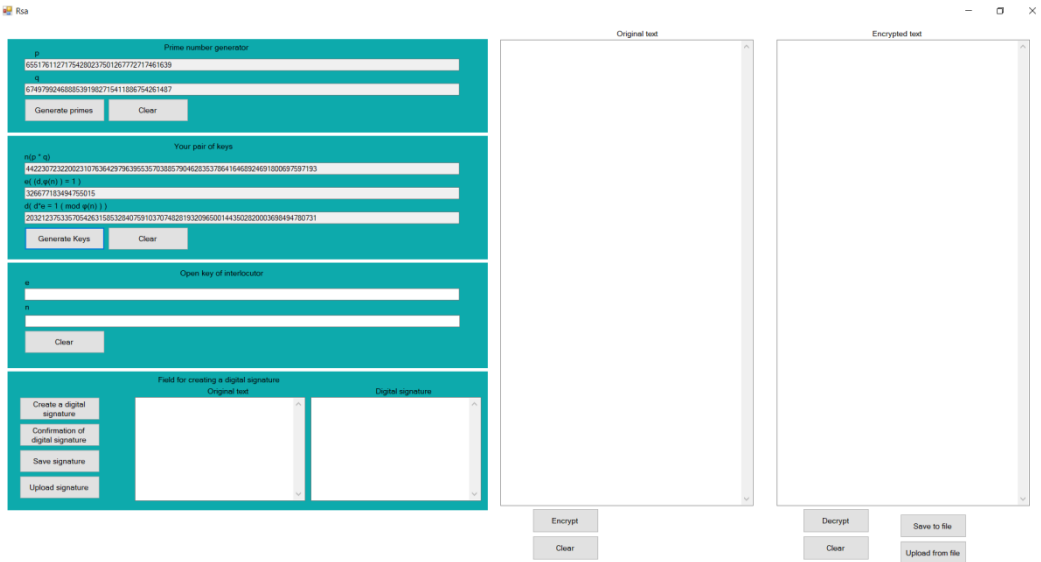


Собеседник Б

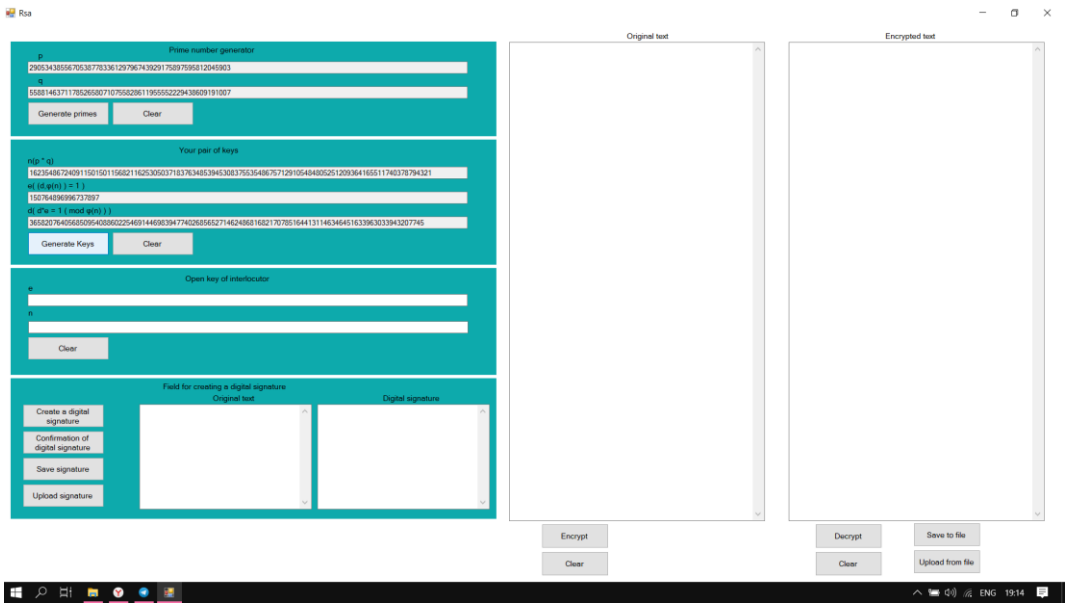


Затем собеседники генерируют свои пары открытого и закрытого ключей (поля с простыми числами и ключами имеют модификатор readonly, что значит что пользователь не может ввести числа сам)

Собеседник А

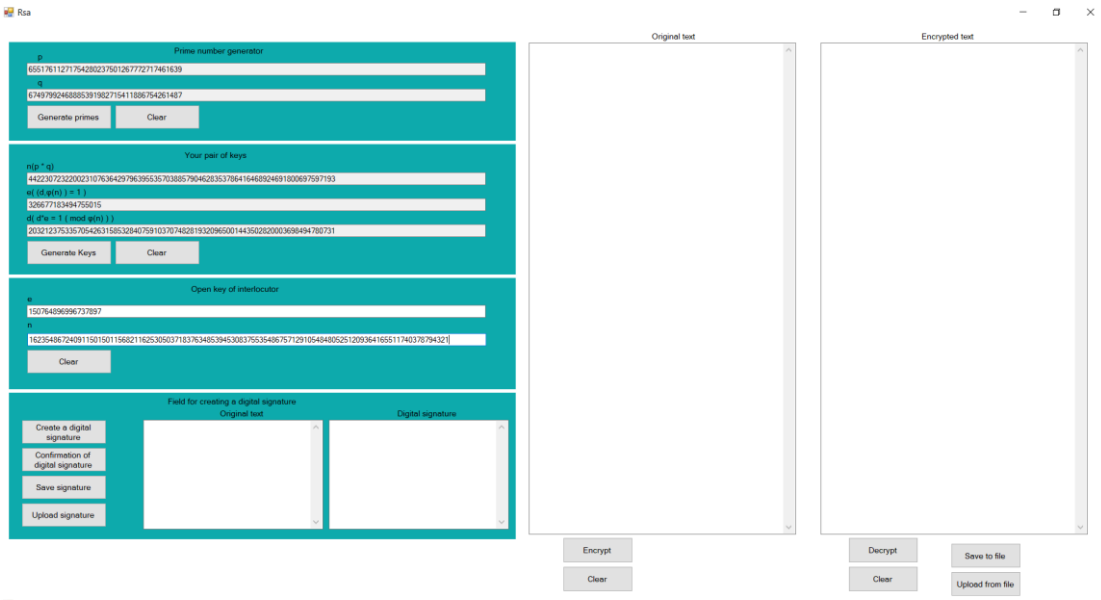


Собеседник Б

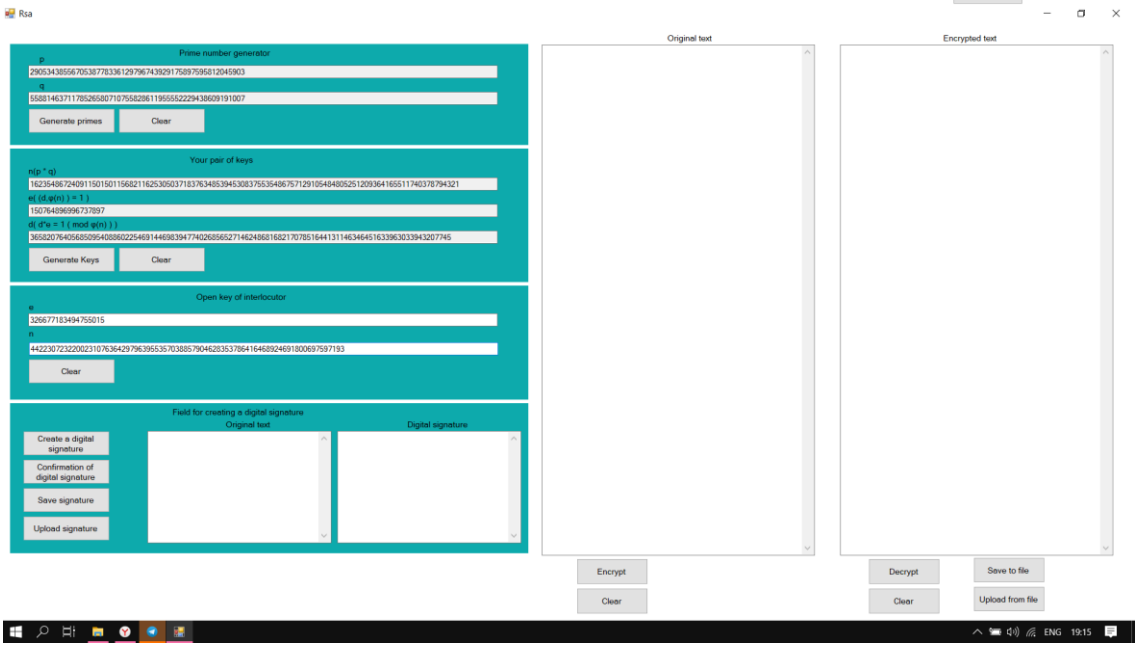


Затем обмениваются открытыми ключами и заполняют соответствующие поля

Собеседник А

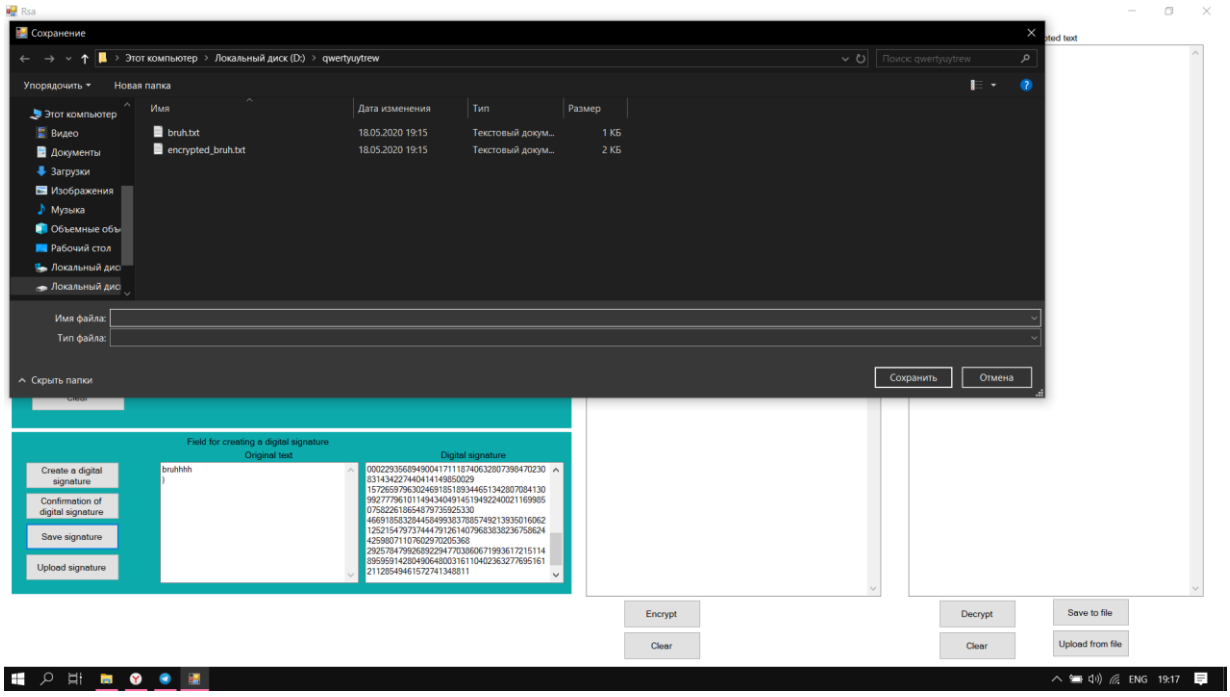


Собеседник Б



В целях безопасности Собеседник Б решил отправить Собеседнику А цифровую подпись. Порядок действий следующий:

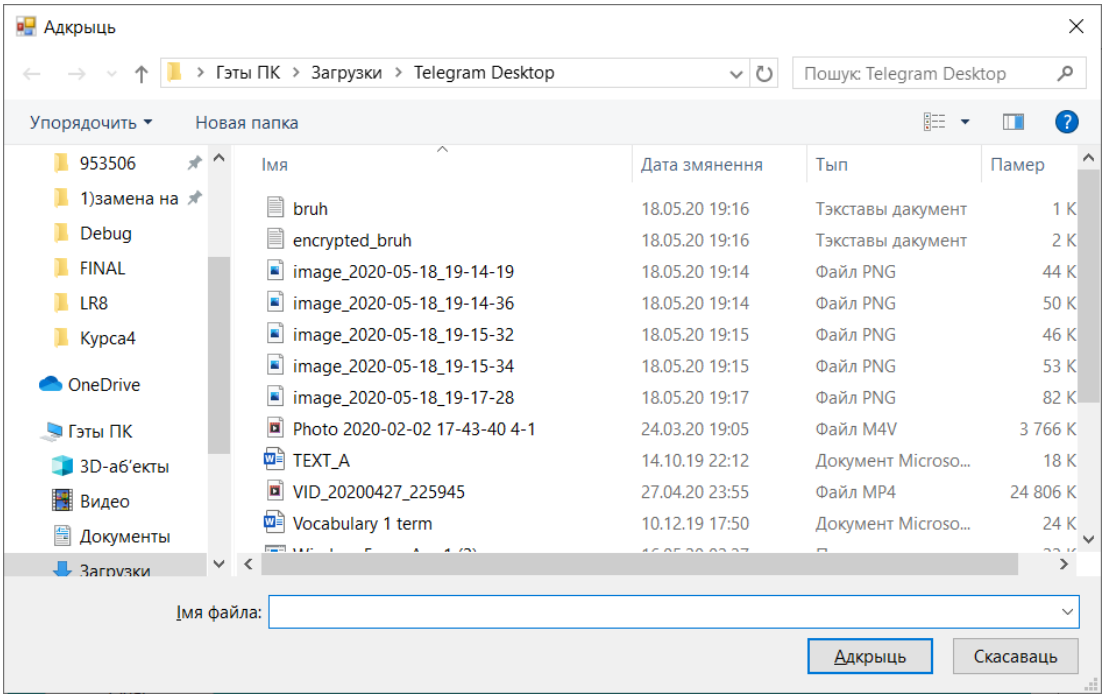
- Вводим текст в поле “Original text”
- Нажимаем кнопку “Create a digital signature”
- Затем текст и шифровка текста может быть отправлена путём копирования из полей ввода или, нажав на кнопку “Save signature”, мы можем сохранить и текст и шифровку в файлы.

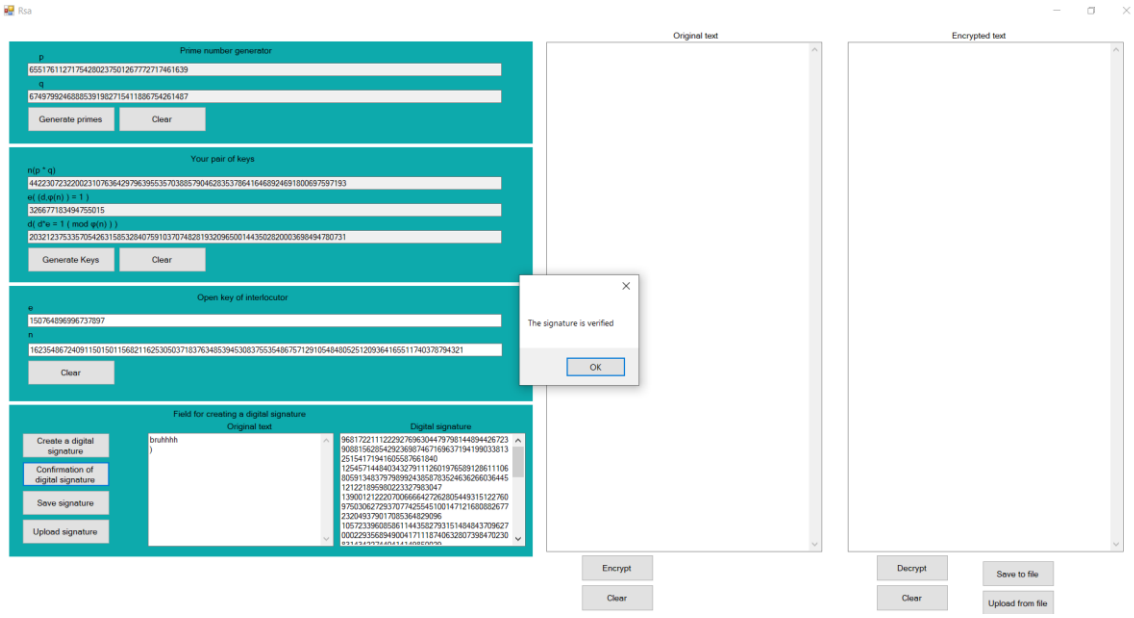


Чтобы проверить цифровую подпись, собеседник А должен выполнить следующий порядок:

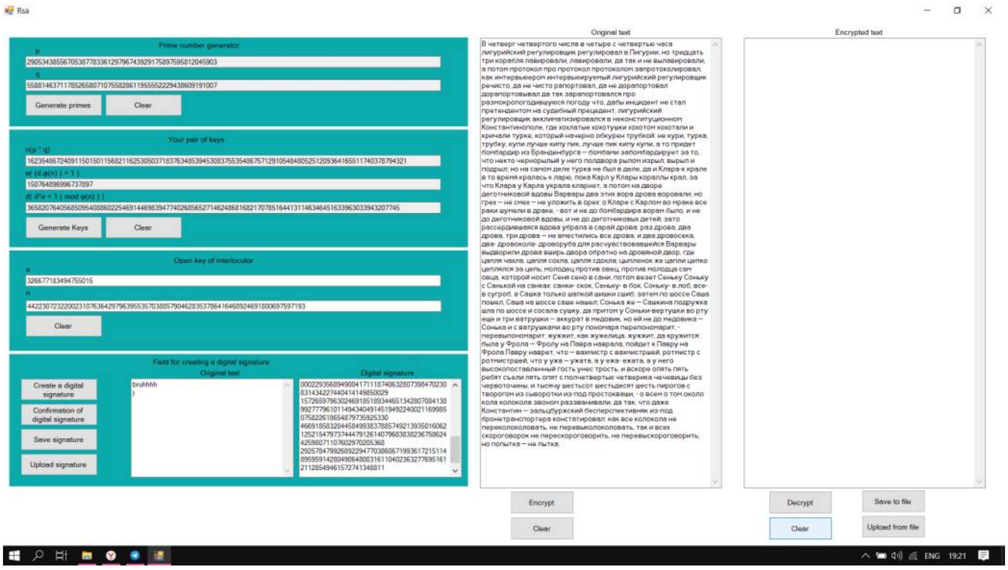
- Либо вставить скопированные тексты в соответствующие поля, либо нажать на кнопку “Upload signature” и выбрать соответствующие файлы.
- Нажать на кнопку “Confirmation of digital signature”

Результатом нажатия кнопки будет уведомление о соответствии или несоответствии подписи

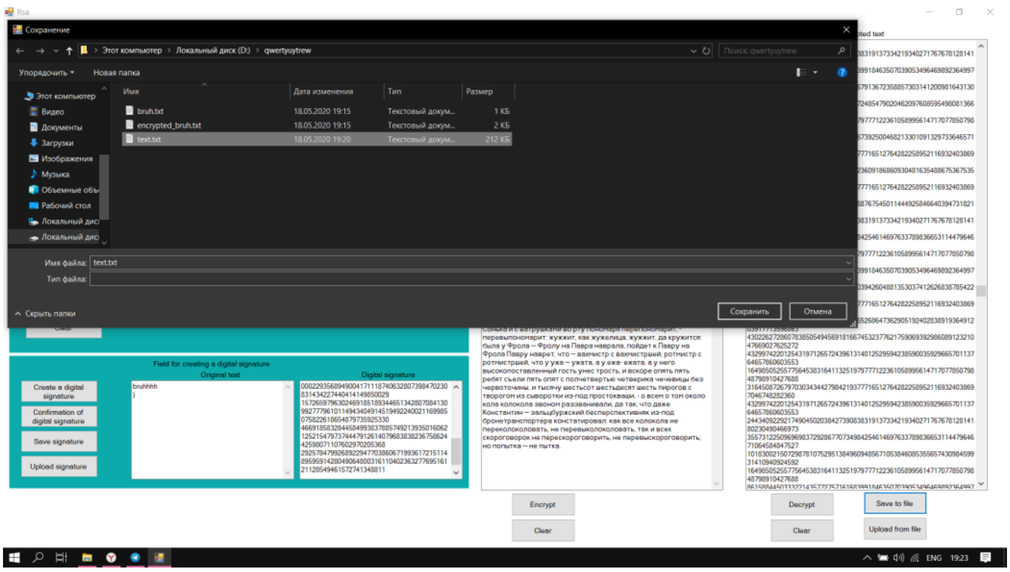




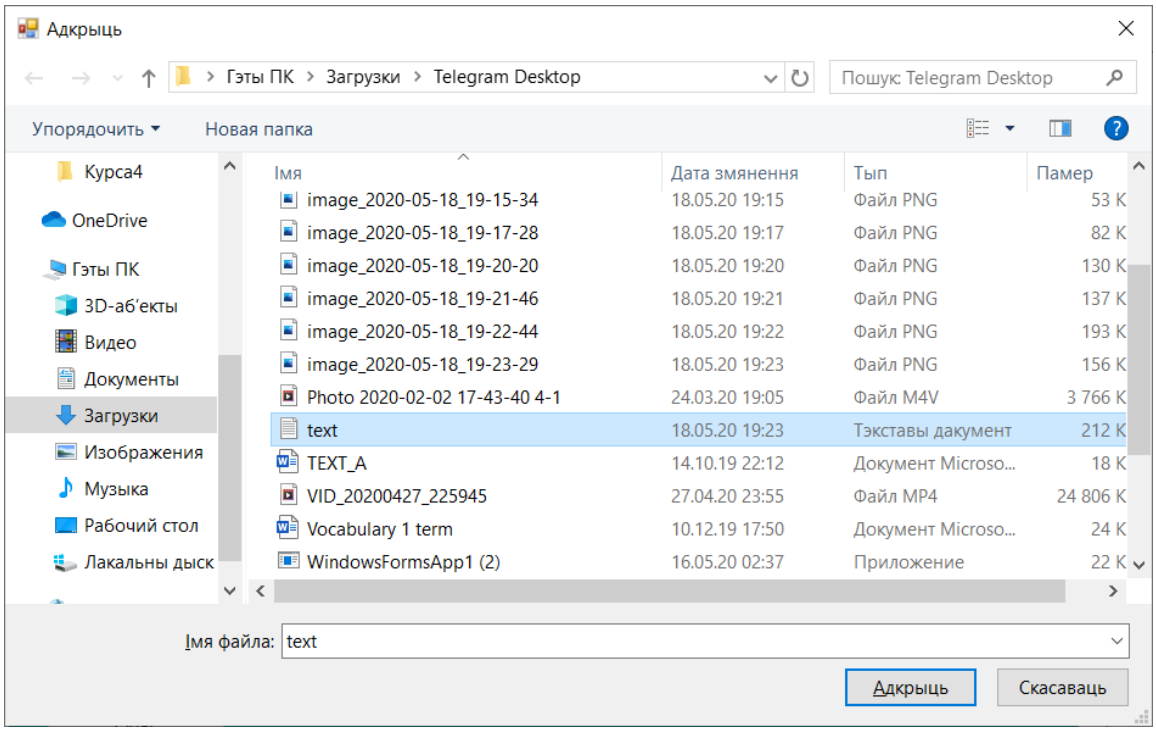
Собеседник Б хочет отправить Собеседнику А сообщение. Он вводит исходный текст в поле “Original text”, после чего нажимает на кнопку “Encrypt” и получает зашифрованный текст. Зашифрованные текст Собеседник Б может либо скопировать, либо сохранить в файл.

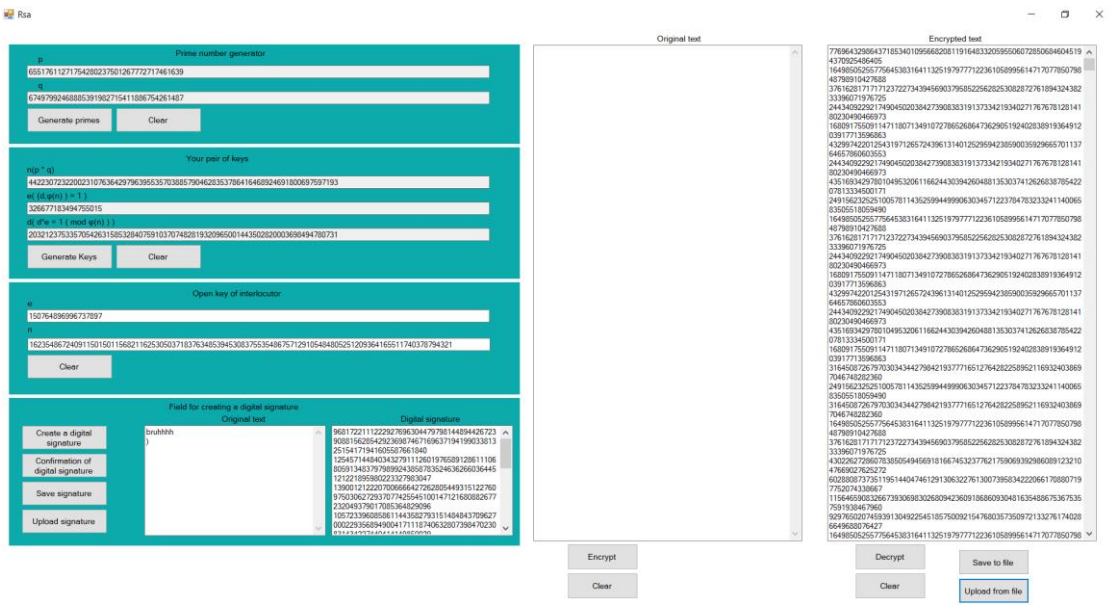

Rsa
Prime number generator
Generate primes
Clear
Your pair of keys
Generate Keys
Open key of interlocutor
Field for creating a digital signature
Original text
Digital signature
Create a digital signature
Confirmation of digital signature
Save signature
Upload signature
Original text
Encrypted text
Encrypt
Decrypt
Save to file
Upload from file


Rsa
Prime number generator
Generate primes
Clear
Your pair of keys
Generate Keys
Open key of interlocutor
Field for creating a digital signature
Original text
Digital signature
Create a digital signature
Confirmation of digital signature
Save signature
Upload signature
Original text
Encrypted text
Encrypt
Decrypt
Save to file
Upload from file


Сохранение
Этот компьютер › Локальный диск (D:) › qwertyuytrew
Упорядочить
Новая папка
Имя
Дата изменения
Тип
Размер
bruh.txt
encrypted_bruh.txt
text.txt
Имя файла: text.txt
Тип файла:
Скрыть папки
Сохранить
Отмена

Собеседник А получает зашифрованный текст или файл с зашифрованным текстом. Текст вставляется в поле “Encrypted text” или загружается из файла после нажатия кнопки “Upload from file”. После нажатия кнопки “Decrypt” Собеседник А получает расшифрованный текст





Rsa
Prime number generator
p
q
Generate primes
Clear
Your pair of keys
n(p * q)
e((d,φ(n)) = 1)
d(d*e = 1 (mod φ(n)))
Generate Keys
Clear
Open key of interlocutor
e
n
Clear
Field for creating a digital signature
Original text
Digital signature
Create a digital signature
Confirmation of digital signature
Save signature
Upload signature
Original text
Encrypt
Clear
Encrypted text
Decrypt
Clear
Save to file
Upload from file

Заключение

Метод шифрования RSA на сегодняшний день довольно популярен и много где используется. Но он все же имеет свой недостаток.

Метод имеет относительно простую математическую модель, но практическая модель показывает довольно низкую скорость шифрования

Из-за низкой скорости шифрования, сообщения обычно шифруют с помощью более производительных симметричных алгоритмов со случайным сеансовым ключом (например, AES, IDEA, Serpent, Twofish), а с помощью RSA шифруют лишь этот ключ, таким образом реализуется гибридная криптосистема. Такой механизм имеет потенциальные уязвимости ввиду необходимости использовать криптографически стойкий генератор псевдослучайных чисел для формирования случайного сеансового ключа симметричного шифрования.

Список использованных источников

1. Сайт https://ru.wikipedia.org – теория
2. Сайт https://habr.com/ - информация о генерация простых чисел

Исходный код приложения

```
using System;
using System.Collections.Generic;
using System.Diagnostics;
using System.IO;
using System.Linq;
using System.Numerics;
using System.Text.RegularExpressions;
using System.Runtime.InteropServices.WindowsRuntime;
using System.Runtime.Remoting;
using System.Text;
using System.Windows.Forms;
using System.Runtime.InteropServices;

namespace WindowsFormsApp1
{
   public partial class Form1 : Form
   {
      //-------------------------------------------------------------------------------Часть кода отвечающая за генерацию простых чисел-------------------------------------------------
      private bool Check(int a)//Функция проверки числа на простотоу
      {
         for(int i=2; i<a; i++)
         {
            if (a % i == 0)
               return false;
         }
         return true;
      }

      private static int LenghtOfBin(BigInteger a)//Функция определяющая кол-во бит в числе
      {
         int lenght = 0;
         while (a != 0)
         {
            a = BigInteger.Divide(a, 2);
            lenght++;
         }
         return lenght;
      }

      private void generatePrimes(List<int> primes, out BigInteger firstPrime, out BigInteger secondPrime)//Функция генерирующая 2 простых числа
      {
         firstPrime = 0;
         secondPrime = 0;

         List<BigInteger> mod3_1 = new List<BigInteger>();

         List<BigInteger> l = new List<BigInteger>();
```

```
BigInteger three = new BigInteger(3), two = new BigInteger(2);

for (int k = 0; k < primes.Count() - 1; k++)
{

   BigInteger seed = new BigInteger(primes[k]);

   BigInteger s2 = BigInteger.Multiply(seed, 2);

   BigInteger r0;
   BigInteger.DivRem(seed, three, out r0);
   // Проверка на остаток = 1
   if (r0 == BigInteger.One) mod3_1.Add(seed);

   for (int i = k + 1; i < primes.Count(); i++)
   {
      BigInteger p = new BigInteger(primes[i]);
      BigInteger r;
      BigInteger.DivRem(p, three, out r);

      if (r == r0) continue;
      else addIfPrime(p, seed, s2, two, l);
   }
}

List<BigInteger> ps = l;

do
{
   l = new List<BigInteger>();
   int size = ps.Count();
   for (int k = 0; k < size; k++)
   {
      BigInteger seed = ps[k];
      BigInteger s2 = BigInteger.Multiply(seed, 2);

      for (int i = 0; i < mod3_1.Count(); i++)
         addIfPrime(mod3_1[i], seed, s2, two, l);

      int n = 100000;
      if (l.Count() > 0)
         if (LenghtOfBin(l[0]) < 700) n = 10;
         else if (LenghtOfBin(l[0]) < 800) n = 20;
         else if (LenghtOfBin(l[0]) < 900) n = 40;
      if (l.Count() > n) break;
   }

   ps = l;

   Random rnd = new Random();
   if (rnd.Next(1, 12) == 1)
   {
      while (true)
      {
         firstPrime = l[rnd.Next(0, l.Count() - 1)];
         secondPrime = l[rnd.Next(0, l.Count() - 1)];
```

```
                if (firstPrime.ToString().Length == secondPrime.ToString().Length)
                {
                    if(firstPrime!=secondPrime)
                        return;
                }
                else
                    continue;
            }
        }
    }
    while (l.Count() > 0);
}

    private static void addIfPrime(BigInteger a, BigInteger b, BigInteger b2, BigInteger two,
List<BigInteger> l)
    {

        BigInteger a2 = BigInteger.Multiply(a, 2), fp = BigInteger.Multiply(b, a2), n = BigInteger.Add(fp,
1);
        BigInteger r = new BigInteger();
        if (BigInteger.Compare(a2, b) < 0)
            r = BigInteger.ModPow(two, a2, n);
        else if (BigInteger.Compare(a, b2) < 0)
            r = BigInteger.ModPow(two, a, n);

        if (r != null && BigInteger.Compare(r, 1) == 0) return;

        r = new BigInteger();

        if (BigInteger.Compare(b2, a) < 0)
            r = BigInteger.ModPow(two, b2, n);
        else if (BigInteger.Compare(b, a2) < 0)
            r = BigInteger.ModPow(two, b, n);

        if (r != null && BigInteger.Compare(r, 1) == 0) return;

        r = BigInteger.ModPow(two, fp / 2, n);

        if (BigInteger.Compare(r, 1) != 0) return;
        l.Add(n);
    }
    //-----------------------------------------------------------------------------Конец генерции чисел-------------
-------------------------------------------------------------------------
    public Form1()
    {
        InitializeComponent();
    }

    //-----------------------------------------------------------------------------Шифрование и дешифрование--
-----------------------------------------------------------------------------
    private List<string> RSA_Endoce(string s, BigInteger e, BigInteger n)//Функция выполняющая
шифрование
    {
        List<string> result = new List<string>();

        BigInteger bi;
```

```
            int size = s.Length;
            for(int i=0; i<size; i++)
            {
                bi = s[i];
                result.Add(BigInteger.ModPow(bi, e, n).ToString());
            }

            return result;
        }

        private string RSA_Dedoce(string[] input, BigInteger d, BigInteger n)//Функция выполняющая дешифрование
        {
            string result = "";

            BigInteger bi;

            foreach (string item in input)
            {
                bi = BigInteger.Parse(item);

                bi = BigInteger.ModPow(bi, d, n);
                try
                {
                    result += (char)bi;
                }
                catch(Exception)
                {
                    MessageBox.Show("Problem of decryption. Check your keys");
                    break;
                }
            }

            return result;
        }
        //----------------------------------------------------------------------------Конец шифрования и дешифрования-------------------------------------------------------------------------------

        private BigInteger EuclidAlgorithm(BigInteger a, BigInteger b)//Алгоритм Евклида для нахождения НОД
        {
            BigInteger reminder;
            BigInteger.DivRem(a, b, out reminder);
            while(reminder!=BigInteger.Zero)
            {
                a = b;
                b = reminder;
                BigInteger.DivRem(a, b, out reminder);
            }
            return b;
        }

        //----------------------------------------------------------------------------Высчитывание публичного и приватного ключей-------------------------------------------------------------
        private BigInteger Calculate_e(BigInteger m)//Вычисление открытого ключа
        {
```

```
            Random rnd = new Random();
            while(true)
            {
                BigInteger d1 = rnd.Next(1000000000);
                BigInteger d2 = rnd.Next(1000000000);

                BigInteger d =BigInteger.Multiply(d1, d2);
                if (d > m)
                    continue;
                else
                {
                    if (EuclidAlgorithm(m, d) == 1)
                        return d;
                }
            }
        }

        private BigInteger Calculate_d(BigInteger a, BigInteger b)//вычисление закрытого ключа
        {
            BigInteger tempA = a, tempB = b;
            BigInteger u1 = BigInteger.One, u2 = BigInteger.Zero, v1 = BigInteger.Zero, v2 =
BigInteger.One;
            BigInteger temp = new BigInteger(), remainder = new BigInteger(), tempU = new BigInteger(),
tempV = new BigInteger();
            while(b!=1)
            {
                BigInteger.DivRem(a, b, out remainder);
                temp = BigInteger.Divide(a, b);
                tempU = BigInteger.Subtract(u1, BigInteger.Multiply(u2, temp));
                tempV = BigInteger.Subtract(v1, BigInteger.Multiply(v2, temp));
                u1 = u2; u2 = tempU;
                v1 = v2; v2 = tempV;
                a = b;
                b = remainder;
            }
            if (v2 < 0)
                v2 = BigInteger.Add(v2, tempA);
            return v2;
        }
        //-----------------------------------------------------------------------------Конец высчитывания
публичного и приватного ключей-----------------------------------------------------------------------------
        private void EncryptionButton_Click(object sender, EventArgs e)//Кнопка шифрования
        {
            textBox_encrypted.Clear();
            if ((textBox_n_interlocutor.Text.Length > 0) && (textBox_e_interlocutor.Text.Length > 0))
            {
                if (textBox_original.Text.Length > 0)
                {
                    string s = textBox_original.Text;

                    BigInteger n = BigInteger.Parse(textBox_n_interlocutor.Text);
                    BigInteger e_ = BigInteger.Parse(textBox_e_interlocutor.Text);

                    List<string> result = RSA_Endoce(s, e_, n);

                    foreach (var text in result)
```

```
                {
                    textBox_encrypted.AppendText(text + "\n");
                }
            }
            else
                MessageBox.Show("The text field is empty");
        }
        else
            MessageBox.Show("The public key is missing!");
    }

    private void DecryptionButton_Click(object sender, EventArgs e)//Кнопка дешифрования
    {
        textBox_original.Clear();
        if ((textBox_d.Text.Length > 0) && (textBox_n.Text.Length > 0))
        {
            if (textBox_encrypted.Text.Length > 0)
            {
                BigInteger d = BigInteger.Parse(textBox_d.Text);
                BigInteger n = BigInteger.Parse(textBox_n.Text);

                string[] input = textBox_encrypted.Text.Split(new char[] { '\n', '\r' },
StringSplitOptions.RemoveEmptyEntries);

                string result = RSA_Dedoce(input, d, n);

                textBox_original.AppendText(result);
            }
            else
                MessageBox.Show("The text field is empty");
        }
        else
            MessageBox.Show("The private key is missing!");
    }

    private void PrimesGen_Click(object sender, EventArgs e)//Кнопка генирации простых чисел
    {
        BigInteger p = new BigInteger();
        BigInteger q = new BigInteger();
        List<int> temp = new List<int>();
        for (int i = 0; i < 10000; i++)
        {
            if (Check(i))
                temp.Add(i);
        }
        generatePrimes(temp, out p, out q);
        textBox_p.Text = p.ToString();
        textBox_q.Text = q.ToString();
    }

    private void KeysGen_Click(object sender, EventArgs e)//Кнопка вычисления ключей
    {
        if ((textBox_p.Text.Length > 0) && (textBox_q.Text.Length > 0))
        {
            BigInteger p = BigInteger.Parse(textBox_p.Text);
            BigInteger q = BigInteger.Parse(textBox_q.Text);
```

```
        BigInteger n = BigInteger.Multiply(p, q);
        BigInteger m = BigInteger.Multiply((p - 1), (q - 1));
        BigInteger e_ = Calculate_e(m);
        BigInteger d = Calculate_d(m, e_);

        textBox_d.Text = d.ToString();
        textBox_n.Text = n.ToString();
        textBox_e.Text = e_.ToString();
    }
    else
        MessageBox.Show("Generate Prime numbers!");
}
//Кнопки очистки полей
private void clear_original_textbox_Click(object sender, EventArgs e)
{
    textBox_original.Clear();
}

private void clear_encrypted_textbox_Click(object sender, EventArgs e)
{
    textBox_encrypted.Clear();
}

private void clear_primes_Click(object sender, EventArgs e)
{
    textBox_p.Clear();
    textBox_q.Clear();
}

private void clear_keys_Click(object sender, EventArgs e)
{
    textBox_n.Clear();
    textBox_e.Clear();
    textBox_d.Clear();
}

private void digital_signature_button_Click(object sender, EventArgs e)
{
    textBox_signature_encrypted.Clear();
    if ((textBox_n.Text.Length > 0) && (textBox_d.Text.Length > 0))
    {
        if (textBox_signature_origin.Text.Length > 0)
        {
            string s = textBox_signature_origin.Text;

            BigInteger n = BigInteger.Parse(textBox_n.Text);
            BigInteger d = BigInteger.Parse(textBox_d.Text);

            List<string> result = RSA_Endoce(s, d, n);

            foreach (var text in result)
            {
                textBox_signature_encrypted.AppendText(text + "\n");
            }
        }
```

```
            else
                MessageBox.Show("The text field is empty");
            }
            else
                MessageBox.Show("The private key is missing!");
        }

        private void confirming_signature_button_Click(object sender, EventArgs e)
        {
            if ((textBox_e_interlocutor.Text.Length > 0) && (textBox_n_interlocutor.Text.Length > 0))
            {
                if (textBox_signature_origin.Text.Length > 0 && textBox_signature_encrypted.Text.Length >
0)
                {
                    BigInteger e_ = BigInteger.Parse(textBox_e_interlocutor.Text);
                    BigInteger n = BigInteger.Parse(textBox_n_interlocutor.Text);

                    string[] input = textBox_signature_encrypted.Text.Split(new char[] { '\n', '\r' },
StringSplitOptions.RemoveEmptyEntries);

                    string result = RSA_Dedoce(input, e_, n);

                    if(result == textBox_signature_origin.Text)
                        MessageBox.Show("The signature is verified");
                    else
                        MessageBox.Show("The signature is not verified");
                }
                else
                    MessageBox.Show("The text field is empty");
            }
            else
                MessageBox.Show("The public key is missing!");
        }
    }
}
```