Министерство науки и высшего образования Российской Федерации
Федеральное государственное автономное образовательное учреждение высшего образования
«Пермский национальный исследовательский политехнический университет»
Электротехнический факультет
Кафедра «Информационные технологии и автоматизированные системы»
Направление 09.03.01 – «Информатика и вычислительная техника»

Дисциплина: «Защита информации»

Профиль: «Автоматизированные системы обработки информации и управления»

Семестр 6

# ОТЧЕТ
## по лабораторной работе №2
### Тема: «Алгоритм RSA»

Выполнил: студент группы АСУ-19-1б
Шеретов М.А. ___________
Проверил: старший преподаватель
Шереметьев В. Г. ___________
Дата ______

Пермь, 2021

## ЦЕЛЬ РАБОТЫ

Получить практические навыки по использованию ассиметричных алгоритмов шифрования, на примере использования алгоритма RSA.

## ЗАДАНИЕ

Выполнить шифрование текстового файла, методом RSA, используя в качестве p и q простые числа с разрядностью не меньшей двадцати.

## ТЕОРЕТИЧЕСКИЕ СВЕДЕНИЯ

Алгоритм RSA (R.Rivest, A.Shamir, L.Adleman) был предложен еще в 1977 году. С тех пор он весьма упорно противостоит различным атакам, и сейчас является самым распространенным криптоалгоритмом в мире. Он входит во многие криптографические стандарты, используется во многих приложениях и секретных протоколах (включая PEM, S-HTTP и SSL).

**Основные принципы работы RSA**

Сначала пара математических определений. Целое число называют простым, если оно делится нацело только на единицу и на само себя, иначе его называют составным. Два целых числа называют взаимно простым, если их наибольший общий делитель (НОД) равен 1.

Алгоритм работы RSA таков. Сначала надо получить открытый и секретный ключи:

1. Выбираются два простых числа p и q
2. Вычисляется их произведение n(=p*q)
3. Выбирается произвольное число e (e<n), такое, что НОД(e,(p-1)(q-1))=1, то есть e должно быть взаимно простым с числом (p-1)(q-1).
4. Методом Евклида решается в целых числах уравнение e*d+(p-1)(q-1)*y=1. Здесь неизвестными являются переменные d и y – метод Евклида как раз и находит множество пар (d,y), каждая из которых является решением уравнения в целых числах.
5. Два числа (e,n) – публикуются как открытый ключ.
6. Число d хранится в строжайшем секрете – это и есть закрытый ключ, который позволит читать все послания, зашифрованные с помощью пары чисел (e,n).

Как же производится собственно шифрование с помощью этих чисел:

1. Отправитель разбивает свое сообщение на блоки, равные $k=[\log_2(n)]$ бит, где квадратные скобки обозначают взятие целой части от дробного числа.
2. Подобный блок, как Вы знаете, может быть интерпретирован как число из диапазона $(0;2^k-1)$. Для каждого такого числа (назовем его $m_i$)

вычисляется выражение $c_i=((m_i)^e)\mathrm{mod}\ n$. Блоки $c_i$ и есть зашифрованное сообщение Их можно спокойно передавать по открытому каналу, поскольку операция возведения в степень по модулю простого числа, является необратимой математической задачей. Обратная ей задача носит название «логарифмирование в конечном поле» и является на несколько порядков более сложной задачей. То есть даже если злоумышленник знает числа e и n, то по $c_i$ прочесть исходные сообщения $m_i$ он не может никак, кроме как полным перебором $m_i$.

А вот на приемной стороне процесс дешифрования все же возможен, и поможет нам в этом хранимое в секрете число d. Достаточно давно была доказана теорема Эйлера, частный случай которой утверждает, что если число n представимо в виде двух простых чисел p и q, то для любого x имеет место равенство $(x^{(p-1)(q-1)})\mathrm{mod}\ n = 1$. Для дешифрования RSA-сообщений воспользуемся этой формулой. Возведем обе ее части в степень (-y) : $(x^{(-y)(p-1)(q-1)})\mathrm{mod}\ n = 1^{(-y)} = 1$. Теперь умножим обе ее части на x : $(x^{(-y)(p-1)(q-1)+1})\mathrm{mod}\ n = 1*x = x$.

А теперь вспомним как мы создавали открытый и закрытый ключи. Мы подбирали с помощью алгоритма Евклида d такое, что e*d+(p-1)(q-1)*y=1, то есть e*d=(-y)(p-1)(q-1)+1. А следовательно в последнем выражении предыдущего абзаца мы можем заменить показатель степени на число (e*d). Получаем $(x^{e*d})\mathrm{mod}\ n = x$. То есть для того чтобы прочесть сообщение $c_i=((m_i)^e)\mathrm{mod}\ n$ достаточно возвести его в степень d по модулю m : $((c_i)^d)\mathrm{mod}\ n = ((m_i)^{e*d})\mathrm{mod}\ n = m_i$.

## ХОД РАБОТЫ

Значения p и q (простые числа) определяются напрямую в коде программы и равны 1000000000007031337 и 1000000000007031389 соответственно. Также в коде определены файлы для ввода данных (input.txt) и вывода зашифрованных данных (output.txt). Расшифрованное сообщение записывается в файл ввода.

В приложении всего доступно 3 кнопки:

1. Encrypt шифрует сообщение из файла ввода и записывает в файл вывода.
2. Decrypt расшифровывает сообщение из файла вывода и записывает в файл ввода
3. Internals показывает все значения, с помощью которых происходит шифрование (e, d, p, q…)

Для вычислений используется тип данных BigInt, так как необходимо работать с очень большими целыми числами.


Рисунок 1. Окно программы

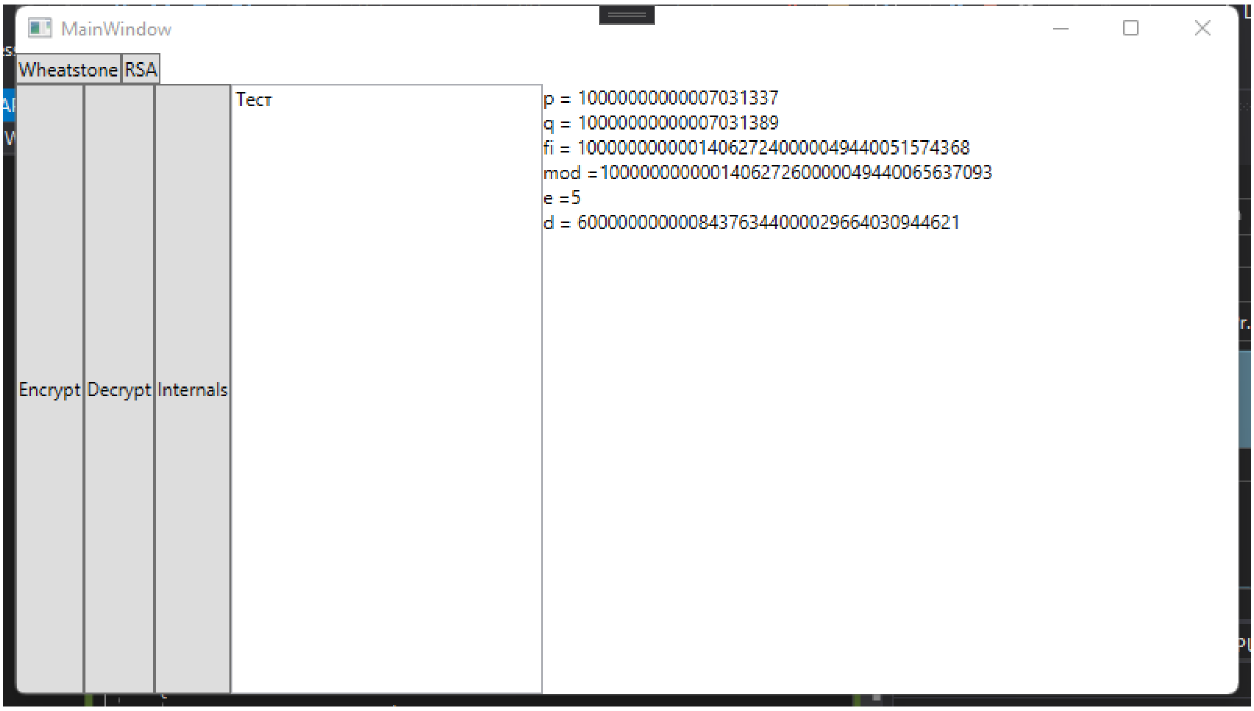


Рисунок 2. Вычисленные значения метода RSA

Ниже на рисунке 3представлены результаты работы программы.

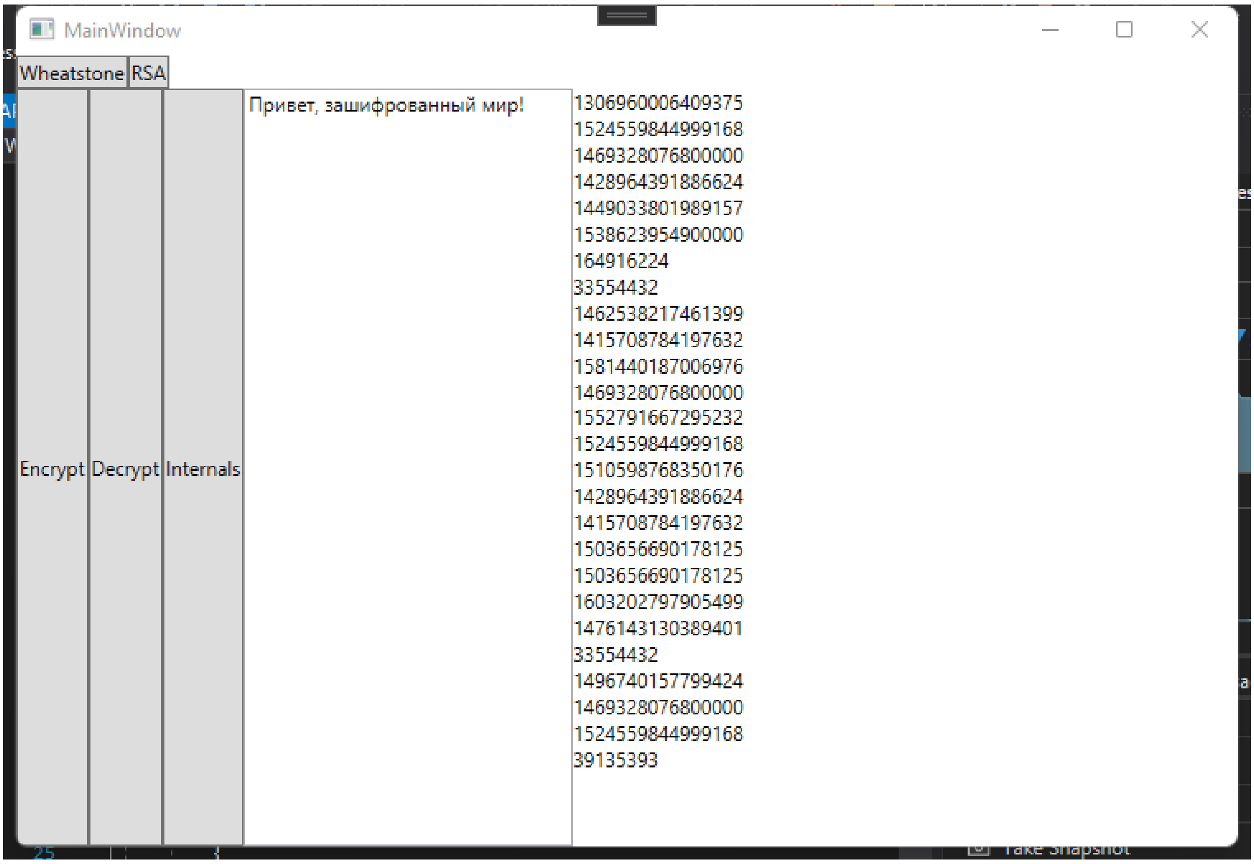


Рисунок 3. Содержимое ввода и соотвествующего вывода

Для шифрования данным методом используется класс RSA, который в конструкторе вычисляет значения функции Эйлера (fi), модуля (mod), открытой экспоненты (e), число d. Конструктор представлен на рисунке .

```
public RSA(BigInteger p, BigInteger q)
{
    P = p;
    Q = q;
    mod = p * q;
    fi = (p - 1) * (q - 1);
    e = FindE(fi);
    e.TryModInverse(fi, out d);
}
```

Рисунок 5. Конструктор класса RSA

Для шифрования используется метод encrypt. Сообщение разбивается на символы, каждый отдельный символ преобразуется в его код, шифруется, а затем записывается в массив.

Для расшифрования используется метод decrypt. Каждое число из массива дешифруется в код символа, а затем преобрзуется в символ и добавляется к строке (рисунок 6)

```
1 reference
public List<BigInteger> Encrypt(string message)
{
    var codes = message.ToCharArray();
    var encryptedMsg = new List<BigInteger>();
    for (int i = 0; i < codes.Length; i++)
    {
        var code = codes[i];
        BigInteger crypdetCode = BigInteger.ModPow((BigInteger)code, e, mod);
        encryptedMsg.Add(crypdetCode);
    }

    return encryptedMsg;
}

1 reference
public string Decrypt(List<BigInteger> message)
{
    var decrypted = "";
    for (int i = 0; i < message.Count; i++)
    {
        var code = message[i];
        BigInteger decryptedCode = BigInteger.ModPow(code, d, mod);
        decrypted += (char)decryptedCode;
    }

    return decrypted;
}
```

Рисунок 6. Методы шифрования и расшифрования

# ПРИЛОЖЕНИЕ А

## Листинг А1 – Класс RSA

```
using System;
using System.Collections.Generic;
using System.Text;
using System.Numerics;

namespace WpfApp1.Pages
{
    public class RSA
    {
        public BigInteger mod;
        public BigInteger fi;
        public BigInteger e;
        public BigInteger d;

        public RSA(BigInteger p, BigInteger q)
        {
            P = p;
            Q = q;
            mod = p * q;
            fi = (p - 1) * (q - 1);
            e = FindE(fi);
            e.TryModInverse(fi, out d);
        }



        public List<BigInteger> Encrypt(string message)
        {
            var codes = message.ToCharArray();
            var encryptedMsg = new List<BigInteger>();
            for (int i = 0; i < codes.Length; i++)
            {
                var code = codes[i];
                BigInteger crypdetCode = BigInteger.ModPow((BigInteger)code, e, mod);
                encryptedMsg.Add(crypdetCode);
            }

            return encryptedMsg;
        }

        public string Decrypt(List<BigInteger> message)
        {
            var decrypted = "";
            for (int i = 0; i < message.Count; i++)
            {
                var code = message[i];
                BigInteger decryptedCode = BigInteger.ModPow(code, d, mod);
                decrypted += (char)decryptedCode;
            }

            return decrypted;
        }

        private BigInteger FindE(BigInteger fi)
        {
            do
            {
                e = GetNextPrime(e);
                if (fi % e != 0)
```

```
                return e;
            } while (e < fi);
            return 0;
        }

        BigInteger GetNextPrime(BigInteger n)
        {
            bool isPrimeNumber = false;

            do
            {
                n++;
                isPrimeNumber = IsPrime(n);
            } while (!isPrimeNumber);
            return n;
        }

        public static bool IsPrime(BigInteger n)
        {
            if (n <= 1) return false;
            if (n <= 3) return true;
            if (n % 2 == 0 || n % 3 == 0)
                return false;

            for (BigInteger i = 5; i*i < n; i += 6)
            {
                if (n % i == 0 || n % (i + 2) == 0)
                    return false;
            }
            return true;
        }

        public BigInteger P { get; private set; }
        public BigInteger Q { get; private set; }
    }

    internal static class BigIntegerExt
    {
        public static BigInteger ModInverse(this BigInteger step, BigInteger m)
        {
            return (1 / step) % m;
        }

        public static bool TryModInverse(this BigInteger number, BigInteger modulo, out
BigInteger result)
        {
            if (number < 1) throw new ArgumentOutOfRangeException(nameof(number));
            if (modulo < 2) throw new ArgumentOutOfRangeException(nameof(modulo));
            BigInteger n = number;
            BigInteger m = modulo, v = 0, d = 1;
            while (n > 0)
            {
                BigInteger t = m / n, x = n;
                n = m % x;
                m = x;
                x = d;
                d = checked(v - t * x); // Just in case
                v = x;
            }
            result = v % modulo;
            if (result < 0) result += modulo;
            if ((long)number * result % modulo == 1L) return true;
            result = default;
            return false;
        }
```

```
    }
}
```