Федеральное государственное автономное
образовательное учреждение
высшего образования
«СИБИРСКИЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ»

Институт Космических и информационных технологий
институт
Кафедра «Информатика»
кафедра

ОТЧЕТ ПО ПРАКТИЧЕСКОЙ РАБОТЕ №2

Основы криптографии с открытым ключом. Алгоритм RSA
тема

Преподаватель А.А. Сидарас
подпись, дата инициалы, фамилия

Студент КИ18-17/1б 031831229 В.А. Прекель
номер группы, зачетной книжки подпись, дата инициалы, фамилия

Красноярск 2020

1 Цель работы

Ознакомиться с основами асимметричного шифрования, ознакомиться с элементами теории чисел, используемых в криптографии с открытым ключом, изучить особенности алгоритма с открытым ключом RSA, получить навыки разработки криптосистем с открытым ключом с использованием языка программирования высокого уровня.

2 Постановка задач

Согласно вашему персональному варианту или индивидуальному заданию преподавателя разработайте и составьте в виде блок-схемы, псевдокода, пошагово на естественном языке алгоритмы шифрования и дешифрования текста. Убедитесь в правильности составления алгоритмов и затем на языке программирования составьте программу, которая реализует данные алгоритмы.

На ряде контрольных примеров (не менее 10) открытого текста, состоящего из различного количества символов, проверьте правильность работы алгоритмов шифрования и дешифрования.

Разработанная Вами программа должна содержать графический интерфейс пользователя.

3 Краткий теоретический материал

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.

4 Ход работы

Для начала разберёмся с шагами и этапами данного алгоритма.

1. Выбираются два больших и простых числа P и Q. В первом варианте задания нужно использовать 31-разрядное число N, поэтому P и Q могут быть 15-разрядными и 16-разрядными соответственно. Поэтому надо генерировать

случаное число P в полуинтервале [10^14; 10^15) и Q в [10^15; 10^16) и проверять их на простоту используя тест Миллера-Рабина. Используется тип BigInteger из .NET.

2. Вычисляется произведение N = P * Q.

3. Вычисляется значение функци Эйляра d = φ(N) = (P - 1)(Q - 1).

4. Выбирается s < d, взаимно простое с d. Генерируется случайное s, пока НОД(s, d) не будет равен 1.

5. Вычисляется e, такое что (e*s)%d=1. Для этого используется расширенный алгоритм евклида. Он вычисляет gcd, e, k в уравнении s * e + d * k = gcd(s, d). Нам нужно значение e.

6. После этого пара (s, N) считается открытым ключом, e - закрытым.

7. Для шифрования строки нужно её разбить на блоки (каждый блок должен быть меньше N). Можно считать каждый символ как блок. Тогда зашифрованная строка будет представленна как массив больших чисел.

8. Для шифрования используется формула c = m^s%N. Для дешифрования m=c^e%N. Эти операции выполняются используя функцию BigInteger.ModPow.

Алгоритм реализован на языке C# 8 используя фреймворк .NET Core 3.1, библиотеку для графического интерфейса AvaloniaUI и фреймфорк юнит-тестирования NUnit.

Листинг 1 – Lab_02/Lab_02.Core/AbstractRsa.cs

```
using System.Numerics;

namespace Lab_02.Core
{
    public abstract class AbstractRsa
    {
        public RSAOpenKey OpenKey { get; } = new RSAOpenKey();

        public class RSAOpenKey
        {
            public BigInteger S { get; internal set; }
            public BigInteger N { get; internal set; }
        }
    }
}
```

Листинг 2 – Lab_02/Lab_02.Core/RsaCrypter.cs

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

namespace Lab_02.Core
{
    public class RsaCrypter : AbstractRsa
    {
        public RsaCrypter(RSAOpenKey openkey)
        {
            OpenKey.N = openkey.N;
            OpenKey.S = openkey.S;
        }

        public BigInteger Crypt(BigInteger m) => BigInteger.ModPow(m, OpenKey.S, OpenKey.N);
```

```
        public IEnumerable<BigInteger> Crypt(string a) => a.Select(i => Crypt(i));
    }
}
```

Листинг 3 – Lab_02/Lab_02.Core/RsaDecrypter.cs

```
using System.Collections.Generic;
using System.Linq;
using System.Numerics;
using System.Security.Cryptography;

namespace Lab_02.Core
{
    public class RsaDecrypter : AbstractRsa
    {
        public RsaDecrypter(int nLength)
        {
            while (E <= 0)
            {
                var p = RandomPrimeBigInt(nLength / 2);
                var q = RandomPrimeBigInt(nLength - nLength / 2);
                OpenKey.N = p * q;
                var d = (p - 1) * (q - 1);
                OpenKey.S = CoprimeLessBigInt(d);
                var (_, e, _) = ExtendedGcd(OpenKey.S, d);
                E = e;
            }
        }

        private BigInteger E { get; }

        private RandomNumberGenerator Random { get; } = RandomNumberGenerator.Create();

        private bool IsProbablePrime(BigInteger source, int certainty)
        {
            if (source == 2 || source == 3)
            {
                return true;
            }

            if (source < 2 || source % 2 == 0)
            {
                return false;
            }

            var d = source - 1;
            var s = 0;

            while (d % 2 == 0)
            {
                d /= 2;
                s += 1;
            }

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

            for (var i = 0; i < certainty; i++)
            {
                do
                {
                    Random.GetBytes(bytes);
                    a = new BigInteger(bytes);
                } while (a < 2 || a >= source - 2);

                var x = BigInteger.ModPow(a, d, source);
                if (x == 1 || x == source - 1)
                {
                    continue;
                }

                for (var 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;
        }

        private BigInteger RandomBigInt(int length) =>
            RandomBigInt(BigInteger.Pow(new BigInteger(10), length - 1),
                BigInteger.Pow(new BigInteger(10), length));

        private BigInteger RandomBigInt(BigInteger min, BigInteger max)
        {
            var bytes = max.ToByteArray();
            while (true)
            {
                Random.GetBytes(bytes);
                bytes[^1] &= 0x7F;
```

```
            var randomBigInt = new BigInteger(bytes);

            if (min <= randomBigInt && randomBigInt < max)
            {
                return randomBigInt;
            }
        }
    }

    private BigInteger RandomPrimeBigInt(int length)
    {
        while (true)
        {
            var r = RandomBigInt(length);
            if (IsProbablePrime(r, length * 3))
            {
                return r;
            }
        }
    }

    private BigInteger CoprimeLessBigInt(BigInteger d)
    {
        while (true)
        {
            var r = RandomBigInt(0, d);
            if (BigInteger.GreatestCommonDivisor(r, d) == 1)
            {
                return r;
            }
        }
    }

    private (BigInteger, BigInteger, BigInteger) ExtendedGcd(BigInteger a, BigInteger b)
    {
        BigInteger x, y;
        if (a == 0)
        {
            x = 0;
            y = 1;
            return (b, x, y);
        }

        var (d, x1, y1) = ExtendedGcd(b % a, a);
        x = y1 - b / a * x1;
        y = x1;
        return (d, x, y);
    }

    public BigInteger Decrypt(BigInteger s) => BigInteger.ModPow(s, E, OpenKey.N);

    public string Decrypt(IEnumerable<BigInteger> a) => new string(a.Select(i => (char) Decrypt(i)).ToArray());
}
}
```

Листинг 4 – Lab_02/Lab_02.Core.Tests/RsaTests.cs

```
using System;
using System.Numerics;

using NUnit.Framework;

namespace Lab_02.Core.Tests
{
    [TestFixture]
    public class RsaTests
    {
        [Test]
        public void RsaRandomTest1()
        {
            var rs = new RsaDecrypter(31);
            var rc = new RsaCrypter(rs.OpenKey);

            var r = new Random();
            for (var i = 0; i < 100; i++)
            {
                var m = new BigInteger(r.Next(1, 10000));
                Assert.AreEqual(m, rs.Decrypt(rc.Crypt(m)));
            }
        }

        [Test]
        public void RsaStringTest1()
        {
            var rs = new RsaDecrypter(31);
            var rc = new RsaCrypter(rs.OpenKey);

            const string m = "qwerty";
            var c = rc.Crypt(m);
            var md = rs.Decrypt(c);

            Assert.AreEqual(m, md);
        }
    }
}
```

На самом деле, изложенный способ шифрования очень слаб. Причина проста — шифрование по буквам. Одна и та же буква будет шифроваться одним

и тем же числом. Если злоумышленник перехватит достаточно большое сообщение, он сможет догадаться о его содержимом. Сперва он обратит внимание на частые коды пробелов и разделит шифровку на слова. Потом он заметит однобуквенные слова и догадается, как кодируются буквы. Путём недолгого перебора, он вычислит дополнительные буквы, по коротким словам. И по более длинным словам без труда восстановит все оставшиеся буквы.

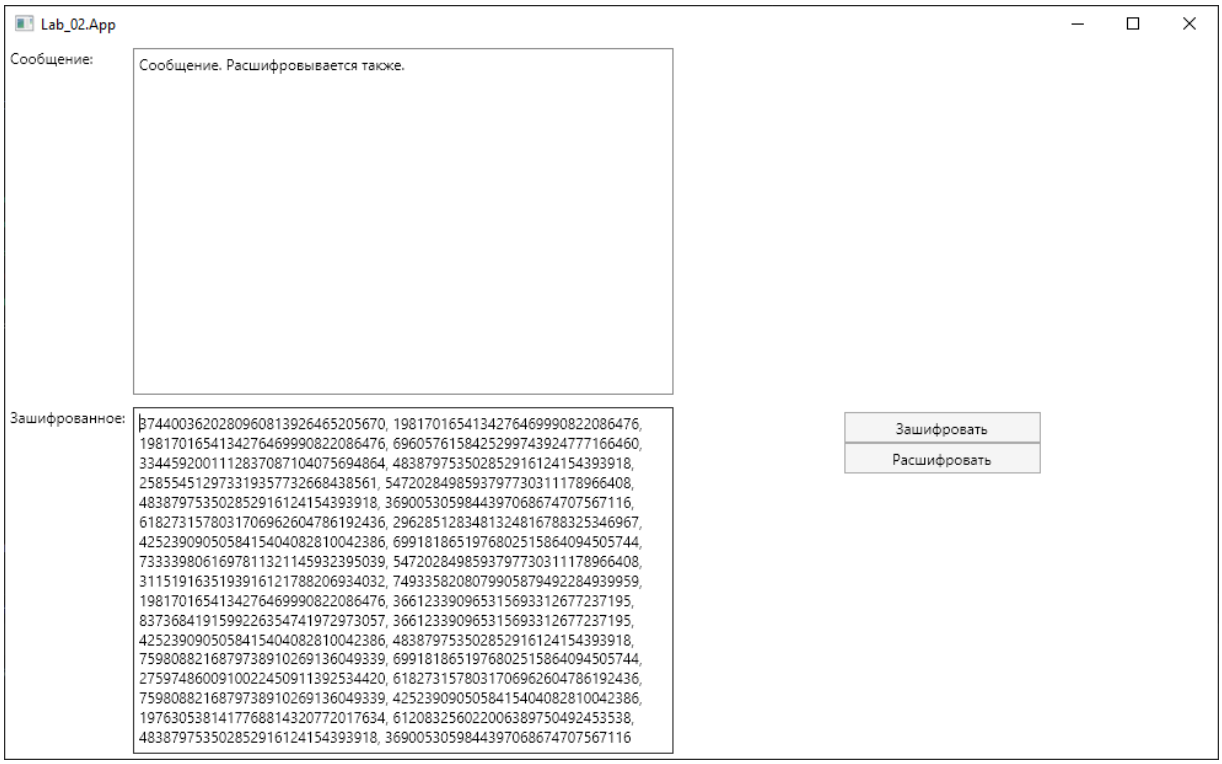

Рисунок 1 – Запуск графического интерфейса.

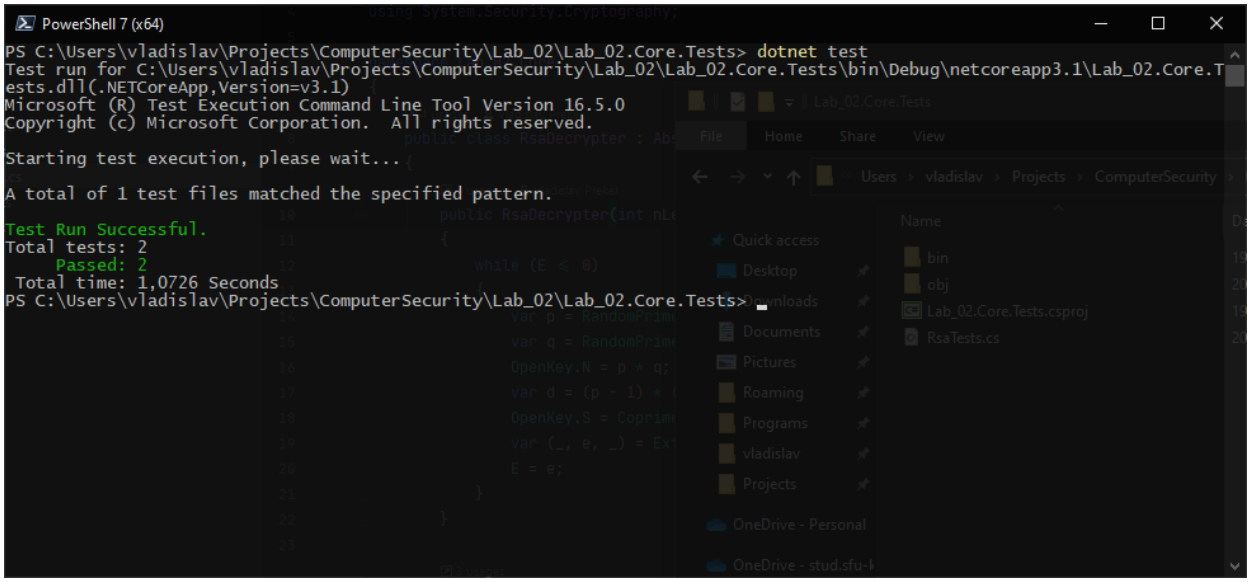

Рисунок 2 – Запуск юнит-тестов.

5 Вывод

В ходе выполнения практического задания нами были получены навыки работы с асимметричными шифрами, написания программного кода для шифрования и дешифрования текста, а также мы смог реализовать алгоритм шифрования RSA, проанализировав, поняли, что алгоритм не является таким уж производительным, а также, что его криптостойкость не является такой уж высокой.