ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ
ВЫСШЕГО ОБРАЗОВАНИЯ МОСКОВСКОЙ ОБЛАСТИ
«УНИВЕРСИТЕТ “ДУБНА”»

ИНСТИТУТ СИСТЕМНОГО АНАЛИЗА И УПРАВЛЕНИЯ

Кафедра системного анализа и управления

Кафедра распределенных информационно-вычислительных систем

КУРСОВАЯ РАБОТА

по дисциплине

«Программирование на языке высокого уровня»

ТЕМА: Реализация и анализ алгоритма шифрования *RSA* на *C#*

(наименование темы)

Выполнил: студент группы **1253**

Згонник Данил Андреевич
(Ф.И.О.)

(подпись студента)

Руководители:
по дисциплине ОИТ
асс. Жаткина К. Н.
(учёная степень, учёное звание, занимаемая должность, Ф.И.О.)

Дата:

Оценка:

(подпись руководителя)

по дисциплине ПЯВУ
доц. Мельникова О. И.
(учёная степень, учёное звание, занимаемая должность, Ф.И.О.)

Дата защиты:

Оценка:

(подпись руководителя)

Содержание

Введение

RSA (аббревиатура от фамилий разработчиков *Rivest, Shamir* и *Adleman*) - самый первый криптографический алгоритм, базирующийся на принципе ассиметричного шифрования. На данный момент большинство продуктов на рынке информационной безопасности используют именно этот метод шифрования, т. к. криптографическая стойкость *RSA* обусловлена вычислительной сложностью решения задачи факторизации больших целых чисел.

Криптосистема *RSA* используется в самых различных продуктах, на различных платформах и во многих отраслях. В настоящее время криптосистема *RSA* встраивается во многие коммерческие продукты, число которых постоянно увеличивается. Также ее используют операционные системы *Microsoft, Apple, Sun* и *Novell*. В аппаратном исполнении *RSA* алгоритм применяется в защищенных телефонах, на сетевых платах *Ethernet*, на смарт-картах, широко используется в криптографическом оборудовании [1].

Тема методов и алгоритмов шифрования привлекает меня, ведь в эпоху научных прорывов и открытий, когда информационные технологии используются повсеместно и уже невозможно представить свою жизнь без них, данная проблематика становится особенно актуальна. Именно поэтому я выбрал данный проект в качестве объекта изучения.

Задание на работу

Целью данной работы является реализация алгоритма шифрования *RSA* на языке *C#* в приложении с графическим интерфейсом *Windows Forms* на платформе *.NET*. Функционал разрабатываемого приложения заключается в возможности шифрование и дешифрование данных при помощи сгенерированной пары ключей. В ходе работы планируется подробно ознакомиться с общей концепцией ассиметричного шифрования, выявить преимущества и недостатки данного метода, а также определить его эффективность.

Теоретическая часть

Концепция криптосистем с открытым ключом была изложена в статье «Новые направления в криптографии» в 1976 г Уайтфилдом Диффи и Мартином Хеллманом в качестве решения проблемы распределения ключей. В соответствии с ней, каждая сторона получает пару ключей — открытый и закрытый. Открытый ключ распространяется свободно, в то время как закрытый держится в тайне. Исходный текст шифруется открытым ключом адресата и передается ему. Обратный процесс в принципе не может быть выполнен с помощью открытого ключа, для расшифровки получатель использует закрытый ключ, который известен только ему. Таким образом, кто угодно может посылать шифрованные сообщения, для этого ему не надо знать ничего тайного, но при этом только владелец закрытого ключа сможет их прочитать.

Изучив эту статью, трое учёных Рональд Ривест, Ади Шамир и Леонард Адлеман (см. рис. 1) из Массачусетского Технологического Института приступили к поискам математической функции, которая бы позволяла реализовать сформулированную Уайтфилдом Диффи и Мартином Хеллманом модель криптографической системы с открытым ключом. После работы над более чем 40 возможными вариантами, им удалось найти алгоритм, основанный на использовании так называемых односторонних функций - функций, вычислить значения которых во много раз проще, чем значения их обратной функции [3].

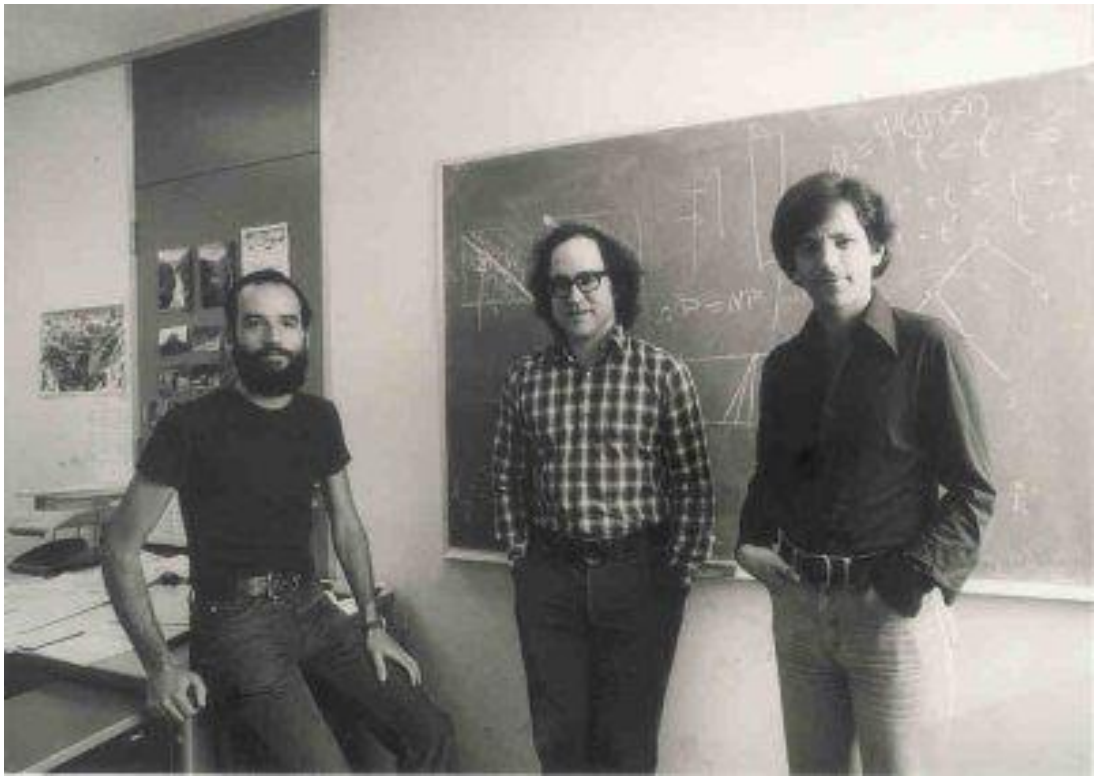

Рис. 1. Создатели *RSA*

Практическая часть

Описание взаимодействия пользователя с программой

Интерфейс приложения для шифрования и дешифрования данных при помощи алгоритма *RSA* представлен на рисунке 2.

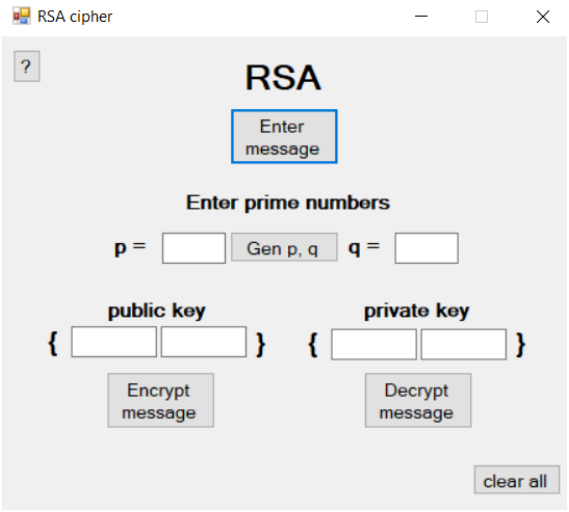

Рис. 2. Пользовательский интерфейса приложения

По нажатию на кнопку «*Enter message*» открывается окно текстового файла *Message.txt* (см рис. 3), куда пользователь вводит данные, которые он хочет зашифровать.

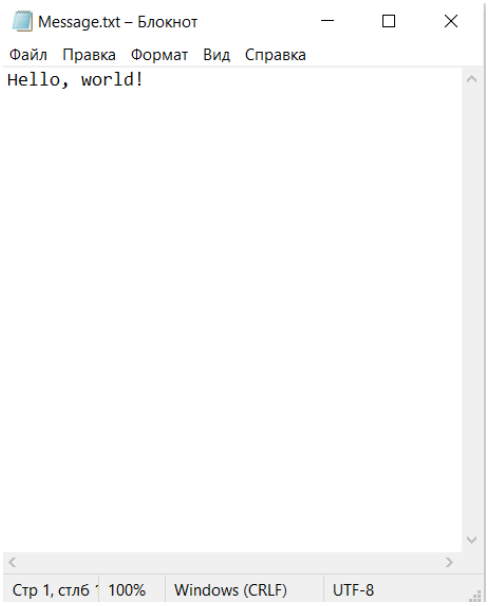

Рис. 3. Окно ввода данных

Далее пользователь вводит простые числа *p* и *q* самостоятельно или генерирует их, нажав кнопку «*Gen p, q*». На основе этих чисел генерируются два ключа: секретный и открытый (см. рис. 4). Затем, по нажатию на кнопку «*Encrypt message*», происходит шифрование данных, полученных из файла *Message.txt*, после чего открывается окно файла *Encrypted_message.txt* (см. рис. 5), в котором пользователь приложения может увидеть свое зашифрованное сообщение.

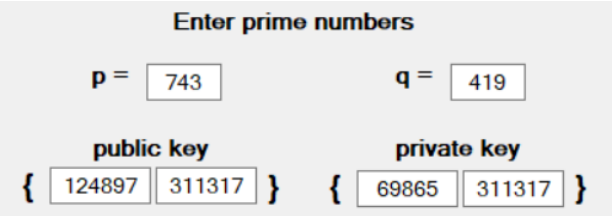

Рис. 4. Открытый и закрытый ключи

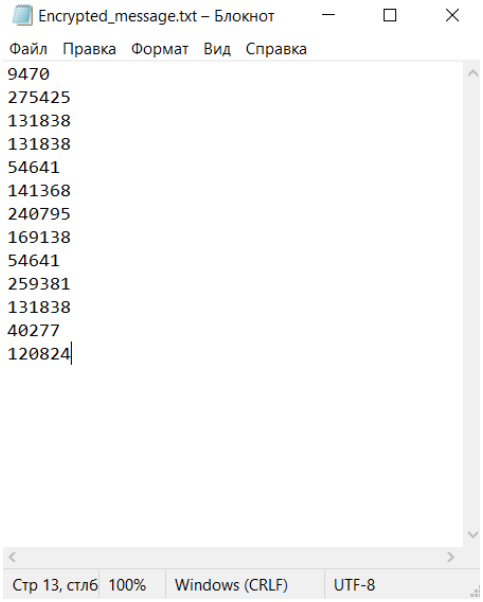

Рис. 5. Окно вывода зашифрованных данных

Чтобы расшифровать данные, полученные в файле *Encrypted_message.txt*, пользователь должен нажать на кнопку «*Decrypt message*», после чего запуститься процесс дешифрования. Его результат можно наблюдать в открывшемся окне файла *Decrypted_message.txt* (см. рис. 6).

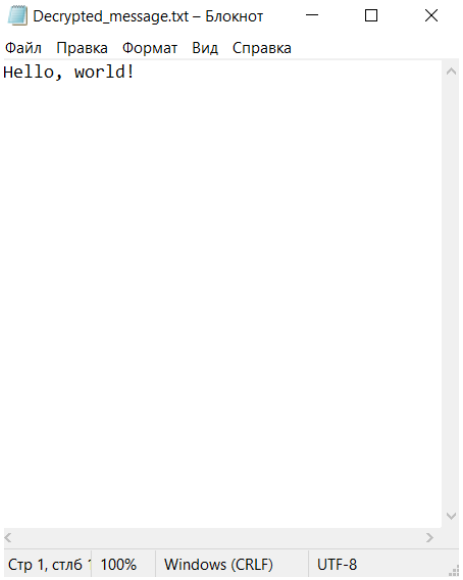

Рис. 6. Окно вывода расшифрованных данных

Итак, приложение успешно справляется со своей задачей: шифрует и дешифрует полученную информацию.

Описание структуры и работы программы

Шаг первый. Подготовка ключей.

Сначала необходимо сгенерировать секретный и приватный ключи, при помощи который информация будет шифроваться и дешифроваться. Делается это следующим образом.

- Необходимо вычислить произведение, полученных p и q: $n = p * q$. Пользователь может сгенерировать p и q, нажав на кнопку «*Gen p, q*». В основе этого действия заложен метод, возвращающий случайное простое число из массива, полученного при помощи алгоритма «Решето Эратосфена». Далее вычисляется функция Эйлера по следующей формуле:

$$f(n) = (p - 1) * (q - 1) \quad (1)$$

  Значением этой функции является количество натуральных простых чисел, не превышающих n и взаимно простых с ним (НОД этих чисел равно 1. Метод, проверяющий, являются ли числа взаимно простыми, был написан отдельно, при использовании алгоритма Евклида).

- Теперь необходимо выбрать открытую экспоненту e, которая должна быть простым числом меньшим $f(n)$ и взаимно простым с ним. Пара чисел $\{e, n\}$ есть открытый ключ, которым шифруется введенное сообщение. На входе метод вычисления e принимает значение функции Эйлера. Код метода вычисления e приведен ниже [5].

```
private long Calculate_exp(long f)
        {
                List<long> array_e = new List<long>();
                Random rnd = new Random();

                for(long i = 2; i < f; i++)
                {
                        if (IsTheNumberSimple(i) && CoprimeNumbers(i, f))
                                array_e.Add(i);
                }
                long e = array_e[rnd.Next(array_e.Count)];
                return e;
        }
```

- Последним этапом подготовки ключей является вычисление закрытой экспоненты d, обратной e по модулю $f(n)$. Иными словами, выполняется следующее условие:

$$(d * e)\%f(n) = 1 \quad (2)$$

Пара чисел $\{d, n\}$ будет секретным, которым введенное сообщение будет расшифровываться. На входе метод вычисления принимает e и f. Код метода вычисления d приведен ниже.

```
private long Calculate_d(long exp, long f)
        {
                long d = 0;
                while (true)
                {
                        if ((d * exp) % f == 1)
                                break;
                        else
                                d++;
                }
                return d;
        }
```

Достоинство ассиметричного метода шифрования заключается в использовании однонаправленных функций, т. к. не существует эффективного метода инвертирования этих функций относительно аргумента (решения задачи факторизации больших чисел). Такая однонаправленная функция и реализована в *RSA* шифровании — функция целочисленного умножения простых чисел (p и q) [4].

Шаг второй. Шифрование данных.

Для шифрования данных информацию сперва надо разбить на блоки. В качестве блока я выбрал один символ и представил его в виде числа *block* (индекс символа в алфавите). Далее этот блок шифруется по следующему принципу:

$$block = block^e \% n \quad (3)$$

Этот метод вычисления на входе принимает данные из файла *Message.txt*, открытую экспоненту и *n*. Код метода шифрования приведен ниже.

```
private List<string> RSA_coder(string str, long exp, long n)
        {
                List<string> result = new List<string>();

                BigInteger block;

                for (int i = 0; i < str.Length; i++)
                {
                        int index = Array.IndexOf(alph, str[i]);

                        block = new BigInteger(index);
                        block = BigInteger.Pow(block, (int)exp);

                        BigInteger n_ = new BigInteger((int)n);

                        block = block % n_;

                        result.Add(block.ToString());
                }

                return result;
        }
```

Т. к. при возведении значения *block* в степень *exp* возникало число, не находящееся в диапазоне ни одного из стандартных целочисленных типов (очень большое), был использован неизменяемый тип *BigInteger* пространства имен *System.Numerics*, представляющий большое целое число произвольной величины. Таким образом была решена проблема хранения огромных чисел [5]. Код кнопки, по нажатию на которую происходит шифрование (см. Приложение 1).

Шаг третий. Дешифрование данных.

Чтобы декодировать зашифрованные данные производится операция, похожая на процесс шифрования:

$$block = block^d \% n. \quad (4)$$

```
private string RSA_decoder(List<string> cipher, long d, long n)
            {
                  string result = "";

                  BigInteger block;

                  foreach (string item in cipher)
                  {
                        block = new BigInteger(Convert.ToDouble(item));
                        block = BigInteger.Pow(block, (int)d);
                        BigInteger n_ = new BigInteger((int)n);
                        block = block % n_;
                        int index = Convert.ToInt32(block.ToString());
                        if (index != -1)
                              result += alph[index].ToString();
                        else
                              continue;
                  }
                  return result;
```

Данный метод вычисления принимает на входе список, элементами которого являются зашифрованные блоки, закрытую экспоненту и *n*. Код метода дешифрования приведен ниже. В данном методе тоже используется тип *BigInteger*, поскольку при дешифровании операции производятся над еще большими числами [2][6]. Код кнопки, по нажатию на которую происходит шифрование (см. Приложение 2).

Аналогично сценарий использования программы пользователем можно представить в виде *UML*-диаграммы (см. рис. 7).

Рис. 7. *UML*-диаграмма вариантов использования программы

Анализ алгоритма

Преимущества алгоритма

Главным преимуществом ассиметричной криптосистемы *RSA*, обнаруженным в ходе работы, является то, что, используя ассиметричные алгоритмы, решается проблема распределения ключей между пользователями. Иными словами, стороны могут свободно связываться и обмениваться данными друг с другом без использования секретных каналов связи, распространяя свои открытые ключи по коммуникационным сетям (см. рис. 7). Кроме того, исчезает квадратичная зависимость числа ключей от числа пользователей (в системе из N пользователей $2N$ ключей).

Рис. 8. Схема передачи данных

Недостатки алгоритма

Т. к. операция генерации пары ключей проводится значительно реже, нежели чем операции шифрования и дешифрования данных, следовательно задача вычисления $block = block^x \% n$ представляет основную вычислительную нагрузку и является весьма ресурсоемкой, из-за чего ассиметричное шифрование достаточно медленное. Еще один недостаток данного метода – необходимость защиты открытых ключей от подмены. На практике было установлено, что процесс расшифровки данных значительно медленнее их шифрования, в силу того, что операции вычисления производятся над еще большими числами (что число d в большинстве случаев больше числа e и приближено к n).

Вывод

В результате работы была реализована программа для шифрования и дешифрования данных при помощи ассиметричного алгоритма шифрования *RSA*. Кроме того, мною были изучены общие принципы информационной безопасности в целом и концепция ассиметричной криптосистемы *RSA* в частности. В ходе работы были выявлены главные достоинства и существенные недостатки данного алгоритма, что, само собой, позволяет эффективно применять криптосистему в тех или иных условиях, учитывая ее особенности. В любом случае, в приложении была реализована лишь базовая логика *RSA* без использования сторонних методов и алгоритмов шифрования, которые применяются для решения реальных задач. В следствие чего, данная программа имеет возможности для расширения функциональности и повышения криптостойкости, а также в дальнейшем может стать основой для создания более крупного проекта.

Список используемой литературы

1. *Cyberguru* [Электронный ресурс] / Статья «Стандарты и применения *RSA*» Режим доступа: *http://www.cyberguru.ru/algorithms/cryptography.html*, свободный (дата обращения: 05.03.2021).
2. *Microsoft* [Электронный ресурс] / Документация к структуре *BidInteger*. Режим доступа:*https://docs.microsoft.com/ru-ru/dotnet/api/system.numerics.biginteger?view=net-5.0*, свободный (дата обращения: 01.04.2021).
3. *Wiki Reading* [Электронный ресурс] / Статья «1977 год. Алгоритм шифрования с открытым ключом *RSA*». Режим доступа: *https://it.wikireading.ru/1000007481*, свободный (дата обращения: 05.03.2021).
4. *Wikipedia* [Электронный ресурс] / Статья «*RSA*». Режим доступа: *https://ru.wikipedia.org/wiki/RSA#%D0%92%D0%B2%D0%B5%D0%B4%D0%B5%D0%BD%D0%B8%D0%B5*, свободный (дата обращения: 20.03.21).
5. *Wikipedia* [Электронный ресурс] / Статья «Функция Эйлера». Режим доступа: *https://ru.wikipedia.org/wiki/%D0%A4%D1%83%D0%BD%D0%BA%D1%86%D0%B8%D1%8F_%D0%AD%D0%B9%D0%BB%D0%B5%D1%80%D0%B0*, свободный (дата обращения: 20.03.21).
6. *Your Private Network* [Электронный ресурс] / Статья «Асимметричные криптосистемы шифрования». Режим доступа: *http://ypn.ru/197/asymmetric-encryption-system/2/*, свободный (дата обращения: 01.04.2021).

Приложение 1

```
private void Encrypt_button_Click(object sender, EventArgs e)
            {
                  if(IsNumber(textBox_p.Text) && IsNumber(textBox_q.Text))
                  {
                        long p = Convert.ToInt32(textBox_p.Text);
                        long q = Convert.ToInt32(textBox_q.Text);

                        if (IsTheNumberSimple(p) && IsTheNumberSimple(q))
                        {
                              long n = p * q;
                              long f = (p - 1) * (q - 1);
                              long exp = Calculate_exp(f);
                              long d = Calculate_d(exp, f);
                              textBox_n1.Text = n.ToString();
                              textBox_n2.Text = n.ToString();
                              textBox_e.Text = exp.ToString();
                              textBox_d.Text = d.ToString();
                              string str = "";
                              StreamReader sr = new StreamReader("Message.txt");

                              while (!sr.EndOfStream)
                              {
                                    str += sr.ReadLine();
                              }

                              sr.Close();

                              List<string> res = RSA_coder(str, exp, n);

                              StreamWriter sw = new StreamWriter("Encrypted_message.txt");
                              foreach (string item in res)
                                    sw.WriteLine(item);
                              sw.Close();

                              Process.Start("Encrypted_message.txt");
                        }
                        else
                              MessageBox.Show("Enter prime numbers p and q");
                  }
                  else
                        MessageBox.Show("Enter prime numbers p and q");
                  }
```

Выше представлен код кнопки, по нажатию на которую происходит шифрование данных.

Приложение 2

```
private void Dencrypt_button_Click(object sender, EventArgs e)
            {
                  if (textBox_d.Text.Length > 0 && textBox_n2.Text.Length
> 0)
                  {
                        long d = Convert.ToInt64(textBox_d.Text);
                        long n = Convert.ToInt64(textBox_n2.Text);

                        List<string> message = new List<string>();
                        StreamReader sr = new StreamReader("En-
crypted_message.txt");

                        while (!sr.EndOfStream)
                        {
                              message.Add(sr.ReadLine());
                        }

                        string res = RSA_decoder(message, d, n);
                        StreamWriter sw = new StreamWriter("De-
crypted_message.txt");
                        sw.WriteLine(res);
                        sw.Close();

                        Process.Start("Decrypted_message.txt");

                  }
                  else
                        MessageBox.Show("Enter private key");

                  }
```

Выше представлен код кнопки, по нажатию на которую происходит дешифрование данных.

Приложение 3

Таблица 1. План работы

№	Этап выполнения	Предполагаемая дата выполнения	Реальная дата выполнения
1	Выбор, согласование и утверждение темы курсовой работы	16.02.2021	16.02.2021
2	Составление и сдача преподавателю плана-графика выполнения курсовой работы	02.03.2021	02.03.2021
3	Сбор информации по теме курсовой работы (если необходимо)	02.03.2021	02.03.2021
4	Разработка начального варианта интерфейса проекта	23.03.2021	23.03.2021
5	Согласование его с преподавателем	23.03.2021	13.04.2021
6	Доработка и согласование с преподавателем окончательного интерфейса проекта	25.03.2021	20.04.2021
7	Написание кода проекта	30.03.2021	20.04.2021
8	Создание и утверждение текстового отчета по курсовой работе	15.04.2021	
9	Защита курсовой работы	30.04.2021	