Министерство науки и высшего образования Российской Федерации  
Федеральное государственное автономное образовательное учреждение высшего образования «Уральский федеральный университет имени первого Президента России Б.Н. Ельцина» (УрФУ)

Институт фундаментального образования

Департамент «школа бакалавриата»



ОТЧЕТ

отчет по лабораторной работе дисциплины «Алгоритмы и анализ

сложности»

«Целочисленная арифметика произвольной точности и RSA-шифрование»

Студент: Чернышев Иван Антонович РИ-210911

Преподаватель: Трофимова Ольга Геннадиевна, Доцент, к.т.н.

Екатеринбург

2021

1. **Оглавление**

[**2.** **Задание** 2](#_Toc66552332)

[**3.** **Теоретическая часть** 2](#_Toc66552333)

[3.1. Описание алгоритма 2](#_Toc66552334)

[3.2. Генерация ключей шифрования 3](#_Toc66552335)

[3.3. Шифрование и дешифровка сообщения 4](#_Toc66552336)

[**4.** **Инструкция пользователя** 5](#_Toc66552337)

[4.1. Режим шифрования/дешифрования текста в консоли 5](#_Toc66552338)

[4.2. Режим шифрования/дешифрования текстового файла 6](#_Toc66552339)

[**5.** **Инструкция программиста** 8](#_Toc66552340)

[5.1. Класс TextCryptographer 9](#_Toc66552341)

[5.1.1. Метод Encrypt(…) 9](#_Toc66552342)

[5.1.2. Метод Decrypt(…) 9](#_Toc66552343)

[5.2. Класс PublicKeyGenerator 10](#_Toc66552344)

[5.2.1. Метод Generate(int p, int q) 10](#_Toc66552345)

[5.3. Класс PrivateKeyGenerator 10](#_Toc66552346)

[5.3.1. Метод Generate(PublicKey publicKey) 11](#_Toc66552347)

[5.4. Класс BigInt 11](#_Toc66552348)

[5.4.1. Метод Add(BigInt a, BigInt b) 11](#_Toc66552349)

[5.4.2. Метод Sub(BigInt a, BigInt b) 11](#_Toc66552350)

[5.4.3. Метод Mul(BigInt a, BigInt b) 11](#_Toc66552351)

[5.4.4. Метод KaratsubaMultiplication(BigInt a, BigInt b) 12](#_Toc66552352)

[5.4.5. Метод Div(BigInt a, BigInt b) 12](#_Toc66552353)

[5.4.6. Метод Mod(BigInt a, BigInt b) 12](#_Toc66552354)

[5.4.7. Метод Pow(BigInt value, BigInt n) 12](#_Toc66552355)

[5.4.8. Метод Inverse(BigInt a, BigInt b) 13](#_Toc66552356)

[5.4.9. Метод GreatestCommonDivisor(BigInt a, BigInt b) 13](#_Toc66552357)

[**6.** **Тестирование** 13](#_Toc66552358)

[6.1. Тестирование класса BigInt 14](#_Toc66552359)

[6.2. Тестирование классов RSA-шифрования 14](#_Toc66552360)

[**7.** **Вывод** 15](#_Toc66552361)

[**8.** **Литература** 16](#_Toc66552362)

[**9.** **Приложение** 17](#_Toc66552363)

[9.1 Класс RsaEncrypting, метод EncryptNumber: 17](#_Toc66552364)

[9.2 Класс TextCryptographer, метод Encrypt: 17](#_Toc66552365)

[9.3 Класс TextCryptographer, метод Decrypt: 17](#_Toc66552366)

[9.4 Класс BigInt, метод KaratsubaMulttiplication: 17](#_Toc66552367)

[9.5 Класс BigInt, метод Inverse: 18](#_Toc66552368)

[9.6 Тестирование класса BigInt: 18](#_Toc66552369)

[9.7 Тестирование RSA-шифрования: 20](#_Toc66552370)

1. **Задание**
   1. Написать класс, который содержит целое число со знаком в виде массива однобайтовых элементов. Реализовать конструкторы, деструктор, перегрузить операции: аддитивные (+, –), мультипликативные (\*, /, %), сравнения (==, !=, <, >), взятие обратного по заданному модулю.
   2. Написать функцию шифрования строки с помощью алгоритм RSA.
   3. Зашифровать/расшифровать текстовый файл с помощью открытого RSA-ключа.
2. **Теоретическая часть**
   1. Описание алгоритма

Асимметричные криптографические системы (коей является и алгоритм RSA-шифрования) основаны на так называемых односторонних функциях. Под односторонней понимается такая функция , которая легко вычисляется при заданном x, причём при известных аргументах k и y значение x восстановить просто, а при неизвестном k – вычислительно сложно.

Подобным свойством обладает операция возведения числа в степень по модулю:

Обратимся к первому выражению. Здесь число *c* получено в результате возведения в степень по модулю числа *m*. Это действие будем называть «шифрование». Тогда становится очевидно, что *m* выступает в роли исходного текста, а *c* – шифровка. Значение *c* будет зависит от степени *e*, в которую возводилось число *m*, и от модуля *n*, по которому был взят результат шифрования.

Таким образом, эта пара чисел *(e, n)* образует «открытый ключ».

Перейдем ко второму выражению. Здесь *d* является параметром, с помощью которого мы получаем исходный текст *m* из шифротекста *c*. Этот параметр называют «закрытый ключ».

* 1. Генерация ключей шифрования

Для генерации открытого и закрытого ключей необходимо проделать следующие действия:

1. Выбрать такое число n, что:

где *p* и *q* – некоторые разные простые числа. Для такого n функция Эйлера имеет вид:

Такой выбор *n* обусловлен следующим. Число *n* передается в составе открытого ключа и теоретически может быть перехвачено злоумышленником. Но чтобы третье лицо смогло вычислить закрытый ключ, ему необходимо произвести факторизацию числа n, т. е. разложить число на простые множители. Но эта операция является вычислительно сложной для больших *n*.

1. Теперь выберем целое число e, которое меньше и взаимно просто ему:
2. Отсюда вычислим *d*:

Отыскание числа, обратного по модулю, производится с помощью расширенного алгоритма Евклида (см. приложение).

На этом этап генерации ключей закончен, получившийся открытый ключ (e, n) публикуется, а закрытый ключ d остается скрытым.

* 1. Шифрование и дешифровка сообщения

Возьмём в качестве сообщения число *m* (). Чтобы его зашифровать, необходимо возвести это число в степень *e* по модулю *n*:

Результат *c* вместе с открытым ключом передается другой стороне ().

Теперь расшифруем шифровку, возведя её в степень закрытого ключа *d,* и взяв результат по модулю *n*:

Для реализации операций с большими числами используется самописный класс BigInt. Чтобы ускорить процесс шифрования/дешифрования, мною был ограничен набор шифруемых символов (обрабатываются только буквы русского и латинского алфавитов, пробел и арабские цифры, в сумме 69 символов). Таким образом, появляется возможность выбрать числа *p* и *q,* равные 7 и 11, соответственно. Отсюда напрямую следуют небольшие значения открытого и закрытого ключа (открытый ключ – (7, 77), закрытый ключ – (43)), с которыми нам придется производить математические операции, что позволяет ускорить вычисление исходного символа из закодированного.

1. **Инструкция пользователя**

После запуска приложения в консоли отображается приветственное сообщение с информацией о программе и предупреждение о том, что программа может зашифровать только текст, состоящий из букв русского и английского алфавита, цифр, а также текст будет приведен к нижнему регистру.

Далее пользователю предлагается выбрать два простых числа P и Q, чтобы начать работу с RSA-алгоритмом (*Рис. 1*).

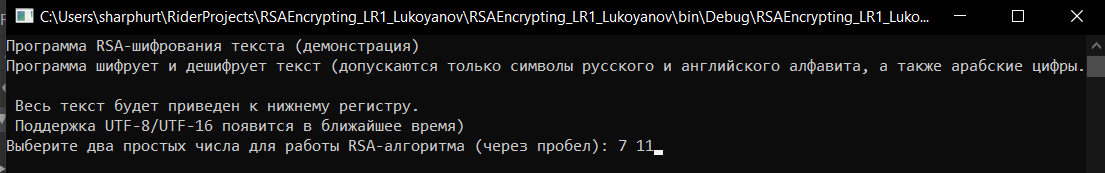


Рис. 1. Приветственное сообщение и предложение выбрать P и Q

После того, как пользователь выбрал 2 простых числа, ему предлагается один из двух доступных вариантов работы: шифрование/дешифрование текста непосредственно в консоли (режим «1»), или шифрование и дешифрование текстового файла (режим «2») (*Рис. 2*).

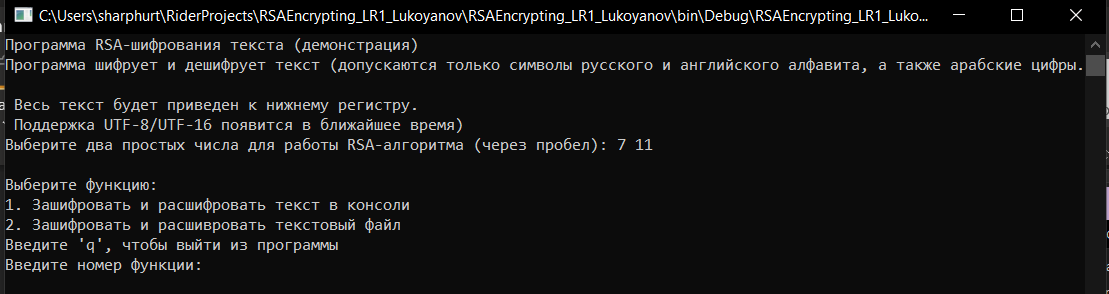


Рис. 2. Приветственное сообщение и диалог выбора режима работы

* 1. Режим шифрования/дешифрования текста в консоли

Чтобы активировать режим, пользователь вводит цифру «1» в ответ на предложение «Введите номер функции: »

После этого пользователю предлагается ввести текст, который необходимо зашифровать. После окончания процесса шифрования (ход процесса отображается в виде полосы прогресса и процента обработанных символов) в консоль выводится массив чисел, каждое из которых соответственно представляет собой зашифрованную числовую интерпретацию символов изначальной строки (*Рис. 3*).

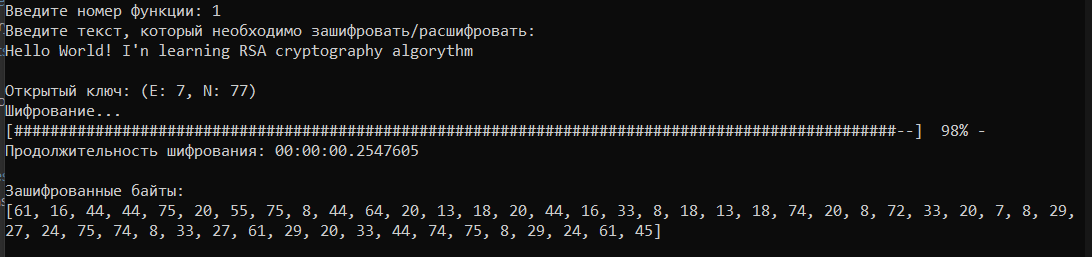


Рис. 3. Процесс шифрования строки

Далее происходит обратная дешифровка полученных байтов в исходный текст (*Рис. 4*).

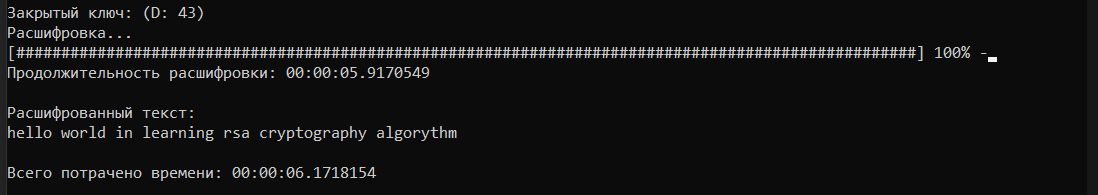


Рис. 4. Дешифрование зашифрованной информации

Как видим, данные дешифровались верно (с учетом ограничений алфавита).

Далее пользователю снова предлагается выбрать номер режима, в котором он хочет работать.

* 1. Режим шифрования/дешифрования текстового файла

Режим работы с файлами похож на режим работы со строками. Здесь пользователю предлагается ввести путь к исходному текстовому файлу с расширением .txt (*Рис. 5*).

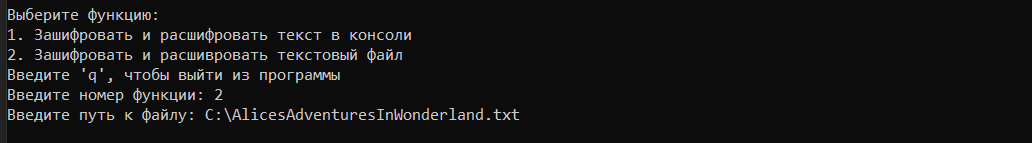


Рис. 5. Режим шифрования/дешифрования файла

Для примера возьмем файл, содержащий отрывок из произведения Льюиса Кэрролла «Алиса в Стране чудес» (*Рис. 6*)

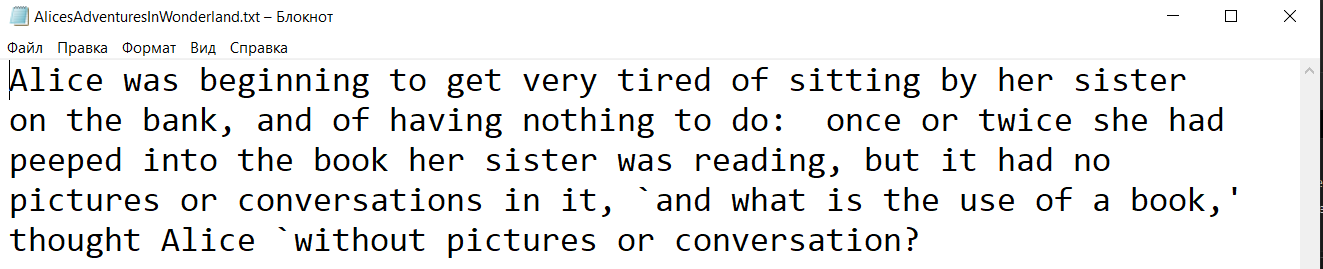


Рис. 6. Отрывок текста для тестирования алгоритма

После того, как пользователь ввел путь к файлу, происходит шифрование файла. Результат шифрования записывается в новый файл, находящийся в той же директории, что и исходный (*Рис. 7*)

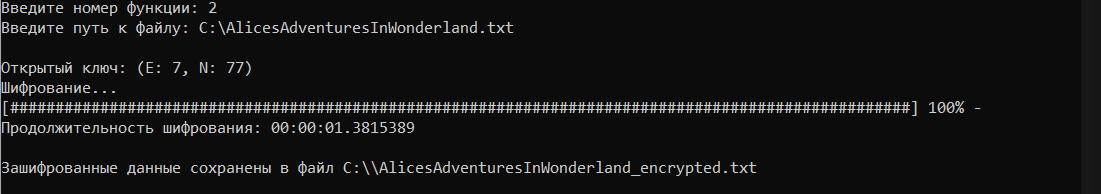


Рис. 7. Процесс шифрования файла

Зашифрованные данные имеют следующий вид (*Рис. 8*). По форме, они ничем не отличаются от данных, которые пользователь получает в режиме «1»

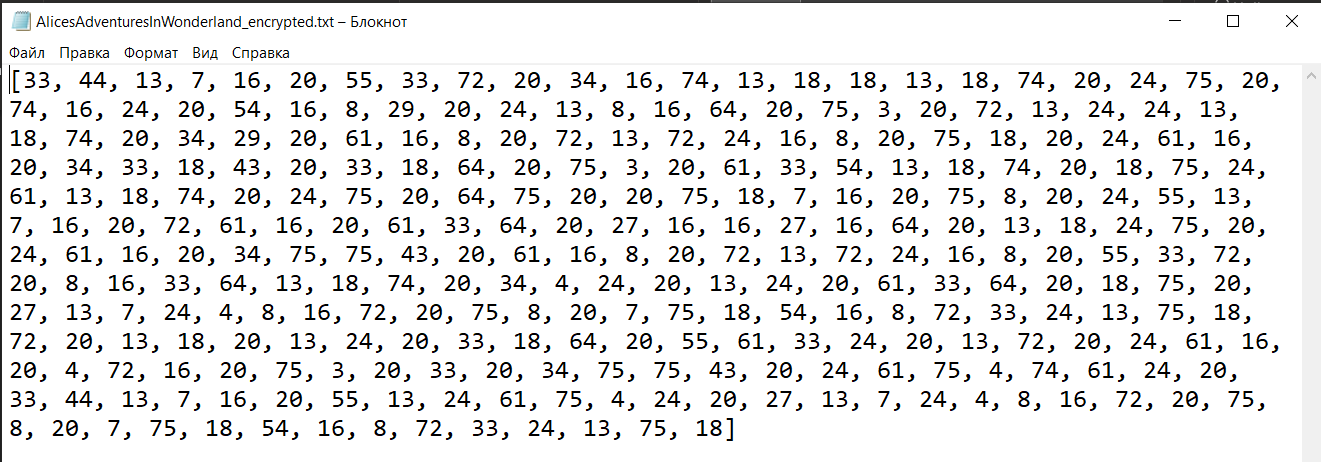


Рис. 8. Зашифрованный файл

Также, как и в предыдущем режиме, дешифрование происходит автоматически, и дешифрованные данные также записываются в новый файл (*Рис. 9, Рис. 10*)

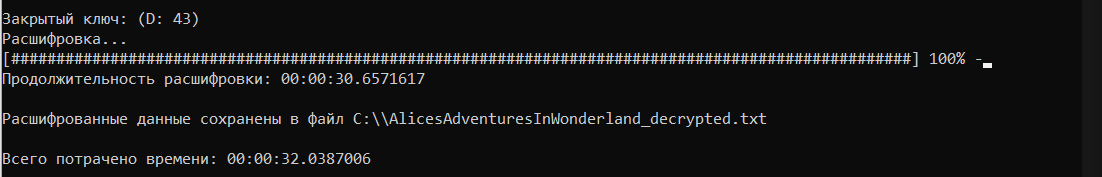


Рис. 9. Дешифрование файла

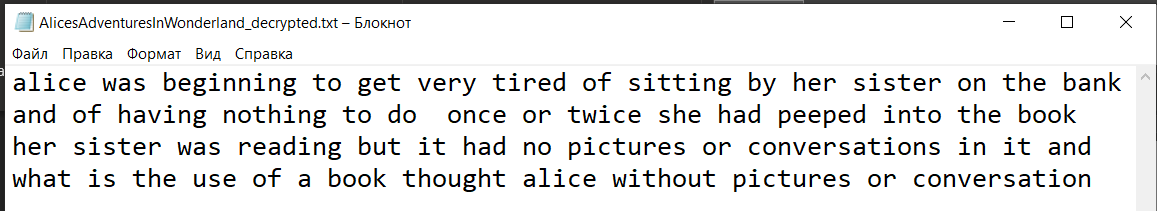


Рис. 10. Дешифрованный текст

Чтобы выйти из программы, пользователю достаточно закрыть окно консоли, либо при выборе режима ввести символ «q» (от англ. «quit» – выход)

1. **Инструкция программиста**

В исходном коде уже написан класс TextCryptographer.cs, позволяющий шифровать и дешифровать текстовые данные. Программист может также написать свои классы для шифрования нетекстовых данных (например, изображения, видео, звук и т. д.).

Класс, написанный программистом, обязательно должен имплементировать интерфейс ICryptographer. Это позволяет без труда изменять способ шифрования в консольном приложении: для этого достаточно создать экземпляр вашего класса в строке 17 файла Program.cs (*Рис. 11*)

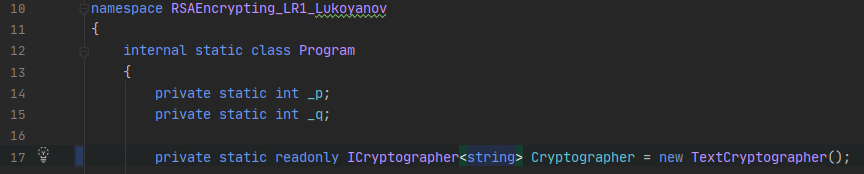


Рис. 11. Создание экземпляра класса TextCryptographer

Методы Encrypt и Decrypt в создаваемом классе должны быть асинхронными.

* 1. Класс TextCryptographer
     1. Метод Encrypt(…)

Метод Encrypt предназначен для кодирования переданных данных с открытым ключом. Делает это в асинхронном режиме, что позволяет отслеживать прогресс операции.

Интерфейс метода Encrypt выглядит следующим образом:

Task<BigInt.BigInt[]> Encrypt(T text, PublicKey publicKey, Progress<double> progress = default);

Здесь в метод передаются 3 аргумента:

T text – данные, предназначенные для шифрования (Тип T задается программистом вручную в зависимости от назначения класса)

PublicKey publicKey – открытый ключ шифрования. Может быть сгенерирован с помощью фабрики PublicKeyGenerator.

IProgress<double> progress – сущность, обрабатывающая прогресс операции (не является обязательным параметром)

Возвращаемое значение метода - Task<BigInt.BigInt[]> - массив объектов BigInt, представляющий собой закодированную информацию.

* + 1. Метод Decrypt(…)

Метод Decrypt предназначен для декодирования информации, закодированной методом Encrypt(…).

Интерфейс метода выглядит следующим образом:

Task<T> Decrypt(BigInt.BigInt[] encrypt, PrivateKey privateKey, IProgress<double> progress = default);

Здесь в метод передаются 3 параметра:

BigInt.BigInt[] encrypt – массив чисел BigInt, представляющий собой закодированные данные

PrivateKey privateKey – закрытый ключ шифрования. Может быть сгенерирован фабрикой PrivateKeyGenerator

IProgress progress - сущность, обрабатывающая прогресс операции (не является обязательным параметром)

Возвращаемое значение метода – объект типа T, представляющий собой расшифрованные данные.

* 1. Класс PublicKeyGenerator

Статический класс PublicKeyGenerator представляет собой фабрику открытых ключей шифрования. Чтобы получить открытый ключ, необходимо вызвать метод Generate(…)

* + 1. Метод Generate(int p, int q)

Метод Generate генерирует открытый ключ, основываясь на переданных значениях P и Q. Отыскание числа Е происходит в цикле, начиная с 3, до тех пор, пока наибольший общий делитель числа e и числа phi не станет равен 1.

Далее возвращается экземпляр класса PublicKey, содержащий в себе вычисленные значения.

* 1. Класс PrivateKeyGenerator

Статический класс PrivateKeyGenerator представляет собой фабрику открытых ключей шифрования. Чтобы получить открытый ключ, необходимо вызвать метод Generate(…)

* + 1. Метод Generate(PublicKey publicKey)

Метод Generate генерирует закрытый ключ, основываясь на данных переданного открытого ключа. Вычисление D производится с помощью метода BigInt.Inverse(…).

Далее возвращается экземпляр закрытого ключа.

* 1. Класс BigInt

Класс BigInt предназначен для работы с большими знаковыми числами (больше 64 бит). Класс реализует все основные операции с числами (сложение/вычитание/умножение/целочисленное деление, взятие остатка, возведение в степень), а также нахождение числа, обратного по модулю к данному, и операции сравнения.

* + 1. Метод Add(BigInt a, BigInt b)

Метод Add принимает два числа типа BigInt, и вычисляет их сумму методом «в столбик».

Возвращает вычисленное значение типа BigInt.

Также этот метод перегружает оператор «+».

* + 1. Метод Sub(BigInt a, BigInt b)

Метод Sub принимает два числа типа BigInt, и вычисляет их разность методом «в столбик».

Возвращает вычисленное значение типа BigInt.

Также этот метод перегружает оператор «–»

* + 1. Метод Mul(BigInt a, BigInt b)

Метод Mul принимает два числа типа BigInt, и вычисляет их произведение методом «в столбик».

Возвращает вычисленное значение типа BigInt.

* + 1. Метод KaratsubaMultiplication(BigInt a, BigInt b)

Метод KaratsubaMultiplication принмает 2 числа типа BigInt, и вычисляет их произведение по алгоритму Карацубы. Гораздо эффективнее метода «в столбик», т. к. имеет сложность

Возвращает вычисленное значение типа BigInt.

Также этот метод перегружает оператор «\*»

* + 1. Метод Div(BigInt a, BigInt b)

Метод Div принимает два числа типа BigInt, и вычисляет частное этих чисел методом «в столбик»

Возвращает вычисленное значение типа BigInt.

Также этот метод перегружает оператор «/»

* + 1. Метод Mod(BigInt a, BigInt b)

Метод Mod принимает два числа типа BigInt, и вычисляет остаток от деления этих двух чисел по формуле (всегда неотрицательный):

Возвращает вычисленное значение типа BigInt.

Также этот метод перегружает оператор «%»

* + 1. Метод Pow(BigInt value, BigInt n)

Метод Pow принимает два числа, и вычисляет число по алгоритму бинарного возведения в степень. Такой алгоритм работает в среднем за операций умножения, а операция умножения Карацубы имеет сложность . Таким образом, общая сложность алгоритма равна .

Возвращает вычисленное значение типа BigInt.

* + 1. Метод Inverse(BigInt a, BigInt b)

Метод Inverse принимает два числа BigInt и возвращает число, обратное числу a по модулю b, т. е. такое, что:

Результат вычисляется с помощью расширенного алгоритма Евклида.

Сложность алгоритма была исследована еще в IXX веке Габриэлем Ламе, и с учетом использования алгоритма Карацубы для умножения составляет:

где n – битовая длина чисел a и b

* + 1. Метод GreatestCommonDivisor(BigInt a, BigInt b)

Метод GreatestCommonDivisor(BigInt a, BigInt b) находит и возвращает НОД(a, b), наибольший общий делитель чисел a и b. Вычисление происходит с помощью алгоритма Евклида.

1. **Тестирование**

Чтобы убедиться в корректной работе алгоритмов, мною были написаны тесты для каждого метода классов BigInt, RSAEncrypting и TextCryptographer.

* 1. Тестирование класса BigInt

Для каждого метода класса BigInt были написаны несколько тест-кейсов, охватывающих различные входные данные (*Рис. 12*). Так, например, для методов сложения/вычитания/умножения/деления и возведения в степень были написаны тесты с отрицательными значениями и тесты с нулями и единицами (примеры тестов см. в приложении).

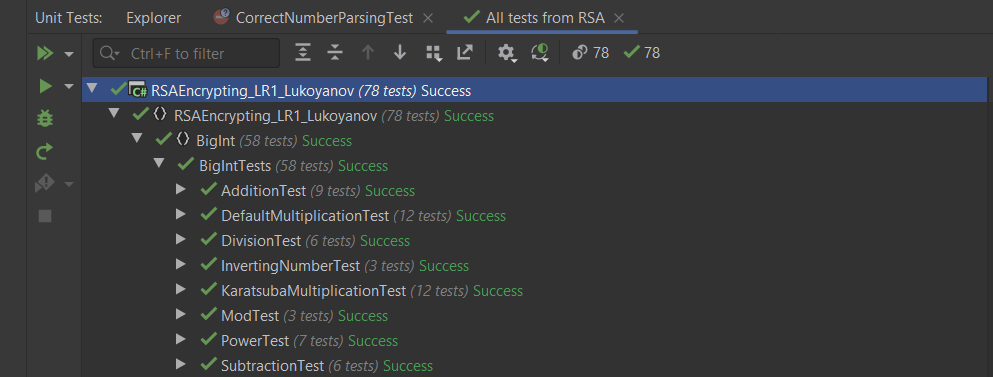


Рис. 12. Тестирование класса BigInt

* 1. Тестирование классов RSA-шифрования

Для тестирования методов шифрования и дешифрования изначально были установлены входные параметры: значения числе P и Q, и на их основе вычислены открытый и закрытый ключ (*Рис. 13*). Тест-кейсы были сгенерированы с помощью онлайн-калькулятора.

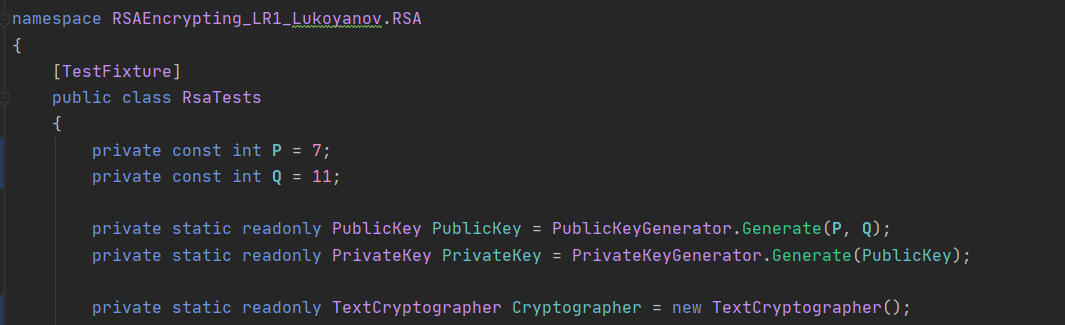


Рис. 13. Начальные параметры для тестирования RSA-шифрования

1. **Вывод**

В ходе работы мною были изучены и реализованы на языке C# алгоритмы для работы с большими знаковыми числами, алгоритмы шифрования и дешифрования чисел и строк. Результатом работы стало написание консольного приложения для шифрования и дешифрования текстовых строк в консоли или текстовых файлов на компьютере пользователя.

1. **Литература**
   1. RSA Calculator [Электронный ресурс] Режим доступа: <https://www.cs.drexel.edu/~jpopyack/IntroCS/HW/RSAWorksheet.html>
   2. Об алгоритме Евклида [Электронный ресурс] Режим доступа: <https://habr.com/ru/post/464949/>
   3. RSA: от простых чисел до электронной подписи [Электронный ресурс] Режим доступа: <https://habr.com/ru/post/534014/>
   4. Длинная арифметика [Электронный ресурс] Режим досупа: <https://ru.wikipedia.org/wiki/Длинная_арифметика>
   5. Умножение длинных чисел методом Карацубы [Электронный ресурс] Режим доступа: <https://habr.com/ru/post/124258/>
   6. Расширенный алгоритм Евклида [Электронный ресурс] Режим доступа: <http://e-maxx.ru/algo/export_extended_euclid_algorithm>
2. **Приложение**
   1. Класс RsaEncrypting, метод EncryptNumber:

public static class RsaEncrypting  
{  
 public static BigInt.BigInt EncryptNumber(int number, PublicKey publicKey) =>  
 BigInt.BigInt.Pow(number, publicKey.E) % publicKey.N;  
   
 public static int DecryptNumber(BigInt.BigInt number, PrivateKey privateKey) =>  
 int.Parse((BigInt.BigInt.Pow(number, privateKey.D)­ % privateKey.N).ToString());  
}

* 1. Класс TextCryptographer, метод Encrypt:

public async Task<BigInt.BigInt[]> Encrypt(string text, PublicKey publicKey, IProgress<double> progress = default)  
{  
 var textBytes = StringUtils.ConvertToNumbers(text.ToLower(), \_alphabet);  
 var encrypted = new BigInt.BigInt[textBytes.Length];  
  
 await Task.Run(() =>  
 {  
 for (var i = 0; i < textBytes.Length; i++)  
 {  
 encrypted[i] = RsaEncrypting.EncryptNumber(textBytes[i], publicKey);  
 progress?.Report(100.0 / text.Length \* i);  
 }  
  
 progress?.Report(100);  
 });  
  
 return encrypted;  
}

* 1. Класс TextCryptographer, метод Decrypt:

public async Task<string> Decrypt(BigInt.BigInt[] encrypt, PrivateKey privateKey, IProgress<double> progress = default)  
{  
 var decrypted = new int[encrypt.Length];  
 await Task.Run(() =>  
 {  
 for (var i = 0; i < encrypt.Length; i++)  
 {  
 decrypted[i] = (byte) RsaEncrypting.DecryptNumber(encrypt[i], privateKey);  
 progress?.Report(Math.Round(100.0 / encrypt.Length \* i));  
 }  
  
 progress?.Report(100);  
 });  
 return StringUtils.ConvertNumbersToString(decrypted, \_alphabet);  
}

* 1. Класс BigInt, метод KaratsubaMulttiplication:

*/// <summary>  
/// Умножение двух чисел (Алгоритм Карацубы)  
/// Рекурсивный алгоритм умножения двух чисел подходит для умножения больших чисел  
/// Гораздо эффективнее умножения "столбиком", т.к. имеет сложность O(n^log\_2(3))  
/// </summary>  
/// <param name="x">Первый множитель</param>  
/// <param name="y">Второй множитель</param>  
/// <returns>Произвдение двух множителей</returns>*public static BigInt KaratsubaMultiplication(BigInt x, BigInt y)  
{  
 var ux = new BigInt(Sign.**Plus**, x.\_digits);  
 var uy = new BigInt(Sign.**Plus**, y.\_digits);  
 var sign = x.Sign == y.Sign ? Sign.**Plus** : Sign.**Minus**;  
  
 var maxSize = Math.Max(x.Size, y.Size);  
  
 if (maxSize < 2)  
 return DefaultMultiplication(x, y);  
  
 maxSize = maxSize / 2 + maxSize % 2;  
  
 var m = Power10(One, maxSize);  
  
 var b = ux % m;  
 var a = ux / m;  
 var c = uy / m;  
 var d = uy % m;  
  
 var z0 = a \* c;  
 var z1 = b \* d;  
 var z2 = (a + b) \* (c + d);  
  
 var result = DefaultMultiplication(Power10(One, maxSize \* 2), z0) + z1 +  
 DefaultMultiplication(z2 - z1 - z0, m);  
 return new BigInt(sign, result.\_digits);  
}

* 1. Класс BigInt, метод Inverse:

*/// <summary>  
/// Число, обратное по модулю (Расширенный алгоритм Евклида)  
/// </summary>  
/// <param name="a">Число, для которого необходимо найти обратное</param>  
/// <param name="n">Модуль, по которому находится обратное</param>  
/// <returns>Возвращает число, обратное A по модулю N (такое, что A \* X mod N = 1)</returns>*public static BigInt Inverse(BigInt a, BigInt n)  
{  
 BigInt i = n, v = Zero, d = One;  
 while (a > Zero)  
 {  
 var t = i / a;  
 var x = a;  
 a = i % x;  
 i = x;  
 x = d;  
 d = v - t \* x;  
 v = x;  
 }  
  
 v %= n;  
 if (v < Zero) v = (v + n) % n;  
 return v;  
}

* 1. Тестирование класса BigInt:

private static void OperationTest(string first, string second, string result, Func<BigInt, BigInt, BigInt> func)  
{  
 var a = new BigInt(first);  
 var b = new BigInt(second);  
 var r = func(a, b);  
 Assert.True(r == new BigInt(result), $"Result is {r}");  
}  
  
[TestCase("373351898", "22242218987747101", "22242219361098999")]  
[TestCase("76264", "668554717734", "668554793998")]  
[TestCase("264523115614", "657282219887", "921805335501")]  
[TestCase("-564508", "-277041158412", "-277041722920")]  
[TestCase("-961428656203", "-355352731734", "-1316781387937")]  
[TestCase("-1303996751177117", "-354670522247", "-1304351421699364")]  
[TestCase("-6982427152952915", "537132162176", "-6981890020790739")]  
[TestCase("114389357256", "-566272303396", "-451882946140")]  
[TestCase("2945541596428113", "-804204726037", "2944737391702076")]  
public void AdditionTest(string a, string b, string result) =>  
 OperationTest(a, b, result, BigInt.Add);  
  
[TestCase("680353957225", "156269694756", "524084262469")]  
[TestCase("-96257428842955", "504256139", "-96257933099094")]  
[TestCase("60450518", "-83758422519287", "83758482969805")]  
[TestCase("246841642", "99246340857053", "-99246094015411")]  
[TestCase("779310694956", "865409862226", "-86099167270")]  
[TestCase("765398045", "650773379169769", "-650772613771724")]  
public void SubtractionTest(string a, string b, string result) =>  
 OperationTest(a, b, result, BigInt.Sub);  
  
[TestCase("0", "0", "0")]  
[TestCase("12472948294", "0", "0")]  
[TestCase("9438429044", "1", "9438429044")]  
[TestCase("337435024824", "648458527447", "218812619306412930344328")]  
[TestCase("87515074", "-11746782888946", "-1028020573788042972004")]  
[TestCase("-416532428020962", "5242874658", "-2183827311106310762580996")]  
[TestCase("-0", "-0", "0")]  
[TestCase("-12472948294", "0", "0")]  
[TestCase("9438429044", "-1", "-9438429044")]  
[TestCase("-170009390880219", "-6279312672", "1067542122513160400835168")]  
[TestCase("-8452134940", "-35385773834565", "299085335406064616201100")]  
[TestCase("-86200735561435224414", "-34107843654934", "2940121211469742033175496268358676")]  
public void DefaultMultiplicationTest(string a, string b, string result) =>  
 OperationTest(a, b, result, BigInt.DefaultMultiplication);  
  
[TestCase("0", "0", "0")]  
[TestCase("12472948294", "0", "0")]  
[TestCase("9438429044", "1", "9438429044")]  
[TestCase("337435024824", "648458527447", "218812619306412930344328")]  
[TestCase("87515074", "-11746782888946", "-1028020573788042972004")]  
[TestCase("-416532428020962", "5242874658", "-2183827311106310762580996")]  
[TestCase("-0", "-0", "0")]  
[TestCase("-12472948294", "0", "0")]  
[TestCase("9438429044", "-1", "-9438429044")]  
[TestCase("-170009390880219", "-6279312672", "1067542122513160400835168")]  
[TestCase("-8452134940", "-35385773834565", "299085335406064616201100")]  
[TestCase("-86200735561435224414", "-34107843654934", "2940121211469742033175496268358676")]  
public void KaratsubaMultiplicationTest(string a, string b, string result) =>  
 OperationTest(a, b, result, BigInt.KaratsubaMultiplication);

[TestCase("-16645927926600509040026869", "883018282001805561", "-18851170")]  
 [TestCase("78722611001381133015934480", "-731822566392073670", "-107570625")]  
 [TestCase("87850140823630", "-965251350836769424", "0")]  
 [TestCase("793485891480702", "943855842", "840685")]  
 [TestCase("1234", "45", "27")]  
 [TestCase("-13547924586232047449092847", "-7069120881602", "1916493551764")]  
 public void DivisionTest(string a, string b, string result) =>  
 OperationTest(a, b, result, BigInt.Div);  
  
  
 [TestCase("235517282517573", "221150240", "17175973")]  
 [TestCase("361851169680067", "54702242139", "50540172721")]  
 [TestCase("348091531494017", "6362305761482863276", "348091531494017")]  
 public void ModTest(string a, string b, string result) =>  
 OperationTest(a, b, result, BigInt.Mod);  
  
 [TestCase("1830184925627198302978", "3921821", "824103")]  
 [TestCase("2408490402825726383127198302978", "4029048129472949", "1968106216971570")]  
 [TestCase("8376327546273912271983029", "27432", "13229")]  
 public void InvertingNumberTest(string number, string modulo, string result) =>  
 OperationTest(number, modulo, result, BigInt.Inverse);  
   
 [TestCase("531", "37", "67374767577675456090964712472670484305696898563133409832099875890530065117852060233269000630135939011")]  
 [TestCase("218", "94", "6529958163664865318517555711970764270146878908402790237793722905845018438374614250241672431880016418454326340956829724578560982344596628673088753944481901021050751991370621051952091577350959887262859055660023054728167424")]  
 [TestCase("849", "20", "37857663344776540371681891561864232691176600687505316008001")]  
 [TestCase("1209204729748", "0", "1")]  
 [TestCase("134890187496287389248", "1", "134890187496287389248")]  
 [TestCase("-8", "2", "64")]  
 [TestCase("8", "-2", "0")]  
 public void PowerTest(string a, string b, string result) =>  
 OperationTest(a, b, result, BigInt.Pow);

* 1. Тестирование RSA-шифрования:

[TestFixture]  
public class RsaTests  
{  
 private const int **P** = 7;  
 private const int **Q** = 11;  
  
 private static readonly PublicKey PublicKey = PublicKeyGenerator.Generate(**P**, **Q**);  
 private static readonly PrivateKey PrivateKey = PrivateKeyGenerator.Generate(PublicKey);  
  
 private static readonly TextCryptographer Cryptographer = new TextCryptographer();  
   
 [TestCase(31, 59)]  
 [TestCase(51, 72)]  
 [TestCase(14, 42)]  
 [TestCase(212, 9)]  
 [TestCase(343, 7)]  
 [TestCase(18, 39)]  
 [TestCase(73, 17)]  
 [TestCase(9, 37)]  
 public void NumberEncryptionTest(int number, int encrypted) =>   
 Assert.AreEqual((BigInt.BigInt) encrypted, RsaEncrypting.EncryptNumber(number, PublicKey));  
  
 [TestCase(18, 46)]  
 [TestCase(3, 38)]  
 [TestCase(330, 22)]  
 [TestCase(19, 61)]  
 [TestCase(343, 63)]  
 [TestCase(68, 19)]  
 [TestCase(46, 74)]  
 [TestCase(12, 12)]  
 public void NumberDecryptionTest(int encrypted, int original) =>  
 Assert.AreEqual(original, RsaEncrypting.DecryptNumber(encrypted, PrivateKey));  
  
 [TestCase("hello world", new []{61, 16, 44, 44, 75, 20, 55, 75, 8, 44, 64})]  
 [TestCase("The quick brown fox jumps over the lazy dog", new []{24, 61, 16, 20, 14, 4, 13, 7, 43, 20, 34, 8, 75, 55, 18, 20, 3, 75, 56, 20, 70, 4, 45, 27, 72, 20, 75, 54, 16, 8, 20, 24, 61, 16, 20, 44, 33, 9, 29, 20, 64, 75, 74})]  
 public void StringEncryptionTest(string toEncrypt, int[] result)  
 {  
 var encrypted = Cryptographer.Encrypt(toEncrypt, PublicKey);  
 Assert.AreEqual(result.Select(x => (BigInt.BigInt) x), encrypted.Result.ToArray());  
 }  
   
 [TestCase(new []{33, 34, 7, 64, 16, 3, 74, 61, 13, 43, 44, 45, 75, 27, 14, 8, 72, 24, 4, 54, 55, 56, 29, 9}, "abcdefghiklmopqrstuvwxyz")]  
 [TestCase(new []{61, 16, 44, 44, 75, 20, 3, 8, 75, 45, 20, 45, 33, 8, 72, 20, 74, 8, 16, 33, 24, 13, 18, 74, 72}, "hello from mars greatings")]  
 public void StringDecryptionTest(int[] encrypt, string result)  
 {  
 var decrypted = Cryptographer.Decrypt(encrypt.Select(x => (BigInt.BigInt) x).ToArray(), PrivateKey);  
 Assert.AreEqual(decrypted.Result, result);  
 }  
}