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

Институт радиоэлектроники и информационных технологий – РТФ

Алгоритм RSA-шифрования

Отчёт по лабораторной работе  
по дисциплине «Алгоритмы, структуры данных и анализ сложности»

Выполнил: студент группы, Аристов В.Е.

Преподаватель: доцент, к.ф.-м.н. Трофимов С.П.

2023

# Оглавление

[Оглавление 2](#_Toc136446213)

[Задание 3](#_Toc136446214)

[Теоретическая часть 4](#_Toc136446215)

[Инструкция пользователя 9](#_Toc136446216)

[Инструкция программиста 12](#_Toc136446217)

[Тестирование 13](#_Toc136446218)

[Выводы 15](#_Toc136446219)

[Литература 18](#_Toc136446220)

[Приложение 19](#_Toc136446221)

# **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.

# **Теоретическая часть**

Алгоритм RSA (Rivest-Shamir-Adleman) является одним из самых широко используемых асимметричных алгоритмов шифрования и основан на сложности задачи факторизации больших простых чисел.

Процесс шифрования и дешифрования в алгоритме RSA основывается на использовании пары ключей: открытого и закрытого. Открытый ключ используется для шифрования сообщений, в то время как закрытый ключ необходим для их дешифрования.

Вот основные шаги алгоритма RSA:

1. Генерация ключей:
   1. Выбор двух больших простых чисел p и q.
   2. Вычисление их произведения n = p \* q. Число n называется модулем.
   3. Вычисление функции Эйлера φ(n) = (p - 1) \* (q - 1).
   4. Выбор открытой экспоненты e, которая должна быть взаимно проста с φ(n) и меньше φ(n).
   5. Вычисление закрытой экспоненты d, которая является обратной к e по модулю φ(n). То есть d \* e ≡ 1 (mod φ(n)).
   6. Пара (e, n) является открытым ключом, а пара (d, n) - закрытым ключом.
2. Шифрование:
   1. Предположим, что у нас есть открытый ключ (e, n) и сообщение M, которое мы хотим зашифровать.
   2. Представление сообщения M в виде целого числа m, где 0 ≤ m < n.
   3. Вычисление зашифрованного сообщения C по формуле: C ≡ m^e (mod n).
3. Дешифрование:
   1. Предположим, что у нас есть закрытый ключ (d, n) и зашифрованное сообщение C, которое мы хотим дешифровать.
   2. Вычисление исходного сообщения m по формуле: m ≡ C^d (mod n).

Алгоритм RSA основывается на математической сложности факторизации больших чисел. Предположительно, чтобы восстановить закрытый ключ по открытому ключу, необходимо выполнить факторизацию числа n на простые множители p и q, что является вычислительно сложной задачей при использовании достаточно больших простых чисел.

RSA широко применяется для шифрования данных, подписи цифровых документов, обеспечения безопасности в сети, аутентификации и других задач, где требуется обмен информацией с конфиденциальностью и целостностью. Однако, важно учитывать, что безопасность RSA основана на сложности факторизации больших чисел, и с появлением квантовых компьютеров, способных решать эту задачу эффективно, может потребоваться переход к квантовоустойчивым алгоритмам шифрования.

Алгоритм RSA обладает несколькими особенностями, которые делают его уникальным и позволяют использовать его в широком спектре приложений:

1. Асимметричное шифрование: RSA является асимметричным алгоритмом шифрования, что означает, что для шифрования и дешифрования используются разные ключи. Открытый ключ используется для шифрования сообщений, в то время как закрытый ключ необходим для их дешифрования. Это обеспечивает простоту и безопасность обмена зашифрованными данными.
2. Безопасность основана на сложности факторизации: Основная безопасность RSA основана на математической сложности факторизации больших простых чисел. Чтобы взломать RSA, необходимо выполнить факторизацию числа n на простые множители p и q. Сложность этой задачи растет экспоненциально с увеличением размера простых чисел. Поэтому использование достаточно больших ключей делает атаку путем перебора неэффективной.
3. Эффективность шифрования небольших объемов данных: RSA хорошо подходит для шифрования небольших объемов данных, таких как ключи шифрования симметричных алгоритмов или хэш-значения. Он обычно не используется для шифрования больших объемов данных, так как процесс шифрования и дешифрования RSA является вычислительно сложным и требует большого объема вычислительных ресурсов.
4. Цифровая подпись: RSA также используется для создания и проверки цифровых подписей. Цифровая подпись позволяет аутентифицировать отправителя и гарантировать целостность сообщения. Отправитель создает подпись, применяя свой закрытый ключ к хэшу сообщения, а получатель проверяет подпись, используя открытый ключ отправителя.
5. Генерация ключей: Генерация безопасных ключей в RSA включает выбор больших простых чисел и вычисление связанных значений. Это процесс, требующий случайности и вычислительных ресурсов. Безопасность RSA напрямую зависит от генерации ключей, и неправильная реализация генерации ключей может привести к уязвимостям.

Несмотря на свои преимущества, RSA также имеет некоторые ограничения. Один из них - относительная медлительность процесса шифрования и дешифрования, особенно для больших объемов данных. Кроме того, с развитием квантовых компьютеров, RSA может стать уязвимым для атак, поскольку они могут эффективно факторизовывать большие числа. Поэтому сейчас проводятся исследования по разработке квантовоустойчивых алгоритмов шифрования для замены RSA в будущем.

Сложность RSA зависит от нескольких факторов, включая размер ключа, операции шифрования/дешифрования и операции генерации ключей. Основная вычислительная сложность RSA связана с факторизацией больших чисел.

1. Генерация ключей: Сложность генерации ключей RSA зависит от размера простых чисел, выбранных для создания ключей. Чем больше размер ключа (т.е. более длинные простые числа), тем больше времени и вычислительных ресурсов требуется для их генерации. Генерация ключей может занять значительное время, особенно для больших ключей.
2. Шифрование и дешифрование: Операции шифрования и дешифрования в RSA требуют выполнения операции возведения в степень по модулю. Сложность этих операций зависит от размера ключа (длины модуля). Обычно время, необходимое для шифрования или дешифрования, пропорционально квадрату длины модуля.
3. Факторизация: Основная вычислительная сложность RSA связана с факторизацией модуля n на простые множители p и q. Сложность факторизации увеличивается экспоненциально с ростом размера модуля (длины ключа). Чем больше размер ключа, тем сложнее выполнить факторизацию и взломать RSA.

Использование достаточно больших ключей является важным для обеспечения безопасности RSA. В настоящее время рекомендуется использовать ключи длиной не менее 2048 бит для надежной защиты данных. Однако с развитием квантовых компьютеров, способных эффективно факторизовывать большие числа, размеры рекомендуемых ключей могут увеличиться в будущем.

Алгоритм RSA не имеет линейной сложности O(n), где n - размер входных данных. Сложность RSA зависит от различных факторов, таких как размер ключа, операции шифрования/дешифрования и факторизации.

Генерация ключей RSA имеет сложность, которая зависит от размера простых чисел, выбранных для создания ключей. Эта операция не является линейной по размеру входных данных.

Операции шифрования и дешифрования в RSA, в основном, зависят от длины модуля (размера ключа). Обычно время, необходимое для выполнения операций шифрования и дешифрования, пропорционально квадрату длины модуля. Это означает, что сложность RSA не является линейной по размеру входных данных.

Самая вычислительно сложная операция в RSA - факторизация модуля на простые множители p и q. Сложность факторизации увеличивается экспоненциально с ростом размера модуля (длины ключа). Факторизация не имеет линейной сложности и является одной из основных вычислительных сложностей RSA

Итак, сложность алгоритма RSA не может быть оценена как O(n), так как она зависит от нескольких факторов и не является линейной по размеру входных данных.

# **Инструкция пользователя**

При запуске Bytes (файла отвечающего за 1 задание) открывается консоль, где пользователь может провести тесты или отказаться от этого.

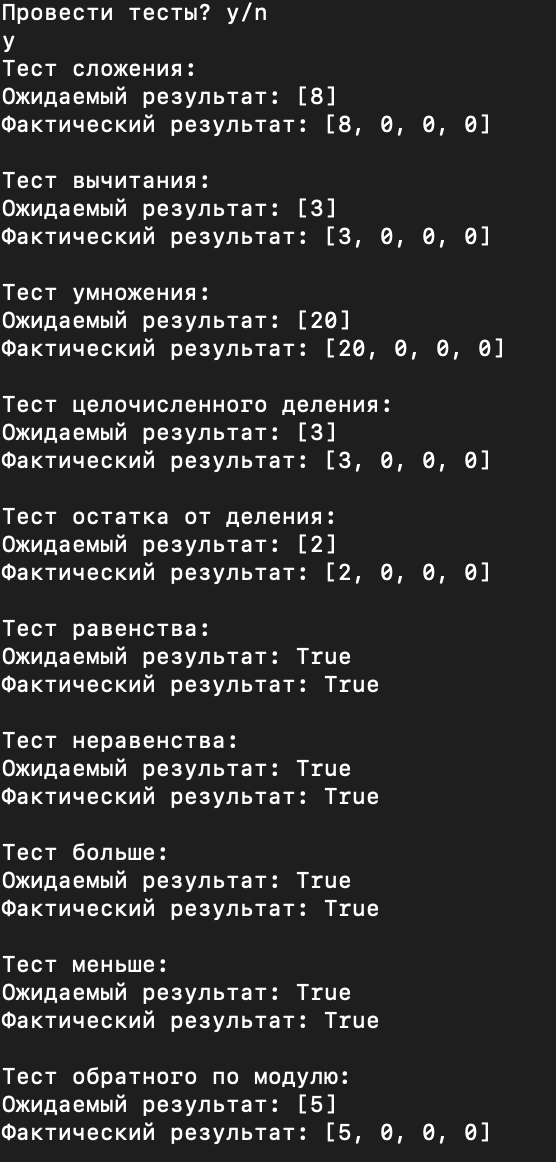


Рисунок 1 – Проведение тестов

Далее, независимо от того, производилось тестирование или нет, пользователю предлагается ввести цифры и провести операции над ними. Увидеть результаты в виде обычных цифр и массиве однобайтовых элементов.

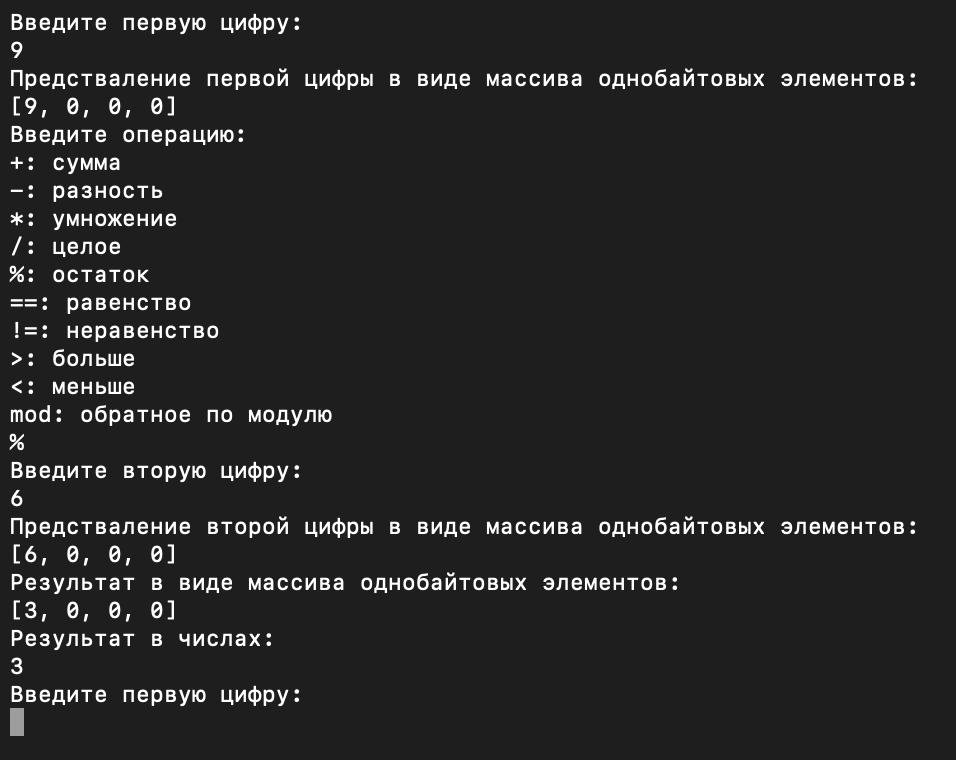
****

Рисунок 2 – Проведение операций с массивами однобайтовых элементов

При запуске RSA(главного файла программы) у пользователя спросят путь к файлам .txt.

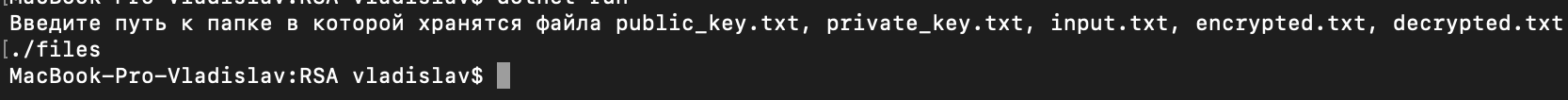


Рисунок 3 – Запуск RSA

Затем файл input.txt будет зашифрован с помощью public\_key.txt и private\_key.txt и сохранен в encrypted.txt, после чего расшифрован в decrypted.txt. После чего пользователю будет предложено пройти проверку отработанного алгоритма.

# **Инструкция программиста**

Программа написана на языке C#. В проекте находится 2 папки – Bytes и RSA для первого и второго задания соответственно.

В папке Bytes находится 3 файла: Program.cs(главный файл программы), SignedByteArray.cs(требуемое класс) и Tests.cs(тестирование).

Компиляция происходит с помощью .NET и команды:

**dotnet build**

В папке RSA находится 1 файл: Program.cs(главный файл программы). В файле Program.cs находится требуемый класс RSAEncryption и главный класс Program в котором находится взаимодействия с пользователем и тестирование.

Компиляция происходит с помощью .NET и команды:

**dotnet build**

# **Тестирование**

При запуске Bytes (файла отвечающего за 1 задание) открывается консоль, где пользователь может провести тесты или отказаться от этого.

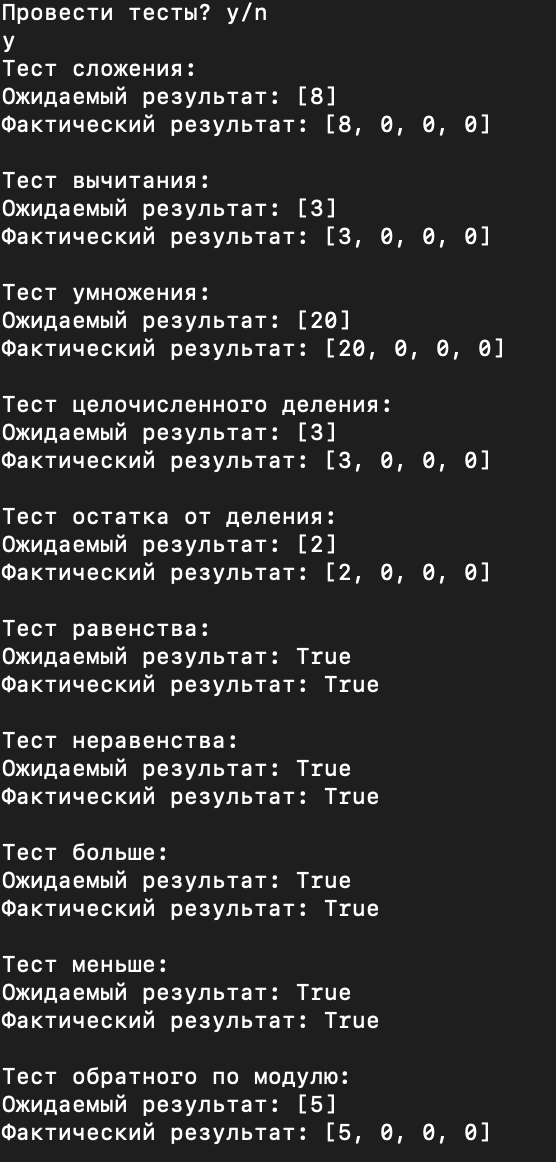


Рисунок 1 – Проведение тестов

При необходимости тесты можно редактировать самостоятельно в файле Tests.cs.

В программе RSA, после того как основная программа отработает, пользователю будет предложено провести тест.A black screen with white text

Description automatically generated with low confidence

Рисунок 4 – тестирование RSA

В случае если оригинальная строка сходится с расшифрованной можно констатировать, что алгоритм отработал корректно, о чем выводится сообщение в консоли. В противном случае выведется "Тест не пройден".

# **Выводы**

В данной работе мы познакомились с алгоритмом RSA (Rivest-Shamir-Adleman) - это криптографический алгоритм, который широко используется для шифрования и подписи данных.

Сложность алгоритма RSA заключается в том, что факторизация больших чисел является вычислительно сложной задачей. Для безопасности RSA используются очень большие числа (обычно несколько сотен бит), которые являются произведением двух простых чисел. Расшифрование или взлом RSA-зашифрованных данных требует факторизации этих больших чисел, что считается вычислительно сложной задачей, особенно при использовании достаточно больших ключей.

Сложность факторизации числа возрастает экспоненциально с увеличением размера числа, что делает атаки на RSA с использованием существующих алгоритмов факторизации практически невозможными для больших достаточно ключей. Поэтому RSA считается безопасным алгоритмом шифрования, при условии использования достаточно больших ключей.

У RSA есть несколько преимуществ и недостатков по сравнению с другими алгоритмами шифрования:

Плюсы RSA:

1. Безопасность: RSA считается одним из самых надежных алгоритмов шифрования. Его безопасность основана на сложности факторизации больших чисел.
2. Поддержка цифровых подписей: RSA может использоваться для создания и проверки цифровых подписей, что позволяет обеспечить подлинность и целостность данных.
3. Распределение открытых ключей: RSA позволяет распространять открытые ключи без необходимости обеспечения конфиденциальности. Это делает алгоритм удобным для реализации криптографических протоколов.
4. Вычислительная эффективность при шифровании небольших объемов данных: RSA эффективно работает с небольшими объемами данных, такими как ключи шифрования или хэши, поскольку операции шифрования и расшифрования могут быть быстро выполнены.

Минусы RSA:

1. Вычислительная сложность при шифровании и расшифровании больших объемов данных: RSA является относительно медленным алгоритмом для шифрования и расшифрования больших объемов данных. Поэтому обычно вместо RSA используются симметричные алгоритмы шифрования для шифрования фактических сообщений, а RSA используется только для зашифрования ключей симметричного шифрования.
2. Длина ключа: Для обеспечения достаточной безопасности, длина ключа RSA должна быть большой, что может занимать больше места и требовать больших вычислительных ресурсов при выполнении операций шифрования и расшифрования.
3. Отсутствие прямой аутентификации: RSA не обеспечивает прямой аутентификации, то есть не позволяет проверить подлинность отправителя сообщения. Для этой цели требуется дополнительное использование цифровых подписей.
4. Уязвимость к атакам с использованием квантовых компьютеров: RSA, основанный на сложности факторизации больших чисел, может быть взломан с помощью квантовых компьютеров с достаточной мощностью. Однако на данный момент квантовые компьютеры с такой мощностью еще не разработаны.
5. Выбор между RSA и другими алгоритмами зависит от конкретных требований безопасности, эффективности и функциональности системы, а также от ожидаемой нагрузки и размера передаваемых данных.

# Литература

1. RSA: от простых чисел до электронной подписи [Электронный ресурс]: URL: <https://habr.com/ru/articles/534014/>
2. RSA - Викиконспекты [Электронный ресурс]: URL: <https://neerc.ifmo.ru/wiki/index.php?title=RSA>
3. Шнайер Б. Прикладная криптография. Протоколы, алгоритмы, исходные тексты на языке Си = Applied Cryptography. Protocols, Algorithms and Source Code in C. — М.: Триумф, 2002. — 816 с.
4. Нильс Фергюсон, Брюс Шнайер. Практическая криптография = Practical Cryptography: Designing and Implementing Secure Cryptographic Systems. — М. : Диалектика, 2004. — 432 с.
5. RSA - Википедия [Электронный ресурс]: URL: <https://ru.wikipedia.org/wiki/RSA>

# **Приложение**

using System;

public class Program

{

public static void Main(string[] args)

{

Console.WriteLine("Провести тесты? y/n");

string answer = Console.ReadLine();

if (answer == "y" || answer == "yes")

{

Tests.Run();

}

while (true) {

Console.WriteLine("Введите первую цифру:");

SignedByteArray a = new SignedByteArray(Convert.ToInt32(Console.ReadLine()));

Console.WriteLine("Предстваление первой цифры в виде массива однобайтовых элементов:");

a.PrintArray();

Console.WriteLine("Введите операцию:");

Console.WriteLine("+: сумма");

Console.WriteLine("-: разность");

Console.WriteLine("\*: умножение");

Console.WriteLine("/: целое");

Console.WriteLine("%: остаток");

Console.WriteLine("==: равенство");

Console.WriteLine("!=: неравенство");

Console.WriteLine(">: больше");

Console.WriteLine("<: меньше");

Console.WriteLine("mod: обратное по модулю");

string oper = Console.ReadLine();

Console.WriteLine("Введите вторую цифру:");

SignedByteArray b = new SignedByteArray(Convert.ToInt32(Console.ReadLine()));

Console.WriteLine("Предстваление второй цифры в виде массива однобайтовых элементов:");

b.PrintArray();

SignedByteArray result;

bool resultBool;

switch (oper)

{

case "+":

result = a + b;

Console.WriteLine("Результат в виде массива однобайтовых элементов:");

result.PrintArray();

Console.WriteLine("Результат в числах:");

Console.WriteLine(result.ToInt());

break;

case "-":

result = a - b;

Console.WriteLine("Результат в виде массива однобайтовых элементов:");

result.PrintArray();

Console.WriteLine("Результат в числах:");

Console.WriteLine(result.ToInt());

break;

case "\*":

result = a \* b;

Console.WriteLine("Результат в виде массива однобайтовых элементов:");

result.PrintArray();

Console.WriteLine("Результат в числах:");

Console.WriteLine(result.ToInt());

break;

case "/":

result = a / b;

Console.WriteLine("Результат в виде массива однобайтовых элементов:");

result.PrintArray();

Console.WriteLine("Результат в числах:");

Console.WriteLine(result.ToInt());

break;

case "%":

result = a % b;

Console.WriteLine("Результат в виде массива однобайтовых элементов:");

result.PrintArray();

Console.WriteLine("Результат в числах:");

Console.WriteLine(result.ToInt());

break;

case "mod":

result = a.ModInverse(b.ToInt());

Console.WriteLine("Результат в виде массива однобайтовых элементов:");

result.PrintArray();

Console.WriteLine("Результат в числах:");

Console.WriteLine(result.ToInt());

break;

case "==":

resultBool = (a == b);

Console.WriteLine("Результат:");

Console.WriteLine(resultBool);

break;

case "!=":

resultBool = (a != b);

Console.WriteLine("Результат:");

Console.WriteLine(resultBool);

break;

case ">":

resultBool = (a > b);

Console.WriteLine("Результат:");

Console.WriteLine(resultBool);

break;

case "<":

resultBool = (a < b);

Console.WriteLine("Результат:");

Console.WriteLine(resultBool);

break;

default:

Console.WriteLine("Неправильный оператор");

continue;

}

}

}

}

using System;

public class SignedByteArray

{

private byte[] value;

// Конструкторы

public SignedByteArray()

{

value = new byte[1];

}

public SignedByteArray(int number)

{

value = BitConverter.GetBytes(number);

}

public SignedByteArray(byte[] bytes)

{

value = bytes;

}

// Деструктор

~SignedByteArray()

{

// Освобождение ресурсов

}

// Перегрузка операторов аддитивных операций (+, -)

public static SignedByteArray operator +(SignedByteArray a, SignedByteArray b)

{

int sum = a.ToInt() + b.ToInt();

return new SignedByteArray(sum);

}

public static SignedByteArray operator -(SignedByteArray a, SignedByteArray b)

{

int difference = a.ToInt() - b.ToInt();

return new SignedByteArray(difference);

}

// Перегрузка операторов мультипликативных операций (\*, /, %)

public static SignedByteArray operator \*(SignedByteArray a, SignedByteArray b)

{

int product = a.ToInt() \* b.ToInt();

return new SignedByteArray(product);

}

public static SignedByteArray operator /(SignedByteArray a, SignedByteArray b)

{

int quotient = a.ToInt() / b.ToInt();

return new SignedByteArray(quotient);

}

public static SignedByteArray operator %(SignedByteArray a, SignedByteArray b)

{

int remainder = a.ToInt() % b.ToInt();

return new SignedByteArray(remainder);

}

// Перегрузка операторов сравнения (==, !=, <, >)

public static bool operator ==(SignedByteArray a, SignedByteArray b)

{

return a.ToInt() == b.ToInt();

}

public static bool operator !=(SignedByteArray a, SignedByteArray b)

{

return a.ToInt() != b.ToInt();

}

public static bool operator <(SignedByteArray a, SignedByteArray b)

{

return a.ToInt() < b.ToInt();

}

public static bool operator >(SignedByteArray a, SignedByteArray b)

{

return a.ToInt() > b.ToInt();

}

// Взятие обратного по заданному модулю

public SignedByteArray ModInverse(int modulus)

{

int valueInt = ToInt();

int modInverse = CalculateModInverse(valueInt, modulus);

return new SignedByteArray(modInverse);

}

// Вспомогательный метод для вычисления обратного по модулю

private int CalculateModInverse(int value, int modulus)

{

int m0 = modulus;

int y = 0, x = 1;

while (value > 1)

{

int quotient = value / modulus;

int temp = modulus;

modulus = value % modulus;

value = temp;

temp = y;

y = x - quotient \* y;

x = temp;

}

if (x < 0)

x += m0;

return x;

}

// Метод преобразования к int

public int ToInt()

{

return BitConverter.ToInt32(value, 0);

}

public override bool Equals(object obj)

{

if (obj == null || GetType() != obj.GetType())

{

return false;

}

SignedByteArray other = (SignedByteArray)obj;

return this.Equals(other);

}

public bool Equals(SignedByteArray other)

{

if (other == null)

{

return false;

}

if (this.value.Length != other.value.Length)

{

return false;

}

for (int i = 0; i < this.value.Length; i++)

{

if (this.value[i] != other.value[i])

{

return false;

}

}

return true;

}

public override int GetHashCode()

{

unchecked

{

int hash = 17;

foreach (byte b in this.value)

{

hash = hash \* 23 + b.GetHashCode();

}

return hash;

}

}

public byte[] GetArray()

{

return this.value;

}

public void SetArray(byte[] newArray)

{

this.value = newArray;

}

public void PrintArray()

{

Console.Write("[");

for (int i = 0; i < value.Length; i++)

{

Console.Write(value[i]);

if (i < value.Length - 1)

{

Console.Write(", ");

}

}

Console.WriteLine("]");

}

}

using System;

public class Tests

{

public static void TestAddition()

{

Console.WriteLine("Тест сложения:");

SignedByteArray a = new SignedByteArray(5);

SignedByteArray b = new SignedByteArray(3);

SignedByteArray result = a + b;

Console.WriteLine("Ожидаемый результат: [8]");

Console.Write("Фактический результат: ");

result.PrintArray();

Console.WriteLine();

}

public static void TestSubtraction()

{

Console.WriteLine("Тест вычитания:");

SignedByteArray a = new SignedByteArray(10);

SignedByteArray b = new SignedByteArray(7);

SignedByteArray result = a - b;

Console.WriteLine("Ожидаемый результат: [3]");

Console.Write("Фактический результат: ");

result.PrintArray();

Console.WriteLine();

}

public static void TestMultiplication()

{

Console.WriteLine("Тест умножения:");

SignedByteArray a = new SignedByteArray(4);

SignedByteArray b = new SignedByteArray(5);

SignedByteArray result = a \* b;

Console.WriteLine("Ожидаемый результат: [20]");

Console.Write("Фактический результат: ");

result.PrintArray();

Console.WriteLine();

}

public static void TestDivision()

{

Console.WriteLine("Тест целочисленного деления:");

SignedByteArray a = new SignedByteArray(15);

SignedByteArray b = new SignedByteArray(4);

SignedByteArray result = a / b;

Console.WriteLine("Ожидаемый результат: [3]");

Console.Write("Фактический результат: ");

result.PrintArray();

Console.WriteLine();

}

public static void TestModulus()

{

Console.WriteLine("Тест остатка от деления:");

SignedByteArray a = new SignedByteArray(17);

SignedByteArray b = new SignedByteArray(5);

SignedByteArray result = a % b;

Console.WriteLine("Ожидаемый результат: [2]");

Console.Write("Фактический результат: ");

result.PrintArray();

Console.WriteLine();

}

public static void TestEquality()

{

Console.WriteLine("Тест равенства:");

SignedByteArray a = new SignedByteArray(5);

SignedByteArray b = new SignedByteArray(5);

bool result = a == b;

Console.WriteLine("Ожидаемый результат: True");

Console.WriteLine("Фактический результат: " + result);

Console.WriteLine();

}

public static void TestInequality()

{

Console.WriteLine("Тест неравенства:");

SignedByteArray a = new SignedByteArray(5);

SignedByteArray b = new SignedByteArray(7);

bool result = a != b;

Console.WriteLine("Ожидаемый результат: True");

Console.WriteLine("Фактический результат: " + result);

Console.WriteLine();

}

public static void TestGreaterThan()

{

Console.WriteLine("Тест больше:");

SignedByteArray a = new SignedByteArray(8);

SignedByteArray b = new SignedByteArray(5);

bool result = a > b;

Console.WriteLine("Ожидаемый результат: True");

Console.WriteLine("Фактический результат: " + result);

Console.WriteLine();

}

public static void TestLessThan()

{

Console.WriteLine("Тест меньше:");

SignedByteArray a = new SignedByteArray(3);

SignedByteArray b = new SignedByteArray(5);

bool result = a < b;

Console.WriteLine("Ожидаемый результат: True");

Console.WriteLine("Фактический результат: " + result);

Console.WriteLine();

}

public static void TestModInverse()

{

Console.WriteLine("Тест обратного по модулю:");

SignedByteArray a = new SignedByteArray(3);

SignedByteArray b = new SignedByteArray(7);

SignedByteArray result = a.ModInverse(b.ToInt());

Console.WriteLine("Ожидаемый результат: [5]");

Console.Write("Фактический результат: ");

result.PrintArray();

Console.WriteLine();

}

public static void Run()

{

TestAddition();

TestSubtraction();

TestMultiplication();

TestDivision();

TestModulus();

TestEquality();

TestInequality();

TestGreaterThan();

TestLessThan();

TestModInverse();

}

}

using System;

using System.IO;

using System.Security.Cryptography;

using System.Text;

public class Program

{

public static void Main(string[] args)

{

// Генерируем RSA-ключи

using (RSACryptoServiceProvider rsa = new RSACryptoServiceProvider())

{

Console.WriteLine("Введите путь к папке в которой хранятся файла public\_key.txt, private\_key.txt, input.txt, encrypted.txt, decrypted.txt");

string path = Console.ReadLine();

RSAParameters publicKey = rsa.ExportParameters(false);

RSAParameters privateKey = rsa.ExportParameters(true);

// Записываем открытый ключ в файл

string publicKeyFile = path + "/public\_key.txt";

string publicKeyText = (publicKey.Modulus != null ? Convert.ToBase64String(publicKey.Modulus) : string.Empty) + ";" +

(publicKey.Exponent != null ? Convert.ToBase64String(publicKey.Exponent) : string.Empty);

File.WriteAllText(publicKeyFile, publicKeyText);

// Записываем закрытый ключ в файл

string privateKeyFile = path + "/private\_key.txt";

string privateKeyText = (privateKey.Modulus != null ? Convert.ToBase64String(privateKey.Modulus) : string.Empty) + ";" +

(privateKey.Exponent != null ? Convert.ToBase64String(privateKey.Exponent) : string.Empty) + ";" +

(privateKey.D != null ? Convert.ToBase64String(privateKey.D) : string.Empty) + ";" +

(privateKey.P != null ? Convert.ToBase64String(privateKey.P) : string.Empty) + ";" +

(privateKey.Q != null ? Convert.ToBase64String(privateKey.Q) : string.Empty) + ";" +

(privateKey.DP != null ? Convert.ToBase64String(privateKey.DP) : string.Empty) + ";" +

(privateKey.DQ != null ? Convert.ToBase64String(privateKey.DQ) : string.Empty) + ";" +

(privateKey.InverseQ != null ? Convert.ToBase64String(privateKey.InverseQ) : string.Empty);

File.WriteAllText(privateKeyFile, privateKeyText);

// Шифруем файл с помощью открытого ключа

string inputFile = path + "/input.txt";

string encryptedFile = path + "/encrypted.txt";

RSAEncryption.EncryptFile(inputFile, encryptedFile, publicKey);

// Расшифровываем файл с помощью закрытого ключа

string decryptedFile = path + "/decrypted.txt";

RSAEncryption.DecryptFile(encryptedFile, decryptedFile, privateKey);

Console.WriteLine("Провести тест? y/n");

if (Console.ReadLine() == "y") {

string origin = File.ReadAllText(inputFile);

string decrypted = File.ReadAllText(decryptedFile);

Console.WriteLine("Оригинальный файл: {0}", origin);

Console.WriteLine("Расшифрованный файл: {0}", decrypted);

if (origin == decrypted) {

Console.WriteLine("Тест успешно пройден");

} else {

Console.WriteLine("Тест не пройден");

}

}

}

}

}

using System;

using System.IO;

using System.Security.Cryptography;

using System.Text;

public class RSAEncryption

{

public static string Encrypt(string plaintext, RSAParameters publicKey)

{

using (RSACryptoServiceProvider rsa = new RSACryptoServiceProvider())

{

rsa.ImportParameters(publicKey);

byte[] data = Encoding.UTF8.GetBytes(plaintext);

byte[] encryptedData = rsa.Encrypt(data, true);

return Convert.ToBase64String(encryptedData);

}

}

public static string Decrypt(string ciphertext, RSAParameters privateKey)

{

using (RSACryptoServiceProvider rsa = new RSACryptoServiceProvider())

{

rsa.ImportParameters(privateKey);

byte[] encryptedData = Convert.FromBase64String(ciphertext);

byte[] decryptedData = rsa.Decrypt(encryptedData, true);

return Encoding.UTF8.GetString(decryptedData);

}

}

public static void EncryptFile(string inputFile, string outputFile, RSAParameters publicKey)

{

string plaintext = File.ReadAllText(inputFile);

string encryptedText = Encrypt(plaintext, publicKey);

File.WriteAllText(outputFile, encryptedText);

}

public static void DecryptFile(string inputFile, string outputFile, RSAParameters privateKey)

{

string encryptedText = File.ReadAllText(inputFile);

string decryptedText = Decrypt(encryptedText, privateKey);

File.WriteAllText(outputFile, decryptedText);

}

}