МИНОБРНАУКИ РФ

ФГБОУ Тверской государственный технический университет

Кафедра «Программного Обеспечения»

Отчет

по дисциплине «Объектно-ориентированное программирование»

на тему: «Методы шифрования и их реализация»

Работу выполнили: студенты группы

Б.ПИН.РИС-22.06

Иванов Алексей, Соболев Роман, Аванесян Григорий,

Борздов Марк, Мельников Михаил, Куликов Матвей

Работу проверил: преподаватель

Биллиг Владимир Арнольдович

Тверь 2023

**Оглавление**

[**Часть 1. Теоретическая часть** 3](#_Toc147174508)

[**Шифрование методом Цезаря** 3](#_Toc147174509)

[**Шифрование методом перестановок** 4](#_Toc147174510)

[**Ассиметричное RSA-шифрование** 5](#_Toc147174511)

[**Часть 2. Практическая часть** 11](#_Toc147174512)

[**Структура решения** 11](#_Toc147174513)

[**Интерфейсный проект** 11](#_Toc147174514)

[**Тестирующий проект** 12](#_Toc147174515)

[**Библиотека классов** 13](#_Toc147174516)

[**Демонстрация работы** 36](#_Toc147174517)

[**Заключение** 37](#_Toc147174518)

[**Использованные ресурсы** 39](#_Toc147174522)

[**Приложение 1. Диалог с chatGPT** 40](#_Toc147174523)

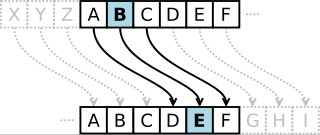
[**Приложение 2. Доказательство долготы конкатенации строк** 46](#_Toc147174524)

**Часть 1. Теоретическая часть**

Прежде, чем перейти к реализации и рассмотрению кода и функциональности алгоритмов, необходимо рассмотреть теоретические основы и доказательства работы используемых в проекте методов.

## **Шифрование методом Цезаря**

Метод Цезаря, так же известный как шифр Цезаря -это один из самых простых методов шифрования, который основан на замене каждой буквы в открытом тексте другой буквой через сдвиг в алфавите. В данном методе используется ключ - число, которое определяет величину сдвига. Например, если ключ равен 4, то каждая буква заменяется на четвертую букву после нее в алфавите.



Для дешифровки текста, зашифрованного методом Цезаря, необходимо знать значение ключа и выполнить обратный сдвиг на такую же величину.

Шифр Цезаря может быть легко взломан даже в случае, когда взломщик знает только зашифрованный текст. Можно рассмотреть две ситуации:

1. Взломщик знает (или предполагает), что использовался простой шифр подстановки, но не знает, что это — схема Цезаря.
2. Взломщик знает, что использовался шифр Цезаря, но не знает значение сдвига.

В первом случае шифр может быть взломан, используя те же самые методы что и для простого шифра подстановки, такие как частотный анализ и т. д. Используя эти методы, взломщик, вероятно, быстро заметит регулярность в решении и поймёт, что используемый шифр — это шифр Цезаря.

Во втором случае взлом шифра является даже более простым. Существует не так много вариантов значений сдвига (26 для английского языка), все они могут быть проверены методом грубой силы.

Шифр Цезаря достаточно понятный в усвоении, поэтому каждый может использовать его каких-либо целях. Однако надо учитывать, что это достаточно легкий шифр и его легко взломать, так что не рекомендуется применять его с целью защиты важной информацией от посторонних лиц.

## **Шифрование методом перестановок**

Процесс шифрования с использованием шифра перестановок включает следующие шаги:

1. Генерация ключа: ключом является перестановка чисел или символов. Например, ключ "3142" означает, что символы будут переставлены в таком порядке: первый символ станет третьим, второй - первым, третий - четвертым и четвертый - вторым.

2. Разбиение текста: исходный текст разбивается на блоки фиксированной длины. Например, если блоки имеют размер 4 символа, исходный текст "Привет, мир!" будет разделен на два блока: "Прив" и "ет, ми".

3. Шифрование: каждый блок символов переставляется в соответствии с ключом. Например, если ключ "3142", то первый блок станет "ивеП", а второй - "те,ирм".

4. Объединение блоков: зашифрованные блоки объединяются вместе для получения зашифрованного текста. Таким образом, в данном примере зашифрованный текст станет "ивеПте,ирм".

Процесс дешифрования с использованием шифра перестановок аналогичен, но ключ используется для обратной перестановки символов в зашифрованном тексте. То есть, если ключом при шифровании была перестановка "3142", то при дешифровании используется ключ "2413", чтобы вернуть символы в исходный порядок.

Шифр перестановок прост в реализации и может быть эффективен при использовании блоков большой длины. Однако, он не обеспечивает высокой степени безопасности, так как его структура легко может быть обнаружена и подвергнута атакам, особенно если длина блоков мала. Поэтому этот метод шифрования обычно применяется в сочетании с другими более сложными алгоритмами для повышения безопасности.

## **Ассиметричное RSA-шифрование**

Метод RSA - это асимметричный криптографический алгоритм, который использует пару ключей: открытый и закрытый. Открытый ключ используется для шифрования сообщений, а закрытый ключ - для их расшифровки. Он основан на сложности факторизации больших чисел.

В наше время он широко используется. Например, звоня через популярный мессенджер или вводя данные своей банковской карточки в онлайн маркетплейсе, перед обменом видеоизображением или данными банковской карты, происходит процедура проверки подлинности и обмена ключом шифрования по алгоритму RSA.

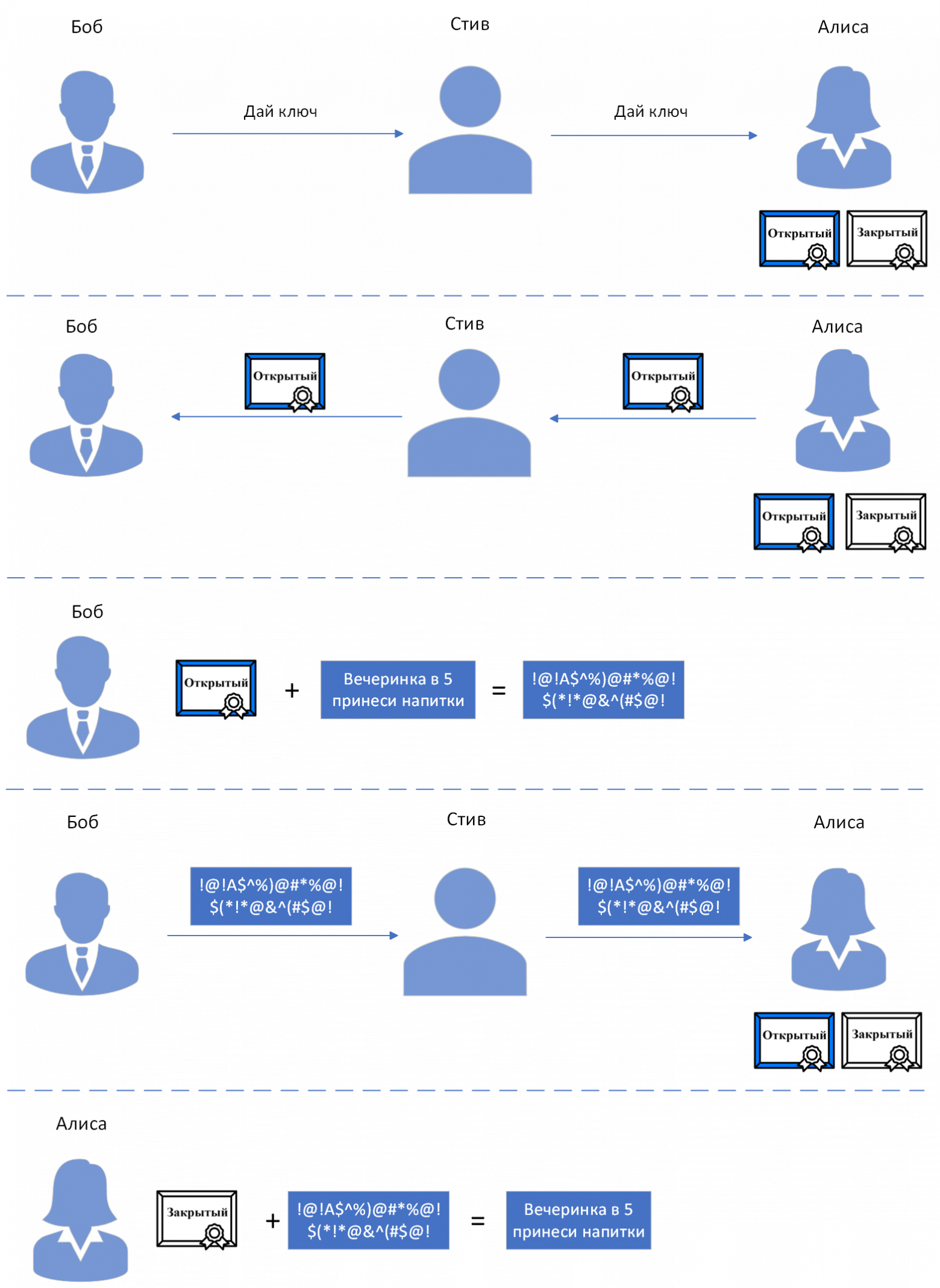
Процесс шифрования RSA состоит из следующих шагов:

1. Генерация ключей: выбор двух больших простых чисел и вычисление других значений.

2. Шифрование сообщения: каждая буква текста заменяется числом, после чего применяются операции по формуле шифрования.

3. Дешифрование сообщения: зашифрованные числа подвергаются операциям по формуле дешифрования, после чего происходит замена чисел на буквы.

Наглядно, данный алгоритм можно объяснить таким образом:

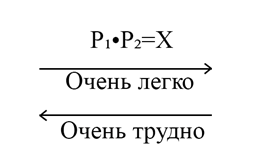


Возникает закономерный вопрос, почему Стив не может подменить ключ Алисы на свой, расшифровать сообщение, а потом, подглядев его, зашифровать обратно на ключ Алисы? Он может это сделать, это называется атака «человек посередине». Сейчас эту проблему решают «Цепочкой доверия», если коротко, то это возможность закрытого ключа — подпись информации.

С базовыми аспектами RSA алгоритма разобрались, теперь давайте заглянем «под капот» и посмотрим, как работает эта магия.

Вся ассиметричная криптография держится на принципе «в одну сторону быстро, в другую неразумно долго».

Например, если мы перемножим числа 592939 и 592967 мы получим число 351593260013. Но как имея только число 351593260013 узнать числа 592939 и 592967? А если каждое из этих двух чисел будут длиной более 1000 знаков? Это называется «сложность задачи факторизации произведения двух больших простых чисел», т.е. в одну сторону просто, а в обратную невероятно сложно.



Теперь рассмотрим процедуру создания публичного и приватного ключей:

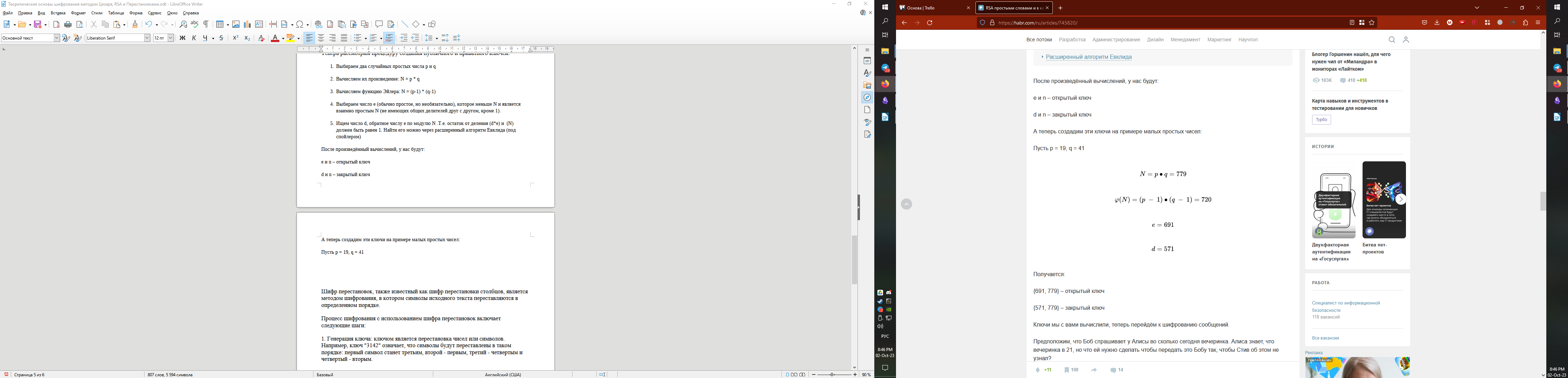
1. Выбираем два случайных простых числа p и q
2. Вычисляем их произведение: N = p \* q
3. Вычисляем функцию Эйлера: N = (p-1) \* (q-1)
4. Выбираем число e (обычно простое, но необязательно), которое меньше N и является взаимно простым N (не имеющих общих делителей друг с другом, кроме 1).
5. Ищем число d, обратное числу e по модулю N .Т.е. остаток от деления (d\*e) и (N) должен быть равен 1. Найти его можно через расширенный алгоритм Евклида (под спойлером)

После произведённых вычислений, у нас будут:

e и n – открытый ключ

d и n – закрытый ключ

А теперь создадим эти ключи на примере малых простых чисел:

Пусть p = 19, q = 41

Получается:

{691, 779} – открытый ключ

{571, 779} – закрытый ключ

**Часть 2. Практическая часть**

Теперь перейдем к нашей программной реализации методов.

## **Структура решения**

Но прежде необходимо рассмотреть общую структуру решения.

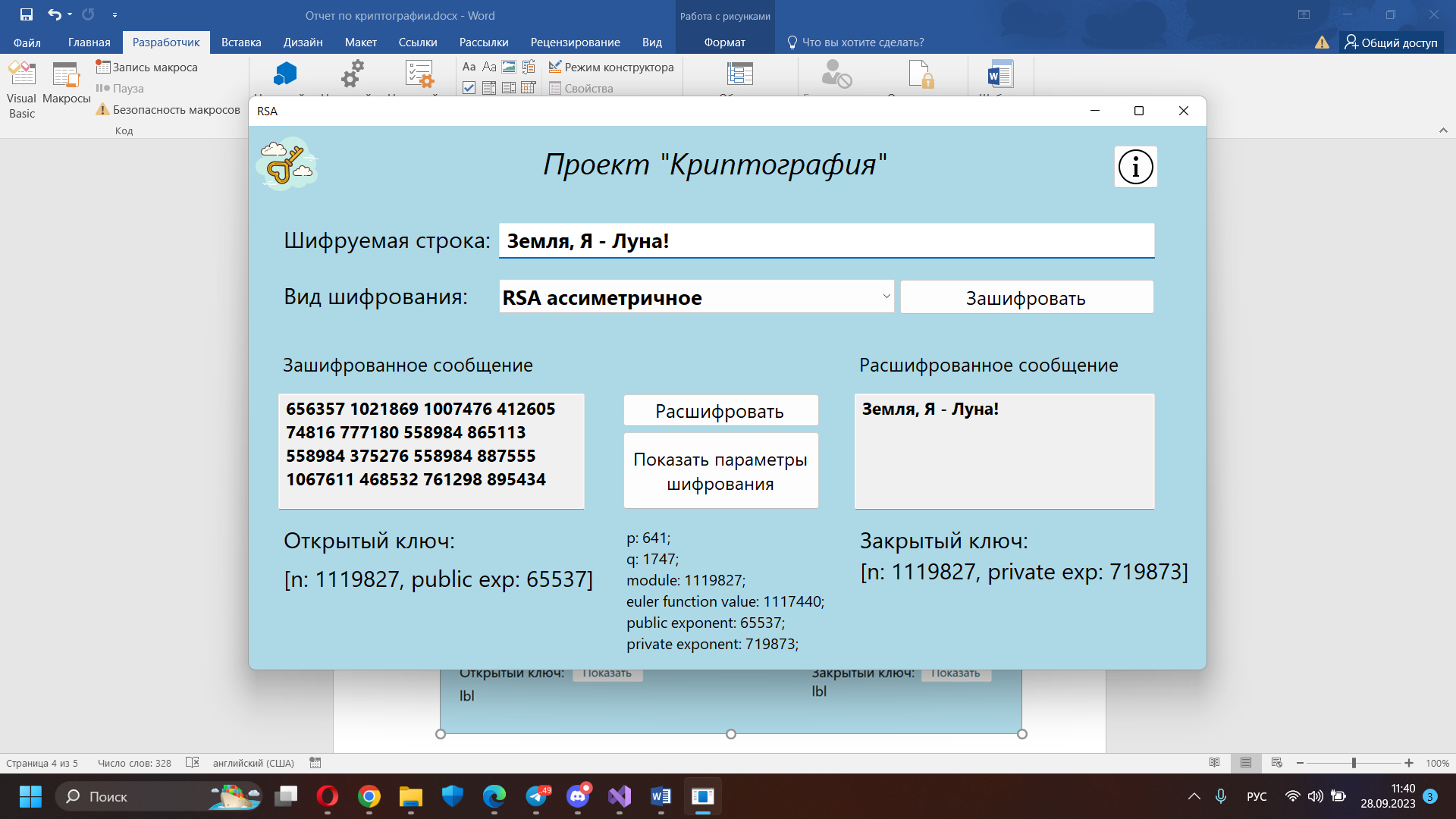
Проект, как и остальные работы, выполненные нашей командой, состоит из 3-ех подпроектов: интерфейсная часть, реализованная посредством Windows Forms, тестирующая часть, состоящая из консольного теста и unit-тестирующего проекта, а также обширная библиотека классов.

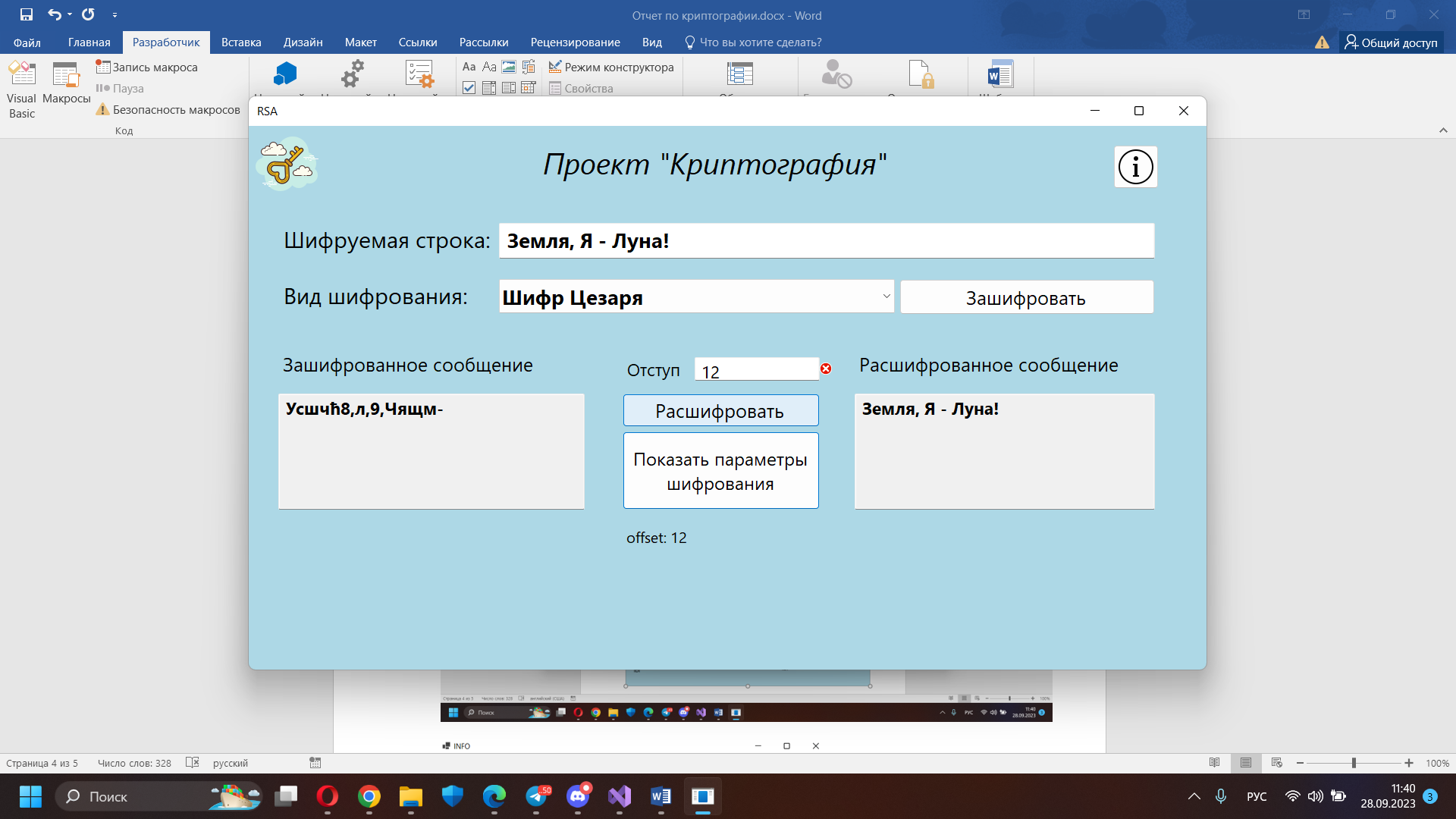
## **Интерфейсный проект**

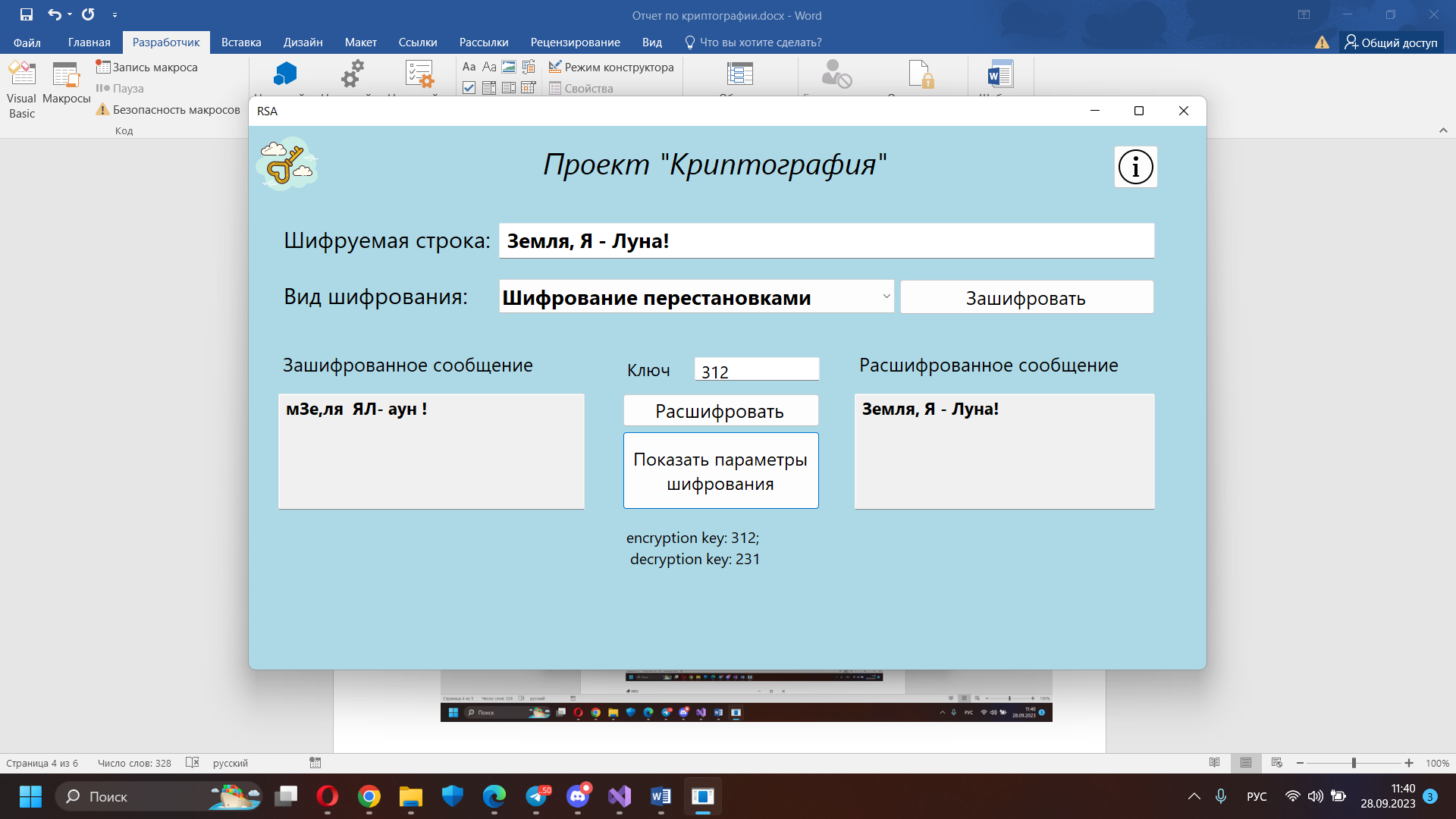
Интерфейсный проект включает в себя 2 формы: основную и информационную, которая содержит текстовую инструкцию к приложению для пользователя.

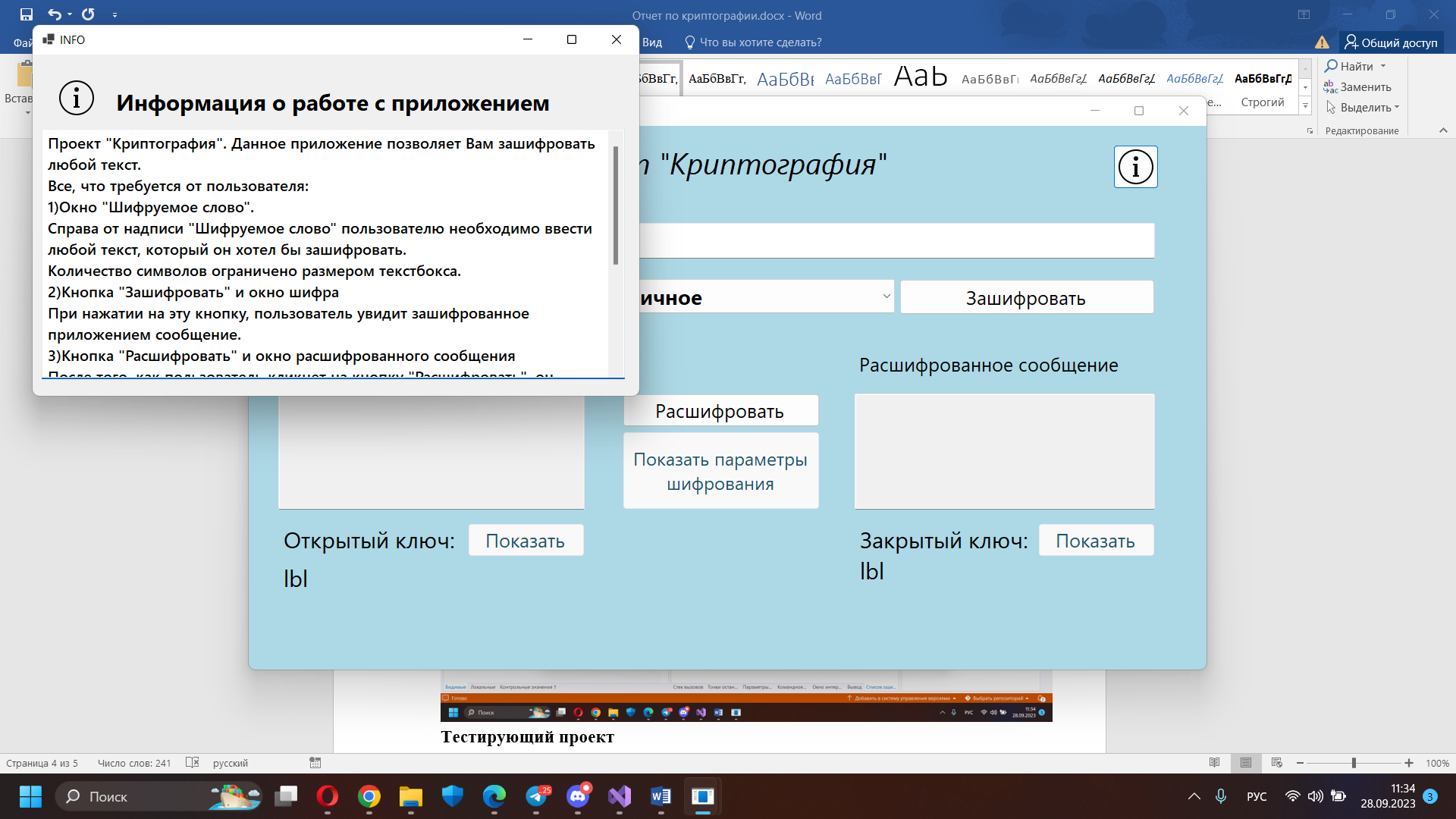
Форма предоставляет возможность ввести шифруемую строку, выбрать тип шифрования, просмотреть сообщение в зашифрованном виде, а также расшифровать, чтобы убедиться в корректности работы, и посмотреть параметры шифрования: для шифра Цезаря это будет смещение, для ассиметричного RSA – открытая и закрытая экспонента, модуль, простые числа и др. Данная программа носит демонстрационный характер и создана, чтобы с помощью яркого интерфейса продемонстрировать пользователю работу некоторых методов шифрования, поэтому и существует такая бесполезная с точки зрения практического шифрования возможность.

Также пользователю предоставлена возможность задать смещение для метода Цезаря и маску для метода перестановок.



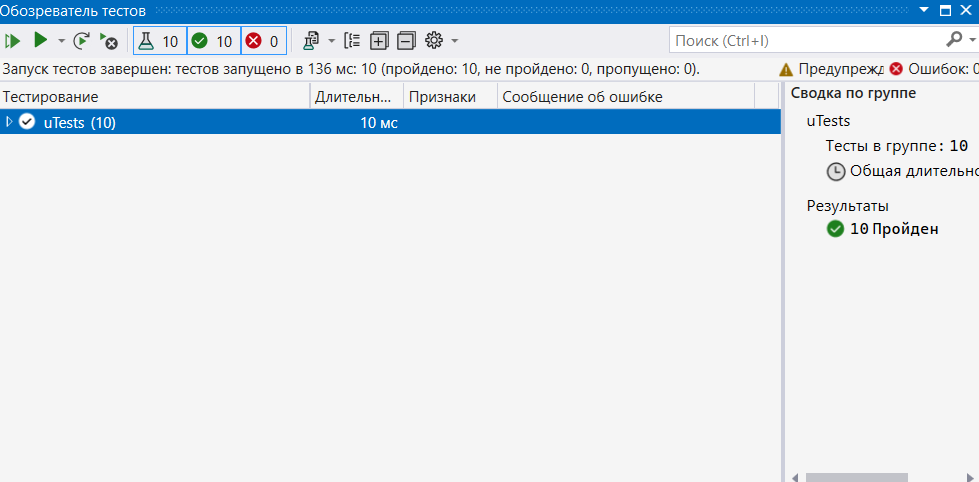






## **Тестирующий проект**

Тестирующий проект помог нам эффективно и быстро проверить корректность работы программы. В нем мы создали отдельные проверки для каждой из функций в построенной библиотеке классов. После запуска все тесты были в успешно пройдены, следовательно созданные нами функции работаю корректно.



## **Библиотека классов**

Библиотека классов крайне обширна. Классы имеют различные отношения между собой, существуют как статические модули, так и полноценные классы.

Они распределены по группам, причем таким, что между классами разных групп не существует отношения наследования. Также выделены в отдельную группу статические классы.

***Группа Modules.***

Набор из двух статических классов, позволяющих осуществлять быстрое возведение в степень по модулю и генерацию простых чисел.

/// <summary>

/// Модуль, позволяющий осуществлять эффективные арифметические действия

/// </summary>

public static class Ariphmetics

{

/// <summary>

/// Функция, позволяющая выполнять быстрее возведение в степень по модулю

/// </summary>

/// <param name="num">Число, над которым будет выполняться операция</param>

/// <param name="pow">Число, в степень которого будет возводиться num</param>

/// <param name="mod">

/// Модуль, по которому будет

/// осуществляться возведение в степень

/// </param>

/// <returns>Число-результат модульной экспоненциации</returns>

public static long ModularExp(long num, long pow, long mod)

{

if (pow == 0)

return 1;

long z = ModularExp(num, pow / 2, mod);

if (pow % 2 == 0)

return (z \* z) % mod;

else

return ((num % mod) \* ((z \* z) % mod)) % mod;

}

}

/// <summary>

/// Модуль, позволяющий осуществлять генерацию простых чисел

/// </summary>

public static class PrimeNumberGenerator

{

/// <summary>

/// Функция, позволяющая создавать простое число в данном диапозоне

/// </summary>

/// <param name="from">

/// Число - начало диапозона, в котором осуществляется генерация

/// </param>

/// <param name="until">

/// Число - конец диапозона, в котором осуществляется генерация

/// </param>

/// <returns>Случайное простое число в данном диапозоне</returns>

public static long generatePrimeNumber(long from, long until)

{

Random random = new Random();

long[] primes = Dividers.AllPrimes((long)from, (long)until);

return primes[random.Next(primes.Length)];

}

}

***Группа StoringClasses.***

Набор из нескольких классов, которые будут использованы для хранения информации о ключах и сообщении. Созданы как оберточные классы для примитивов, чтобы сделать код более понятным и наглядным, а также на случай изменения спецификации (например, если понадобится изменить способ вывода ключа, то в данном случае нужно будет лишь переписать одну функцию в классе, вывод которого нужно изменить – это просто, понятно и очевидно для поиска). Также установлены отношения наследования: PrivateKey и PublicKey наследуются от класса Key. Эти классы разделены с целью увеличить понятность дальнейшего кода.

/// <summary>

/// Класс, предоставляющий возможность хранить одно поле -

/// значение ключа

/// </summary>

public class Key

{

/// <summary>

/// Поле - значение ключа

/// </summary>

private long n;

/// <summary>

/// Конструктор, позволяющий создать экземпляр класса на основе данного значения ключа

/// </summary>

/// <param name="n">Значение ключа n </param>

public Key(long n)

{

this.n = n;

}

/// <summary>

/// Возвращает значение ключа

/// </summary>

/// <returns>N - значение ключа</returns>

public long getN()

{

return n;

}

/// <summary>

/// Позволяет получить строку, отображающую основные сведения

/// о текущем экземпляре ключа

/// </summary>

/// <returns>

/// Строка, отображающая текущее n-значение ключа

/// </returns>

public override string ToString()

{

return $"[{n}]";

}

}

/// <summary>

/// Класс, хранящий в себе данные, необходимые для создания открытого ключа:

/// значение открытой экспоненты и значение ключа

/// </summary>

public class PublicKey : Key

{

/// <summary>

/// Значение октрытой экспоненты

/// </summary>

private long publicExp;

/// <summary>

/// Конструктор, позволяющий создать экземпляр класса на основе

/// данного значения открытой экспоненты и значения ключа n

/// </summary>

/// <param name="n">Значение ключа n </param>

/// <param name="publicExp">Значение открытой экспоненты </param>

public PublicKey(long n, long publicExp) : base(n)

{

this.publicExp = publicExp;

}

/// <summary>

/// Позволяет получить значение открытой экспоненты

/// данного открытого ключа

/// </summary>

/// <returns>Значение открытой экспоненты</returns>

public long getPublicExp()

{

return this.publicExp;

}

/// <summary>

/// Позволяет получить строку, отображающую основные сведения

/// о текущем экземпляре ключа

/// </summary>

/// <returns>

/// Строка, отображающая текущее n-значение ключа,

/// а также значение открытой экспоненты

/// </returns>

public override string ToString()

{

return $"[n: {this.getN()}, public exp: {getPublicExp()}]";

}

}

/// <summary>

/// Класс, хранящий в себе данные, необходимые для создания закрытого ключа:

/// значение закрытой экспоненты и значение ключа

/// </summary>

public class PrivateKey : Key

{

/// <summary>

/// Значение закрытой экспоненты

/// </summary>

private long privateExp;

/// <summary>

/// Конструктор, позволяющий создать экземпляр класса на основе

/// данного значения открытой экспоненты и значения ключа n

/// </summary>

/// <param name="n">Значение ключа n </param>

/// <param name="privateExp">Значение закрытой экспоненты </param>

public PrivateKey(long n, long privateExp) : base(n)

{

this.privateExp = privateExp;

}

/// <summary>

/// Позволяет получить значение закрытой экспоненты

/// данного открытого ключа

/// </summary>

/// <returns>Значение открытой экспоненты</returns>

public long getPrivateExp()

{

return this.privateExp;

}

/// <summary>

/// Позволяет получить строку, отображающую основные сведения

/// о текущем экземпляре ключа

/// </summary>

/// <returns>

/// Строка, отображающая текущее n-значение ключа,

/// а также значение закрытой экспоненты

/// </returns>

public override string ToString()

{

return $"[n: {this.getN()}, private exp: {getPrivateExp()}]";

}

}

/// <summary>

/// Класс, позволяющий хранить сообщение, состоящее из одного символа

/// </summary>

public class Message

{

/// <summary>

/// Символ - значение сообщения

/// </summary>

private char message;

/// <summary>

/// Конструктор, позволяющий создавать экземпляр класса

/// на основе символа

/// </summary>

/// <param name="message">Символ - сообщение</param>

public Message(char message)

{

this.message = message;

}

/// <summary>

/// Конструктор, позволяющий создавать экземпляр класса

/// на основе числа в формате Int64

/// </summary>

/// <param name="message">Код символа в формате Int64</param>

public Message(long message)

{

this.message = (char)message;

}

/// <summary>

/// Конструктор, позволяющий создавать экземпляр класса

/// на основе числа в формате Int32

/// </summary>

/// <param name="message">Код символа в формате Int32</param>

public Message(int message)

{

this.message = (char)message;

}

/// <summary>

/// Позволяет получить код сообщения в формате Int32

/// </summary>

/// <returns>Код сообщения в формате Int32</returns>

public int getMessage()

{

return this.message;

}

/// <summary>

/// Позволяет получить строку, содержающую один символ - данное сообщение

/// </summary>

/// <returns>

/// Строка, состоящая из одно символа - текущего символа

/// </returns>

public override string ToString()

{

return message.ToString();

}

***Группа Cesaurus.***

Классы этой группы позволяют осуществлять шифровку и расшифровку методом Цезаря. Разделены энкриптор и декриптор, чтобы исключить лишние зависимости между классами, уменьшить количество ошибок, а также сделать код проще и нагляднее, но, так как задача проекта – продемонстрировать работу, они объединяются в классе CryptoProcessCaesar, в котором реализованы методы, которые будут вызываться пользователем в зависимости от его действий относительно интерфейса. Объединены они, чтобы в интерфейсной части не пришлось создавать дополнительные поля, переменные и объекты, которые сделают код излишне растянутым.

Схема группы выглядит следующим образом (*стрелочки демонстрируют отношение вложенности*):

StringEncryptorCaesar:  
Encrypt();

StringDecryptorCaesar:  
Decrypt();

CryptoProcessCaesar: Encrypt();  
Decrypt();

showMsg();

showOffset();

showEncrypt();

showDecrypt();

/// <summary>

/// Класс, позволяющий осуществлять зашировку сообщений, представленных строками

/// С помощью шифрования Цезаря

/// </summary>

public class StringEncryptorCaesar

{

/// <summary>

/// Поле - сдвиг

/// </summary>

private int k;

/// <summary>

/// Поле - шифруемое сообщение

/// </summary>

private string message;

/// <summary>

/// Конструктор, позволяющий создавать экземпляр класса на основе данных

/// сдвига и сообщения

/// </summary>

/// <param name="k">Сдвиг</param>

/// <param name="message">Сообщение в формате строки</param>

public StringEncryptorCaesar(int k, string message)

{

this.message = message;

this.k = k;

}

/// <summary>

/// Позволяет зашифровать сообщение методом шифрования Цезаря

/// </summary>

/// <returns>Шифруемое сообщение в формате строки</returns>

public string Encrypt()

{

StringBuilder encryptedMessage = new StringBuilder();

foreach (char character in message.ToCharArray())

{

char c = (char)(character + k);

encryptedMessage.Append(c);

}

return encryptedMessage.ToString().Trim();

}

}

/// <summary>

/// Класс, позволяющий осуществлять расшифровку сообщений,

/// зашифрованных с помощью шифра Цезаря

/// </summary>

public class StringDecryptorCaesar

{

/// <summary>

/// Поле - сдвиг в шифре Цезаря

/// </summary>

private int offset;

/// <summary>

/// Поле - расшировываемое сообщение

/// </summary>

private string encryptedMessage;

/// <summary>

/// Конструктор, позволяющий создавать экземпляр класса на основе данных

/// закрытого ключа и сообщения

/// </summary>

/// <param name="offset">Cдвиг в шифре Цезаря</param>

/// <param name="message">Зашифрованное сообщение</param>

public StringDecryptorCaesar(int offset, string encryptedMessage)

{

this.encryptedMessage = encryptedMessage;

this.offset = offset;

}

/// <summary>

/// Позволяет расшифровать зашифрованное сообщение

/// </summary>

/// <returns>Расшифровываемое сообщение</returns>

public string Decrypt()

{

StringBuilder decryptedMessage = new StringBuilder();

foreach (char c in encryptedMessage)

{

decryptedMessage.Append((char)(c - offset));

}

return decryptedMessage.ToString();

}

}

/// <summary>

/// Класс, позволяющий зашифровывыать и расшифровыывать строку

/// </summary>

public class CryptoProcessCaesar

{

/// <summary>

/// Строка, которую шифруем

/// </summary>

private string message;

/// <summary>

/// Сдвиг в шифре Цезаря

/// </summary>

private int offset;

public CryptoProcessCaesar(string message, int offset)

{

this.message = message;

this.offset = offset;

}

/// <summary>

/// Позволяет шифровать сроку

/// </summary>

/// <returns></returns>

public string encrypt()

{

StringEncryptorCaesar SE = new StringEncryptorCaesar(offset, message);

string EncMsg = SE.Encrypt();

return EncMsg.ToString().Trim();

}

/// <summary>

/// Позволяет выводить изначальную строку

/// </summary>

/// <returns></returns>

public string showMsg()

{

return message;

}

/// <summary>

/// Позволяет выводить сдвиг

/// </summary>

/// <returns></returns>

public string showOffset()

{

return offset.ToString();

}

/// <summary>

/// Позволяет расшифровывать зашифрованную строку

/// </summary>

/// <returns></returns>

public string decrypt()

{

StringDecryptorCaesar SD = new StringDecryptorCaesar(offset, encrypt());

string DecMsg = SD.Decrypt();

return DecMsg.ToString().Trim();

}

/// <summary>

/// Выводит зашифрованную строку

/// </summary>

/// <returns></returns>

public string showEncrypt()

{

return encrypt().ToString().Trim();

}

/// <summary>

/// Выводит расшифрованную строку

/// </summary>

/// <returns></returns>

public string showDecrypt()

{

return decrypt().ToString().Trim();

}

}

/// <summary>

/// Класс, позволяющий осуществлять зашировку сообщений методом перестановки

/// </summary>

***Группа Transposition.***

Группа спроектирована аналогично группе Caesaurus, отличается только тем, что здесь реализован метод шифрования перестановками.

Схема группы выглядит следующим образом (*стрелочки демонстрируют отношение вложенности*):

StringEncryptorTransposition:  
Encrypt();

StringDecryptorCaesar:  
Decrypt();

StringDecryptorTransposition:

Encrypt();  
Decrypt();

showMsg();

showEncKey();

showDecKey();

showEncrypt();

showDecrypt();

/// <summary>

/// Класс, позволяющий осуществлять шифрование сообщений

/// методом перестановки

/// </summary>

public class StringEncryptorTransposition

{

/// <summary>

/// Поле - ключ шифрования

/// </summary>

private int[] encKey;

/// <summary>

/// Полк - шифруемое сообщение

/// </summary>

private string message;

/// <summary>

/// Конструктор, позволяющий создавать экземпляр класса на основе данных

/// ключа шифрования и сообщения

/// </summary>

/// <param name="encKey">Ключ шифрования</param>

/// <param name="message">Сообщение в формате строки</param>

public StringEncryptorTransposition(int[] encKey, string message)

{

this.message = message;

this.encKey = encKey;

}

/// <summary>

/// Позволяет зашифровать сообщение методом шифрования Цезаря

/// </summary>

/// <returns>Шифруемое сообщение в формате строки</returns>

public string Encrypt()

{

while (message.Length % encKey.Length != 0)

{

message += " ";

}

StringBuilder encryptedMessage = new StringBuilder();

string temp = "";

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

{

temp += message[i];

if (temp.Length == encKey.Length)

{

for (int j = 0; j < temp.Length; j++)

{

encryptedMessage.Append(temp[encKey[j] - 1]);

}

temp = "";

}

}

return encryptedMessage.ToString();

}

}

/// <summary>

/// Класс, позволяющий осуществлять расшифровку сообщений,

/// зашифрованных методом перестановки

/// </summary>

public class StringDecryptorTransposition

{

/// <summary>

/// Поле - ключ дешифрования

/// </summary>

private int[] decKey;

/// <summary>

/// Поле - расшировываемое сообщение

/// </summary>

private string encryptedMessage;

/// <summary>

/// Конструктор, позволяющий создавать экземпляр класса на основе данных

/// закрытого ключа и сообщения

/// </summary>

/// <param name="decKey">Ключ дешифрования</param>

/// <param name="message">Зашифрованное сообщение</param>

public StringDecryptorTransposition(int[] decKey, string encryptedMessage)

{

this.encryptedMessage = encryptedMessage;

this.decKey = decKey;

}

/// <summary>

/// Позволяет расшифровать зашифрованное сообщение

/// </summary>

/// <returns>Расшифровываемое сообщение</returns>

public string Decrypt()

{

StringBuilder decryptedMessage = new StringBuilder();

string temp = "";

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

{

temp += encryptedMessage[i];

if (temp.Length == decKey.Length)

{

for (int j = 0; j < temp.Length; j++)

{

decryptedMessage.Append(temp[decKey[j]]);

}

temp = "";

}

}

return decryptedMessage.ToString();

}

}

/// <summary>

/// Класс, позволяющий зашифровывыать и расшифровыывать строку

/// </summary>

public class CryptoProcessTransposition

{

/// <summary>

/// Строка, которую шифруем

/// </summary>

private string message;

/// <summary>

/// Ключ, которым зашифровано сообщение

/// </summary>

private int[] encKey;

/// <summary>

/// Ключ, которым зашифровано сообщение

/// </summary>

private int[] decKey;

public CryptoProcessTransposition(string message, string encKey)

{

this.message = message;

this.encKey = new int[encKey.Length];

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

{

this.encKey[i] = int.Parse(encKey[i].ToString());

}

this.decKey = new int[encKey.Length];

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

{

this.decKey[this.encKey[i] - 1] = i;

}

}

/// <summary>

/// Позволяет шифровать сроку

/// </summary>

/// <returns></returns>

public string encrypt()

{

StringEncryptorTransposition SE = new StringEncryptorTransposition(encKey, message);

string EncMsg = SE.Encrypt();

return EncMsg.ToString();

}

/// <summary>

/// Позволяет выводить изначальную строку

/// </summary>

/// <returns></returns>

public string showMsg()

{

return message;

}

/// <summary>

/// Позволяет выводить ключ шифрования

/// </summary>

/// <returns></returns>

public string showEncKey()

{

string res = "";

foreach (int i in encKey)

{

res += i.ToString();

}

return res;

}

/// <summary>

/// Позволяет выводить ключ дешифрования

/// </summary>

/// <returns></returns>

public string showDecKey()

{

string res = "";

foreach (int i in decKey)

{

res += (i + 1).ToString();

}

return res;

}

/// <summary>

/// Позволяет расшифровывать зашифрованную строку

/// </summary>

/// <returns></returns>

public string decrypt()

{

StringDecryptorTransposition SD = new StringDecryptorTransposition(decKey, encrypt());

string DecMsg = SD.Decrypt();

return DecMsg.ToString();

}

/// <summary>

/// Выводит зашифрованную строку

/// </summary>

/// <returns></returns>

public string showEncrypt()

{

return encrypt().ToString().Trim();

}

/// <summary>

/// Выводит расшифрованную строку

/// </summary>

/// <returns></returns>

public string showDecrypt()

{

return decrypt().ToString().Trim();

}

}

***Группа Dividers.***

Представляет собой статический модуль, написанный ранее для другого проекта, используется для генерации простых чисел.

/// <summary>

/// Модуль, позволяющий работать с натуральными числами и их делителями

/// </summary>

public static class Dividers

{

/// <summary>

/// Определяет, является ли натуральное число d

/// делителем натурального числа n

/// </summary>

/// <param name="n">Проверяемое число в диапозоне от 1 до Int.MaxValue</param>

/// <param name="d">Проверяемый делитель в диапозоне от 1 до Int.MaxValue</param>

/// <returns>

/// True: n - делитель d,

/// False: n - не делитель d

/// </returns>

public static bool IsDivider(long n, long d)

{

return n % d == 0;

}

/// <summary>

/// Находит разложение натурального числа на простые делители и их степени

/// </summary>

/// <param name="n">Проверяемое натуральное число от 2 до Int.MaxValue</param>

/// <returns>

/// Кортеж, состоящий из двух массивов:

/// массив целочисленных положительных делителей

/// и массив целочисленных положительных степеней соответсвующих делителей

/// </returns>

public static (long[], long[]) Factorize(long n)

{

List<long> dividers = new List<long>();

List<long> powers = new List<long>();

long[] primes = AllPrimes(2, n);

long i = 0;

int k = 0;

while (n > 1)

{

if (n % primes[i] == 0)

{

dividers.Add(primes[i]);

powers.Add(0);

while (n % primes[i] == 0)

{

powers[k]++;

n /= primes[i];

}

k++;

}

i++;

}

return (dividers.ToArray(), powers.ToArray());

}

/// <summary>

/// Проверяет, является ли данное натуральное число простым

/// </summary>

/// <param name="n">Проверяемое натуральное число в диапозоне от 1 до Int.MaxValue</param>

/// <returns>

/// True: число простое

/// False: число составное

/// </returns>

public static bool IsPrime(long n)

{

return AllDividers(n).Length == 2;

}

/// <summary>

/// Получает список всех натуральных делителей

/// данного натурального числа

/// </summary>

/// <param name="n">

/// Натуральное число, список делителей которого необходимо получить,

/// в диапозоне от 1 до Int.MaxValue

/// </param>

/// <returns>Массив всех делителей числа</returns>

public static long[] AllDividers(long n)

{

List<long> result = new List<long>();

for (int i = 1; i <= Math.Sqrt(n); i++)

{

if (Dividers.IsDivider(n, i))

{

result.Add(i);

result.Add(n / i);

}

}

return result.Distinct().ToArray();

}

/// <summary>

/// Находит все простые числа на отрезке

/// от данного натурального d

/// до данного натурального n

/// с помощью метода решета Эратосфена,

/// </summary>

/// <param name="d">

/// Начало проверяемого отрезка,

/// натуральное число от 2 до Int.MaxValue

/// </param>

/// <param name="n">

/// Конец проверяемого отрезка,

/// натуральное число от n до Int.MaxValue,

/// </param>

/// <returns>Массив простых чисел на отрезке [d; n]</returns>

public static long[] AllPrimes(long d, long n)

{

List<long> numbers = new List<long>();

bool[] isNotPrime = new bool[n + 1];

for (int j = 2; j \* j <= n; j++)

{

if (!isNotPrime[j])

for (int i = j \* j; i <= n; i += j)

isNotPrime[i] = true;

}

long start = d >= 2 ? d : 2;

for (long i = start; i <= n; i++)

if (!isNotPrime[i])

numbers.Add(i);

return numbers.ToArray();

}

/// <summary>

/// Находит все простые числа на отрезке

/// от данного натурального d

/// до данного натурального n

/// методом перебора

/// </summary>

/// <param name="d">

/// Начало проверяемого отрезка,

/// натуральное число от 1 до Int.MaxValue

/// </param>

/// <param name="n">

/// Конец проверяемого отрезка,

/// натуральное число от n до Int.MaxValue

/// </param>

/// <returns>Массив простых чисел на отрезке</returns>

public static long[] AllPrimesByCheckingAll(long d, long n)

{

List<long> numbers = new List<long>();

for (long i = d; i <= n; i++)

{

if (IsPrime(n))

numbers.Add(n);

}

return numbers.ToArray();

}

/// <summary>

/// Находит на заданном отрезке количество чисел

/// со строго определенным количеством небазовых делителей

/// </summary>

/// <param name="amountOfDividers">

/// Заданное целое количество делителей,

/// </param>

/// <param name="start">

/// Начало проверяемого отрезка,

/// натуральное число от 1 до Int.MaxValue</param>

/// <param name="end">

/// Конец проверяемого отрезка,

/// натуральное число от start до Int.MaxValue

/// </param>

/// <returns>

/// Массив чисел, находящихся на данном отрезке,

/// с определенным количеством делителей

/// </returns>

public static long[] FindNumsWithDividers(long amountOfDividers, long start, long end)

{

if (amountOfDividers == 3)

return Dividers.FindNumsWithThreeDividers(start, end);

List<long> list = new List<long>();

for (long k = start; k < end; k++)

{

long[] divider = AllDividers(k);

if (divider.Length == amountOfDividers + 2)

list.Add(k);

}

return list.ToArray();

}

/// <summary>

/// Находит на заданном отрезке количество чисел

/// с ровно тремя небазовыми делителями

/// </summary>

/// <param name="start">

/// Начало проверяемого отрезка,

/// натуральное число от 1 до Int.MaxValue</param>

/// <param name="end">

/// Конец проверяемого отрезка,

/// натуральное число от start до Int.MaxValue

/// </param>

/// <returns>

/// Массив чисел, находящихся на данном отрезке,

/// с ровно тремя небазовыми делителями

/// </returns>

public static long[] FindNumsWithThreeDividers(long start, long end)

{

List<long> list = new();

foreach (long i in AllPrimes(start, (long)(Math.Sqrt(Math.Sqrt(end))) + 1))

{

long sqare = i \* i;

long fourth = sqare \* sqare;

list.Add(fourth);

}

return list.ToArray();

}

}

***Группа RSA.***

Главная группа в проекте, реализующая самый технически сложный метод – ассиметричное RSA шифрование. Это задача весьма нетривиальная, и решена она была посредством широкой декомпозиции на уровне классов.

Был необходим генератор ключей для начала – был создан соответсвующий класс. Далее была реализована возможность шифровать и расшифровывать сообщения класса Message – т.е. односимвольные, далее функционал расширяется до полноценного энкриптора и декриптора строк, а позже они объединяются с той же целью, что и в других классах: спецификация проекта – продемонстрировать принципы шифрования, и это будет делать удобнее, используя один класс, предоставляющий весь функционал.

Схема группы выглядит следующим образом (*стрелочки демонстрируют отношение вложенности*):

RSAEncryptor:  
encrypt();

RSADecryptor:  
decrypt();

RSAStringEncryptor:  
encrypt();

RSAStringDecryptor:  
decrypt();

CryptoProcessRSA:

Encrypt();  
Decrypt();

showMsg();

showPrivateKey();

showPublicKey();

showEncrypt();

showDecrypt();

showKeyGen()

RSAKeyGenerator:  
calcModule();

calcEulerFunc();

generatePublicExp();

generatePrivateExp();

/// <summary>

/// Класс, предоставляющий возможность генерировать

/// ключи для ассиметричного шифрования

/// </summary>

public class RSAKeyGenerator

{

/// <summary>

/// Константа - общепринятое значение открытой экспоненты,

/// равное 2^16 + 1

/// </summary>

public const int PUBLIC\_EXP = 65537;

/// <summary>

/// Простое число p, использующееся для генерации ключей

/// </summary>

private long p;

/// <summary>

/// Простое число q, использующееся для генерации ключей

/// </summary>

private long q;

/// <summary>

/// Конструктор, позволяющий создавать экземпляр данного объекта,

/// на основе данных значений простых чисел p и q

/// </summary>

/// <param name="p">

/// Простое число p, использующееся для генерации ключей

/// </param>

/// <param name="q">

/// Простое число q, использующееся для генерации ключей

/// </param>

public RSAKeyGenerator(long p, long q)

{

this.p = p;

this.q = q;

}

/// <summary>

/// Функция, позволяющая высчитать модуль для данных ключей

/// </summary>

/// <returns>

/// Модуль - произведение простых чисел p и q

/// </returns>

public long calcModule()

{

return p \* q;

}

/// <summary>

/// Функция, позволяющая высчитать значение функции Эйлера

/// </summary>

/// <returns>

/// Значение функции Эйлера - произведение простых чисел,

/// уменьшенных на единицу

/// </returns>

public long calcEulerFunc()

{

return (p - 1) \* (q - 1);

}

/// <summary>

/// Позволяет получить значение открытой экспоненты

/// </summary>

/// <returns>Значение открытой экспоненты</returns>

public long generatePublicExp()

{

return PUBLIC\_EXP;

}

/// <summary>

/// Позволяет получить значение закрытой экспоненты

/// </summary>

/// <returns>Значение закрытой экспоненты</returns>

public long generatePrivateExp()

{

long e = generatePublicExp();

long fi = calcEulerFunc();

long d = Extendedgcd(e, fi).x;

if (d < 0)

return d + fi;

else return d;

}

/// <summary>

/// Функция, позволяющая получить значение наибольшего общего делителя чисел

/// </summary>

/// <param name="a">Первое число</param>

/// <param name="b">Второе число</param>

/// <returns>

/// Кортеж, состоящий из 3 чисел:

/// НОД первого и второго числа,

/// первое число,

/// второе число

/// </returns>

private (long gcd, long x, long y) Extendedgcd(long a, long b)

{

long gcd, x, y, x1, y1;

if (b == 0) return (a, 1, 0);

(gcd, x1, y1) = Extendedgcd(b, a % b);

x = y1;

y = x1 - (a / b) \* y1;

return (gcd, x, y);

}

/// <summary>

/// Возвращает данные о текущем объекте в формате строки

/// </summary>

/// <returns>

/// Данные о текущем объекте:

/// значения p, q, модуль, Эйлерова функция,

/// открытая и закрытая экспоненты

/// </returns>

public override string ToString()

{

return

$"p: {p};\n" +

$"q: {q};\n" +

$"module: {calcModule()};\n" +

$"euler function value: {calcEulerFunc()};\n" +

$"public exponent: {generatePublicExp()};\n" +

$"private exponent: {generatePrivateExp()};\n";

}

}

/// <summary>

/// Класс, позволяющий осуществлять зашифровку сообщений методом RSA

/// </summary>

public class RSAEncryptor

{

/// <summary>

/// Поле - открытый ключ

/// </summary>

private PublicKey publicKey;

/// <summary>

/// Поле - шифруемое сообщение

/// </summary>

private Message message;

/// <summary>

/// Конструктор, позволяющий создавать экземпляр класса на основе данных

/// открытого ключа и сообщения

/// </summary>

/// <param name="publicKey">Открытый ключ, с помощью которого шифруется сообщения</param>

/// <param name="message">Шифруемое сообщение</param>

public RSAEncryptor(PublicKey publicKey, Message message)

{

this.message = message;

this.publicKey = publicKey;

}

/// <summary>

/// Позволяет зашифровать сообщение методом RSA

/// </summary>

/// <returns>Зашифрованное сообщение</returns>

public Message encrypt()

{

return

new Message(

Ariphmetics.ModularExp(message.getMessage(),

publicKey.getPublicExp(),

publicKey.getN()

)

);

}

}

/// <summary>

/// Класс, позволяющий осуществлять расшифровку сообщений,

/// зашифрованных методом RSA

/// </summary>

public class RSADecryptor

{

/// <summary>

/// Поле - закрытый ключ

/// </summary>

private PrivateKey privateKey;

/// <summary>

/// Поле - расшировываемое сообщение

/// </summary>

private Message encryptedMessage;

/// <summary>

/// Конструктор, позволяющий создавать экземпляр класса на основе данных

/// закрытого ключа и сообщения

/// </summary>

/// <param name="privateKey">Закрытый ключ, с помощью которого зашифровано сообщение</param>

/// <param name="message">Зашифрованное сообщение</param>

public RSADecryptor(PrivateKey privateKey, Message message)

{

this.privateKey = privateKey;

this.encryptedMessage = message;

}

/// <summary>

/// Позволяет расшифровать зашифрованное сообщение

/// </summary>

/// <returns>Расшифровываемое сообщение</returns>

public Message decrypt()

{

return

new Message(

Ariphmetics.ModularExp(

encryptedMessage.getMessage(),

privateKey.getPrivateExp(),

privateKey.getN()

)

);

}

}

/// <summary>

/// Класс, позволяющий осуществлять расшировку сообщений, представленных строками

/// </summary>

public class RSAStringEncryptor

{

/// <summary>

/// Поле - открытый ключ

/// </summary>

private PublicKey publicKey;

/// <summary>

/// Шифруемое сообщение

/// </summary>

private string message;

/// <summary>

/// Конструктор, позволяющий создавать экземпляр класса на основе данных

/// открытого ключа и сообщения

/// </summary>

/// <param name="publicKey">Открытый ключ</param>

/// <param name="message">Сообщение в формате строки</param>

public RSAStringEncryptor(PublicKey publicKey, string message)

{

this.message = message;

this.publicKey = publicKey;

}

/// <summary>

/// Позволяет зашифровать сообщение методом RSA

/// </summary>

/// <returns>Шифруемое сообщение в формате строки</returns>

public string Encrypt()

{

StringBuilder encryptedMessage = new StringBuilder();

foreach (char character in message.ToCharArray())

{

long charValue = (long)character;

long encryptedCharValue = Ariphmetics.ModularExp(charValue, publicKey.getPublicExp(), publicKey.getN());

encryptedMessage.Append(encryptedCharValue).Append(" ");

}

return encryptedMessage.ToString().Trim();

}

}

/// <summary>

/// Класс, позволяющий осуществлять расшифровку сообщений,

/// зашифрованных методом RSA

/// </summary>

public class RSAStringDecryptor

{

/// <summary>

/// Поле - закрытый ключ

/// </summary>

private PrivateKey privateKey;

/// <summary>

/// Поле - расшировываемое сообщение

/// </summary>

private string encryptedMessage;

/// <summary>

/// Конструктор, позволяющий создавать экземпляр класса на основе данных

/// закрытого ключа и сообщения

/// </summary>

/// <param name="privateKey">Закрытый ключ, с помощью которого зашифровано сообщение</param>

/// <param name="message">Зашифрованное сообщение</param>

public RSAStringDecryptor(PrivateKey privateKey, string encryptedMessage)

{

this.encryptedMessage = encryptedMessage;

this.privateKey = privateKey;

}

/// <summary>

/// Позволяет расшифровать зашифрованное сообщение

/// </summary>

/// <returns>Расшифровываемое сообщение</returns>

public string Decrypt()

{

StringBuilder decryptedMessage = new StringBuilder();

string[] encryptedValues = encryptedMessage.Split(' ');

foreach (string encryptedValue in encryptedValues)

{

if (!string.IsNullOrWhiteSpace(encryptedValue))

{

long encryptedCharValue = long.Parse(encryptedValue);

long decryptedCharValue = Ariphmetics.ModularExp(encryptedCharValue, privateKey.getPrivateExp(), privateKey.getN());

char decryptedChar = (char)decryptedCharValue;

decryptedMessage.Append(decryptedChar);

}

}

return decryptedMessage.ToString();

}

}

/// <summary>

/// Класс, позволяющий зашифровывыать и расшифровыывать строку

/// </summary>

public class CryptoProcessRSA

{

/// <summary>

/// Строка, которую шифруем

/// </summary>

private string message;

/// <summary>

/// Начало отрезка, на котором мы выбираем простые числа

/// </summary>

const int from = 100;

/// <summary>

/// Конец отрезка, для выбора простых чисел

/// </summary>

const int to = 3000;

long p;

long q;

/// <summary>

/// Генерация открытого и закрытого ключа

/// </summary>

RSAKeyGenerator keyGenerator;

PublicKey pK1;

PrivateKey prK1;

public CryptoProcessRSA(string message)

{

this.message = message;

p = PrimeNumberGenerator.generatePrimeNumber(from, to);

q = PrimeNumberGenerator.generatePrimeNumber(from, to);

keyGenerator = new RSAKeyGenerator(p, q);

pK1 = new PublicKey(keyGenerator.calcModule(), keyGenerator.generatePublicExp());

prK1 = new PrivateKey(keyGenerator.calcModule(), keyGenerator.generatePrivateExp());

}

/// <summary>

/// Позволяет шифровать сроку

/// </summary>

/// <returns></returns>

public string encrypt()

{

RSAStringEncryptor SE = new RSAStringEncryptor(pK1, message);

string EncMsg = SE.Encrypt();

return EncMsg.ToString().Trim();

}

/// <summary>

/// Позволяет выводить изначальную строку

/// </summary>

/// <returns></returns>

public string showMsg()

{

return message;

}

/// <summary>

/// Позволяет выводить закрытый ключ

/// </summary>

/// <returns></returns>

public string showPrivateKey()

{

return prK1.ToString();

}

/// <summary>

/// Позволяет выводить открытый ключ

/// </summary>

/// <returns></returns>

public string showPublicKey()

{

return pK1.ToString();

}

/// <summary>

/// Позволяет расшифровывать зашифрованную строку

/// </summary>

/// <returns></returns>

public string decrypt()

{

RSAStringDecryptor SD = new RSAStringDecryptor(prK1, encrypt());

string DecMsg = SD.Decrypt();

return DecMsg.ToString().Trim();

}

/// <summary>

/// Выводит зашифрованную строку

/// </summary>

/// <returns></returns>

public string showEncrypt()

{

return encrypt().ToString().Trim();

}

/// <summary>

/// Выводит расшифрованную строку

/// </summary>

/// <returns></returns>

public string showDecrypt()

{

return decrypt().ToString().Trim();

}

/// <summary>

/// Получает генератор ключей, приведенный к строке

/// </summary>

/// <returns>Строка, содержащая данные о генераторе ключей</returns>

public string showKeyGen()

{

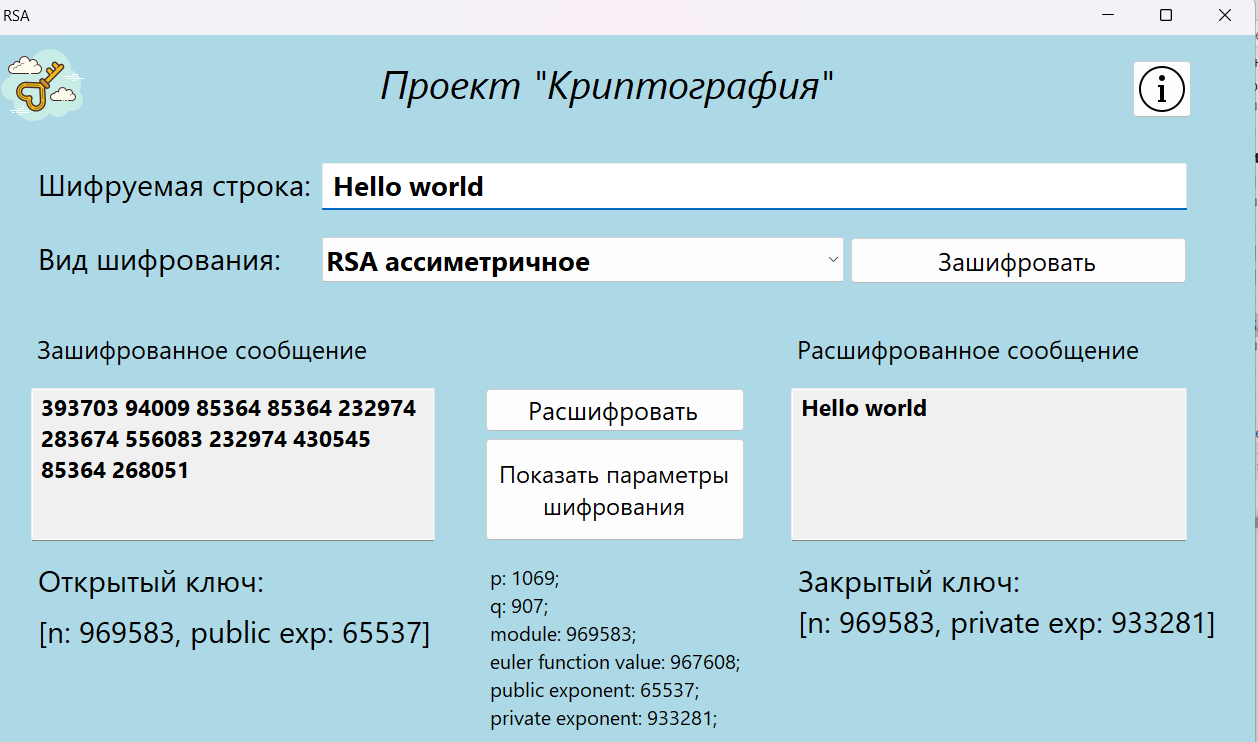
return keyGenerator.ToString();

}

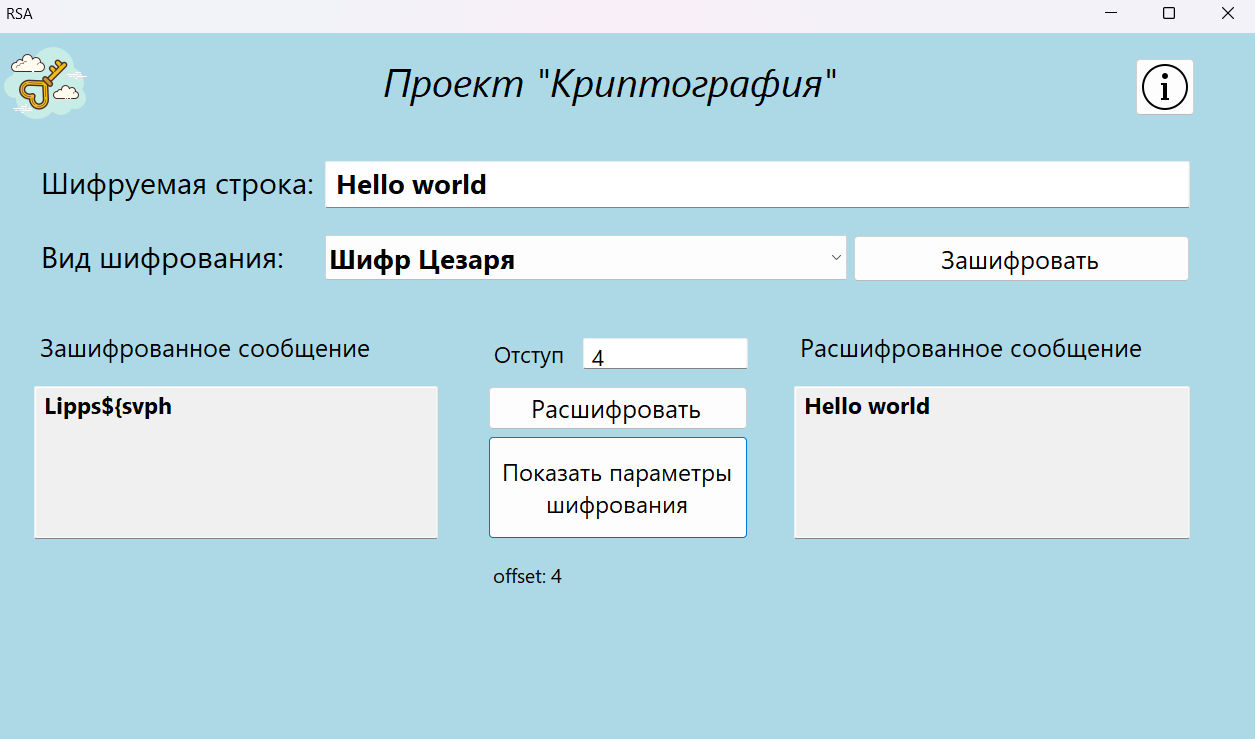
## **Демонстрация работы**

На примере входных и выходных данных продемонстрируем работу проекта «Криптография».

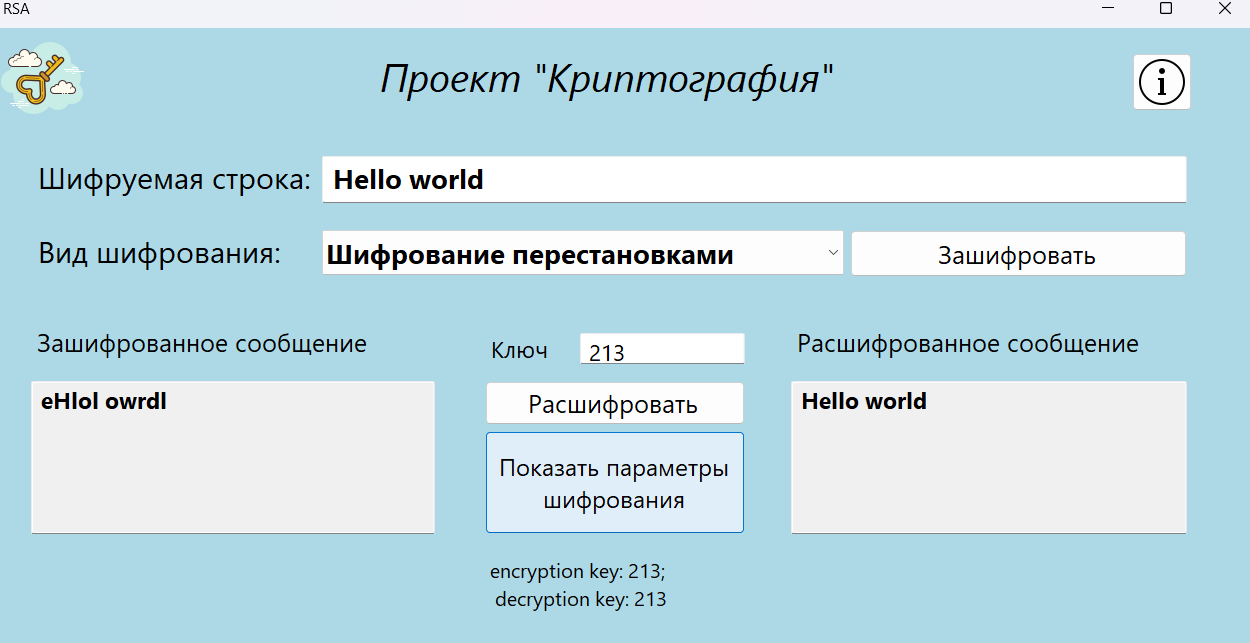
1) Пример работы RSA шифрования



2) Пример работы Шифрования Цезаря



3) Пример работы шифрования перестановками



**Заключение**

Криптография является неотъемлемой частью современного информационного общества и имеет множество применений. Она обеспечивает конфидециальность информации, делая сообщение непонятным для посторонних лиц. Также методы шифрования предоставляют защиту от различных атак, таких как подбор паролей или взлом. Сложные алгоритмы создают высокий уровень защиты данных. Без криптографии нарушители могли бы перехватывать важную информацию и использовать ее в корыстных целях. Поэтому криптография необходима для безопастности и конфиденциальности информации.

В нашем проекте мы реализовали три метода шифрования: RSA шифрование, шифрование Цезаря, шифрование перестановками. Все эти методы способны шифровать любые введенные пользователем символы. Также в нашем проекте пользователь может посмотреть по каким параметрам было произведено шифрование сообщение, а окно справки помогает правильно использовать весь функционал проекта.

По мере создания проекта мы узнали о многих видах шифрования, реализовав некоторые из них в своем проекте. Также мы осознали важность шифрования в современном мире, изучили различные алгоритмы и формулы, необходимые зашифровки и расшифровки сообщений.

**Использованные ресурсы**

* 1. Шифры перестановки. Шифры гаммирования. Лекция Академии ФСБ России.
  2. https://habr.com/ru/articles/745820/
  3. https://e-nigma.ru/stat/rsa/
  4. https://neerc.ifmo.ru/wiki/index.php?title=RSA
  5. <https://planetcalc.ru/1434/>
  6. <https://intuit.ru/studies/courses/3633/875/lecture/32498>
  7. <https://neerc.ifmo.ru/wiki/index.php?title=RSA>

**Приложение 1. Диалог с chatGPT**

***Калибр***

ChatGPT расскажи мне про методы шифрования сообщений

***CHATGPT 4***

Существует множество методов шифрования сообщений, которые используются для обеспечения конфиденциальности и безопасности передачи информации. Вот некоторые из наиболее распространенных методов шифрования:

1. Шифр замены: Это один из самых простых методов шифрования, где каждой букве алфавита присваивается другая буква или символ. Примером может служить шифр Цезаря, который сдвигает каждую букву на определенное количество позиций.

2. Шифр перестановки: При использовании этого метода буквы или символы меняются местами в заданном порядке. Это может быть сделано с использованием ключевого слова или числа, чтобы определить порядок перестановки.

3. Асимметричное шифрование: Этот метод использует пары ключей - публичный и приватный. Публичный ключ используется для шифрования сообщений, а приватный ключ используется для расшифровки сообщений. RSA является одним из самых популярных асимметричных алгоритмов шифрования.

4. Симметричное шифрование: В этом методе используется только один ключ как для шифрования, так и для расшифровки сообщений. Некоторые популярные алгоритмы симметричного шифрования включают DES (Data Encryption Standard) и AES (Advanced Encryption Standard).

5. Шифрование по открытому ключу (Public Key Infrastructure, PKI): Это комбинация асимметричного и симметричного шифрования. Открытый ключ используется для шифрования сообщений, и только соответствующий закрытый ключ может быть использован для расшифровки.

6. Квантовое шифрование: Это метод шифрования, который использует принципы квантовой механики для обработки и передачи информации. Квантовое шифрование обеспечивает более высокий уровень безопасности и невозможность подслушивания.

Каждый из этих методов имеет свои преимущества и ограничения, а также различную степень безопасности. Выбор метода шифрования зависит от конкретных потребностей и требований конкретного случая.

***Калибр***

Какой из этих видов шифрования наиболее эффективный?

***CHATGPT 4***

Каждый метод шифрования имеет свои преимущества и ограничения, и эффективность зависит от конкретной ситуации и требований. Оценка эффективности шифрования может основываться на следующих критериях:

1. Уровень безопасности: Некоторые методы шифрования обеспечивают более высокий уровень безопасности, чем другие. Квантовое шифрование, например, считается наиболее безопасным среди всех методов, но в настоящее время его применение ограничено.

2. Сложность взлома: Чем сложнее алгоритм шифрования, тем труднее его взломать. Некоторые методы шифрования, такие как AES с достаточно длинным ключом, считаются очень безопасными в отношении возможности взлома.

3. Эффективность алгоритма: Некоторые методы шифрования могут быть более эффективными с точки зрения вычислительной сложности и времени, требующихся для шифрования и расшифровки сообщений. Например, AES является очень эффективным алгоритмом с высокой скоростью шифрования.

4. Доступность и распространенность: Некоторые методы шифрования широко распространены и доступны для использования в различных приложениях и платформах. Это может упростить процесс реализации и внедрения шифрования.

В конечном счете, эффективность шифрования зависит от конкретных требований и контекста использования. Рекомендуется использовать комбинацию различных методов шифрования для повышения безопасности и достижения нужного уровня защиты информации.

***Калибр***

Расскажи подробнее про метод шифрования по открытому ключу. Приведи пример такого алгоритма написанного на C#

***CHATGPT 4***

Метод шифрования с открытым ключом (или асимметричное шифрование) - это криптографический метод, в котором используется пара ключей: открытый ключ для шифрования данных и закрытый ключ для их расшифровки. Открытый ключ может быть свободно распространен и известен всем, в то время как закрытый ключ должен оставаться в секрете, доступ к нему должен иметь только владелец.

Пример алгоритма шифрования с открытым ключом на языке C# - RSA (Rivest-Shamir-Adleman):

using System;

using System.Security.Cryptography;

public class RSAExample

{

public static void Main()

{

// Создание объекта RSA для использования алгоритма RSA

using (RSA rsa = RSA.Create())

{

// Генерация пары открытого и закрытого ключей

RSAParameters publicKey = rsa.ExportParameters(false); // Открытый ключ

RSAParameters privateKey = rsa.ExportParameters(true); // Закрытый ключ

// Шифрование данных с использованием открытого ключа

byte[] dataToEncrypt = Encoding.UTF8.GetBytes("Hello, world!");

byte[] encryptedData = RSAEncrypt(dataToEncrypt, publicKey);

// Расшифровка данных с использованием закрытого ключа

byte[] decryptedData = RSADecrypt(encryptedData, privateKey);

// Вывод расшифрованных данных

Console.WriteLine(Encoding.UTF8.GetString(decryptedData));

}

}

// Метод для шифрования данных с использованием открытого ключа RSA

public static byte[] RSAEncrypt(byte[] dataToEncrypt, RSAParameters publicKey)

{

using (RSA rsa = RSA.Create())

{

rsa.ImportParameters(publicKey);

byte[] encryptedData = rsa.Encrypt(dataToEncrypt, RSAEncryptionPadding.Pkcs1);

return encryptedData;

}

}

// Метод для расшифровки данных с использованием закрытого ключа RSA

public static byte[] RSADecrypt(byte[] encryptedData, RSAParameters privateKey)

{

using (RSA rsa = RSA.Create())

{

rsa.ImportParameters(privateKey);

byte[] decryptedData = rsa.Decrypt(encryptedData, RSAEncryptionPadding.Pkcs1);

return decryptedData;

}

}

}

В этом примере используется класс RSA из пространства имен System.Security.Cryptography для реализации алгоритма шифрования RSA. Метод RSAEncrypt принимает данные для шифрования и открытый ключ в формате RSAParameters, а метод RSADecrypt принимает зашифрованные данные и закрытый ключ в том же формате. Оба метода используют соответствующие методы из класса RSA для выполнения операций шифрования и расшифровки.

Криптография с открытым ключом широко применяется для обмена зашифрованными данными, обеспечения безопасности при передаче информации через незащищенные каналы связи или для создания цифровых подписей. RSA - один из наиболее популярных алгоритмов шифрования с открытым ключом, но существуют и другие, такие как ECC (эллиптическая кривая шифрования) и DSA (цифровая подпись алгоритма).

***Калибр***

Какова сложность написанного тобой алгоритма?

***CHATGPT* 4**

Алгоритм шифрования RSA имеет некоторую вычислительную сложность, которая зависит от размера ключа. В данном примере используется ключ длиной 2048 бит

Сложность алгоритма RSA определяется главным образом операцией возведения в степень с большими числами и вычислением модуля. Сложность шифрования RSA составляет примерно O(n^2), где n - длина ключа. Для расшифровки данных сложность составляет примерно O(n^3).

Таким образом, чем длиннее ключ, тем больше времени потребуется для выполнения операций шифрования и расшифровки.

Стандартные длины ключа для RSA на сегодняшний день обычно равны 2048 или 3072 бит. Длина ключа определяется компромиссом между безопасностью и производительностью: более длинные ключи обеспечивают большую степень безопасности, но требуют больше времени для выполнения операций.

Важно отметить, что в приведенном примере использованы базовые методы шифрования RSA. Для реального применения в безопасных системах рекомендуется использовать библиотеки или фреймворки, которые оптимизированы для работы с криптографией и имеют реализацию, которая обеспечивает высокую степень безопасности и производительности.

**Приложение 2. Доказательство долготы конкатенации строк**

Конкатенация строк - это операция объединения двух или более строк в одну строку. Для доказательства того, что конкатенация строк является долгой операцией, можно рассмотреть несколько аспектов:

1. Временная сложность: Когда производится конкатенация строк, для этого необходимо создать новую строку и скопировать в нее все символы из каждой исходной строки. Это требует выделения новой памяти и последующее копирование символов, что занимает время. В худшем случае, когда строки имеют достаточно большую длину, время, затраченное на выделение памяти и копирование символов, может быть значительным.

С этой точки зрения, конкатенация строк в языке программирования C# имеет линейную сложность, то есть O(n), где n - суммарная длина всех строк, подлежащих конкатенации.

При использовании оператора "+" для конкатенации строк, происходит создание новой строки, содержащей символы из обеих исходных строк. Копирование символов происходит за линейное время, пропорциональное суммарной длине конкатенируемых строк.

Однако, когда конкатенация строк выполняется в цикле или при многократных операциях, сложность может стать квадратичной, то есть O(n^2). Например, если строки конкатенируются n раз в цикле, при каждой итерации происходит создание и копирование новой строки. Такая конкатенация может привести к увеличению времени выполнения в квадратичном масштабе относительно общей длины строк.

2. Память: Конкатенация строк требует выделения нового блока памяти для хранения объединенной строки. Исходные строки не изменяются, и поэтому они по-прежнему занимают память. Если операция конкатенации строк выполняется многократно или для больших строк, это может привести к значительному расходу памяти.

С точки зрения использования памяти, конкатенация строк в языке программирования C# может требовать значительной дополнительной памяти, особенно при многократном выполнении операции или для больших строк.

Каждое объединение строк при использовании оператора "+" порождает новую строку, что приводит к выделению нового блока памяти для хранения объединенной строки. Таким образом, затраты памяти при конкатенации строк будут прямо пропорциональны размеру объединенной строки. Если размеры конкатенируемых строк достигают больших значений или операция выполняется многократно, это может привести к значительному расходу памяти.

Например, предположим, что у нас есть две строки:

string str1 = "123456789";

string str2 = "123456789";

При конкатенации этих строк таким образом:

string result = str1 + str2;

Будет создана новая строка "123456789123456789", которая будет занимать дополнительную память.

Если операция конкатенации строк выполняется многократно, например, в цикле, то при каждой итерации будет создаваться новая строка, что приведет к накоплению отходов памяти и может привести к освобождению памяти через сборку мусора.

3. Неэффективное использование памяти: При каждой операции конкатенации создается новая строка, что может привести к фрагментации памяти и неэффективному использованию доступной памяти.

В целом, хотя время и сложность операции конкатенации строк зависят от контекста, конкатенация строк обычно является долгой операцией, особенно когда строки имеют большую длину или когда операция выполняется в цикле или многократно.