Правительство Российской Федерации
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ
ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
«НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ
«ВЫСШАЯ ШКОЛА ЭКОНОМИКИ»
(НИУ ВШЭ)

Московский институт электроники и математики им. А.Н. Тихонова

ОТЧЕТ
О ПРАКТИЧЕСКОЙ РАБОТЕ № 3
по дисциплине «Криптографические методы защиты информации»

Современные симметричные шифры

Студент гр. БИБ233
Д.А. Гутников
«26» февраля 2025 г.

Руководитель
Заведующий кафедрой информационной безопасности киберфизических систем
канд. техн. наук, доцент
_____О.О. Евсютин
«___» _____ 2025 г.

Москва – 2025 г.

1 ЦЕЛЬ РАБОТЫ

Целью данной работы является приобретение навыков программной реализации современных алгоритмов симметричного шифрования.

2 КРАТКИЕ ТЕОРЕТИЧЕСКИЕ СВЕДЕНИЯ

2.1 Общие сведения о симметричных шифрах

Основное отличие между современными и историческими (классическими) шифрами, к последним из которых относятся все алгоритмы шифрования докомпьютерной эпохи, заключается в способе представления данных. Современные шифры обрабатывают данные в цифровом виде как битовые последовательности, природа которых не имеет значения, в то время как исторические шифры могли работать лишь с текстами, что накладывало определенные ограничения на использовавшийся математический аппарат.

При этом основные криптографические преобразования остались прежними, лишь усложнился способ их реализации (здесь следует отметить, что речь идет исключительно о симметричных шифрах).

Основным методом криптоанализа исторических шифров является статистический криптоанализ, когда криптоаналитик изучает статистику встречаемости символов в шифртексте и, используя полученную информацию, может осуществить дешифрование зашифрованных данных.

До появления компьютерной криптографии данные, которые необходимо было защищать с помощью шифрования, практически всегда представляли собой текст на естественном языке, обладающем высокой избыточностью. Именно высокая избыточность открытых текстов является причиной успешного применения различных статистических атак для взлома классических шифров. В связи с этим К. Шенноном были предложены два основных криптографических метода сокрытия избыточности открытого текста: перемешивание и рассеивание.

Перемешивание устраняет зависимость между открытым текстом и шифртекстом, затрудняет попытки найти в шифртексте избыточность и статистические закономерности. Перемешивание осуществляется с помощью подстановок (замен).

Рассеивание перераспределяет избыточность открытого текста, распространяя ее на весь шифртекст. Каждый бит открытого текста при шифровании должен повлиять на максимальное число других бит открытого текста. Простейшим способом создания рассеивания является перестановка.

По отдельности ни перемешивание, ни рассеивание не могут обеспечить надежного шифрования, поэтому классические подстановочные и перестановочные шифры

неприменимы в современном мире, однако криптосистемы, в которых реализованы оба этих метода, обладают высокой криптостойкостью.

Можно выделить следующие основные элементарные операции над данными, объединение и многократное повторение которых положено в основу всех современных симметричных алгоритмов шифрования:

- замена элементов данных (короткие битовые последовательности) с помощью специальных таблиц;
- перестановка элементов данных;
- битовый сдвиг;
- операции сложения/вычитания по модулю;
- поразрядное сложение по модулю 2.

Современная криптография выделяет два типа симметричных криптосистем: блочные и поточные.

Поточные шифры преобразуют открытый текст в шифртекст последовательно по одному биту путем сложения битов открытого текста с битами гаммы по модулю 2. Гамма необходимой длины формируется из секретного ключа шифрования фиксированной длины. При этом поточный шифр может быть построен на основе блочного шифра при использовании того в специальном режиме.

Блочные шифры обрабатывают открытый текст, разбивая его на блоки равного размера, причем все современные блочные шифры являются раундовыми. Это означает, что зашифрование одного блока открытого текста (расшифрование одного блока шифртекста) заключается в многократном применении к нему некоторой последовательности элементарных криптографических операций. Соответственно, однократное применение такой последовательности к блоку данных называется раундом.

Кратко опишем современные стандарты симметричного шифрования.

2.2 ГОСТ 28147-89

Первый российский стандарт симметричного шифрования описан в нормативном документе «ГОСТ 28147-89. Системы обработки информации. Защита криптографическая. Алгоритм криптографического преобразования». В настоящее время данный стандарт является устаревшим, он утратил силу 31 декабря 2015 г.

ГОСТ 28147-89 является блочным шифром, построенным на основе сети Фейстеля и работающим с блоками длиной 64 бита. Ключ шифрования составляет 256 бит.

ГОСТ 28147-89 предназначен для работы в следующих режимах:

- режим простой замены;
- режим гаммирования;

- режим гаммирования с обратной связью;
- режим выработки имитовставки.

Базовым режимом работы ГОСТ 28147-89 является режим простой замены. В этом режиме блоки открытого текста последовательно и независимо друг от друга преобразуются в блоки шифртекста. Зашифрование одного блока данных происходит в течение 32-х раундов основного криптографического преобразования.

В режиме гаммирования открытый текст, разбитый на 64-битовые блоки, зашифровывается путем поразрядного сложения по модулю 2 с гаммой шифра, которая вырабатывается блоками по 64 бита. Выработка блоков гаммы осуществляется следующим образом. С помощью простого рекуррентного соотношения для каждого блока открытого текста вычисляется некоторое значение той же длины, что и блок. Данное значение зашифровывается в режиме простой замены и образует очередной блок гаммы. Для инициализации данного процесса необходимо задать 64-битовое начальное значение, называемое синхропосылкой.

Режим гаммирования с обратной связью очень похож на режим гаммирования и отличается от него только способом выработки гаммы. Сначала, как и в режиме гаммирования, необходимо задать 64-битовую синхропосылку, которая затем зашифровывается в режиме простой замены. Полученное значение является первым блоком гаммы, который накладывается на первый блок открытого текста путем побитного сложения по модулю 2.

Полученный в результате первый блок шифртекста зашифровывается в режиме простой замены, и результат шифрования используется в качестве второго блока гаммы, то есть побитно складывается по модулю 2 со вторым блоком открытого текста.

Таким образом, каждый блок гаммы, за исключением первого, получается путем шифрования в режиме простой замены предыдущего блока шифртекста.

Рассмотренные режимы работы стандарта шифрования ГОСТ 28147-89 обеспечивают непосредственно шифрование данных, то есть служат для обеспечения конфиденциальности. Кроме того, в ГОСТ 28147-89 предусмотрен режим, предназначенный для решения задачи обеспечения контроля целостности информации, – режим выработки имитовставки

Имитовставка – это контрольная комбинация, зависящая от открытого текста и секретного ключа, использующаяся для обнаружения всех случайных или преднамеренных изменений в открытом тексте.

Выработка имитовставки осуществляется следующим образом: открытый текст разбивается на блоки длиной 64 бита, первый блок зашифровывается 16-ю циклами режима

простой замены, полученное значение побитно складывается по модулю 2 со вторым блоком открытого текста, результат зашифровывается 16 циклами режима простой замены и побитно складывается по модулю 2 с третьим блоком открытого текста. Эти действия повторяются для всех блоков открытого текста, и в итоге мы получим 64-битовое значение. В качестве имитовставки можно взять любое количество бит этого значения.

В режиме выработки имитовставки используется то же основное криптографическое преобразование, что и в режиме простой замены, но течении 16-ти циклов, а не 32-х. Длина ключа так же составляет 256 бит.

Имитовставка добавляется отправителем к защищаемым данным, которые могут передаваться и в открытом виде, если задача обеспечения конфиденциальности не ставится. Получатель, знающий секретный ключ, вычисляет имитовставку и сравнивает вычисленное значение с полученным. Если обнаруживается несовпадение, полученные данные отвергаются как ложные.

Таким образом, для потенциального злоумышленника практически неразрешимы следующие задачи: вычисление имитовставки для заданного открытого текста, и подбор открытого текста для заданного значения имитовставки. Имитовставка обеспечивает невозможность имитации и подмены данных.

2.3 ГОСТ Р 34.12-2015

Действующий российский стандарт симметричного шифрования описан в документе «ГОСТ Р 34.12-2015. Информационная технология. Криптографическая защита информации. Блочные шифры». Данный стандарт был разработан на смену ГОСТ 28147-89 и введен 1 января 2016 г.

ГОСТ Р 34.12-2015 описывает два симметричных блочных шифра:

- шифр «Магма» с длиной блока 64 бита и длиной ключа шифрования 256 бит;
- шифр «Кузнечик» с длиной блока 128 бит и длиной ключа шифрования 256 бит.

Шифр «Магма» представляет собой тот же самый шифр, который был описан в стандарте ГОСТ 28147-89. Отличие заключается лишь в том, что в шифре «Магма» определена фиксированная таблица замен (один из этапов основного криптографического преобразования в данном шифре заключается в замене 4-битовых подблоков блока данных с помощью специальной таблицы), в то время как в ГОСТ 28147-89 данная таблица не определена и ее выбор оставлен на усмотрение разработчиков средств криптографической защиты информации, реализующих соответствующий алгоритм шифрования.

Шифр «Кузнечик» является вновь разработанным шифром. Его создание было обусловлено потребностью в криптографическом алгоритме, работающем с большей длиной блока, нежели 64 бита. В отличие от шифра «Магма» шифр «Кузнечик» основан не

на сети Фейстеля, а на SP-сети (подстановочно-перестановочной сети). Количество раундов работы данного шифра равно 10.

Важным отличием ГОСТ Р 34.12-2015 от ГОСТ 28147-89 является то, что в новом стандарте определены лишь базовые блочные шифры и не определены режимы их работы. Здесь под базовым блочным шифром понимается шифр, реализующий при каждом фиксированном значении ключа одно обратимое отображение множества блоков открытого текста фиксированной длины в блоки шифртекста такой же длины. Поэтому одновременно со стандартом ГОСТ Р 34.12-2015 был введен дополняющий его стандарт, определяющий режимы работы блочных шифров.

2.4 ГОСТ Р 34.13-2015

Данный стандарт описан в документе «ГОСТ Р 34.13-2015. Информационная технология. Криптографическая защита информации. Режимы работы блочных шифров» и определяет следующие режимы работы алгоритмов блочного шифрования:

- режим простой замены (Electronic Codebook, ECB);
- режим гаммирования (Counter, CTR);
- режим гаммирования с обратной связью по выходу (Output Feedback, OFB);
- режим простой замены с зацеплением (Cipher Block Chaining, CBC);
- режим гаммирования с обратной связью по шифртексту (Cipher Feedback, CFB);
- режим выработки имитовставки (Message Authentication Code algorithm).

В режиме простой замены блоки открытого текста последовательно и независимо друг от друга преобразуются в блоки шифртекста. Другими словами, открытый текст разбивается на блоки, и к каждому блоку применяется базовый блочный шифр.

Режим гаммирования, определенный в ГОСТ Р 34.13-2015, очень похож на режим гаммирования, определенный в ГОСТ 28147-89. Зашифрование в режиме гаммирования заключается в покомпонентном сложении открытого текста с гаммой шифра, которая вырабатывается путем зашифрования последовательности значений счетчика базовым алгоритмом блочного шифрования с последующим усечением. Начальное значение счетчика зависит от синхропосылки (в терминологии нового стандарта – вектора инициализации), каждое последующее значение увеличивается на единицу. Таким образом, в режиме гаммирования значение очередного блока гаммы зависит от номера соответствующего блока открытого текста.

В режиме гаммирования с обратной связью по выходу используется регистр сдвига, начальным заполнением которого является значение синхропосылки. В ходе выработки гаммы часть разрядов регистра сдвига зашифровывается с помощью базового блочного шифра и полученное значение после усечения используется в качестве блока гаммы. При

этом заполнение регистра сдвига сдвигается на количество использованных разрядов, а на освободившиеся позиции записывается выходное значение базового блочного шифра. Таким образом, в режиме гаммирования с обратной связью по выходу в формировании очередного блока гаммы участвуют предыдущие блоки гаммы.

В режиме простой замены с зацеплением также используется регистр сдвига. В данном режиме очередной блок шифртекста получается путем зашифрования результата покомпонентного сложения значения очередного блока открытого текста со значением *n* разрядов регистра сдвига с большими номерами. Затем регистр сдвигается на один блок в сторону разрядов с большими номерами. В разряды с меньшими номерами записывается значение блока шифртекста. Другими словами, каждый блок открытого текста перед зашифрованием сцепляется (с помощью поразрядного сложения по модулю 2) с блоком, сформированным из предшествующего шифртекста. Само зашифрование блока открытого текста осуществляется с помощью базового блочного шифра.

Режим гаммирования с обратной связью по шифртексту похож на режим гаммирования с обратной связью по выходу. Отличие заключается в том, что в регистр сдвига записываются блоки шифртекста вместо блоков гаммы. В результате в режиме гаммирования с обратной связью по шифртексту в формировании очередного блока гаммы участвуют предыдущие блоки шифртекста.

Режим выработки имитовставки мало отличается от аналогичного режима, определенного в ГОСТ 28147-89, он имеет то же самое предназначение и схожее устройство.

2.5 DES

Стандартом симметричного шифрования в США долгое время являлся DES (Data Encryption Standard).

В 1972 году американское Национальное бюро стандартов (в настоящее время переименованное в Национальный институт стандартов и технологий), выступило инициатором создания единого стандартного криптографического алгоритма шифрования данных для частного сектора. Такой стандарт был необходим для обеспечения возможности согласованной работы криптографических средств от различных производителей. Было проведено два конкурса, победителем второго из которых стала разработка фирмы IBM, выполненная исследовательской группой под руководством Хорста Фейстеля.

В конце 1976 года DES был принят в качестве федерального стандарта и разрешен к использованию, в начале 1977 года было опубликовано официальное описание стандарта FIPS PUB 46.

DES представляет собой блочный шифр, работающий с 64-битовыми блоками. Длина ключа составляет 64 бита, но каждый восьмой бит используется для проверки четности, поэтому реальная длина ключа шифрования равна 56 битам.

Шифрование заключается в многократном повторении основного криптографического преобразования, называемого раундом или циклом.

В основе DES лежит сеть Фейстеля, названная так по имени ее создателя. Можно сказать, что ГОСТ 28147-89 и DES по своим конструктивным особенностям являются представителями одного семейства шифров.

2.6 AES

За годы, прошедшие с момента создания DES, он подвергался многочисленным исследованиям и попыткам криптоанализа. Были разработаны такие методы криптоанализа как дифференциальный и линейный, проведены успешные атаки на некоторые варианты DES с меньшим числом циклов. Но в целом все проведенные исследования подтвердили надежность DES. Поэтому главным его недостатком стала длина ключа. Быстрое развитие средств вычислительной техники уже в первой половине 90-х годов XX века сделало возможным полный перебор всех ключей DES за реальное время.

В связи с этим в 1997 году НИСТ объявил конкурс на создание нового общенационального стандарта шифрования данных, который должен был прийти на замену DES. На конкурс были представлены 15 алгоритмов симметричного шифрования из разных стран, из которых затем отобрали 5 финалистов. В итоге лучшим был признан шифр Rijndael, разработанный двумя бельгийскими криптографами Винсентом Рэйменом и Йоаном Дайменом. В 2001 году опубликован стандарт FIPS PUB 197, содержащий улучшенный стандарт шифрования данных AES (Advanced Encryption Standard), основанный на Rijndael.

Одним из требований к новому стандарту была возможность работы с информационными блоками и ключами шифрования различной длины. Поэтому Rijndael был сконструирован как шифр с переменной длиной блока и переменной длиной ключа, которые независимо могут приниматься равными 128, 192 и 256 бит. AES представляет собой усеченную версию шифра Rijndael с варьируемой длиной ключа 128, 192 или 256 бит, но с постоянной длиной блока 128 бит.

AES является итерационным шифром, что означает, как уже было сказано, многократное повторение основного криптографического преобразования. При этом число раундов не фиксировано, а зависит от длины ключа: 10 раундов для ключа длиной 128 бит, 12 раундов для ключа длиной 192 бита и 14 раундов для ключа длиной 256 бит.

3 Выбор языка для программной реализации

Выбор был сделан в пользу C#, поскольку это современный язык программирования, а также содержит типы, позволяющие работать с числами в поле остатков от деления на 2^{32} и с числами в контексте битовых последовательностей.

4 Тестирование программной реализации

Протестируем данную реализацию шифра Магма на основе текстового файла text.txt (рисунок 1).

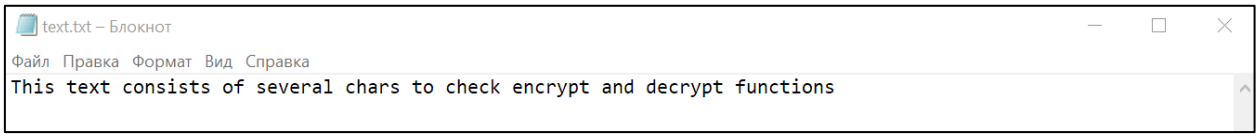

Рисунок 1. Файл открытого текста.

Выберем 8 32-битных ключей для шифрования, зашифруем и расшифруем данную символьную последовательность с целью проверки данной программной реализации. Ключи представлены на рисунке 2.

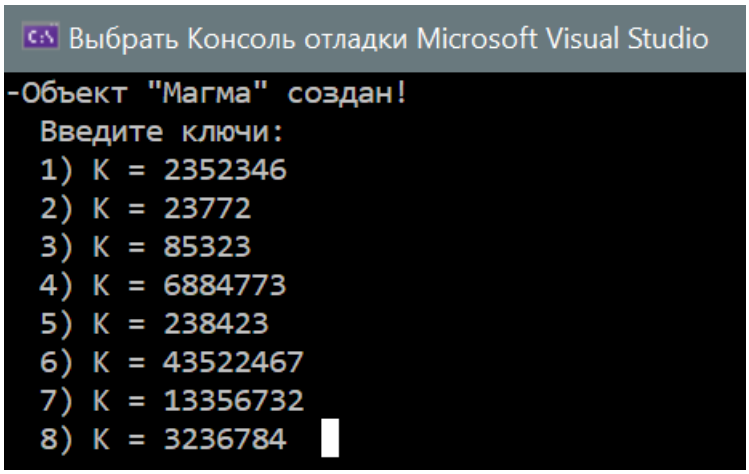

Рисунок 2. Ключи шифрования.

В результате работы программы в папке исполняемого файла появились новые файлы: encrypted_txt.mgm и decrypted_txt.txt (рисунок 3).

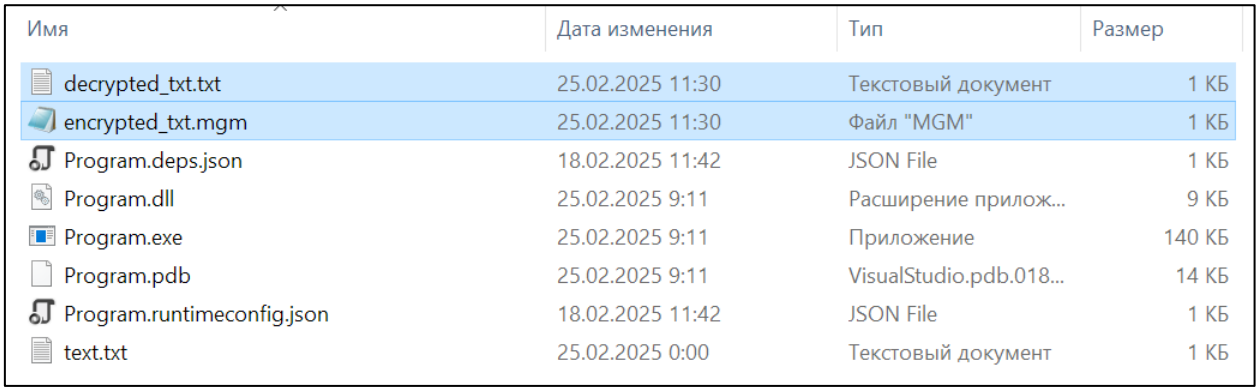

Рисунок 3. Новые файлы в папке исполняемого файла.

Если открыть эти файлы и изучить содержимое, то можно посмотреть на результат работы программы (рисунок 4 и рисунок 5).

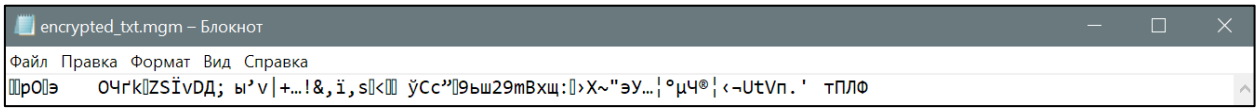

Рисунок 4. Содержимое зашифрованного файла.

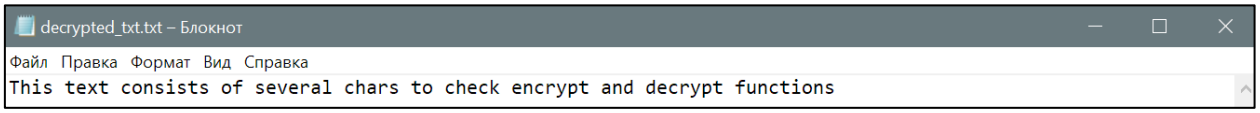

Рисунок 5. Содержимое расшифрованного файла.

Стоит отметить, что во время работы программы возможны коллизии из-за конфликта кодировок с некоторыми кодами, полученными в результате зашифрования, что неоднократно случалось во время тестирования.

5 Вывод по проделанной работе

В результате практической работы была написана программная реализация шифра Магма и проведено зашифрование и расшифрование некоторого сообщения.

6 Источники

1. Материалы Smart LMS
2. Хабр – Реализация шифра «Магма» на языке RUST URL=https://habr.com/ru/articles/866574/

7 Приложение

Program.cs:

```
namespace Program
{
    public static class Program
    {
        public static void Main()
        {
            Magma magma = new Magma();

            string file = "text.txt";
            string destination = "encrypted_txt.mgm";
            string destination2 = "decrypted_txt.txt";

            magma.EncryptFile(file, destination);

            magma.DecryptFile(destination, destination2);
        }
    }
}
```

Magma.cs:

```
namespace Program
{
    public class Magma
```

```
{
    public uint[] Keys = new uint[32];
    private uint[,] TransformTable = new uint[8, 16]
    {
        {12, 4, 6, 2, 10, 5, 11, 9, 14, 8, 13, 7, 0, 3, 15, 1},
        {6, 8, 2, 3, 9, 10, 5, 12, 1, 14, 4, 7, 11, 13, 0, 15},
        {11, 3, 5, 8, 2, 15, 10, 13, 14, 1, 7, 4, 12, 9, 6, 0},
        {12, 8, 2, 1, 13, 4, 15, 6, 7, 0, 10, 5, 3, 14, 9, 11},
        {7, 15, 5, 10, 8, 1, 6, 13, 0, 9, 3, 14, 11, 4, 2, 12},
        {5, 13, 15, 6, 9, 2, 12, 10, 11, 7, 8, 1, 4, 3, 14, 0},
        {8, 14, 2, 5, 6, 9, 1, 12, 15, 4, 11, 0, 13, 10, 3, 7},
        {1, 7, 14, 13, 0, 5, 8, 3, 4, 15, 10, 6, 9, 12, 11, 2}
    };

    public Magma()
    {
        Console.WriteLine("-Объект \"Магма\" создан!");

        Console.WriteLine("  Введите ключи:");
        for (int i = 0; i < 8; i++)
        {
            Console.Write("  {0}) K = ", i + 1);
            Keys[i] = (uint)Int32.Parse(Console.ReadLine());
            Keys[i + 8] = Keys[i];
            Keys[i + 16] = Keys[i];
            Keys[31 - i] = Keys[i];
        }
    }

    public Number64 EncryptNumberBlock(Number64 block)
    {
        for (int i = 0; i < 32; i++)
        {
            uint current_key = Keys[i];
            block = Round(block, current_key);
        }

        // Поменять местами левую и правую части
        uint temp = block.left;
        block.left = block.right;
        block.right = temp;

        return block;
    }

    public string Encrypt(string input)
    {
        ulong[] numbers = TextHelper.ConvertToLong(input);
        ulong[] numbers2 = new ulong[numbers.Length];

        for (int i = 0; i < numbers.Length; i++)
        {
            numbers2[i] = (ulong)EncryptNumberBlock(numbers[i]);
        }

        return TextHelper.ConvertToString(numbers2);
    }

   public void EncryptFile(string file, string destination)
    {
        if (File.Exists(file))
        {
            byte[] file_bytes = File.ReadAllBytes(file);

            string text = System.Text.Encoding.Unicode.GetString(file_bytes);
```

```
        string encrypted = Encrypt(text);

        byte[] bytes = System.Text.Encoding.Unicode.GetBytes(encrypted);

        File.WriteAllBytes(destination, bytes);
    }
}

public Number64 DecryptNumberBlock(Number64 block)
{
    for (int i = 31; i >= 0; i--)
    {
        uint current_key = Keys[i];
        block = Round(block, current_key);
    }

    // Поменять местами левую и правую части
    uint temp = block.left;
    block.left = block.right;
    block.right = temp;

    return block;
}

public string Decrypt(string input)
{
    ulong[] numbers = TextHelper.ConvertToLong(input);
    ulong[] numbers2 = new ulong[numbers.Length];

    for (int i = 0; i < numbers.Length; i++)
    {
        numbers2[i] = (ulong)DecryptNumberBlock(numbers[i]);
    }

    return TextHelper.ConvertToString(numbers2);
}

public void DecryptFile(string file, string destination)
{
    if (File.Exists(file))
    {
        byte[] file_bytes = File.ReadAllBytes(file);

        string text = System.Text.Encoding.Unicode.GetString(file_bytes);

        string decrypted = Decrypt(text);

        byte[] bytes = System.Text.Encoding.Unicode.GetBytes(decrypted);

        File.WriteAllBytes(destination, bytes);
    }
}

public uint G(uint number, uint key)
{
    number += key;
    number = T_full(number);
    number = RotateLeft(number);

    return number;
}

public uint RotateLeft(uint number)
{
```

```
            return (number << 11) | (number >> (32 - 11));
        }

        public Number64 Round(Number64 number, uint current_key)
        {
            uint left = number.left;
            uint right = number.right;
            uint right_memory = right;

            right = G(right, current_key);
            right ^= left;

            left = right_memory;
            ulong new_number = ((ulong)left << 32) + (ulong)right;
            number.Recreate(new_number);

            return number;
        }

        public uint T_full(uint number)
        {
            int step = 28;
            uint ones = 15;
            uint current;
            for (int i = 0; i < 8; i++)
            {
                current = number >> step;
                current &= 15;              // Взяли последние 4 бита

                current = T(current, 7-i);
                number &= ~(ones << step);    // Занулили единицы, где надо
                number |= current << step;  // Вставили единицы, где надо

                step -= 4;
            }

            return number;
        }

        public uint T(uint number, int row)
        {
            return TransformTable[row, number];
        }
    }
}
```

Number64.cs:

```
namespace Program
{
    public class Number64
    {
        private uint left_;
        private uint right_;

        public Number64()
        {

        }

        public Number64(uint left, uint right)
        {
            this.left_ = left;
            this.right_ = right;
        }

        public uint left
```

```
        {
            get { return left_; }
            set { left_ = value; }
        }

        public uint right
        {
            get { return right_; }
            set { right_ = value; }
        }

        public Number64 Recreate(ulong number)
        {
            this.left = (uint)(number >> 32);
            this.right = (uint)number;

            return this;
        }

        public override string ToString()
        {
            return this.left + "" + this.right;
        }

        public static implicit operator Number64(ulong x)
        {
            uint left = (uint)(x >> 32);
            uint right = (uint)x;

            return new Number64(left, right);
        }

        public static explicit operator ulong(Number64 x)
        {
            return (ulong)(x.left_) << 32 | x.right_;
        }
    }
}
```

TextHelper.cs:

```
using System.Text;

namespace Program
{
    public static class TextHelper
    {
        public static ulong[] ConvertToLong(string text)
        {
            int length = text.Length;
            int ulongCount = (length + 3) / 4;
            ulong[] numbers = new ulong[ulongCount];
            int currentIndex = 0;
            int shift = 48;

            foreach (char c in text)
            {
                numbers[currentIndex] |= (ulong)c << shift;
                shift -= 16;

                if (shift < 0)
                {
                    currentIndex++;
                    shift = 48;
                }
            }
```

```
            return numbers;
        }

        public static string ConvertToString(ulong[] numbers)
        {
            StringBuilder sb = new StringBuilder();

            foreach (ulong number in numbers)
            {
                for (int shift = 48; shift >= 0; shift -= 16)
                {
                    ushort charValue = (ushort)((number >> shift) & 0xFFFF);
                    sb.Append((char)charValue);
                }
            }

            // Удаляем нули в конце строки
            string result = sb.ToString();
            result = result.TrimEnd('\0'); // Удаляем все нулевые символы в конце

            return result;
        }

        public static string OpenFile(string path)
        {
            if (File.Exists(path))
            {
                Console.WriteLine("File found!");
                return File.ReadAllText(path);
            }
            else
            {
                Console.WriteLine("No file found");
                return null;
            }
        }
    }
}
```