Учреждение образования «БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ  
ТЕХНОЛОГИЧЕСКИЙ УНИВЕРСИТЕТ»

Факультет информационных технологий

[Кафедра информационных](https://www.belstu.by/fakultety/fit/vm) систем и технологий

Специальность 1-40 01 01 Программное обеспечение информационных технологий

**Отчёт по лабораторной работе №7**

по дисциплине Информационная безопасность

Тема: Исследование блочных шифров

Студент: Станчик М.А.

ФИТ курс 3 группа 4

Преподаватель: Савельева М.Г.

Минск, 2025

**Лабораторная работа №7**

**Цель:** изучение и приобретение практических навыков разработки и использования приложений для реализации блочных шифров.

**Задачи:**

1. Закрепить теоретические знания по алгебраическому описанию, алгоритмам реализации операций зашифрования/расшифрования и оценке криптостойкости блочных шифров.

2. Разработать приложение для реализации указанных преподавателем методов блочного зашифрования/расшифрования.

3. Выполнить анализ криптостойкости блочных шифров.

4. Оценить скорость зашифрования/расшифрования реализованных шифров.

5. Результаты выполнения лабораторной работы оформить в виде описания разработанного приложения, методики выполнения экспериментов с использованием приложения и результатов эксперимента

# Теоретические сведения

Все стандарты и алгоритмы блочных шифров (БШ) строятся на основе подстановочных и перестановочных шифров, т. е. являются комбинационными. БШ относятся также к классу симметричных.

Блочное зашифрование (расшифрование) предполагает разбиение исходного открытого (зашифрованного) текста на равные блоки, к которым применяется однотипная процедура зашифрования (расшифрования). Указанная однотипность характеризуется прежде всего тем, что процедура зашифрования (расшифрования) состоит из совокупности повторяющихся наборов преобразований, называемых раундами.

В основе криптостойкости блочных шифров лежит идея К. Шеннона о представлении составного шифра таким образом, чтобы он обладал двумя важными свойствами: рассеиванием и перемешиванием. Рассеивание должно скрыть отношения между зашифрованным текстом и исходным текстом. Рассеивание подразумевает, что каждый символ (или бит) в зашифрованном тексте зависит от одного или всех символов в исходном тексте. Другими словами, если единственный символ в исходном тексте изменен, несколько или все символы в зашифрованном тексте будут также изменены.  Идея относительно перемешивания заключается в том, что оно должно скрыть отношения между зашифрованным текстом и ключом

Алгоритм DES строится на основе сети Фейстеля. Входной блок данных, состоящий из 64 битов, преобразуется в выходной блок идентичной длины. В алгоритме широко используются рассеивания (подстановки) и перестановки битов текста. Комбинация двух указанных методов преобразования образует фундаментальный строительный блок DES, называемый раундом или циклом.

Один блок данных подвергается преобразованию (и при зашифровании, и при расшифровании) в течение 16 раундов. После первоначальной перестановки и разделения 64-битного блока данных на правую (R0) и левую (L0) половины длиной по 32 бита выполняются 16 раундов одинаковых действий.

# 2 Описание алгоритма DES-EDE2

Существует модификация DES, называемая DES-EDE2. Double Encryption with Two Keys — это модификация стандартного алгоритма DES (Data Encryption Standard), использующая двойное шифрование с двумя разными ключами для повышения безопасности. Смысл модификации в том, что алгоритм DES выполняется три раза, но используется только два ключа, причем при первой и третьей итерации используется один и тот же ключ.

Процесс шифрования:

* Первый этап: исходный текст шифруется с использованием первого ключа (*K1*) алгоритмом DES,
* Второй этап: результат первого этапа дешифруется с использованием второго ключа (*K2*) алгоритмом DES;
* Третий этап: результат второго этапа снова шифруется с использованием первого ключа (*K1*) алгоритмом DES.

Процесс дешифрования:

* Первый этап: исходный зашифрованный текст шифруется с использованием первого ключа (*K1*),
* Второй этап: результат первого этапа дешифруется с использованием второго ключа (*K2*);
* Третий этап: результат второго этапа снова шифруется с использованием первого ключа (*K1*).

Также преимуществом DES-EDE2 является сохранение совместимости

с существующими системами, использующими DES.

# 3 Описание приложения, реализующего алгоритм DES-EEE2

Для выполнения шифрования и дешифрования с помощью алгоритма DES-EEE2 использовалась библиотека System.Security.Cryptography из .NET Framework для выполнения операции шифрования и дешифрования. Используется класс TripleDESCryptoServiceProvider, который реализует алгоритм шифрования Triple DES.

Метод Encrypt принимает как параметры строку, представляющую первый ключ для шифрования (должен быть длиной 16 байт), строку, представляющую второй ключ для шифрования (должен быть длиной 16 байт) и строку, которую необходимо зашифровать. Возвращает зашифрованное сообщение в виде строки, закодированной в Base64. Метод предназначен для шифрования текстового сообщения с использованием алгоритма DES-EDE2, обеспечивая повышенный уровень безопасности по сравнению с обычным DES. Код метода представлен в листинге 3.1.

public static string process(string keyOne, string keyTwo, string message)

{

using (TripleDESCryptoServiceProvider tdes = new TripleDESCryptoServiceProvider())

{

tdes.Mode = CipherMode.ECB;

tdes.Padding = PaddingMode.Zeros;

byte[] key1 = Encoding.ASCII.GetBytes(keyOne);

byte[] key2 = Encoding.ASCII.GetBytes(keyTwo);

string encryptedMessage = "";

using (ICryptoTransform encryptor1 = tdes.CreateEncryptor(key1, new byte[8]))

using (ICryptoTransform decryptor2 = tdes.CreateDecryptor(key2, new byte[8]))

using (ICryptoTransform encryptor3 = tdes.CreateEncryptor(key1, new byte[8]))

{

byte[] encryptedBytes = encryptor1.TransformFinalBlock(Encoding.UTF8.GetBytes(message), 0, message.Length);

byte[] decryptedBytes = decryptor2.TransformFinalBlock(encryptedBytes, 0, encryptedBytes.Length);

byte[] reencryptedBytes = encryptor3.TransformFinalBlock(decryptedBytes, 0, decryptedBytes.Length);

encryptedMessage = Convert.ToBase64String(reencryptedBytes);

Console.WriteLine("Зашифрованное сообщение: " + encryptedMessage);

}

return encryptedMessage;

}

}

Листинг 3.1 – Код метода шифрования

Для дешифрования была разработан метод Decrypt. Принимает как параметры строку, представляющую первый ключ для дешифрования (должен быть длиной 16 байт), строку, представляющую второй ключ для дешифрования (должен быть длиной 16 байт), строку, содержащую зашифрованное сообщение в формате Base64. Возвращает расшифрованное сообщение в виде строки. Метод предназначен для восстановления оригинального сообщения из зашифрованного текста, обеспечивая корректное применение алгоритма DES-EDE2. Код метода представлен в листинге 3.2.

public static string Decrypt(string keyOne, string keyTwo, string message)

{

string decryptedMessage = "";

using (TripleDESCryptoServiceProvider tdes = new TripleDESCryptoServiceProvider())

{

tdes.Mode = CipherMode.ECB;

tdes.Padding = PaddingMode.Zeros;

byte[] key1 = Encoding.ASCII.GetBytes(keyOne);

byte[] key2 = Encoding.ASCII.GetBytes(keyTwo);

using (ICryptoTransform decryptor1 = tdes.CreateDecryptor(key1, new byte[8]))

using (ICryptoTransform encryptor2 = tdes.CreateEncryptor(key2, new byte[8]))

using (ICryptoTransform decryptor3 = tdes.CreateDecryptor(key1, new byte[8]))

{

byte[] encryptedBytes = Convert.FromBase64String(message);

byte[] decryptedBytes = decryptor1.TransformFinalBlock(encryptedBytes, 0, encryptedBytes.Length);

byte[] reencryptedBytes = encryptor2.TransformFinalBlock(decryptedBytes, 0, decryptedBytes.Length);

byte[] finalDecryptedBytes = decryptor3.TransformFinalBlock(reencryptedBytes, 0, reencryptedBytes.Length);

decryptedMessage = Encoding.UTF8.GetString(finalDecryptedBytes);

Console.WriteLine("Расшифрованное сообщение: " + decryptedMessage);

}

}

return decryptedMessage;

}

Листинг 3.2 – Код метода дешифрования

Для замера времени выполнения операций шифрования и дешифрования используется класс Stopwatch из пространства имен System.Diagnostics. В приложении используются обычные, слабые и полуслабые ключи для шифрования и дешифрования. Код вызовов методов и замера их выполнения приведен в листинге 3.3.

public class Program

{

public static string originalMessage = "Hello my name is Maxim Stanchik";

public static void Main()

{

Console.WriteLine("//---------- Использование обычных ключей 1 ----------//");

Stopwatch sw = new Stopwatch();

Console.WriteLine($"Изначальный текст: {originalMessage}");

string firstKey = "1110110110101100";

string secondKey = "1110111001010101";

sw.Start();

string encryptMessage = DESS.Encrypt(firstKey, secondKey, originalMessage);

sw.Stop();

Console.WriteLine($"Время шифрования: {sw.Elapsed.TotalMilliseconds} ms");

sw.Restart();

originalMessage = DESS.Decrypt(firstKey, secondKey, encryptMessage);

sw.Stop();

Console.WriteLine($"Время расшифрования: {sw.Elapsed.TotalMilliseconds} ms");

Console.WriteLine("Размер текста: " + (originalMessage).Length \* 8);

Console.WriteLine("Размер зашифрованного текста: " + (encryptMessage).Length \* 6);

Console.WriteLine("//---------- Использование обычных ключей 2 ----------//");

sw = new Stopwatch();

Console.WriteLine($"Изначальный текст: {originalMessage}");

firstKey = "1110110110101101";

secondKey = "1110111001010101";

sw.Start();

encryptMessage = DESS.Encrypt(firstKey, secondKey, originalMessage);

sw.Stop();

Console.WriteLine($"Время шифрования: {sw.Elapsed.TotalMilliseconds} ms");

sw.Restart();

originalMessage = DESS.Decrypt(firstKey, secondKey, encryptMessage);

sw.Stop();

Console.WriteLine($"Время расшифрования: {sw.Elapsed.TotalMilliseconds} ms");

Console.WriteLine("Размер текста: " + (originalMessage).Length \* 8);

Console.WriteLine("Размер зашифрованного текста: " + (encryptMessage).Length \* 6);

Console.WriteLine("//---------- Использование слабых ключей ----------//");

sw = new Stopwatch();

Console.WriteLine($"Изначальный текст: {originalMessage}");

firstKey = "1F1F1F1F1F1F1F1F";

secondKey = "FEFEFEFEFEFEFEFE";

sw.Start();

encryptMessage = DESS.Encrypt(firstKey, secondKey, originalMessage);

sw.Stop();

Console.WriteLine($"Время шифрования: {sw.Elapsed.TotalMilliseconds} ms");

sw.Restart();

originalMessage = DESS.Decrypt(firstKey, secondKey, encryptMessage);

sw.Stop();

Console.WriteLine($"Время расшифрования: {sw.Elapsed.TotalMilliseconds} ms");

Console.WriteLine("Размер текста: " + (originalMessage).Length \* 8);

Console.WriteLine("Размер зашифрованного текста: " + (encryptMessage).Length \* 6);

Console.WriteLine("//---------- Использование полуслабых ключей ----------//");

sw = new Stopwatch();

Console.WriteLine($"Изначальный текст: {originalMessage}");

firstKey = "FEE0FEE0FEE0FEE0";

secondKey = "0E010E010E010E01";

sw.Start();

encryptMessage = DESS.Encrypt(firstKey, secondKey, originalMessage);

sw.Stop();

Console.WriteLine($"Время шифрования: {sw.Elapsed.TotalMilliseconds} ms");

sw.Restart();

originalMessage = DESS.Decrypt(firstKey, secondKey, encryptMessage);

sw.Stop();

Console.WriteLine($"Время расшифрования: {sw.Elapsed.TotalMilliseconds} ms");

Console.WriteLine("Размер текста: " + (originalMessage).Length \* 8);

Console.WriteLine("Размер зашифрованного текста: " + (encryptMessage).Length \* 6);

}

}

Листинг 3.3 – Код вызовов методов и замера их выполнения

Результат работы приложения с исходным текстом «Hello my name is Maxim Stanchik» приведен на рисунке 3.1.

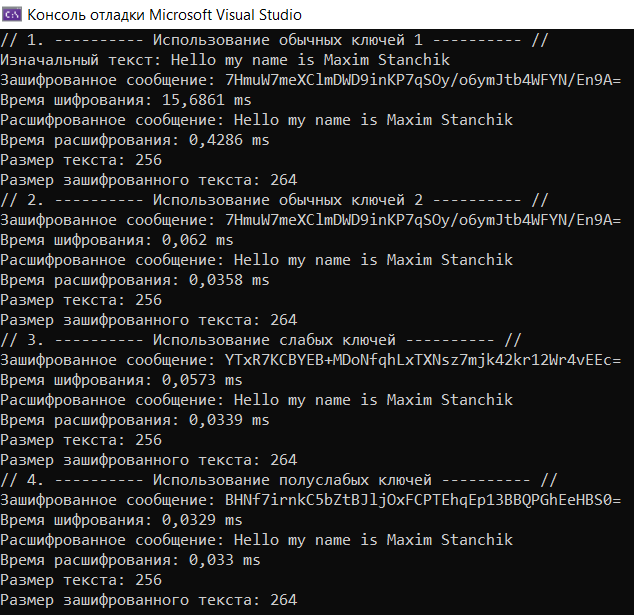


Рисунок 3.1 – Результат работы приложения

Текст, который шифруется, а потом расшифровывается, соответствует исходному тексту, следовательно, приложение работает правильно.

# Оценка лавинного эффекта

Лавинный эффект – криптографическое свойство для шифрования, которое означает, что изменение значения малого количества битов во входном тексте или в ключе ведет к «лавинному» изменению значений выходных битов шифротекста. Другими словами, это зависимость всех выходных битов от каждого входного бита. При реализации приложения в первый раз шифрование происходит ключами «1110110110101101» и «1110111001010101», а во второй раз изменим второй ключ на один символ (во второй раз ключ равен «1110111001010100»), а первый ключ оставим без изменений. Результат выполнения программы на рисунке 4.1.

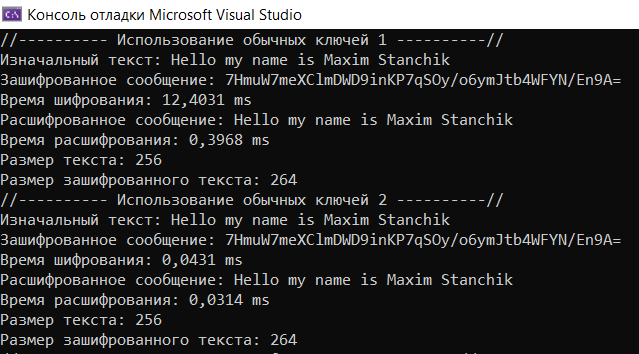


Рисунок 4.1 – Демонстрация лавинного эффекта

Из рисунка видно, что при изменении ключа всего лишь на один символ, благодаря лавинному эффекту, зашифрованные сообщения получаются абсолютно разными.

Получилось, что у нас поменялся не один символ сообщения, а сообщение целиком, что повышает крипкостойкость,  так как криптоаналитик не может сделать предположение о входной информации, основываясь на выходной.

Вспомним, что первоначальное значение разделяется на две половины, каждая из которых сдвигается независимо. Если все биты каждой половины равны 0 или 1, то для всех раундов алгоритма используется один и тот же ключ. Это может произойти, если ключ состоит из одних 1, из одних 0, или если одна половина ключа состоит из одних 1, а другая – из одних 0. Кроме того, некоторые пары ключей при зашифровании переводят открытый текст в идентичный шифртекст. Иными словами, один из ключей пары может расшифровать сообщения, зашифрованные другим ключом пары.

Эти ключи, называемые полуслабыми. Для примера возьмем ключ для шифрования, который равен «1010101010101010» и для расшифрования «1111111100000000». Оба ключа являются полуслабыми. Результат представлен на рисунке 4.2.

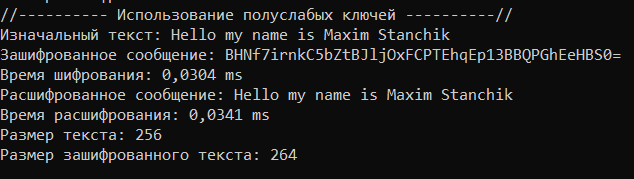


Рисунок 4.2 – Полуслабые ключи

Также осуществлялось шифрование с использованием слабых ключей. Результат приведен на рисунке 4.3.

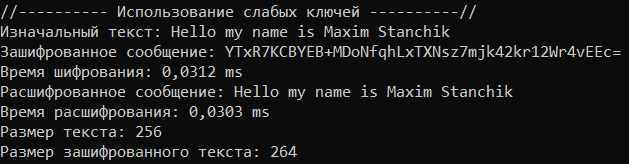


Рисунок 4.3 – Слабые ключи

Судя по результатам на рисунках 4.2 и 4.3 можно сделать вывод, что сообщение, зашифрованное одним ключом и расшифрованное другим, является эквивалентным исходному сообщению, в связи с этим рекомендуется использовать сильные ключи для шифрования текста.

Эффект лавины значительно уменьшается с использованием слабых и полуслабых ключей, поскольку при разделении первоначального значения ключа на две половины, получаются части, в которые все биты равны 0 или 1, которые в последующем не изменяются в каждом раунде алгоритма. В случае полуслабых ключей, каждый из подключей используется 8 раз, что также снижает действие лавинного эффекта.

# Время выполнения зашифрования и расшифрования

Для оценки времени выполнения операций зашифрования и расшифрования построим графики при шифровании алгоритмом DES-EEE2 с разной длиной входных текстов. График представлен на рисунке 5.1.

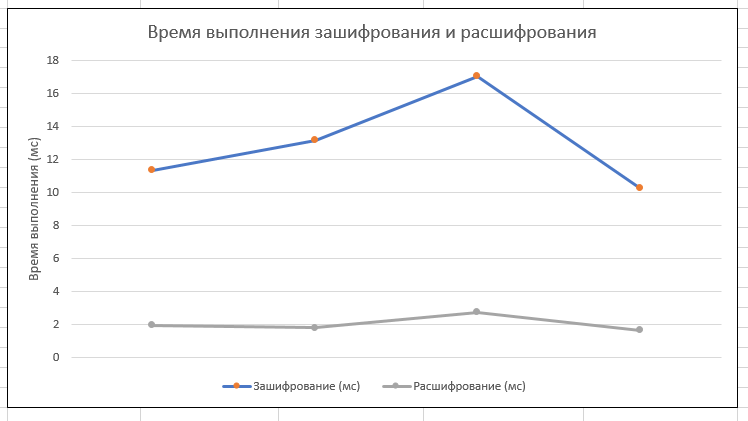


Рисунок 5.1 – Время выполнения расшифрования и зашифрования

При сравнении данного графика с временем выполнения рассмотренных ранее шифров можно заметить, что алгоритм DES-EDE2 выполняет операции зашифрования и расшифрования примерно в 7­–10 раз медленнее, чем рассмотренные ранее алгоритмы шифрования. Однако, данный алгоритм предоставляет более высокую криптостойкость в сравнении с остальными ранее рассмотренными алгоритмами. Шифрование и дешифрование алгоритмом DES выполняется достаточно быстро из-за малой длины ключёа.

# Оценка степени сжатия шифротекста

Одним из свойств алгоритма DES является то, что шифротекст практически нельзя сжать. Это происходит потому, что шифрование преобразует информацию таким образом, что в итоге создается псевдослучайная последовательность битов. Эта последовательность битов не имеет никакой структуры, что делает ее сложно поддающейся современным алгоритмам сжатия.

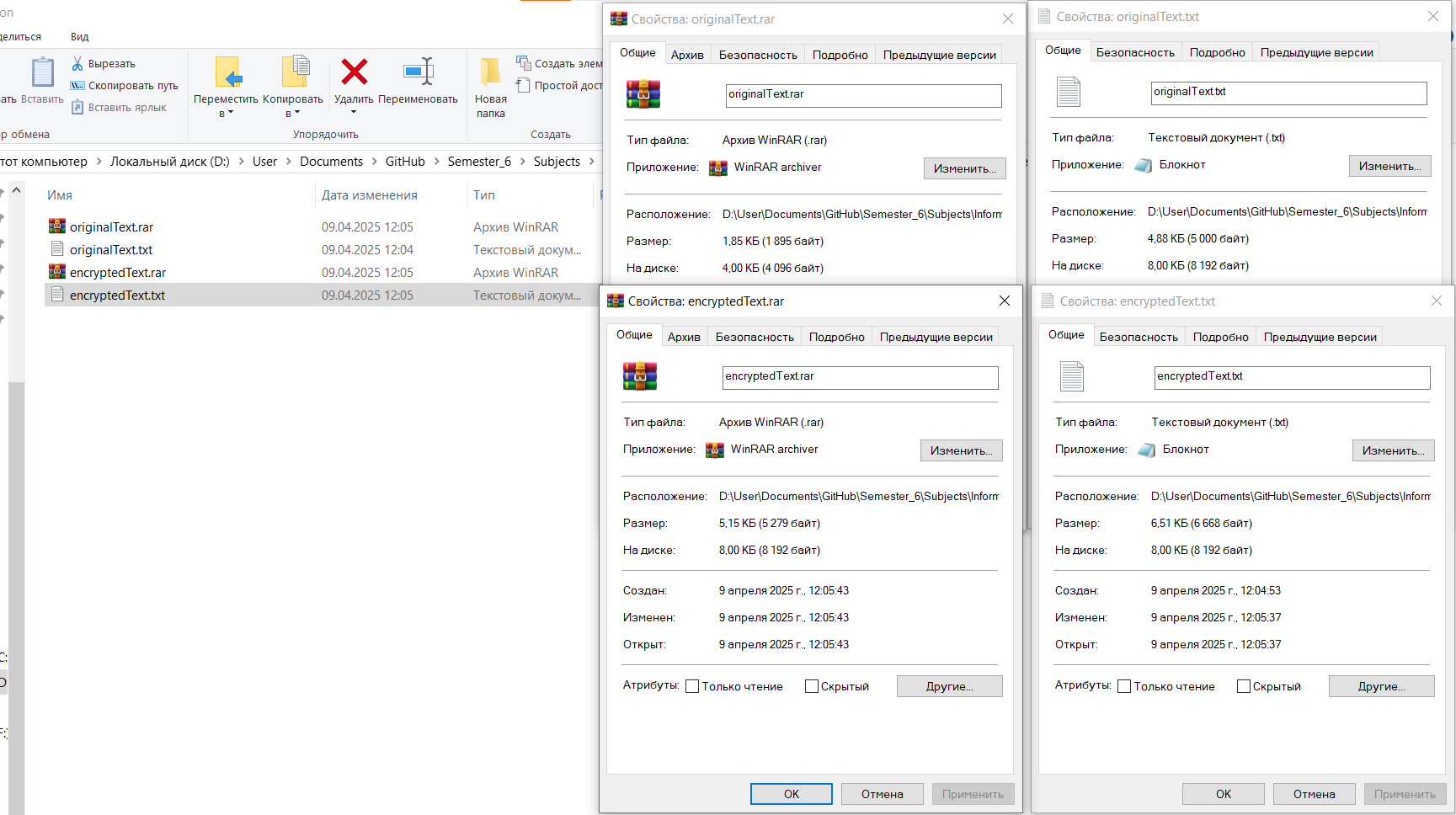


Рисунок 6.1 – Сжатие открытого текста и шифртекста

Из рисунка можно получить следующие данные: текстовый документ с незашифрованным текстом весит 5000 байт, документ с шифртекстом уже 6668 байт. Архив, содержащий текстовый документ с шифртекстом, весит 5279 байт, то есть сжатие уменьшило изначальный вес на 20,83 %. Архив, содержащий текстовый документ с обычным текстом, весит 1895 байт, то есть сжатие уменьшило изначальный вес на 62,10 %.

**Вывод:** в ходе лабораторной работы были изучены блочные шифры, в частности алгоритм DES и его модификация DES-EDE2. Разработанное приложение продемонстрировало шифрование и дешифрование с использованием двух ключей, что повысило безопасность по сравнению с классическим DES. Анализ показал, что слабые и полуслабые ключи значительно снижают защитные свойства шифрования, подчеркивая важность выбора надежных ключей. Хотя DES-EDE2 работает медленнее, он обеспечивает более высокий уровень безопасности, что подтверждается лавинным эффектом при изменении ключей. Шифротекст не поддается эффективному сжатию, и рекомендуется исследовать современные алгоритмы, такие как AES, для дальнейших улучшений.