Учреждение образования

«Белорусский государственный технологический университет»

**Отчет по лабораторной работе №5**

по дисциплине: «Защита информации и надежность информационных систем»

Исполнитель:

Воликов Д. А., ФИТ 4-7

Руководитель:

Асс. Савельева М. Г.

Минск 2024

# Лабораторная работа №5. Исследование блочных шифров

**Цель**: изучение и приобретение практических навыков разработки и использования приложений для реализации блочных шифров (рассчитана на 4 часа аудиторных занятий).

**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 отчета о проведенных исследованиях, методике выполнения практической части задания и оценке криптостойкости шифров.

## Теоретические сведения

В 1972 г. Национальное бюро стандартов США инициировало программу защиты каналов связи и компьютерных данных. Одна из целей – разработка единого стандарта криптографического шифрования. Основными критериями оценки алгоритма являлись следующие:

* алгоритм должен обеспечить высокий уровень защиты;
* алгоритм должен быть понятен и детально описан;
* криптостойкость алгоритма должна зависеть только от
* ключа;
* алгоритм должен допускать адаптацию к различным приме-
* нениям;
* алгоритм должен быть разрешен для экспорта.

В качестве начального варианта нового алгоритма рассматривался Lucifer– разработка компании IBM начала семидесятых годов. В основе указанного алгоритма использовались два запатентованных в 1971 г. Хорстом Фейстелем (Horst Feistel) устройства, реализующие различные алгоритмы шифрования, позже получившие название шифр (сеть) Фейстеля (Feistel Cipher, Feistel Network). В первой версии проекта Lucifer сеть Фейстеля не использовалась.

Алгоритм DES. Входной блок данных, состоящий из 64 битов, преобразуется в выходной блок идентичной длины. В алгоритме широко используются рассеивания (подстановки) и перестановки битов текста, о которых мы упоминали выше. 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.

Левая и правая ветви каждого промежуточного значения обрабатываются как отдельные 32-битные значения, обозначенные *Li* и *Ri*.

Вначале правая часть блока *Ri* расширяется до 48 битов с использованием таблицы, которая определяет перестановку плюс расширение на 16 битов. Эта операция приводит размер правой половины в соответствие с размером ключа для выполнения операции XOR. Кроме того, за счет выполнения этой операции быстрее возрастает зависимость всех битов результата от битов исходных данных и ключа (это называется «лавинный эффект»).

После выполнения перестановки с расширением для полученного 48-битного значения выполняется операция XOR с 48-битным подключом *Ki*. Затем полученное 48-битное значение подается на вход блока подстановки *S*, результатом которой является 32-битное значение. Подстановка выполняется в восьми блоках подстановки или восьми *S*-блоках (*S*-boxes). При выполнении этой операции 48 битов данных делятся на восемь 6-битных подблоков, каждый из которых по соответствующей таблице замен замещается четырьмя битами. Подстановка с помощью S-блоков является одним из важнейших этапов DES. Таблицы замен для этой операции специально спроектированы так, чтобы обеспечивать максимальную криптостойкость. В результате выполнения этого этапа получаются восемь 4-битных блоков, которые вновь объединяются в единое 32-битное значение.

Далее полученное 32-битное значение обрабатывается с помощью перестановки *Р*, которая не зависит от используемого ключа. Целью перестановки является такое максимальное переупорядочивание битов, чтобы в следующем раунде шифрования каждый бит с большой вероятностью обрабатывался другим *S*-блоком.

И наконец, результат перестановки объединяется с помощью операции XOR с левой половиной первоначального 64-битного блока данных. Затем левая и правая половины меняются местами, и начинается следующий раунд.

После выполнения 16-раундового зашифрования 64-битного блока данных осуществляется конечная перестановка (IP−1). Она является обратной к перестановке IP.

Каждый 8-й бит исходного 64-битного ключа отбрасывается. Эти 8 битов, находящих в позициях 8, 16, 24, 32, 40, 48, 56, 64, изначально добавляются в исходный ключ таким образом, чтобы каждый байт содержал четное число единиц. Это используется для обнаружения ошибок при обмене и хранении ключей по известным алгоритмам избыточного кодирования. Один избыточный бит в ключе DES формируется, как видим, в соответствии с кодом простой четности. Этот код позволяет в кодовом слове (в нашем случае – в каждом байте ключа) обнаруживать ошибки, количество которых нечетно.

При расшифровании на вход алгоритма подается зашифрованный текст. Единственное отличие состоит в обратном порядке использования частичных ключей *Ki*. Ключ *K16* используется в первом раунде, *K1* – в последнем.

После последнего раунда процесса расшифрования две половины выхода меняются местами так, чтобы вход заключительной перестановки был составлен из подблоков *R16* и *L16*. Выходом этой стадии является расшифрованный текст.

## 1.2 Практическое задание

**Вариант 2**. По условию исходное сообщение нужно зашифровать алгоритмом DES-EEE3. Это значит, что исходное сообщение нужно шифровать три раза с тремя разными ключами на каждом этапе (*K1* = «Информац», *K2* =  
= «лаборато», *K3* = «зопаснос»).

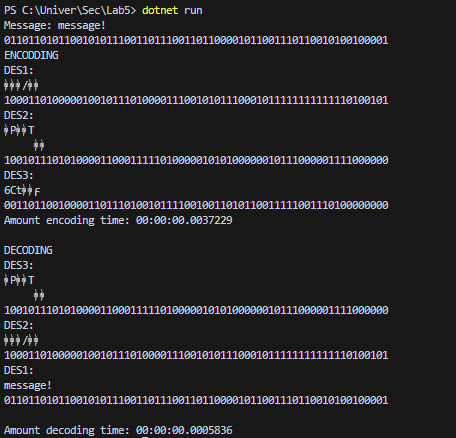


Рисунок 1 – Результат работы программы

Оценим время зашифрования и расшифрования одного и того же сообщения.

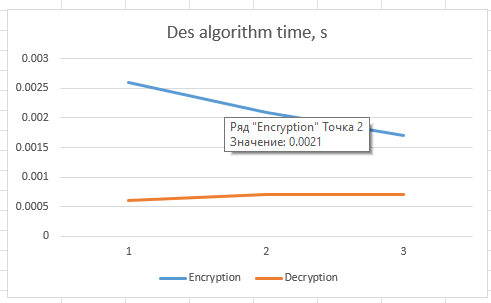


Рисунок 2 – Графики времени зашифрования/расшифрования

Теперь оценим лавинный эффект. Для этого в битовом представлении исходного сообщения изменим первый бит и сравним результаты.

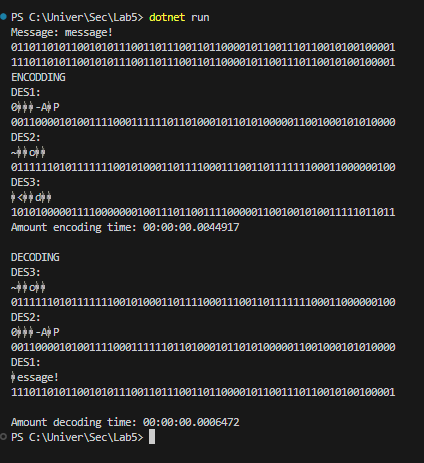


Рисунок 3 – Шифрование битого сообщения

Как можем увидеть, битовые представления зашифрованного сообщения существенно различаются.

Теперь оценим результат работы программы с использование слабого ключа (64 битовая последовательность нулей и единиц).

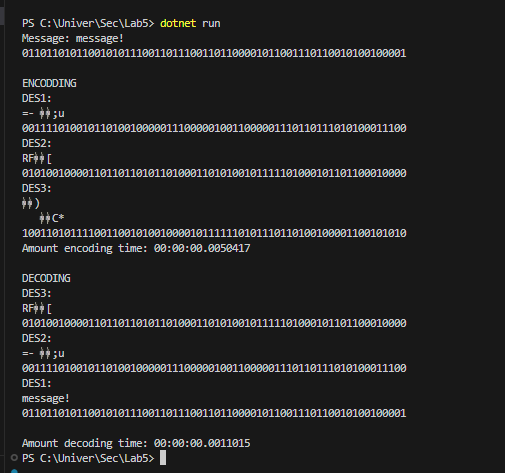


Рисунок 4 – Шифрование сообщения слабым ключом

Можно было ожидать, что зашифрованное сообщение могло равняться исходному, но данный ключ и последовательности 0 и 1 не даёт такой результат. Теперь оценим лавинный эффект для слабого ключа.

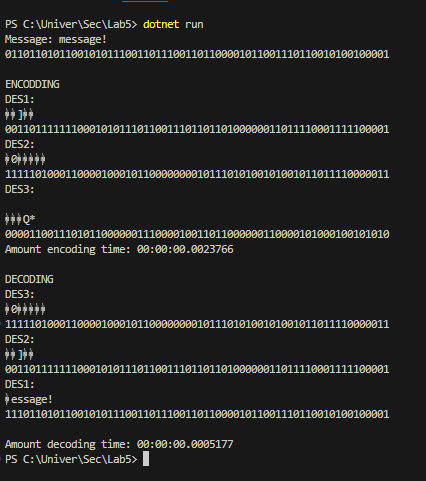


Рисунок 5 – Шифрование битого сообщения слабым ключом

Как можем заметить, разница между битовыми сообщениями существенна, что доказывает наличие лавинного эффекта.

Теперь оценим результат работы программы с использование полу слабого ключа.

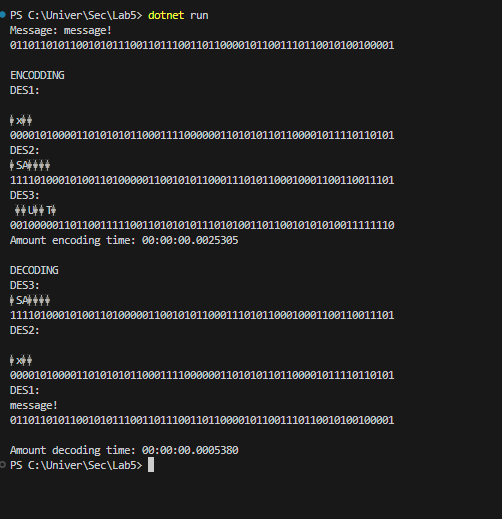


Рисунок 6 – Шифрование сообщения полу слабым ключом

Теперь оценим лавинный эффект для полу слабого ключа.

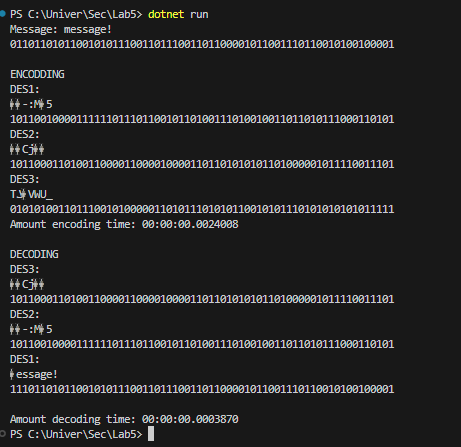


Рисунок 7 – Шифрование битого сообщения полу слабым ключом

Как можем заметить, разница между битовыми сообщениями существенна, что доказывает наличие лавинного эффекта.

Оценим сжатие исходного и зашифрованного сообщений средствами архиватора ZIP.

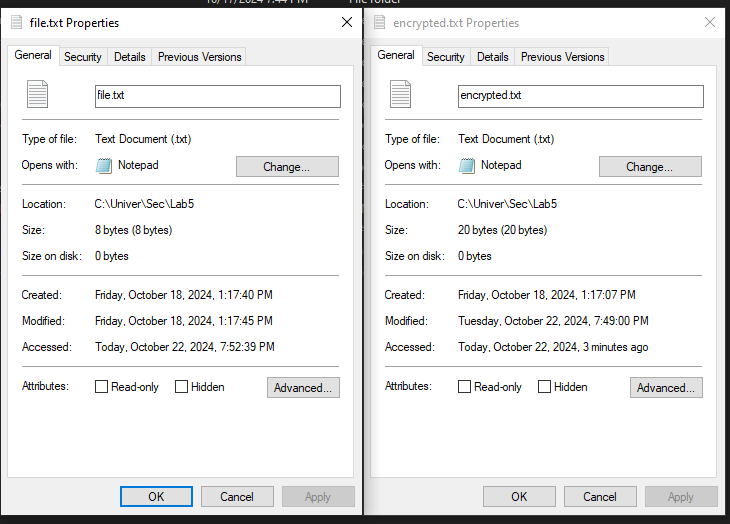


Рисунок 8 – Размеры файлов до сжатия

Как можно заметить, размер зашифрованного сообщения в 2 раза больше исходного сообщения.

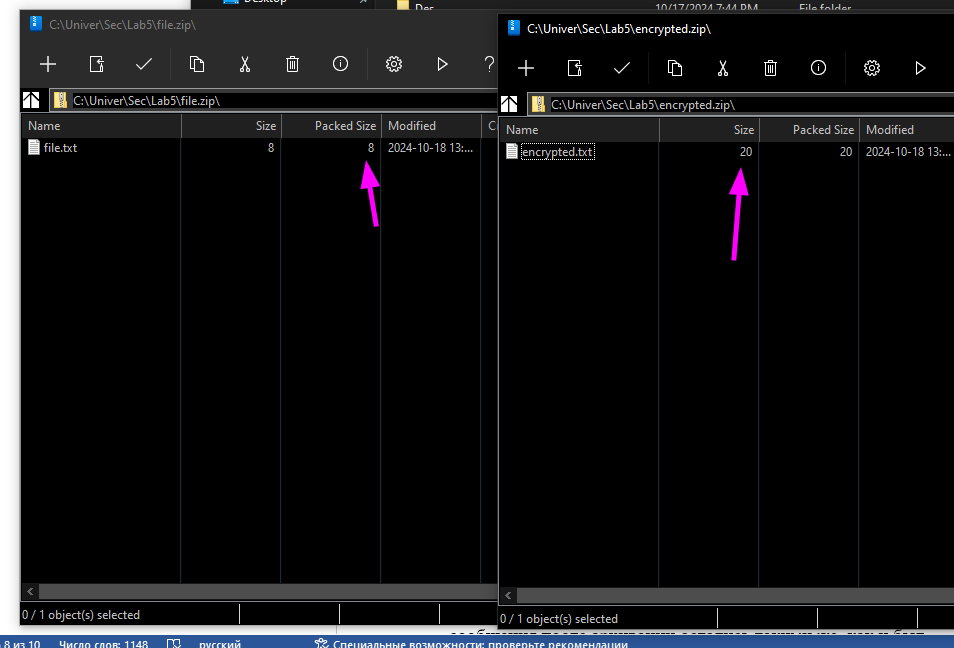


Рисунок 9 – Размеры файлов после сжатия

Как можем заметить по параметру «Packed Size», сжатия не было, а значит сообщения после архивации остались такими же, как и были.

Вывод: в ходе этой лабораторной работы я изучил принцип работы алгоритма шифрования DES. Написал программу, которая зашифровывает и расшифровывает сообщение, а также на основе которой можно реализовать алгоритмы типа DES-3EEE и т.п. Оценил лавинный эффект с использованием ключей различной стойкости.

# Листинг «Лабораторная работа №5»

using Lab5.Helpers;

using Lab5.Des;

using System.Text;

static string AlignMessage(string message)

{

int missingBits = 64 - message.Length % 64;

return missingBits == 64 ? message : message.PadRight(message.Length + missingBits, '0');

}

string message = "message!";

StringBuilder binaryMessage = new(AlignMessage(BinaryConverter.UTF8ToBinary(message)));

binaryMessage[0] = binaryMessage[0] == '0' ? '1' : '0';

string key1 = "Информац";

string key2 = "зопаснос";

string key3 = "лаборато";

var des1 = new Des(BinaryConverter.UTF8ToBinary(key1));

var des2 = new Des(BinaryConverter.UTF8ToBinary(key2));

var des3 = new Des(BinaryConverter.UTF8ToBinary(key3));

Console.WriteLine($"Message: {message}");

Console.WriteLine(BinaryConverter.UTF8ToBinary(message));

Console.WriteLine("\nENCODDING");

var watch = System.Diagnostics.Stopwatch.StartNew();

//string encodedMessage1 = des1.Encode(AlignMessage(binaryMessage.ToString()));

string encodedMessage1 = des1.Encode(AlignMessage(BinaryConverter.UTF8ToBinary(message)));

string encodedMessage2 = des2.Encode(encodedMessage1);

string encodedMessage3 = des3.Encode(encodedMessage2);

watch.Stop();

Console.WriteLine($"DES1:");

Console.WriteLine(BinaryConverter.BinaryToUTF8(encodedMessage1));

Console.WriteLine(encodedMessage1);

Console.WriteLine($"DES2:");

Console.WriteLine(BinaryConverter.BinaryToUTF8(encodedMessage2));

Console.WriteLine(encodedMessage2);

Console.WriteLine($"DES3:");

Console.WriteLine(BinaryConverter.BinaryToUTF8(encodedMessage3));

Console.WriteLine(encodedMessage3);

Console.WriteLine($"Amount encoding time: {watch}\n");

watch = System.Diagnostics.Stopwatch.StartNew();

string decodedMessage3 = des3.Decode(encodedMessage3);

string decodedMessage2 = des2.Decode(decodedMessage3);

string decodedMessage1 = des1.Decode(decodedMessage2);

watch.Stop();

Console.WriteLine($"DECODING\nDES3:");

Console.WriteLine(BinaryConverter.BinaryToUTF8(decodedMessage3));

Console.WriteLine(decodedMessage3);

Console.WriteLine($"DES2:");

Console.WriteLine(BinaryConverter.BinaryToUTF8(decodedMessage2));

Console.WriteLine(decodedMessage2);

Console.WriteLine($"DES1:");

Console.WriteLine(BinaryConverter.BinaryToUTF8(decodedMessage1));

Console.WriteLine(decodedMessage1);

Console.WriteLine($"\nAmount decoding time: {watch}");

Листинг 5.1 – Код программы