Учреждение образования

«БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНОЛОГИЧЕСКИЙ УНИВЕРСИТЕТ»

Лабораторная работа №7

**ИССЛЕДОВАНИЕ БЛОЧНЫХ ШИФРОВ**

Студент: Сосновец М.И.

ФИТ 3 курс 4 группа

Преподаватель: Нистюк О.А.

Минск 2025

**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 описания разработанного приложения, методики выполнения экспериментов с использованием приложения и результатов эксперимента

**Теоретические сведения**

В 1972 г. Национальное бюро стандартов США (ныне – Национальный институт стандартов и технологии, National Institute of Standarts & Technology – NIST) инициировало программу защиты каналов связи и компьютерных данных. Одна из целей – разработка единого стандарта криптографического шифрования. Основными критериями оценки алгоритма являлись следующие [5]:

* 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 применениям;
* алгоритм должен быть разрешен для экспорта.

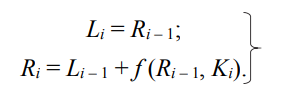
В качестве начального варианта нового алгоритма рассматривался Lucifer – разработка компании IBM начала семидесятых годов. В основе указанного алгоритма использовались два запатентованных в 1971 г. Хорстом Фейстелем (Horst Feistel) устройства, реализующие различные алгоритмы шифрования, позже получившие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

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

Основные требования к шифрам рассматриваемого класса можно сформулировать следующим образом:

* даже незначительное изменение исходного сообщения должно приводить к существенному изменению зашифрованного сообщения;
* устойчивость к атакам по выбранному тексту;
* алгоритмы зашифрования/расшифрования должны быть реализуемыми на различных платформах;
* алгоритмы должны базироваться на простых операциях;
* алгоритмы должны быть простыми для написания кода, вероятность появления программных ошибок должна быть низкой;
* алгоритмы должны допускать их модификацию при переходе на иные требования по уровню криптостойкости.

Сеть Фейстеля. Само название конструкции Фейстеля (сеть) означает ее ячеистую топологию [28]. Формально одна ячейка сети соответствует одному раунду зашифрования или расшифрования сообщения. При зашифровании сообщение разбивается на блоки одинаковой (фиксированной) длины (как правило – 64 или 128 битов). Полученные блоки называются входными. В случае если длина входного блока меньше, чем выбранный размер, блок удлиняется установленным способом. Каждый входной блок шифруемого сообщения изначально делится на два подблока одинакового размера: левый (L0) и правый (R0). Далее в каждом i-м раунде выполняются преобразования в соответствии с формальным представлением ячейки сети Фейстеля:



В своей статье [28] Х. Фейстель описывает два блока преобразований с использованием функции f (Ri – 1, Ki):

* блок подстановок (S-блок, англ. S-box);
* блок перестановок (P-блок, англ. P-box). Блок подстановок состоит:
* из дешифратора, преобразующего n-разрядное двоичное число в одноразрядный сигнал по основанию 2n ;
* внутреннего коммутатора;
* шифратора, преобразующего сигнал из одноразрядного 2n - ричного в n-разрядный двоичный.

При большом размере блоков шифрования (128 битов и более) реализация такой конструкции Фейстеля на 32-разрядных архитектурах может вызвать затруднения, поэтому применяются модифицированные варианты этой конструкции. Обычно используются сети с четырьмя ветвями.

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.

**Практическое задание**

Разработать авторское приложение в соответствии с целью лабораторной работы. При этом можно воспользоваться готовыми библиотеками либо программными кодами, реализующими некоторые блочные алгоритмы, из приложения в [5].

Приложение должно реализовывать следующие операции:

* разделение входного потока данных на блоки требуемой длины с необходимым дополнением последнего блока;
* выполнение требуемых преобразований ключевой информации;
* выполнение операций зашифрования/расшифрования;
* оценка скорости выполнения операций зашифрования/расшифрования;
* пошаговый анализ лавинного эффекта с подсчетом количества изменяющихся символов по отношению к исходному слову.

Исследуемый метод шифрования и ключевая информация – в соответствии с вариантом из табл. 5.5.

|  |  |  |
| --- | --- | --- |
| Вариант | Алгоритм | Ключ |
| 2 | DES-EEE3 |  |

Алгоритм DES-EEE3 (Triple DES) использует три последовательных этапа шифрования данных с тремя разными ключами. Для шифрования данные проходят через три DES-криптографические функции:

1. Шифрование с первым ключом (Key1).
2. Шифрование результата второго раз с использованием второго ключа (Key2).
3. Шифрование результата третьим ключом (Key3).

Реализация разделения входного потока данных на блоки требуемой длины с необходимым дополнением последнего блока показана в листинге 1.1.

|  |
| --- |
| function alignMessage(message, blockSize = 64) {  const missingBits = blockSize - (message.length % blockSize);  return missingBits === blockSize ? message : message.padEnd(message.length + missingBits, '0');  }  function main() {  const message = "diagram!";  const binaryMessage = BinaryConverter.utf8ToBinary(message);  const alignedMessage = alignMessage(binaryMessage);  console.log(`Original message: "${message}"`);  console.log(`Binary: ${binaryMessage}`);  console.log(`Aligned Binary: ${alignedMessage}`);  } |

Листинг 1.1 – Реализация разделения входного потока данных на блоки требуемой длины с необходимым дополнением последнего блока

Преобразование ключевой информации в алгоритме DES-EEE3 включает следующие этапы:

1. Преобразование ключа из символьного представления в бинарное. Каждый ключ, представленный текстом, преобразуется в бинарный формат UTF-8.
2. Применение начальной перестановки к ключу. Используется таблица keyPermutationTo56BitsTable, чтобы сократить ключ до 56 бит, удалив избыточные биты.
3. Разделение ключа на левую и правую части. После сокращения ключа до 56 бит он делится на две половины по 28 бит: левая (L) и правая (R).
4. Циклические сдвиги (ротация). Для каждой из 16 раундов выполняются сдвиги половин ключа (L и R) влево на заданное число позиций согласно таблице keyShiftTable.
5. Преобразование в 48-битный ключ. После сдвигов объединенный ключ (L + R) подвергается дальнейшей перестановке с помощью таблицы keyPermutationTo48BitsTable, чтобы получить итоговый 48-битный ключ для конкретного раунда.
6. Генерация набора раундовых ключей. Все 16 ключей сохраняются в массиве и используются последовательно для шифрования/дешифрования данных

Реализация преобразований ключевой информации представлена в листинге 1.2.

|  |
| --- |
| class KeysGenerator {    static generateKeys(key) {      const keys = new Array(16);      let key56Bits = Permutator.permutate(key, desTables.keyPermutationTo56BitsTable);      for (let i = 0; i < 16; i++) {        key56Bits = this.shiftKey(key56Bits, desTables.keyShiftTable[i]);        keys[i] = Permutator.permutate(key56Bits, desTables.keyPermutationTo48BitsTable);      }      return keys;    }    static shiftKey(key, shiftsNumber) {      const left = key.substr(0, 28);      const right = key.substr(28);      return left.substr(shiftsNumber) + left.substr(0, shiftsNumber) +             right.substr(shiftsNumber) + right.substr(0, shiftsNumber);    }  } |

Листинг 1.2 – Реализация преобразований ключевой информации

Реализация операций зашифрования/расшифрования показана в листинге 1.3.

|  |
| --- |
| encode(message, encode = true) {      let encodedMessage = '';      for (let i = 0; i < message.length; i += 64) {        const block = message.substr(i, 64).padEnd(64, '0');        const permutatedBlock = Permutator.permutate(block, desTables.initialPermutationTable);        let left = permutatedBlock.substr(0, 32);        let right = permutatedBlock.substr(32);        for (let j = 0; j < 16; j++) {          const temp = right;          right = Permutator.permutate(right, desTables.rExtensionTo48BitsTable);          right = this.xor(right, encode ? this.keys[j] : this.keys[15 - j]);          right = Permutator.sBlockPermutate(right, desTables.sConversionTable, desTables.rPermutationTable);          right = this.xor(right, left);          left = temp;        }        encodedMessage += Permutator.permutate(right + left, desTables.finalPermutationTable);      }      return encodedMessage;    }    decode(message) {      return this.encode(message, false);    } |

Листинг 1.3 – Реализация операций зашифрования/расшифрования

Результат работы программы, а также оценка скорости выполнения операций зашифрования/расшифрования показан на рисунке 1.1.

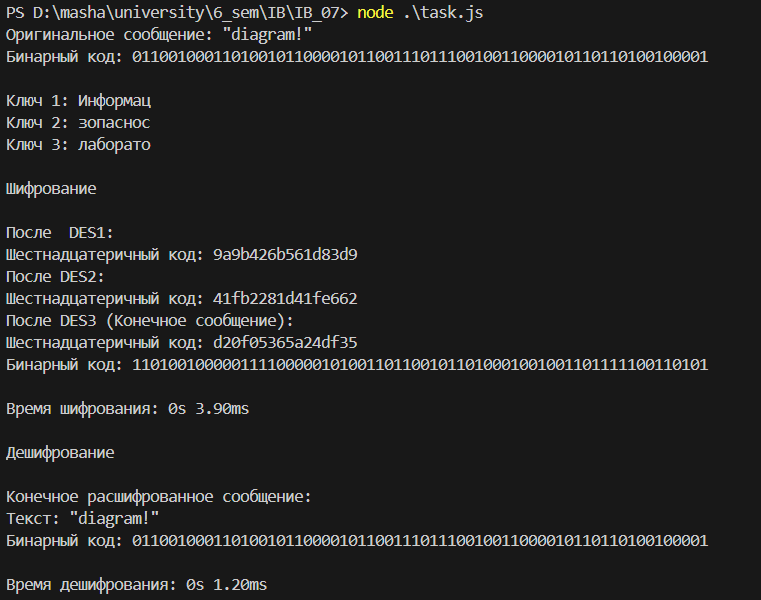


Рисунок 1.1 – Результат работы программы

Оценка скорости выполнения операций зашифрования/расшифрования в виде графиков показана на рисунке 1.2.

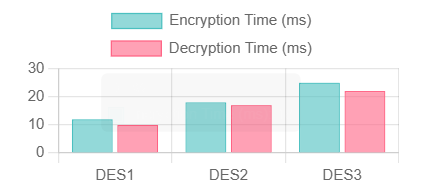


Рисунок 1.2 – Оценка скорости выполнения операций зашифрования/расшифрования

Лавинный эффект заключается в том, что в исходном сообщении изменим первый бит. Первый бит был 0, а станет 1 (инверсия) и новое бинарное сообщение будет выглядеть так 11100100 01101001 01100001 01100111 01110010 01100001 01101101 00100001. Далее шифруем измененное сообщение тем же ключом DES, получаем новый зашифрованный текст 8645a5117eba8cdd и далее переводим в бинарный вид 10000110 01000101 10100101 00010001 01111110 10111010 10001100 11011101.

Теперь оценим лавинный эффект. Реализация лавинного эффекта показана в листинге 1.4.

|  |
| --- |
| function calculateAvalancheEffect(originalMessage, desInstance) {    const binaryOriginalMessage = BinaryConverter.utf8ToBinary(originalMessage);    const alignedMessage = alignMessage(binaryOriginalMessage);    const encryptedOriginal = desInstance.encode(alignedMessage);    console.log(`Зашифрованное исходное сообщение: ${BinaryConverter.binaryToHex(encryptedOriginal)}`);    let modifiedBinaryMessage = binaryOriginalMessage.split('');    modifiedBinaryMessage[0] = modifiedBinaryMessage[0] === '0' ? '1' : '0'; // Меняем первый бит    modifiedBinaryMessage = modifiedBinaryMessage.join('');    console.log(`Модифицированное бинарное сообщение (первый бит изменен): ${modifiedBinaryMessage}`);    const alignedModifiedMessage = alignMessage(modifiedBinaryMessage);    const encryptedModified = desInstance.encode(alignedModifiedMessage);    console.log(`Зашифрованное модифицированное сообщение: ${BinaryConverter.binaryToHex(encryptedModified)}`);    let diffCount = 0;    for (let i = 0; i < encryptedOriginal.length; i++) {      if (encryptedOriginal[i] !== encryptedModified[i]) {        diffCount++;      }    }    console.log(`Количество изменённых битов: ${diffCount}`);    return diffCount;  } |

Листинг 1.4 – Реализация лавинного эффекта

Результат работы лавинного эффекта показан на рисунке 1.3.

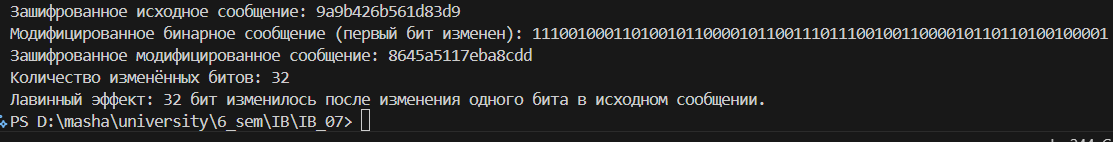


Рисунок 1.3 – Результат работы лавинного эффекта

Из рисунка 1.3 видно, битовые представления зашифрованного сообщения существенно различаются, так как исходное сообщение изменилось на 32 бит.

Реализация влияния слабых и полуслабых ключей на результат шифрования и лавинный эффект показана на рисунке 1.4.

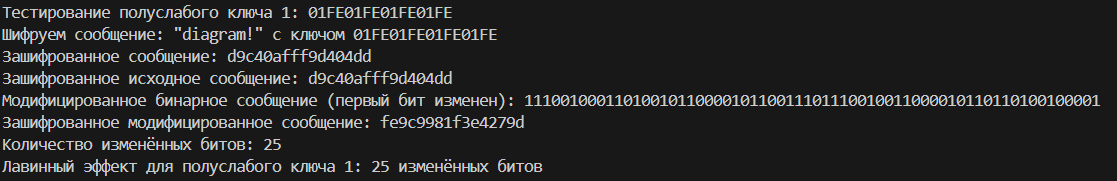
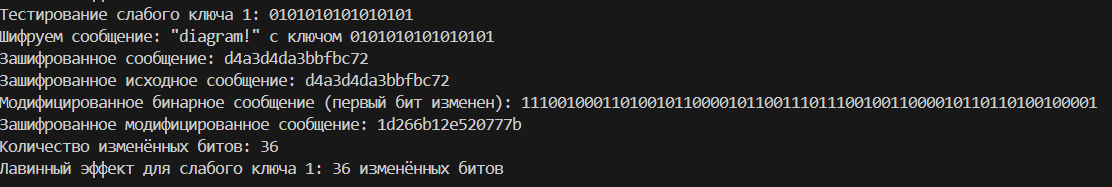


Рисунок 1.4 – Реализация влияния слабых и полуслабых ключей на результат шифрования и лавинный эффект

Тестирование слабого ключа (0101010101010101) показало высокий лавинный эффект: изменение одного бита в исходном сообщении привело к изменению 36 битов в зашифрованном тексте. Это свидетельствует о значительном воздействии на результат шифрования, несмотря на слабость ключа.

В случае полуслабого ключа (01FE01FE01FE01FE) лавинный эффект оказался меньшим — всего 25 изменённых битов. Это подтверждает, что полуслабые ключи, хотя и не идеальны, обеспечивают большую устойчивость по сравнению со слабыми, оказывая менее выраженное влияние на результат шифрования при тех же изменениях в исходном сообщении.

Таким образом, полуслабый ключ демонстрирует лучшее поведение в плане лавинного эффекта, чем слабый, что делает его предпочтительным вариантом для шифрования.

Оценка степени сжатия открытого текста показана на рисунке 1.5.

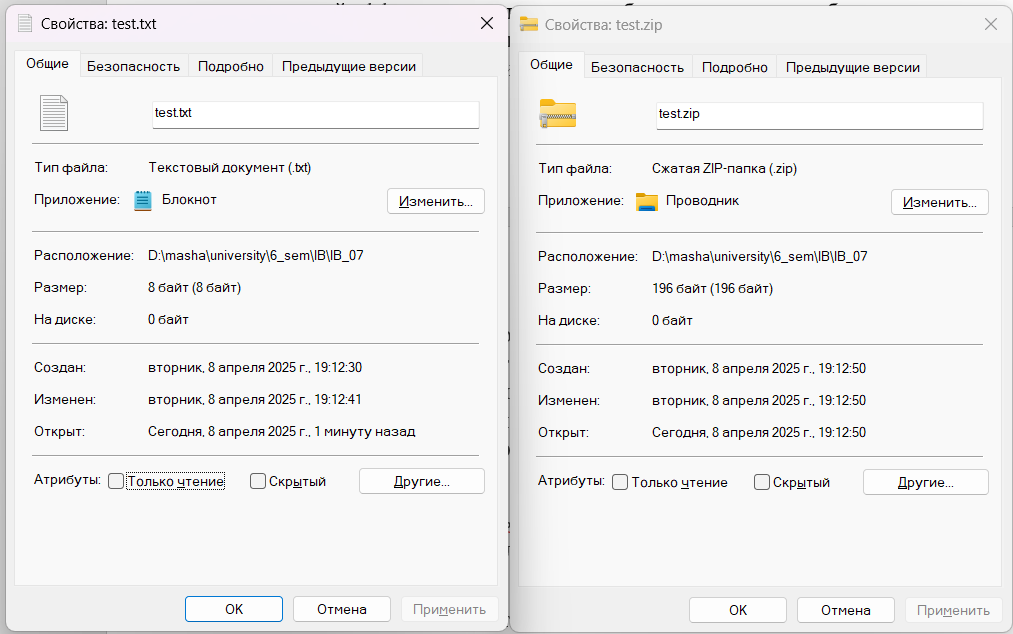


Рисунок 1.5 – Оценка степени сжатия открытого текста

Когда сжали файл, содержащий текст "diagram!" размером 8 байт, и получается архив размером 196 байт, это вероятно связано с тем, что для очень маленьких файлов сжатие не всегда эффективно из-за добавления метаданных и особенностей алгоритма сжатия. Кроме того, отсутствие повторяющихся данных в файле не позволило алгоритму сжатия уменьшить его размер.

Оценка степени сжатия зашифрованного текста показана на рисунке 1.6.

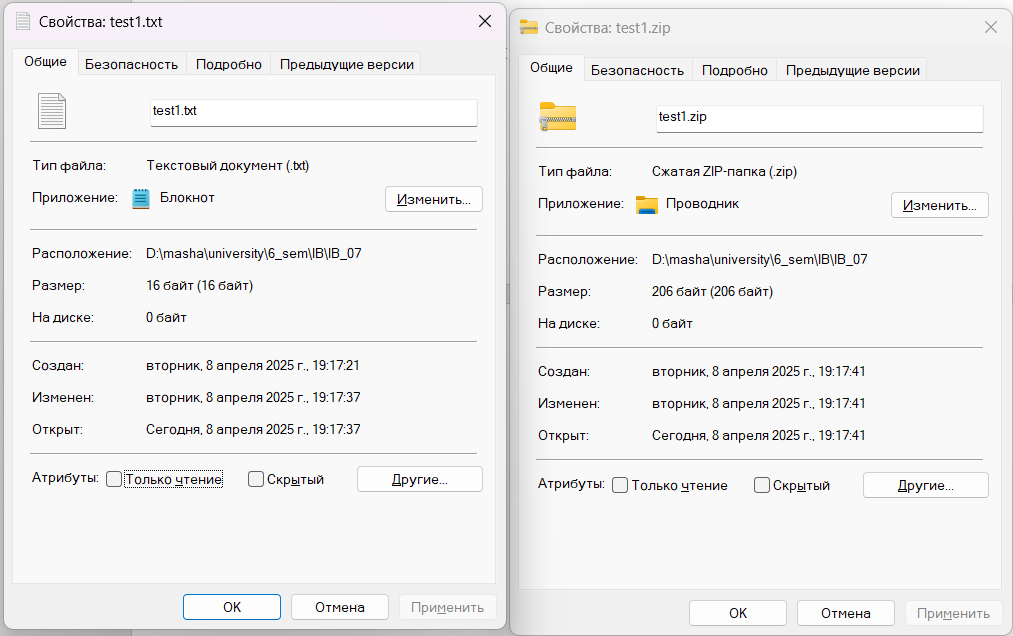


Рисунок 1.6 – Оценка степени сжатия зашифрованного текста

Это происходит из-за того, что шифрованные данные имеют высокую энтропию и случайность, что делает их практически несжимаемыми.

**Вывод**: В ходе этой лабораторной работы были изучены принцип работы алгоритма шифрования DES. Написана программу, которая зашифровывает и расшифровывает сообщение, а также на основе которой можно реализовать алгоритмы типа DES-3EEE и т.п. Оценка лавинного эффекта с использованием ключей различной стойкости.