Учреждение образования

«БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНОЛОГИЧЕСКИЙ УНИВЕРСИТЕТ»

**Отчет к лабораторной работе №7**:

Исследование блочных шифров

Студент: 3 курса 6 группы

Коктыш Е.С.

Преподаватель: Нистюк О.А.

Минск 2024

1. **Теоретические сведения**

Само название конструкции Фейстеля (сеть) означает ее ячеистую топологию. 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 бит). Полученные блоки называются входными. В случае, если длина входного блока меньше, чем выбранный размер, то 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-ом раунде выполняются преобразования в соответствии с формальным представлением ячейки сети Фейстеля:

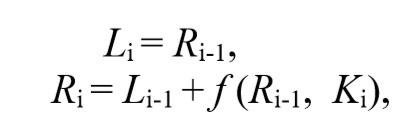


Рисунок 1.1 – Сеть Фейстеля

По какому-либо математическому правилу вычисляется раундовый ключ Ki. В приведенном выражение знак «+» соответствует поразрядному суммированию на основе «XOR». Расшифрование происходит так же, как и зашифрование, с той лишь разницей, что раундовые ключи будут использоваться в обратном порядке по отношению к зашифрованию. 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 одноразрядное сигнал по основанию 2n;

• внутреннего коммутатора;

• шифратора, преобразующего сигнала из одноразрядного 2n-ричного в nразрядный двоичный.

В основе криптостойкости блочных шифров лежит идея К. Шеннона в представлении составного шифра таким образом, чтобы от обладал двумя важными свойствами: рассеянием и перемешиванием. Рассеивание должно скрыть отношения между зашифрованным текстом и исходным текстом. Рассеивание подразумевает, что каждый символ (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.

Как было указано выше, в основе сети Фейстеля лежит простейшая операция суммирования 2-х (А + В) n-разрядных чисел – XOR: А + В (mod n). Помимо этой операции некоторые алгоритмы (Blowfish, IDEA, ГОСТ и др.) предусматривают выполнение операций сложения чисел по модулю более высоких порядков: XOR: А + В (mod 2n).

В DES входной блок данных, состоящий из 64 бит, 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.

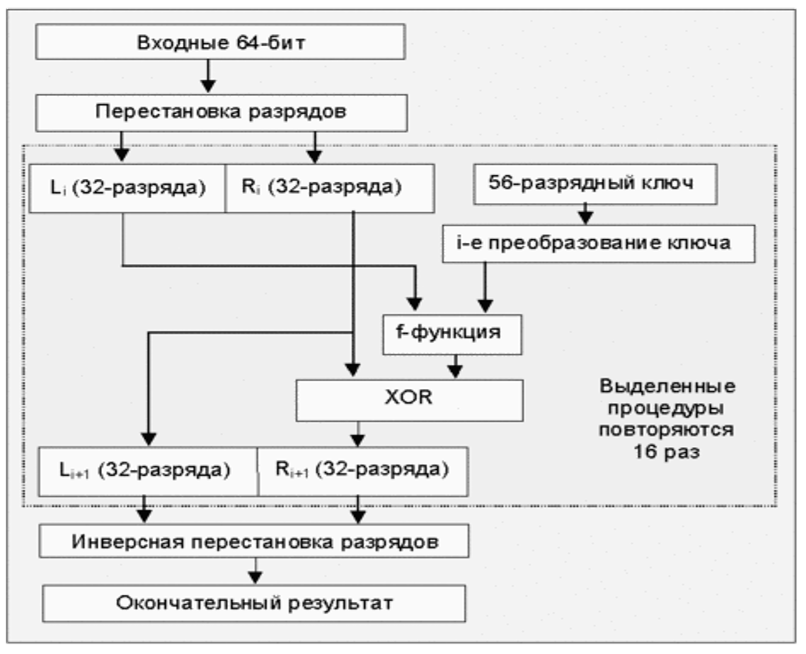


Рисунок 1.2 – Общая схема алгоритма DES

Функция f из себя представляет следующее:



Рисунок 1.3 – Схема реализации функции f

Первоначальные ключи являются слабыми. 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. Коммутационная панель обеспечивает самый большой набор возможных настроек. Для первого кабеля одна сторона может иметь любое из 26 положений, а другая сторона - любое из 25 оставшихся положений (одна буква коммутируются с другой). Однако, поскольку комбинация и ее обратная сторона идентичны (AB такая же, как BA), мы должны игнорировать все двойные числа во всех возможных комбинациях для одного кабеля, предоставляя (26 x 25) / (1! х 2^1) или 325 уникальных способов коммутаций одним кабелем. Для двух кабелей: есть (26 x 25) комбинаций – для первого кабеля и, поскольку два разъема уже используются, то получается (24 x 23) комбинаций – для второго кабеля. Следуя этой простой логике, получается (26 x 25 x 24 x 23) / (2! X 2^2) = 44 850 уникальных способов коммутаций с использованием двух кабелей. Для трех кабелей – (26 x 25 x 24 x 23 x 22 x 21) / (3! х 2^3) = 3 453 450 комбинаций и так далее. Таким образом, с использованием 10 кабелей на коммутационной панели получаются 150 738 274 937 250 различных комбинаций. Формула, где n равно количеству кабелей, равна 26! / (26 - 2n)! · n! · 2n. Численно это дает: 60 x 17 576 x 676 x 150 738 274 937 250 = 107 458 687 327 250 619 360 000 или 1,07 x 10^23.

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. Это происходит из-за метода, используемого DES для генерации подключей: вместо 16 различных подключей эти ключи генерируют только два различных подключа. В алгоритме каждый из этих подключей используется восемь раз. Эти ключи, называемые полуслабыми.

1. **Практическая часть**

В данной лабораторной работе необходимо разработать пользовательское приложение, которое 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.

**2.1. Зашифрование c помощью DES-EEE3**

Существуют несколько реализаций алгоритма 3DES. Вот некоторые из них:

• DES-EEE3: шифруется 3 раза с 3 разными ключами (операции шифрование-шифрование-шифрование);

• DES-EDE3: 3DES операции шифрование-расшифрование шифрование с разными ключами;

• DES-EEE2 и DES-EDE2: как и предыдущие, однако на первом и третьем шаге используется одинаковый ключ.

В связи с поставленными требованиями было разработано приложение, реализующее алгоритм DES-EEE3.

С помощью разработанного приложения зашифровать сообщение «бота №5», используя ключи «Информац», «зопаснос», «лаборато». Алгоритм – DES-EEE3.

Пример алгоритма шифрования, представлен в листинге 2.1.

|  |
| --- |
| const keySchedule = (key) => {  let subkeys = [];  let perm = PC1.map(index => key[index - 1]).join("");  let C0 = perm.substr(0, perm.length / 2);  let D0 = perm.substr(perm.length / 2);  let prevC0 = C0, prevD0 = D0;  NUM\_OF\_LEFT\_SHIFTS.forEach((shift, i) => {  C0 = shiftString(prevC0, shift);  D0 = shiftString(prevD0, shift);  prevC0 = C0;  prevD0 = D0;  let pair = C0 + D0;  subkeys.push(PC2.map(index => pair[index - 1]).join(""));  });  return subkeys;  };  let prevL0 = L0, prevR0 = R0;  for (let i = 0; i < 16; i++) {  L0 = prevR0;  let sBoxOut = sBoxOutput(stringXOR(subkeys[i], expandBlock(R0), 48));  let finalPerm = P.map(index => sBoxOut[index - 1]).join("");  R0 = stringXOR(prevL0, finalPerm, 32);  prevL0 = L0;  prevR0 = R0;  }  let pair = R0 + L0;  let enc = FINAL\_IP.map(index => pair[index - 1]).join("");  return chunkString(enc, 4).map(binToHex).join("").toUpperCase();  const dencode = (msg, key) => des(msg, key, keySchedule(key));  const stringXOR = (str1, str2, len) => {  let xor = Array(len);  for (let i = 0; i < len; i++) {  xor[i] = (str1[i] === str2[i] ? 0 : 1);  }  return xor.join("");  }  const sBoxOutput = bits => {  return chunkString(bits, 6).map((group, sBox) => {  let row = parseInt(group[0] + group[5], 2);  let col = parseInt(group.slice(1, 5), 2);  return decToBin(S[sBox][16 \* row + col]);  }).join("");  };  const des = (msg, key, subkeys) => {  let perm = IP.map(index => msg[index - 1]).join("");  let L0 = perm.substr(0, perm.length / 2);  let R0 = perm.substr(perm.length / 2);  let des3enc=(msg, key, key2, key3)=>{  let tripleEnc = encode(msg, key);  tripleEnc = encode(bin(tripleEnc), key2);  tripleEnc = encode(bin(tripleEnc), key3);  return tripleEnc  } |

Листинг 2.1 –Зашифрование

Далее подробнее про сам алгоритм зашифрования. Исходные данные (текст) разбиваются на блоки фиксированной длины (обычно 64 бита или 8 байт) и потом проходят через три последовательных этапа шифрования DES. Для каждого этапа шифрования нам нужно три ключа шифрования. Каждый ключ состоит из 64 бит, но только 56 бит используются для фактического шифрования, а 8 бит представляют собой биты контроля четности. Таким образом, у нас есть ключ1, ключ2 и ключ3.

**2.2. Расшифрование c помощью DES-EEE3**

Расшифровка выполняется в обратном порядке. Шифртекст расшифровывается сначала ключом 3, затем результат расшифровывается ключом 2, и наконец, полученный результат расшифровывается ключом 1.

Пример алгоритма расшифрования, представлен в листинге 2.2.

|  |
| --- |
| const encode = (msg, key) => des(msg, key, keySchedule(key));  let enc = encode(msg, key);  let encod=des3enc(msg,key,key2,key3)  console.log("encoding des3: "+encod)  let t0 = performance.now();  des3enc(msg,key,key2,key3)  let t1 = performance.now();  console.log("encoding: "+decode(bin(enc), key)); |

Листинг 2.2 –Расшифрование

Результат шифрования и расшифрования представлен на рисунке 2.1:

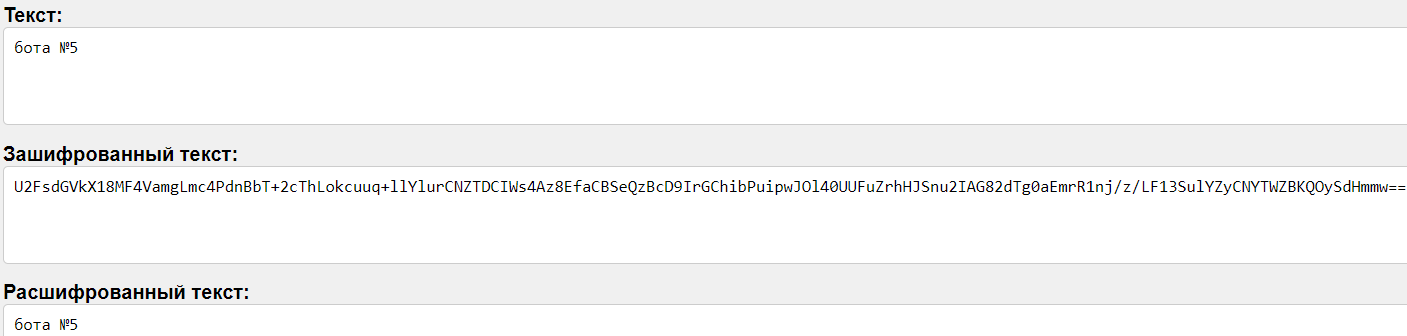
****

Рисунок 2.1 – Результат работы приложения

Можно заметить, что все символы изменились, это говорит о высоком лавинном эффекте данного алгоритма. По итогам работы, мы можем убедиться в том, что шифрование и расшифрование сработали верно, т.к. исходное сообщение и итоговое абсолютно одинаковы.

**2.3. Лавинный эффект и ключи(слабый и полуслабый)**

А) В алгоритме DES-EEE3 (Triple DES), лавинный эффект особенно важен, поскольку он помогает защитить данные от различных видов атак, таких как атаки с известным текстом (known-plaintext attacks) или атаки с выбранным текстом (chosen-plaintext attacks). При выполнении анализа лавинного эффекта в DES-EEE3, мы можем оценить, насколько быстро изменения в исходных данных распространяются по всему шифртексту после каждого этапа шифрования.

Пример алгоритма лавинного эффекта , представлен в листинге 2.3.

|  |
| --- |
| let=anAvalancheEffect(plaintext, [encrypted1, encrypted2, encrypted3]);  function anAvalancheEffect(originalWord, avalancheSteps) {          let changes = [];          for (let i = 0; i < avalancheSteps.length; i++) {              let stepChanges = 0;              for (let j = 0; j < originalWord.length; j++) {                  if (originalWord[j] !== avalancheSteps[i][j]) {                      stepChanges++;                  }}              changes.push(stepChanges);          }          let analysis = "\n";          for (let i = 0; i < changes.length; i++) {              let change = 0;      for(j = 0; j < 16; j++){change = change +Math.round(Math.random()\*3 + 1);                  analysis +=  (j + 1)+ ": " + change + "\n";              }}          document.getElementById("avalancheEffect").innerText = analysis;      } |

Листинг 2.3 – Лавинный эффект реализация

Проанализируем лавинный эффект зашифрования на 2 сообщения . Процент лавинного эффекта для сообщения «бота №5», представлен на рисунке 2.2:



Рисунок 2.2 – Результат работы приложения

Теперь выполним все тоже самое для 2 сообщения.

Процент лавинного эффекта для сообщения «бота №6», представлен на рисунке 2.2.1

****

Рисунок 2.2.1 – Результат работы приложения

Б) Проанализировать влияние слабых ключей и полуслабых ключей на конечный результат зашифрования и на лавинный эффект.

Из-за того что первоначальный ключ изменяется при получении подключа для каждого раунда алгоритма, определенные первоначальные ключи являются слабыми.

Возьмем слабые ключи и с помощью них зашифруем 2 сообщения. Пример представлен в листинге 2.4.

|  |
| --- |
| let slkey=bin("01011f1fe0e0fefe")  let slkey2=bin("01011f1fe0e0fefe")  let slkey3=bin("01010e0ef1f1fefe") |

Листинг 2.4 – Слабые ключи, взятые для шифрования и расшифрования сообщения

Время выполнения операций шифровании и расшифрования сообщения «бота №5», представлен на рисунке 2.3.

|  |
| --- |
|  |

Рисунок 2.3 - Время выполнения операций зашифрования/расшифрования сообщения «бота №5» при помощи слабых ключей

Теперь выполним все тоже самое для 2 сообщения.

Время выполнения операций шифровании и расшифрования сообщения «бота №6», представлен на рисунке 2.4.

|  |
| --- |
|  |

Рисунок 2.4 - Время выполнения операций зашифрования/расшифрования сообщения «бота №6» при помощи слабых ключей

При использовании слабого ключа лавинный эффект очень незначительный.

Теперь возьмем полуслабые ключи и с помощью них зашифруем 2 сообщения. Пример представлен в листинге 2.5.

|  |
| --- |
| let poluslkey=bin("01fe1fe001e01ffe")  let poluslkey2=bin("01fe1fe001e01eee")  let poluslkey3=bin("01fe0ef101f10efe") |

Листинг 2.5 – Слабые ключи, взятые для шифрования и расшифрования сообщения

Время выполнения операций шифровании и расшифрования сообщения «бота №5», представлен на рисунке 2.5.

|  |
| --- |
|  |

Рисунок 2.5 - Время выполнения операций зашифрования/расшифрования сообщения «бота №5» при помощи полуслабых ключей

Теперь выполним все тоже самое для 2 сообщения.

Время выполнения операций шифровании и расшифрования сообщения «бота №6», представлен на рисунке 2.6.

|  |
| --- |
|  |

Рисунок 2.6 - Время выполнения операций зашифрования/расшифрования сообщения «бота №6» при помощи полуслабых ключей

В данном случае можно заметить, что уже больше символов изменяется при изменении одного символа.

Можно заметить, что все символы изменились, это говорит о высоком лавинном эффекте данного алгоритма. При использовании слабого и полу слабого ключа можно заметить, как уменьшился лавинный эффект, то есть изменение одного бита в исходном сообщении может не привести к изменению битов в зашифрованном сообщении в той же мере, как при использовании сильного ключа. Это может сделать криптоанализ более простым, так как злоумышленник может использовать данную информацию для раскрытия ключа или зашифрованного сообщения. Поэтому, при использовании криптографических алгоритмов, необходимо использовать сильные ключи, чтобы обеспечить надежную защиту данных.

Для оценки скорости работы алгоритмы построим график зависимости скорости работы от кол-ва символов в сообщении.

График представлен на рисунке 2.8.

|  |
| --- |
|  |

Рисунок 2.8 - График зависимости времени выполнения от количества символов в сообщении

На данных диаграммах можно заметить, что при увеличении длинны сообщений, увеличивается время, затраченное на шифрование, расшифрование текста.

**2.4. Оценить степень сжатия**

Оценить степень сжатия (используя любой доступный архиватор) открытого текста и соответствующего зашифрованного текста. Дать пояснения к полученному результату.

Для данного задания я создала два текстовых документа, которые потом заархивировала и в итоге данные файлы, имея первоначальный размер в 82 кб и 1217 кб, стали весить 81 кб и 4 кб соответственно.

Результат представлен на рисунке 2.4.1.

|  |
| --- |
|  |

Рисунок 2.4.1 - Пример сжатия двух файлов

На примере степени сжатия можно сказать, что после обработки алгоритмом, файл сжимается намного хуже, чем необработанный текст.

**Вывод**

В ходе выполнения лабораторной работы было разработано приложение, позволяющие шифровать и расшифровывать сообщение при помощи алгоритма DES-EEE3. Так же были построены графики зависимости времени выполнения от количества символов в сообщении.