МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ

Учреждение образования «БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНОЛОГИЧЕСКИЙ УНИВЕРСИТЕТ»

**Лабораторная работа № 7**

**ИССЛЕДОВАНИЕ БЛОЧНЫХ ШИФРОВ**

Разработала: Бай И.О.

ФИТ 3 курс 5 группа

Преподаватель: Савельева М.Г.

Минск 2023

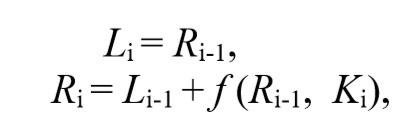
**Цель:** изучение и приобретение практических навыков разработки и использования приложений для реализации блочных шифров (рассчитана на 4 часа аудиторных занятий).

**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**

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:

 (1)

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 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 2*n*;

• внутреннего коммутатора;

• шифратора, преобразующего сигнала из одноразрядного 2*n*-ричного в *n-*разрядный 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.

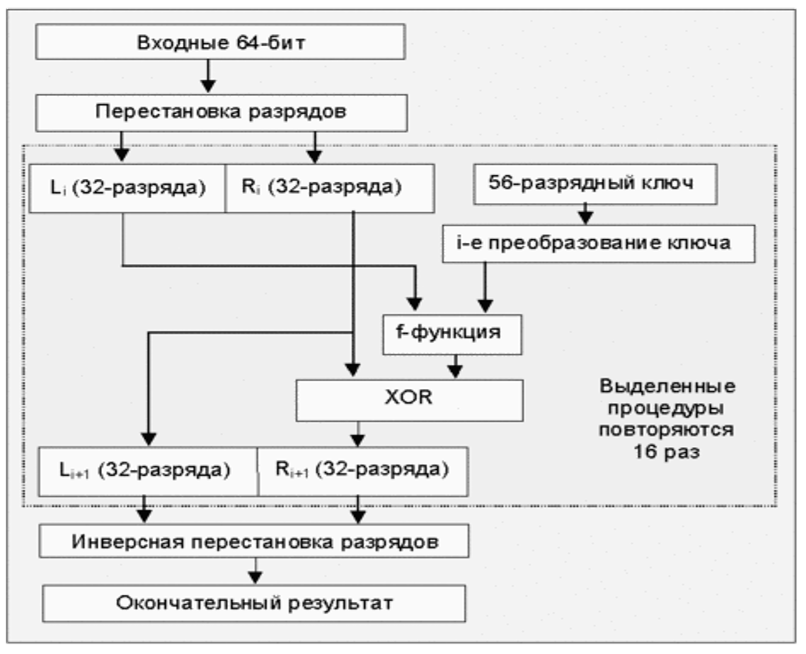


Рисунок 2 – Общая схема алгоритма DES

Функция *f* из себя представляет следующее:



Рисунок 3 – Схема реализации функции f

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.

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.

**Практическая часть**

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

Алгоритм зашифрования:

private void button1\_Click(object sender, EventArgs e)

{

if(textBox1.Text.Length == 8)

{

key = textBox1.Text;

if (openFileDialog1.ShowDialog() == DialogResult.OK)

{

string source = openFileDialog1.FileName;

string fileText = File.ReadAllText(source);

richTextBox1.Text = fileText;

saveFileDialog1.Filter = "des files |\*.des";

if (saveFileDialog1.ShowDialog() == DialogResult.OK)

{

string destination = saveFileDialog1.FileName;

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

EncryptFile(source, destination, key);

startTime.Stop();

var resultTime = startTime.Elapsed;

// elapsedTime - строка, которая будет содержать значение затраченного времени

string elapsedTime = String.Format("{0:00}:{1:00}:{2:00}.{3:000}",

resultTime.Hours,

resultTime.Minutes,

resultTime.Seconds,

resultTime.Milliseconds);

label1.Text += elapsedTime.ToString();

}

}

}

else

{

MessageBox.Show("Ключ должен быть длиной 8!");

}

}

private void EncryptFile(string source, string destination, string key)

{

FileStream fsInput = new FileStream(source, FileMode.Open, FileAccess.Read);

FileStream fsEncrypted = new FileStream(destination, FileMode.Create, FileAccess.Write);

DESCryptoServiceProvider DES = new DESCryptoServiceProvider();

try

{

DES.Key = ASCIIEncoding.ASCII.GetBytes(key);

DES.IV = ASCIIEncoding.ASCII.GetBytes(key);

ICryptoTransform desencrypt = DES.CreateEncryptor();

CryptoStream cryptoStream = new CryptoStream(fsEncrypted, desencrypt, CryptoStreamMode.Write);

byte[] bytearrayinput = new byte[fsInput.Length - 0];

fsInput.Read(bytearrayinput, 0, bytearrayinput.Length);

cryptoStream.Write(bytearrayinput, 0, bytearrayinput.Length);

cryptoStream.Close();

}

catch(Exception ex)

{

MessageBox.Show(ex.Message);

return;

}

string fileText = File.ReadAllText(destination);

richTextBox3.Text = fileText;

fsInput.Close();

fsEncrypted.Close();

}

private void DecryptFile(string source, string destination, string key)

{

FileStream fsInput = new FileStream(source, FileMode.Open, FileAccess.Read);

FileStream fsDecrypted = new FileStream(destination, FileMode.Create, FileAccess.Write);

DESCryptoServiceProvider DES = new DESCryptoServiceProvider();

try

{

DES.Key = ASCIIEncoding.ASCII.GetBytes(key);

DES.IV = ASCIIEncoding.ASCII.GetBytes(key);

ICryptoTransform desencrypt = DES.CreateDecryptor();

CryptoStream cryptoStream = new CryptoStream(fsDecrypted, desencrypt, CryptoStreamMode.Write);

byte[] bytearrayinput = new byte[fsInput.Length - 0];

fsInput.Read(bytearrayinput, 0, bytearrayinput.Length);

cryptoStream.Write(bytearrayinput, 0, bytearrayinput.Length);

cryptoStream.Close();

}

catch

{

MessageBox.Show("Ошибка");

return;

}

string fileText = File.ReadAllText(destination);

richTextBox2.Text = fileText;

fsInput.Close();

fsDecrypted.Close();

}

private void button2\_Click(object sender, EventArgs e)

{

key = textBox1.Text;

openFileDialog1.Filter = "des files |\*.des";

if (openFileDialog1.ShowDialog() == DialogResult.OK)

{

string source = openFileDialog1.FileName;

saveFileDialog1.Filter = "txt files | \*.txt";

if (saveFileDialog1.ShowDialog() == DialogResult.OK)

{

string destination = saveFileDialog1.FileName;

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

DecryptFile(source, destination, key);

startTime.Stop();

var resultTime = startTime.Elapsed;

// elapsedTime - строка, которая будет содержать значение затраченного времени

string elapsedTime = String.Format("{0:00}:{1:00}:{2:00}.{3:000}",

resultTime.Hours,

resultTime.Minutes,

resultTime.Seconds,

resultTime.Milliseconds);

label2.Text += elapsedTime.ToString();

}

}

}

Листинг 2 – Реализация расшифрования алгоритмом DES

Результат выполнения программы представлен на рисунках 4:

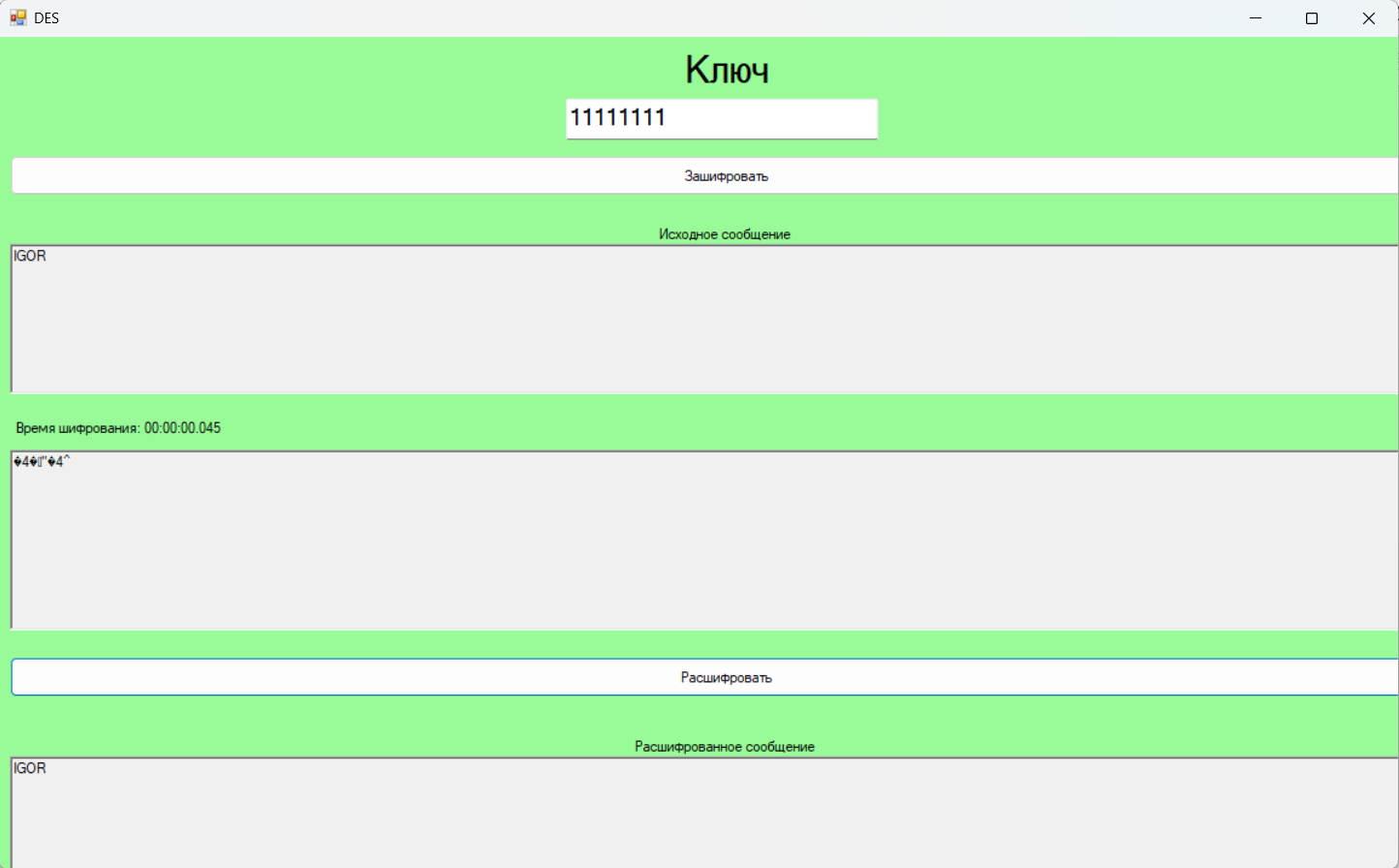


Рисунок 4 – Работа приложения

Как видно из результатов работы приложения исходный и зашифрованный текст совпали, значит алгоритмы шифрования и дешифрования работают исправно.

Слабые ключи:

Четыре ключа из 256 *возможных ключей* называются **слабыми ключами**. Слабые ключи — это одни из тех, которые после операции удаления проверочных бит состоят из всех нулей или всех единиц или половины нулей и половины единиц. Такие ключи показаны в таблице 1.

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Таблица 1 — Слабые ключи   |  |  | | --- | --- | | **Ключи до удаления проверочных бит (64 бита)** | **Действующие ключи (56 бит)** | | 0101 0101 0101 0101 | 0000000 0000000 | | 1F1F 1F1F 1F1F 1F1F | 0000000 FFFFFFF | | E0E0 E0E0 E0E0 E0E0 | FFFFFFF 0000000 | | FEFE FEFE FEFE FEFE | FFFFFFF FFFFFFF | |

Ключи раунда, созданные от любого из этих слабых ключей, — те же самые и имеют тот же самый тип, что и ключ шифра. Например, эти шестнадцать ключей раунда создают первый ключ, который состоит из всех нулей или всех единиц или наполовину из нулей и единиц.

Это происходит по той причине, что алгоритм генерирования ключей сначала делит ключ шифра на две половины. Смещение или перестановка блока не изменяют блок, если он состоит из всех нулей, или всех единиц, или наполовину из нулей и единиц.

В чем опасность использования слабых ключей? Если мы зашифровали блок слабым ключом и впоследствии расшифровали результат тем же самым слабым ключом, мы получаем первоначальный блок. Процесс создает один и тот же первоначальный блок, если мы расшифровываем блок дважды. Другими словами, каждый слабый ключ есть инверсия самого себя: как это показано на [рис. 5](https://intuit.ru/studies/professional_retraining/940/courses/408/lecture/9362?page=4#image.8.11).

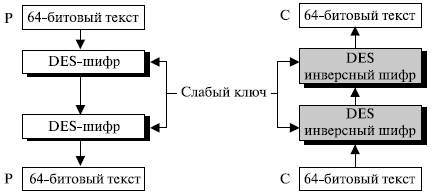


Рис. 5 —Двойное шифрование и дешифрование со слабым ключом

Слабых ключей надо избегать, потому что противник может легко распознать их на перехваченном шифре. Если после двух этапов дешифрации результат тот же самый, противник определяет, что он нашел ключ.

Сжатие файлов:

Таблица 2 – Сжатие файлов

|  |  |
| --- | --- |
|  |  |
|  |  |

Рис 6 – Зависимость времени шифрования/расшифрования от кол-ва символов

**Вывод:**

DES — алгоритм для симметричного шифрования, разработанный фирмой IBM и утверждённый правительством США в 1977 году как официальный стандарт. Размер блока для DES равен 64 битам. В основе алгоритма лежит сеть Фейстеля с 16 циклами и ключом, имеющим длину 56 бит.

В основе алгоритма лежит сеть Фейстеля с 16 циклами (раундами) и ключом, имеющим длину 56 бит. Алгоритм использует комбинацию нелинейных (S-блоки) и линейных (перестановки E, IP, IP-1) преобразований.