МИНОБРНАУКИ РОССИИ

Федеральное государственное бюджетное образовательное учреждение высшего образования

НИЖЕГОРОДСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ

УНИВЕРСИТЕТ им. Р.Е.АЛЕКСЕЕВА

Институт радиоэлектроники и информационных технологий

ОТЧЕТ

по лабораторной работе №3

«Шифр DES»

по дисциплине

«Методы и средства защиты информации»

РУКОВОДИТЕЛЬ:

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_ Капранов С. Н.\_\_

(подпись) (фамилия, и.,о.)

СТУДЕНТ:

\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Сухоруков В.А.\_\_\_

(подпись) (фамилия, и.,о.)

\_\_\_\_\_\_19-ВМ\_\_\_\_\_\_\_\_

(шифр группы)

Работа защищена «\_\_\_»\_\_\_\_\_\_\_\_\_\_\_\_

С оценкой \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Нижний Новгород 2022

# Задание №3.1

# Реализовать стандарт шифрования данных DES в режиме «Электронная кодовая книга».

# Описание алгоритма.

# 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 (раундами) и ключом, имеющим длину 56 бит. Алгоритм использует комбинацию нелинейных (S-блоки) и линейных (перестановки E, IP, IP-1) преобразований.

Схема шифрования алгоритма DES

# Исходный текст — блок 64 бит.

# Процесс шифрования состоит из начальной перестановки, 16 циклов шифрования и конечной перестановки.

# Исходный текст Т (блок 64 бит) преобразуется c помощью *начальной перестановки*  IP которая определяется таблицей 1:

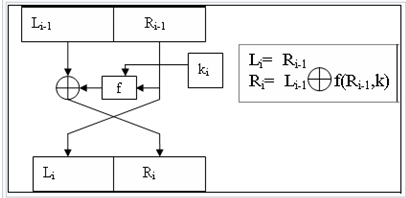


# Полученный после начальной перестановки 64-битовый блок IP(T) участвует в *16 циклах преобразования Фейстеля*:

# Разбить IP(T) на две части  L0,R0, где L0,R0 — соответственно 32 старших битов и 32 младших битов блока  T\_0

# Пусть  Ti-1=Li-1 Ri-1результат (i-1) итерации, тогда результат i-ой итерации Ti=Li Riопределяется:





# Левая половина равна правой половине предыдущего вектора . А правая половина — это битовое сложение Li-1 и  f(Ri-1, Ki) по модулю 2.

# В 16-циклах преобразования Фейстеля [функция](https://ru.wikipedia.org/wiki/%D0%A4%D1%83%D0%BD%D0%BA%D1%86%D0%B8%D1%8F_(%D0%BC%D0%B0%D1%82%D0%B5%D0%BC%D0%B0%D1%82%D0%B8%D0%BA%D0%B0)) f играет роль [шифрования](https://ru.wikipedia.org/wiki/%D0%A8%D0%B8%D1%84%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5). Рассмотрим подробно функцию f.

### *Основная функция шифрования (функция Фейстеля)*

# [Аргументами](https://ru.wikipedia.org/wiki/%D0%90%D1%80%D0%B3%D1%83%D0%BC%D0%B5%D0%BD%D1%82_(%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5)) функции f являются 32-битовый [вектор](https://ru.wikipedia.org/wiki/%D0%98%D0%BD%D0%B4%D0%B5%D0%BA%D1%81%D0%BD%D1%8B%D0%B9_%D0%BC%D0%B0%D1%81%D1%81%D0%B8%D0%B2)  Ri-1 и 48-битовый ключ Ki ,который является результатом преобразования 56-битового исходного ключа шифра  К. Для вычисления функции f последовательно используются

# [функция](https://ru.wikipedia.org/wiki/%D0%A4%D1%83%D0%BD%D0%BA%D1%86%D0%B8%D1%8F_(%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5)) расширения E,

# сложение по модулю 2 с ключом  Ki

# преобразование S, состоящее из 8 преобразований S-блоков ,

# [перестановка](https://ru.wikipedia.org/wiki/%D0%9F%D0%B5%D1%80%D0%B5%D1%81%D1%82%D0%B0%D0%BD%D0%BE%D0%B2%D0%BA%D0%B0) P.

# Функция E расширяет 32-битовый [вектор](https://ru.wikipedia.org/wiki/%D0%98%D0%BD%D0%B4%D0%B5%D0%BA%D1%81%D0%BD%D1%8B%D0%B9_%D0%BC%D0%B0%D1%81%D1%81%D0%B8%D0%B2) Ri-1 до 48-битового вектора ERi-1 путём дублирования некоторых битов из Ri-1.

# Полученный после перестановки блок ERi-1 (R\_{i-1})} складывается по модулю 2 с ключами Ki и затем представляется в виде восьми последовательных блоков  B1,B2,...B8.

# Каждый  Bj является 6-битовым блоком. Далее каждый из блоков Bj трансформируется в 4-битовый [блок](https://ru.wikipedia.org/wiki/%D0%91%D0%BB%D0%BE%D0%BA) B’j с помощью преобразований Sj.

# *Генерирование ключей*

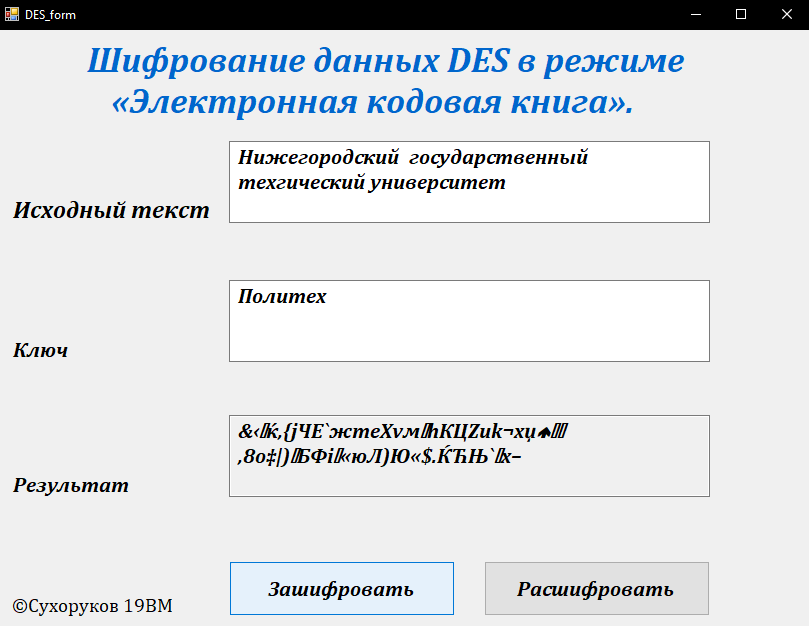
# Ключи Ki получаются путём побитового сдвига влево на определённое количество бит в зависимости от раунда шифрования.

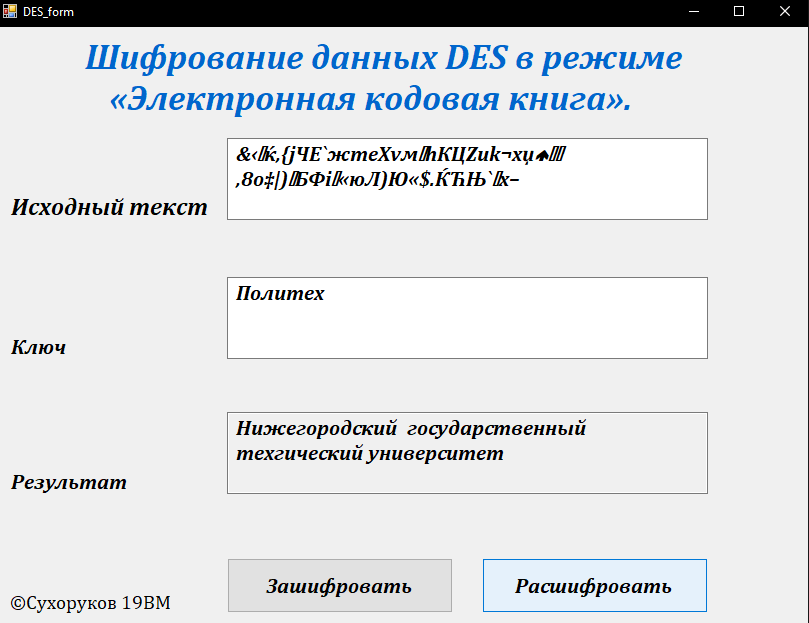
# *Конечная перестановка*

# Конечная [перестановка](https://ru.wikipedia.org/wiki/%D0%9F%D0%B5%D1%80%D0%B5%D1%81%D1%82%D0%B0%D0%BD%D0%BE%D0%B2%D0%BA%D0%B0) IP -1 действует на  T16-1 =R16+L16 и является обратной к первоначальной перестановке.

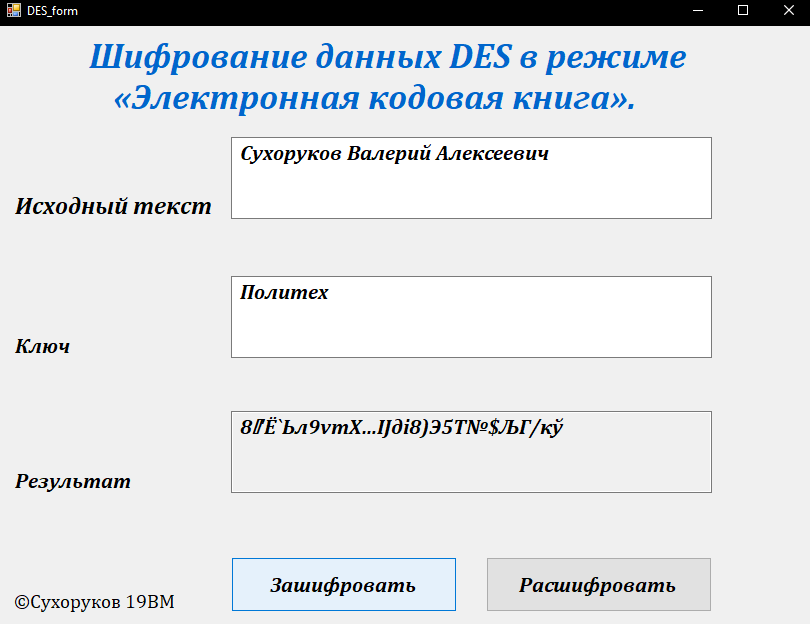
# Примеры шифрования и расшифровывания

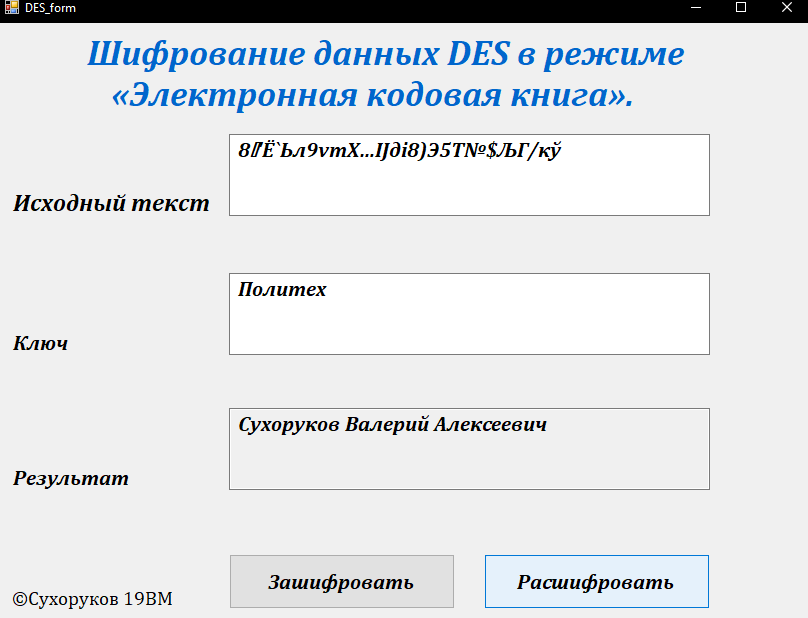
* Нижегородский государственный технический университет





* Сухоруков Валерий Алексеевич





# Текст программы

* Функция зашифровки всего текста

//Функция, вызываемая при нажатии на кнопку "Зашифровать" в форме.

//Устанавливает в текстовое поле 3 результат функции шифрования.

System::Void LR3UI::DES\_form::button1\_Click(System::Object^ sender, System::EventArgs^ e) {

string text = "";

string key = "";

string result = "";

text = marshal\_as<string>(textBox1->Text);

key = marshal\_as<string>(textBox2->Text);

string coderText = "";

bitset<64> resKey(0);

//смещение

size\_t offset = 0;

//конвертация ключа в последовательность бит

//проход по каждому символу

for (size\_t itKey = 0; itKey < key.length(); itKey++) {

//конвертируем символ в биты

bitset<8> val(key[itKey]);

//Записываем значение 8 бит текущего символа

for (size\_t i = 0; i < 8; i++) {

//заполняем результирущий список

resKey[offset + i] = val[i];

}

//Сдвигаемся на 8 бит

offset += 8;}

//Если длина ключа меньше 8 символов,

//заполняем оставшуюся часть нулями

for (size\_t itKey = offset; itKey < 64; itKey++) {

resKey[itKey] = 0;

}

//Конвертация строки открытого текста в последовательность бит

for (size\_t itText = 0; itText < text.length(); itText = itText + 8)

{

bitset<64> res(0);

offset = 0;

//Выделяем блоки длиной 8 символов - 64 бита

for (size\_t i = 0; i < 8; i++) {

//Один символ

bitset<8> val;

//Если длина строки некратна 8, то заполняем нулевыми битами

if ((itText + i) < text.length()) {

val = bitset<8>(text[itText + i]);

}

else {

val = bitset<8>(0);

}

//Записываем 8 бит в 64 битовую последовательность

for (size\_t i = 0; i < 8; i++) { res[offset + i] = val[i]; }

//Сдвигаемся на 8 символов

offset += 8;

}

//Вызываем функцию шифрации для полученных 64 бит

result += coder(res, resKey);

}

String^ myString = marshal\_as<System::String^>(result);

textBox3->Text = myString;

return System::Void();

}

* Функция зашифровки 64 бит текста

//Функция шифрования методом DES в режиме "Электронная книга".

string coder(bitset<64> res, bitset<64> resKey) {

//разделение сообщения на блоки

bitset<64> resIP(0);

size\_t spot = 0;

//Начальная перестановка бит по таблице IP

for (size\_t i = 0; i < 8; i++) {

for (size\_t j = 0; j < 8; j++) {

spot = IP[i][j];

resIP[i \* 8 + j] = res[spot - 1];

}

}

//правые и левые биты сообщения

bitset<32> L(0); bitset<32> R(0);

//Разделение битов

sortLR(L, R, resIP);

//Две половины ключа

bitset<28> C(0); bitset<28> D(0);

//Получение 56 битного ключа

sortCD(C, D, resKey);

//Ключ i-того раунда

bitset<48> Ki(0);

//16 Раундов шифрования

for (size\_t c = 0; c < 16; c++) {

getKi(C, D, c, Ki);

bitset<32> resB(0);

F(R, Ki, resB);

bitset<32> tmp1 = L;

bitset<32> tmp2 = R;

L = tmp2;

R = tmp1 ^ resB;

}

//Составляем из двух половин целую строку 64 бит

bitset<64> resultLR(0);

for (size\_t i = 0; i < 64; i++) {

if (i < 32) { resultLR[i] = L[i]; }

else { resultLR[i] = R[i - 32]; }

}

//Конечная перестановка

bitset<64> FP(0);

spot = 0;

for (size\_t i = 0; i < 8; i++) {

for (size\_t j = 0; j < 8; j++) {

spot = finalIP[i][j];

FP[i \* 8 + j] = resultLR[spot - 1];

}

}

//конвертируем биты в символы

string result = convertToString(FP);

return result;

}

* Функция F сети Фестеля

//Функция F алгоритма шифрования DES

System::Void F(bitset<32>& R, bitset<48>& Ki, bitset<32>& resB) {

//Расширение части текста по таблице Е

bitset<48> ER(0);

size\_t spot = 0;

size\_t k = 0;

for (size\_t i = 0; i < 8; i++) {

for (size\_t j = 0; j < 6; j++, k++) {

spot = E[i][j];

ER[k] = R[spot - 1];

}

}

//Сложение расширенной правой части с ключом по модулю 2

ER ^= Ki;

//Делим полученное значение на 8 блоков по 6 бит

bitset<6> ER1(0); bitset<6> ER2(0);

bitset<6> ER3(0); bitset<6> ER4(0);

bitset<6> ER5(0); bitset<6> ER6(0);

bitset<6> ER7(0); bitset<6> ER8(0);

//заносим значения в данные блоки по порядку

for (size\_t i = 0; i < 6; i++) {

ER1[i] = ER[i]; ER2[i] = ER[i + 6];

ER3[i] = ER[i + 12]; ER4[i] = ER[i + 18];

ER5[i] = ER[i + 24]; ER6[i] = ER[i + 30];

ER7[i] = ER[i + 36]; ER8[i] = ER[i + 42];

}

//переменная, хранящая номер строки

bitset<2> row(0);

//переменная, хранящая номер столбца

bitset<4> column(0);

bitset<32> value\_from\_S(0);

//Поиск элемента в талбицах S

for (size\_t i = 0; i < 8; i++) {

bitset<6> tmp\_ER;

if (i == 0) { tmp\_ER = ER1; } if (i == 1) { tmp\_ER = ER2; }

if (i == 2) { tmp\_ER = ER3; } if (i == 3) { tmp\_ER = ER4; }

if (i == 4) { tmp\_ER = ER5; } if (i == 5) { tmp\_ER = ER6; }

if (i == 6) { tmp\_ER = ER7; } if (i == 7) { tmp\_ER = ER8; }

//Поиск положения элемента в таблице S

row[0] = tmp\_ER[0]; row[1] = tmp\_ER[5];

column[0] = tmp\_ER[1]; column[1] = tmp\_ER[2];

column[2] = tmp\_ER[3]; column[3] = tmp\_ER[4];

//Переменная, хранящая значение нужной ячейки

size\_t number;

//Поиск значения в таблицах S

if (i == 0){ number = S1[row.to\_ulong()][column.to\_ulong()] - 1; }

if (i == 1){ number = S2[row.to\_ulong()][column.to\_ulong()] - 1; }

if (i == 2){ number = S3[row.to\_ulong()][column.to\_ulong()] - 1; }

if (i == 3){ number = S4[row.to\_ulong()][column.to\_ulong()] - 1; }

if (i == 4){ number = S5[row.to\_ulong()][column.to\_ulong()] - 1; }

if (i == 5){ number = S6[row.to\_ulong()][column.to\_ulong()] - 1; }

if (i == 6){ number = S7[row.to\_ulong()][column.to\_ulong()] - 1; }

if (i == 7){ number = S8[row.to\_ulong()][column.to\_ulong()] - 1; }

//Получаем последовательность из 4 бит

bitset<4> B(number);

//Составляем последовательность из 32 бит

for (size\_t j = 0; j < 4; j++) {

value\_from\_S[i \* 4 + j] = B[j];

}

}

//Перестановка бит в соответствии с таблицей P

spot = 0;

for (size\_t i = 0; i < 8; i++) {

for (size\_t j = 0; j < 4; j++) {

spot = P[i][j];

resB[i \* 4 + j] = value\_from\_S[spot - 1];

}

}

return System::Void();

}

* Функция расшифровки текста

//Функция, вызываемая при нажатии на кнопку "Расшифровать" в форме.

//Устанавливает в текстовое поле 3 результат функции расшифрования.

System::Void LR3UI::DES\_form::button2\_Click(System::Object^ sender, System::EventArgs^ e) {

string result = "";

string key = "";

string decoderResult = "";

result = marshal\_as<string>(textBox1->Text);

key = marshal\_as<string>(textBox2->Text);

bitset<64> resKey(0);

//Смещение

size\_t offset = 0;

//конвертация ключа в последовательность бит

//проход по каждому символу

for (size\_t itKey = 0; itKey <key.length(); itKey++) {

//конвертируем символ в биты

bitset<8> val(key[itKey]);

//Записываем значение 8 бит текущего символа

for (size\_t i = 0; i < 8; i++) {

//заполняем результирущий список

resKey[offset + i] = val[i];

}

//Сдвигаемся на 8 бит

offset += 8;

}

//Если длина ключа меньше 8 символов,

//заполняем оставшуюся часть нулями

for (size\_t itKey = offset; itKey < 64; itKey++){

resKey[itKey] = 0;

}

//Конвертация строки закрытого текста в последовательность бит

for (size\_t itResultText = 0; itResultText < result.length(); itResultText = itResultText+8) {

bitset<64> res(0);

offset = 0;

//Выделяем блоки длиной 8 символов - 64 бита

for (size\_t i = 0; i < 8; i++){

//Один символ

bitset<8> val;

//Если длина строки некратна 8, то заполняем нулевыми битами

if ((itResultText + i)< result.length()){

val= bitset<8> (result[itResultText + i]);

}

else {

val = bitset<8>(0);

}

//Записываем 8 бит в 64 битовую последовательность

for (size\_t i = 0; i < 8; i++){res[offset + i] = val[i];}

//Сдвигаемся на 8 символов

offset += 8;

}

//Вызываем функцию дешифрации для полученных 64 бит

decoderResult += deCoder(res, resKey);

}

//Приведение типа для вывода строки в текстовое поле

String^ myString = marshal\_as<System::String^>(decoderResult);

textBox3->Text = myString;

return System::Void();

}

* Функция расшифровки 64 бит текста

//Функция расшифрования методом DES в режиме "Электронная книга".

string deCoder(bitset<64> res, bitset<64> resKey) {

bitset<64> resIP(0);

size\_t spot = 0;

//Конечная перестановка

//Присваиваем результирующему списку значение бита по таблице IP наоборот

for (size\_t i = 0; i < 8; i++) {

for (size\_t j = 0; j < 8; j++) {

spot = finalIP[i][j];

resIP[spot - 1] = res[i \* 8 + j];

}

}

bitset<32> L(0);

bitset<32> R(0);

//разделение битов на левые и правые

sortLR(L, R, resIP);

bitset<28> C(0);

bitset<28> D(0);

sortCD(C, D, resKey);

bitset<48> Ki(0);

//Проходим 16 раундов в обратном порядке

for (size\_t c = 16; c > 0; c--) {

getDecoderKi(C, D, c, Ki);

bitset<32> resB(0);

F(L, Ki, resB);

bitset<32> tmp1 = L;

bitset<32> tmp2 = R;

R = tmp1;

L = tmp2 ^ resB;

}

//Составляем из двух половин целую строку 64 бит

bitset<64> resultLR(0);

for (size\_t i = 0; i < 64; i++) {

if (i < 32) { resultLR[i] = L[i]; }

else { resultLR[i] = R[i - 32]; }

}

bitset<64> FP(0);

spot = 0;

//Начальная перестановка в обратном порядке

for (size\_t i = 0; i < 8; i++) {

for (size\_t j = 0; j < 8; j++) {

spot = IP[i][j];

FP[spot - 1] = resultLR[i \* 8 + j];

}

}

//конвертируем биты в символы

string result = convertToString(FP);

return result;

}