Федеральное государственное бюджетное образовательное учреждение высшего образования

**«НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ МОСКОВСКИЙ ГОСУДАРСТВЕННЫЙ СТРОИТЕЛЬНЫЙ УНИВЕРСИТЕТ»**

*Институт ЭУИС*

*Кафедра ИСТАС*

**Практическая работа №3**

по дисциплине:

«Защита информации»

**Вариант: DES-шифрование.**

|  |
| --- |
| Сдал: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  (подпись, дата) |
| Торговцев В.П. ИЭУИС 4-2 |
| Принял: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  (подпись, дата) |
| Алексеевская Я.А. |

г. Москва

2020 г.

**Описание программы.**

На рисунке 1 представлен вид главного окна программы.

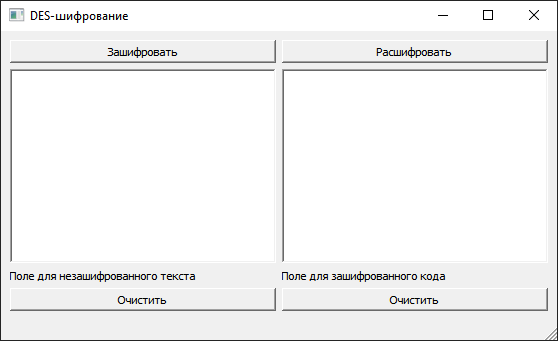


Рисунок 1 - Главное окно

Сверху имеется 2 кнопки – “Зашифровать” и “Расшифровать” для запуска шифрования и дешифрования текста, снизу 2 кнопки для очистки левого и правого поля ввода.

Между этими двумя блоками кнопок находится 2 поля для ввода.

Левое поле предназначено для ввода текста для шифрования.

Правое поле предназначено для ввода зашифрованного кода.

**Инструкция.**

Запуск:

Вариант 1. Открыть файл Torgovtsev\_Practice\_3.exe в папке « ProjectForWindows».

Вариант 2. Если установлен Qt, запустить Torgovtsev\_Practice\_3.pro в папке «project for Qt5.14.2», нажать кнопку «Настроить проект» и запустить проект комбинацией клавиш Ctrl+R.

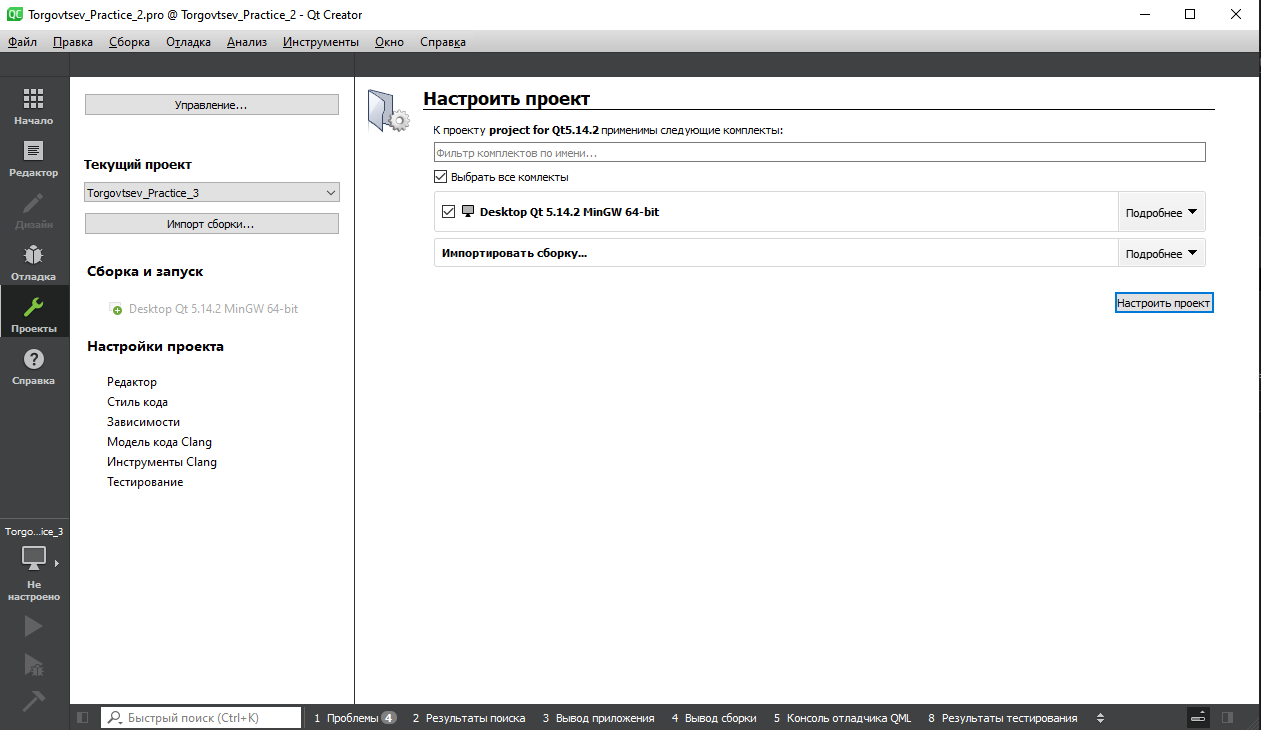


Рисунок 2 – Запуск программы

Ход программы:

1. Для того, чтобы зашифровать текст:
   1. Введите текст для шифрования в поле над надписью “Поле для незашифрованного текста”. Рисунок 3

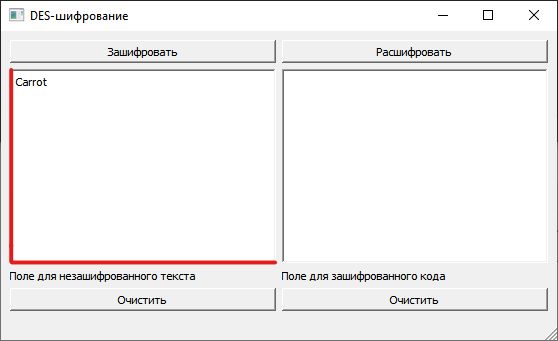
Текст можно вводить как на русском, так и на английском языках.

Рисунок 3 – Ввод текста для шифрования

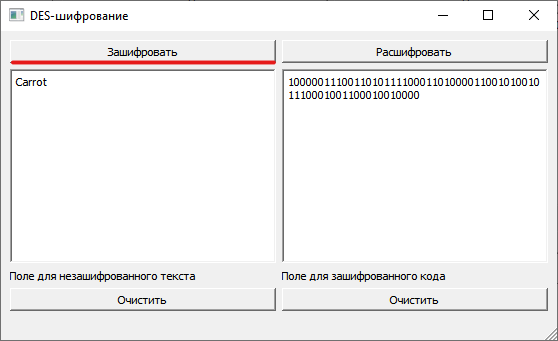
* 1. Нажмите на кнопку «Зашифровать», результат появится в правом поле вывода над надписью «Поле для зашифрованного кода». Рисунок 4

Рисунок 4 – Результат шифрования

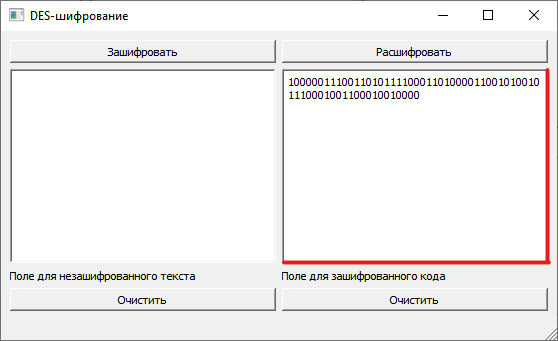
1. Для того, чтобы расшифровать двоичный код:
   1. Если вы до этого производили процесс шифрования, то нажмите на клавишу «Очистить» под левым полем ввода. Если нет, то введите двоичный код в поле над надписью ”Поле для зашифрованного кода”. Рисунок 5

Рисунок 5 – Ввод двоичного кода для дешифрования

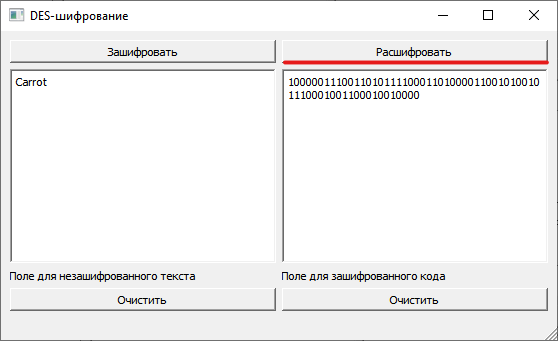
* 1. Нажмите на кнопку «Расшифровать», результат появится в левом поле вывода над надписью «Поле для незашифрованного текста». Рисунок 6

Рисунок 6 – Результат дешифрования

Исходный код программы:

Файл “mainwindow.h”:

#ifndef MAINWINDOW\_H

#define MAINWINDOW\_H

#include <QMainWindow>

#include <iostream>

#include <bitset>

QT\_BEGIN\_NAMESPACE

namespace **Ui** { class **MainWindow**; }

QT\_END\_NAMESPACE

class **MainWindow** : public QMainWindow

{

Q\_OBJECT

public:

**MainWindow**(QWidget \*parent = nullptr);

~***MainWindow***();

private slots:

void **on\_pushButton\_encrypt\_clicked**();

void **on\_pushButton\_decrypt\_clicked**();

void **on\_actionClear\_Plain\_Text\_clicked**();

void **on\_actionClear\_Encrypted\_Text\_clicked**();

private:

Ui::MainWindow \*ui;

QString key = "";

QString **ewIDES**(QString plainText, std::string key);

QString **dwIDES**(QString encryptedText, std::string key);

std::vector<std::string> **keyPreparation**(std::string key);

std::vector< std::string > **textToBinaryAscii**(std::string str);

std::string **binaryAsciiToText**(std::string str);

QString **DESEncryption**(std::string dataBlock, std::vector< std::string > keys);

std::string **charToBinaryAscii**(char ch);

char **binaryAsciiToChar**(std::string binaryAscii);

std::string **apply\_func\_E**(std::string str);

std::string **apply\_func\_F**(std::string str1, std::string str2);

std::string **apply\_xor**(std::string str1, std::string str2);

};

#endif // MAINWINDOW\_H

Файл “main.cpp”:

#include "mainwindow.h"

#include <QApplication>

int main(int argc, char \*argv[])

{

QApplication a(*argc*, argv);

MainWindow w;

w.show();

return a.exec();

}

Файл “mainwindow.cpp”:

#include "mainwindow.h"

#include "ui\_mainwindow.h"

MainWindow::**MainWindow**(QWidget \*parent)

: QMainWindow(parent)

, ui(new Ui::MainWindow)

{

ui->setupUi(this);

for (int f = 0; f < 8; f++)//----------Генерация ключа

{

key[f] = rand()%'z'+'a';

}

}

MainWindow::~***MainWindow***()

{

delete ui;

}

void MainWindow::**on\_pushButton\_encrypt\_clicked**()//----Кнопка начать шифрование

{

ui->plainTextEdit\_encrypted->clear();

QString plainText = ui->plainTextEdit\_plain->toPlainText();

QString encryptedText = "to be implemented";

encryptedText = ewIDES(plainText, key.toStdString());

ui->plainTextEdit\_encrypted->document()->setPlainText(encryptedText);

}

void MainWindow::**on\_pushButton\_decrypt\_clicked**()//--Кнопка начать дешифровку

{

ui->plainTextEdit\_plain->clear();

QString encryptedText = ui->plainTextEdit\_encrypted->toPlainText();

QString plainText = "to be implemented";

plainText = dwIDES(encryptedText, key.toStdString());

ui->plainTextEdit\_plain->document()->setPlainText(plainText);

}

void MainWindow::**on\_actionClear\_Plain\_Text\_clicked**()//--Кнопка очистить левый экран

{

ui->plainTextEdit\_plain->clear();

}

void MainWindow::**on\_actionClear\_Encrypted\_Text\_clicked**()//--Кнопка очистить правый экран

{

ui->plainTextEdit\_encrypted->clear();

}

QString MainWindow::**ewIDES**(QString plainText, std::string key)//--Запуск Шифрования

{

std::vector< std::string > keys = keyPreparation(key);

std::vector< std::string > blocks = textToBinaryAscii(plainText.toStdString());

QString encryptedText;

for(int i=0; i<blocks.size(); ++i)

encryptedText += DESEncryption(blocks[i], keys);

// encryptedText = QString::fromStdString(binaryAsciiToText(encryptedText.toStdString()));

return encryptedText;

}

QString MainWindow::**dwIDES**(QString encryptedText, std::string key)//--Запуск Дешифрования

{

std::vector< std::string > keys = keyPreparation(key);

std::reverse(keys.begin(), keys.end());

std::string stdPlainText = encryptedText.toStdString();

std::string allPlainText = "";

for(int i=0; i<encryptedText.size()/64; ++i)

{

allPlainText += DESEncryption(stdPlainText.substr(i\*64, 64), keys).toStdString();

}

return QString::fromStdString(binaryAsciiToText(allPlainText));

}

std::vector< std::string > MainWindow::**keyPreparation**(std::string key)//----подготовка ключа

{

std::string permutedKey = "";

int PC1\_permutations[56] = {

57, 49, 41, 33, 25, 17, 9, 1, 58, 50, 42, 34, 26, 18,

10, 2, 59, 51, 43, 35, 27, 19, 11, 3, 60, 52, 44, 36,

63, 55, 47, 39, 31, 23, 15, 7, 62, 54, 46, 38, 30, 22,

14, 6, 61, 53, 45, 37, 29, 21, 13, 5, 28, 20, 12, 4

};

for(int i=0;i<56; ++i)

{

permutedKey += key[PC1\_permutations[i]-1];

}

// std::cout << key << std::endl;

// std::cout << permutedKey << std::endl;

// std::cout << permutedKey.length() << std::endl;

int shifts[16] = {1, 1, 2, 2, 2, 2, 2, 2, 1, 2, 2, 2, 2, 2, 2, 1};

std::vector< std::pair<std::string, std::string> > keys;

std::string C0 = permutedKey.substr(0,28);

std::string D0 = permutedKey.substr(28,56);

// std::cout << "C0 : " << C0 << std::endl;

// std::cout << "D0 : " << D0 << std::endl;

keys.push\_back(std::make\_pair(*C0*, *D0*));

for(int i=1 ; i<17 ; ++i)

{

std::string C = keys[i-1].first;

std::string D = keys[i-1].second;

C = C.substr(shifts[i-1], C.length()) + C.substr(0, shifts[i-1]);

D = D.substr(shifts[i-1], D.length()) + D.substr(0, shifts[i-1]);

keys.push\_back(std::make\_pair(*C*, *D*));

// std::cout << "C" << i << " : " << C << std::endl;

// std::cout << "D" << i << " : " << D << std::endl;

}

int PC2\_permutations[48] = {

14, 17, 11, 24, 1, 5, 3, 28, 15, 6, 21, 10,

23, 19, 12, 4, 26, 8, 16, 7, 27, 20, 13, 2,

41, 52, 31, 37, 47, 55, 30, 40, 51, 45, 33, 48,

44, 49, 39, 56, 34, 53, 46, 42, 50, 36, 29, 32

};

std::vector<std::string> finalKeys;

for(int i=0;i<16;++i)

{

std::string k = keys[i+1].first + keys[i+1].second;

std::string fk = "";

for(int j=0;j<48;++j)

{

fk += k[PC2\_permutations[j]-1];

}

finalKeys.push\_back(fk);

// std::cout << fk << std::endl;

}

std::cout << std::endl << ">[BEGIN] Keys generated Successfully" << std::endl << std:: endl;

// for(int i=0; i<finalKeys.size(); ++i)

// {

// std::cout << "Key " << i+1 << ": " << finalKeys[i] << std::endl;

// }

// std::cout << std::endl << ">[END] Keys generated Successfully" << std::endl << std:: endl;

return finalKeys;

}

std::vector< std::string > MainWindow::**textToBinaryAscii**(std::string str)//-----текст в двоичный код

{

std::vector< std::string > blocksOfData;

for(int i=0; i<str.size()/8; ++i)

{

std::string blockStr = str.substr(i\*8, 8);

std::string block = "";

for(int i=0; i<8; ++i)

block += charToBinaryAscii(blockStr[i]);

blocksOfData.push\_back(block);

}

if(str.size()%8 != 0)

{

int start = ((int)str.size()/8)\*8;

int length = (int)str.size()- start;

std::string blockStr = str.substr(start, length);

for(int i=0; i<(8-length); ++i)

blockStr += " ";

std::string block = "";

for(int i=0; i<8; ++i)

block += charToBinaryAscii(blockStr[i]);

blocksOfData.push\_back(block);

}

return blocksOfData;

}

std::string MainWindow::**binaryAsciiToText**(std::string str)//-----Двоичный код в текст

{

std::string blocksOfData = "";

for(int i=0; i<str.size()/64; ++i)

{

std::string blockStr = str.substr(i\*64, 64);

std::string block = "";

for(int i=0; i<8; ++i)

{

std::string ascii = blockStr.substr(i\*8, 8);

block += binaryAsciiToChar(ascii);

}

blocksOfData += block;

}

return blocksOfData;

}

QString MainWindow::**DESEncryption**(std::string dataBlock, std::vector< std::string > keys)//---Шифрование

{

// 0000000100100011010001010110011110001001101010111100110111101111

std::string permutedBlock = "";

int initPermutation[64] = {

58, 50, 42, 34, 26, 18, 10, 2,

60, 52, 44, 36, 28, 20, 12, 4,

62, 54, 46, 38, 30, 22, 14, 6,

64, 56, 48, 40, 32, 24, 16, 8,

57, 49, 41, 33, 25, 17, 9, 1,

59, 51, 43, 35, 27, 19, 11, 3,

61, 53, 45, 37, 29, 21, 13, 5,

63, 55, 47, 39, 31, 23, 15, 7

};

for(int i=0 ; i<64 ; ++i)

{

permutedBlock += dataBlock[initPermutation[i]-1];

}

std::string L0 = permutedBlock.substr(0, 32);

std::string R0 = permutedBlock.substr(32, 64);

std::cout << "L0 : " << L0 << std::endl;

std::cout << "R0 : " << R0 << std::endl;

std::vector< std::pair<std::string, std::string> > data;

data.push\_back(std::make\_pair(*L0*, *R0*));

for(int i=1;i<17;++i)

{

std::string L = data[i-1].second;

std::string R = apply\_xor(

data[i-1].first,

apply\_func\_F(apply\_func\_E(data[i-1].second), keys[i-1])

);

data.push\_back(std::make\_pair(*L*, *R*));

std::cout << "L" << i << " : " << L << std::endl;

std::cout << "R" << i << " : " << R << std::endl;

}

std::string encryptedDataReversedKey = "";

encryptedDataReversedKey += data[data.size()-1].second;

encryptedDataReversedKey += data[data.size()-1].first;

std::string finalPermutedBlock = "";

int finalPermutation[64] = {

40, 8, 48, 16, 56, 24, 64, 32,

39, 7, 47, 15, 55, 23, 63, 31,

38, 6, 46, 14, 54, 22, 62, 30,

37, 5, 45, 13, 53, 21, 61, 29,

36, 4, 44, 12, 52, 20, 60, 28,

35, 3, 43, 11, 51, 19, 59, 27,

34, 2, 42, 10, 50, 18, 58, 26,

33, 1, 41, 9, 49, 17, 57, 25

};

for(int i=0 ; i<64 ; ++i)

{

finalPermutedBlock += encryptedDataReversedKey[finalPermutation[i]-1];

}

return QString(finalPermutedBlock.c\_str());

}

std::string MainWindow::**charToBinaryAscii**(char ch)//---Перевод Char в двоичный код

{

return std::bitset<8>(int(ch)).to\_string();

}

char MainWindow::**binaryAsciiToChar**(std::string binaryAscii)//----Перевод Двоичного кода в Char

{

return char(std::bitset<8>(binaryAscii).to\_ulong());

}

std::string MainWindow::**apply\_func\_E**(std::string str)//---Выполнение функции

{

std::string result = "";

int ePermutations[48] = {

32, 1, 2, 3, 4, 5,

4, 5, 6, 7, 8, 9,

8, 9, 10, 11, 12, 13,

12, 13, 14, 15, 16, 17,

16, 17, 18, 19, 20, 21,

20, 21, 22, 23, 24, 25,

24, 25, 26, 27, 28, 29,

28, 29, 30, 31, 32, 1

};

for(int i=0;i<48;++i)

{

result += str[ePermutations[i]-1];

}

return result;

}

std::string MainWindow::**apply\_func\_F**(std::string str1, std::string str2)//---Выполнение функции

{

// result is 48 bit

std::string result = apply\_xor(str1, str2);

int sboxes[8][4][16] =

{

{

{ 14, 4, 13, 1, 2, 15, 11, 8, 3, 10, 6, 12, 5, 9, 0, 7 },

{ 0, 15, 7, 4, 14, 2, 13, 1, 10, 6, 12, 11, 9, 5, 3, 8 },

{ 4, 1, 14, 8, 13, 6, 2, 11, 15, 12, 9, 7, 3, 10, 5, 0 },

{ 15, 12, 8, 2, 4, 9, 1, 7, 5, 11, 3, 14, 10, 0, 6, 13 }

},

{

{ 15, 1, 8, 14, 6, 11, 3, 4, 9, 7, 2, 13, 12, 0, 5, 10 },

{ 3, 13, 4, 7, 15, 2, 8, 14, 12, 0, 1, 10, 6, 9, 11, 5 },

{ 0, 14, 7, 11, 10, 4, 13, 1, 5, 8, 12, 6, 9, 3, 2, 15 },

{ 13, 8, 10, 1, 3, 15, 4, 2, 11, 6, 7, 12, 0, 5, 14, 9 }

},

{

{ 10, 0, 9, 14, 6, 3, 15, 5, 1, 13, 12, 7, 11, 4, 2, 8 },

{ 13, 7, 0, 9, 3, 4, 6, 10, 2, 8, 5, 14, 12, 11, 15, 1 },

{ 13, 6, 4, 9, 8, 15, 3, 0, 11, 1, 2, 12, 5, 10, 14, 7 },

{ 1, 10, 13, 0, 6, 9, 8, 7, 4, 15, 14, 3, 11, 5, 2, 12 }

},

{

{ 7, 13, 14, 3, 0, 6, 9, 10, 1, 2, 8, 5, 11, 12, 4, 15 },

{ 13, 8, 11, 5, 6, 15, 0, 3, 4, 7, 2, 12, 1, 10, 14, 9 },

{ 10, 6, 9, 0, 12, 11, 7, 13, 15, 1, 3, 14, 5, 2, 8, 4 },

{ 3, 15, 0, 6, 10, 1, 13, 8, 9, 4, 5, 11, 12, 7, 2, 14 }

},

{

{ 2, 12, 4, 1, 7, 10, 11, 6, 8, 5, 3, 15, 13, 0, 14, 9 },

{ 14, 11, 2, 12, 4, 7, 13, 1, 5, 0, 15, 10, 3, 9, 8, 6 },

{ 4, 2, 1, 11, 10, 13, 7, 8, 15, 9, 12, 5, 6, 3, 0, 14 },

{ 11, 8, 12, 7, 1, 14, 2, 13, 6, 15, 0, 9, 10, 4, 5, 3 }

},

{

{ 12, 1, 10, 15, 9, 2, 6, 8, 0, 13, 3, 4, 14, 7, 5, 11 },

{ 10, 15, 4, 2, 7, 12, 9, 5, 6, 1, 13, 14, 0, 11, 3, 8 },

{ 9, 14, 15, 5, 2, 8, 12, 3, 7, 0, 4, 10, 1, 13, 11, 6 },

{ 4, 3, 2, 12, 9, 5, 15, 10, 11, 14, 1, 7, 6, 0, 8, 13 }

},

{

{ 4, 11, 2, 14, 15, 0, 8, 13, 3, 12, 9, 7, 5, 10, 6, 1 },

{ 13, 0, 11, 7, 4, 9, 1, 10, 14, 3, 5, 12, 2, 15, 8, 6 },

{ 1, 4, 11, 13, 12, 3, 7, 14, 10, 15, 6, 8, 0, 5, 9, 2 },

{ 6, 11, 13, 8, 1, 4, 10, 7, 9, 5, 0, 15, 14, 2, 3, 12 }

},

{

{ 13, 2, 8, 4, 6, 15, 11, 1, 10, 9, 3, 14, 5, 0, 12, 7 },

{ 1, 15, 13, 8, 10, 3, 7, 4, 12, 5, 6, 11, 0, 14, 9, 2 },

{ 7, 11, 4, 1, 9, 12, 14, 2, 0, 6, 10, 13, 15, 3, 5, 8 },

{ 2, 1, 14, 7, 4, 10, 8, 13, 15, 12, 9, 0, 3, 5, 6, 11 }

}

};

std::string output = "";

std::string outerBits = "";

std::string innerBits = "";

for(int i=0, s=0 ; i<48 ; i+=6, s++)

{

outerBits += result[i];

outerBits += result[i+5];

innerBits += result[i+1];

innerBits += result[i+2];

innerBits += result[i+3];

innerBits += result[i+4];

int row = std::stoi(outerBits, nullptr, 2);

int column = std::stoi(innerBits, nullptr, 2);

int valInSBox = sboxes[s][row][column];

output += std::bitset<4>(valInSBox).to\_string();

outerBits = "";

innerBits = "";

}

std::string permutedOutput = "";

int permutations[32] = {

16, 7, 20, 21,

29, 12, 28, 17,

1, 15, 23, 26,

5, 18, 31, 10,

2, 8, 24, 14,

32, 27, 3, 9,

19, 13, 30, 6,

22, 11, 4, 25

};

for(int i=0; i<32; ++i)

{

permutedOutput += output[permutations[i]-1];

}

return permutedOutput;

}

std::string MainWindow::**apply\_xor**(std::string str1, std::string str2)// -- Исключаещее ИЛИ

{

if(str1.length() != str2.length())

std::cout << "Error in XORed Strings, Length Not Equal" << std::endl;

std::string result = "";

for(int i=0;i<str1.length();++i)

{

if(str1[i] == str2[i])

result += '0';

else

result += '1';

}

return result;

}