

Лабораторная работа №3

Тема: Шифрование с помощью алгоритма DES

Цель: Выполнить программную реализацию алгоритма DES

Ход работы

Задание. Разработать программу, которая будет выполнять шифрование симметричным ключём открытого текста из файла и сохранять результат в другой файл. Ключ так же считывается из файла.

Изучим алгоритм работы DES (рисунок 1.1 и 1.2), обратив внимание на 16ти- раундный алгоритм генерации промежуточных ключей.

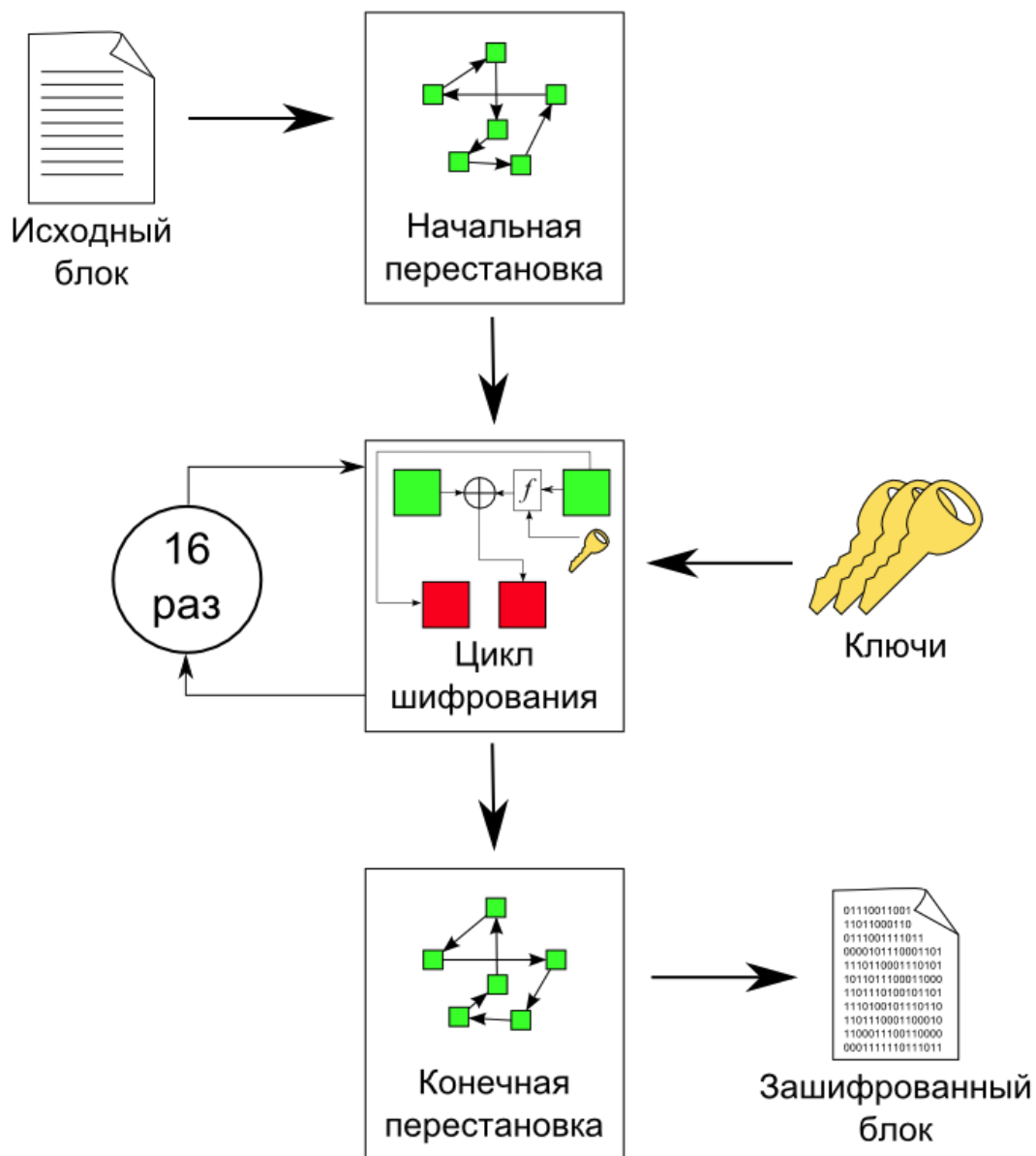


Рисунок 1.1. Общая схема работы алгоритма

Рассмотрим цикл шифрования подробнее. На каждой итерации используется сеть Фейстейля (рисунок 1.2 и 1.3), вместе с которой происходит наложение каждого из промежуточного ключей.

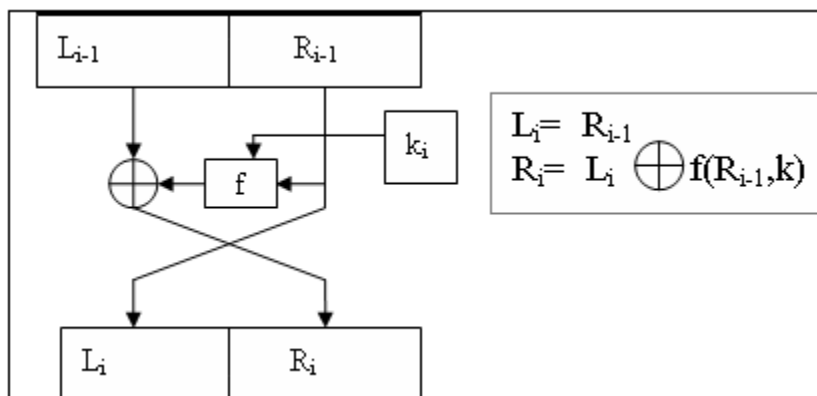


Рисунок 1.2. Прямое шифрование сетью Фейстейля

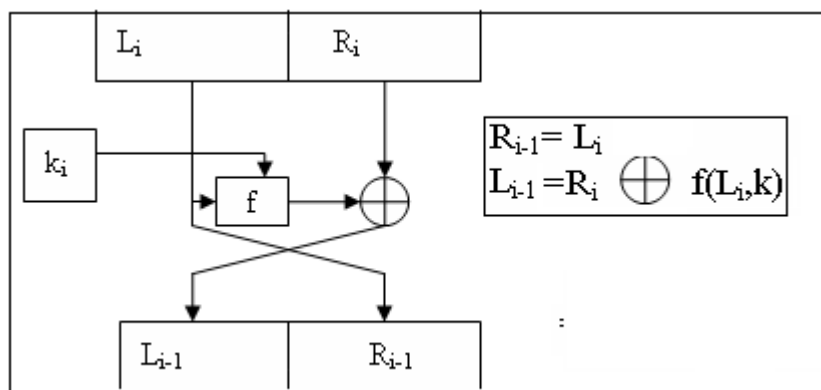


Рисунок 1.3. Обратное шифрование сетью Фейстейля

Аргументами функции f являются 32-битовый вектор R_{i-1} и 48-битовый ключ k_i , который является результатом преобразования 56-битового исходного ключа шифра k .

Для вычисления функции последовательно используются:

1. функция расширения E
2. сложение по модулю 2 с ключом k_i
3. преобразование S , состоящее из 8 преобразований (перестановок) S -блоков

Более подробный алгоритм будет выглядеть следующим образом (рисунок 1.5)

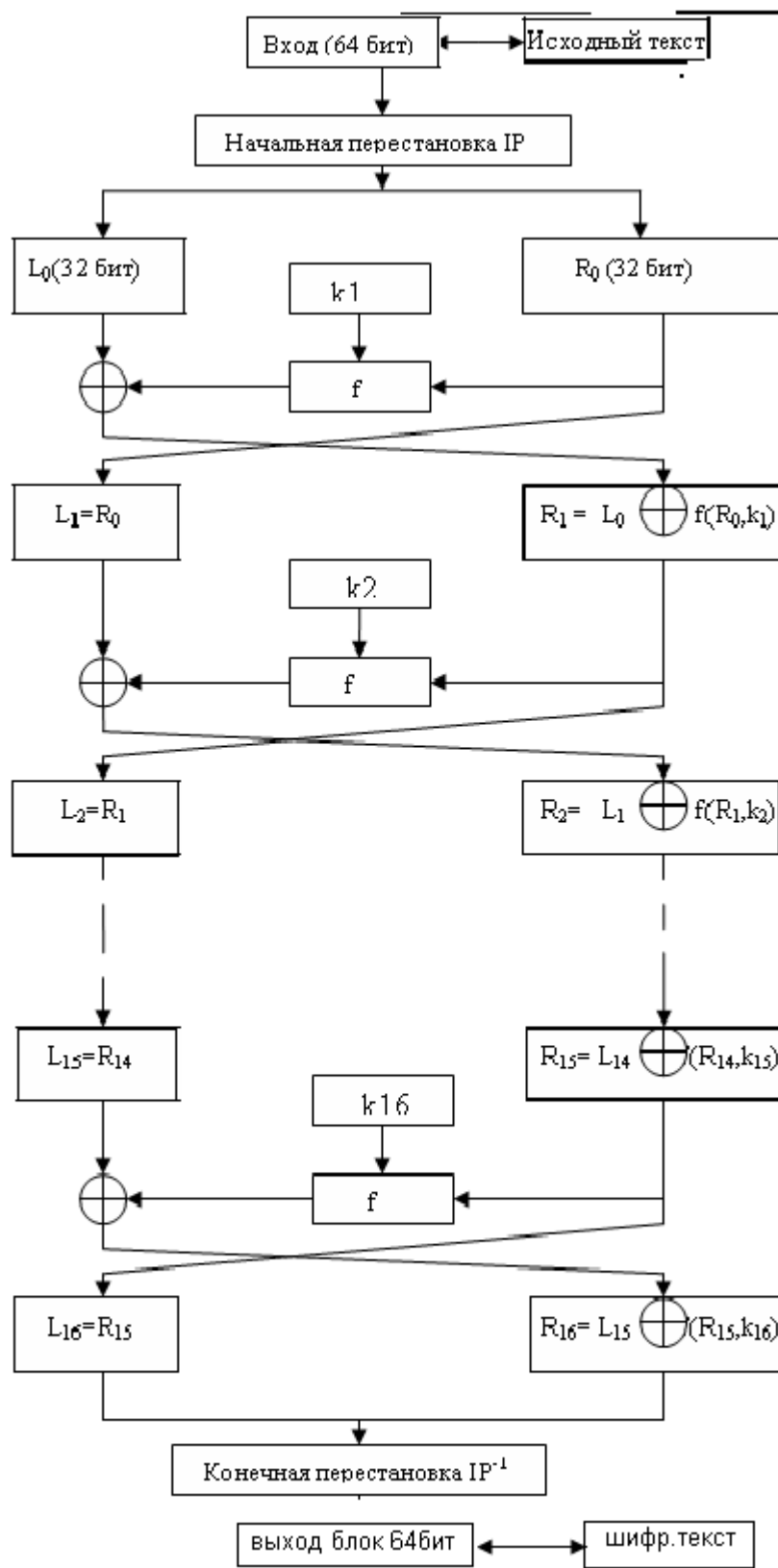


Рисунок 1.4. Полный алгоритм работы DES

Организуем структуру программы следующим образом (рисунок 2.1). Текстовые файлы с ключём и исходным текстом мы положим в папку “resources”. Шифрованное и дешифрованное сообщения также будем сохранять в “resources”. Логика работы алгоритма будет реализована в .h-файлах, которые будут находится в “lib”.

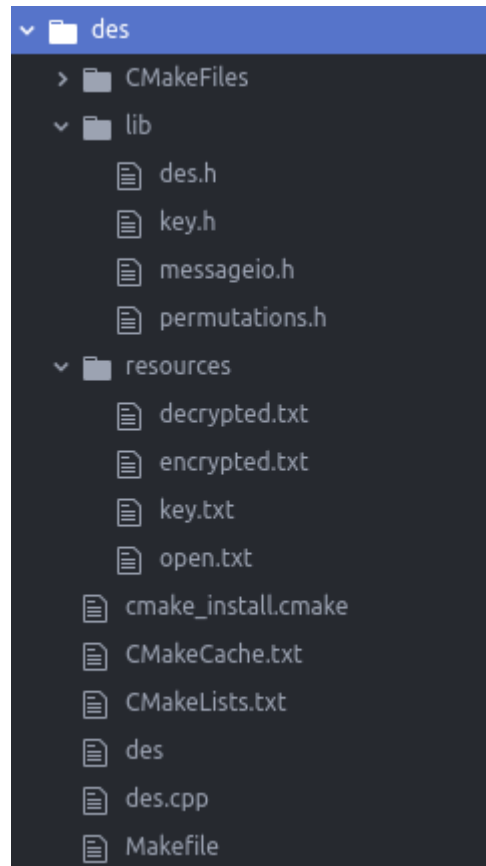


Рисунок 2. Файловая структура

В “messageio.h” мы определим класс-контейнер “Message”, который будет конструируется из имени файла, который содержит текст для шифрования либо расшифровки. Его задачей будет считывание и преобразование текста с файла в массив байт, соответствующий типу в C++uint8_t, а также запись результата в файл. Так же в классе определим метод для строкового представления считанных данных.

Так же создадим класс “DES” (des.h), который конструируется из класса “Message” (паттерн композиция). Его задачей будет обеспечение дополнительного уровня абстракции для манипуляции с байтами сообщения и реализации методов шифрования и расшифрования с помощью сети Фейстейля. Полученные таким путём шифротекст или расшифровка будут интерпретированы как Message-контейнеры. После записаны в текстовые файлы.

Код из файлов key.h и permutations.h связан между собой. “Permutations.h” содержит в себе статические данные и предвычисленные перестановки для

промежуточных ключей, а так же низкоуровневые операции для работы с битами. Такие как их перестановка, выделение левой/правой части числа, крайних бит и т. д. В свою очередь key.h использует функции и константные данные из предыдущего файла для генерации промежуточных ключей и операции над 32ти-, 48ми-, 56ти- и 64х- битными числами.

Общий вид вызываемой части программы реализуем следующим образом.

Листинг 1. des.cpp.

```
#include "lib/des.h"

#define OPEN_RESOURCE "resources/open.txt"
#define DECR_RESOURCE "resources/decrypted.txt"
#define ENCR_RESOURCE "resources/encrypted.txt"
#define KEY_RESOURCE "resources/key.txt"

int main(void) {

    Message OpenMsg = Message(OPEN_RESOURCE);
    cout << OpenMsg.repr_string() << endl;
    DES Cipher = DES(OpenMsg, KEY_RESOURCE);

    Message EncryptedMsg = Cipher.encrypt();
    cout << "Encrypted: " << endl;
    EncryptedMsg.print_array();
    EncryptedMsg.write_txt(ENCR_RESOURCE);

    Message DecryptedMsg = Cipher.decrypt();
    cout << "Decrypted: " << endl;
    DecryptedMsg.print_array();
    DecryptedMsg.write_txt(DECR_RESOURCE);

    cout << "As string: " << DecryptedMsg.repr_string() << endl;
    return 0;
}
```

Рассмотрим содержимое заголовочных файлов подробнее (листинги 1.2, 1.3, 1.4, 1.5).

Листинг 2. messageio.h

```
#ifndef MESSAGGEIO_H
#define MESSAGGEIO_H

#include "iostream"
#include "fstream"
#include "stdint.h"
#include "string.h"
#include "stddef.h"

#define BUFF_SIZE 1024
#define FILE_ERROR "Non-existent file"

using std::ifstream;
using std::ofstream;
using std::getline;
using std::cout;
using std::endl;
using std::cerr;
using std::size_t;
using std::string;

class Message {
public:
    Message(const char * file_path);
    ~Message() = default;
    static string read_txt(const char * file_path);
    void write_txt(const char * file_path);
    string repr_string();
    void print_array();

    uint8_t * data;
    size_t data_len;
};

Message :: Message(const char * file_path){
    string buffer = read_txt(file_path);
    this->data = (uint8_t*)buffer.c_str();
    this->data_len = buffer.length();
}
```

```

void Message :: write_txt(const char * file_path){
    ofstream file_stream(file_path);
    if(file_stream.is_open()) file_stream << this->data;
    else {
        cerr << FILE_ERROR << endl;
        exit(1);
    }
    file_stream.close();
}

string Message :: read_txt(const char * file_path) {
    ifstream file_stream(file_path);
    if(file_stream.is_open()) {
        file_stream.seekg(0, file_stream.end);
        size_t actual_len = file_stream.tellg();
        file_stream.seekg(0, file_stream.beg);
        size_t aligned_len = actual_len % 8 == 0 ? actual_len :
actual_len + (8 - (actual_len % 8));
        auto * buffer = new char [aligned_len];
        file_stream.read(buffer, actual_len);
        file_stream.close();
        return string(buffer);
    }
    else {
        cerr << FILE_ERROR << endl;
        exit(1);
    }
}

void Message :: print_array() {
    cout << "[ ";
    for (size_t i = 0; i < this->data_len; ++i)
        cout << this->data[i];
    cout << " ]" << endl;
}

string Message :: repr_string(){
    return string((char*)this->data);
}
#endif

```

Листинг 3. permutations.h

```
#ifndef PERMUTATIONS_H
#define PERMUTATIONS_H

#include <stdint.h>
#include <stddef.h>

using std::size_t;

uint32_t substitutions(uint64_t block48b);
void substitution_6bits_to_4bits(uint8_t * blocks6b,
uint8_t * blocks4b);
uint8_t extreme_bits(uint8_t block6b);
uint8_t middle_bits(uint8_t block6b);

uint32_t permutation(uint32_t block32b);
uint64_t expansion_permutation(uint32_t block32b);
uint64_t initial_permutation(uint64_t block64b);
uint64_t final_permutation(uint64_t block64b);

void split_64bits_to_32bits(uint64_t block64b, uint32_t *
block32b_1, uint32_t * block32b_2);
void split_64bits_to_8bits(uint64_t block64b, uint8_t *
blocks8b);
void split_48bits_to_6bits(uint64_t block48b, uint8_t *
blocks6b);

uint64_t join_32bits_to_64bits(uint32_t block32b_1,
uint32_t block32b_2);
uint64_t join_28bits_to_56bits(uint32_t block28b_1,
uint32_t block28b_2);
uint64_t join_8bits_to_64bits(uint8_t * blocks8b);
uint32_t join_4bits_to_32bits(uint8_t * blocks8b);

const uint8_t __Sbox[8][4][16] = {
    { // 0
        {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}},
    },
    { // 1
        {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 },
    },
    { // 2
        {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}},
    },
    { // 3
        {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}},
    },
    { // 4
        {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 },
    },
    { // 5
        {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},
    },
    { // 6
        {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},
    },
    { // 7
        {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},
    },
};

```

```

const uint8_t __IP[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,  
};
```

```
const uint8_t __FP[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,  
};
```

```
const uint8_t __K1P[28] = {  
    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,  
};
```

```
const uint8_t __K2P[28] = {  
    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,  
};
```

```
const uint8_t __CP[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,  
};
```

```
const uint8_t __EP[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,  
};
```

```
const uint8_t __P[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,
};

```

```

uint64_t expansion_permutation(uint32_t block32b) {
    uint64_t block48b = 0;
    for (uint8_t i = 0 ; i < 48; ++i) {
        block48b |= (uint64_t)((block32b >> (32 - __EP[i]))
& 0x01) << (63 - i);
    }
    return block48b;
}

```

```

uint32_t substitutions(uint64_t block48b) {
    uint8_t blocks4b[4], blocks6b[8] = {0};
    split_48bits_to_6bits(block48b, blocks6b);
    substitution_6bits_to_4bits(blocks6b, blocks4b);
    return join_4bits_to_32bits(blocks4b);
}

```

```

void substitution_6bits_to_4bits(uint8_t * blocks6b,
uint8_t * blocks4b) {
    uint8_t block2b, block4b;

    for (uint8_t i = 0, j = 0; i < 8; i += 2, ++j) {
        block2b = extreme_bits(blocks6b[i]);
        block4b = middle_bits(blocks6b[i]);
        blocks4b[j] = __Sbox[i][block2b][block4b];

        block2b = extreme_bits(blocks6b[i+1]);
        block4b = middle_bits(blocks6b[i+1]);
        blocks4b[j] = (blocks4b[j] << 4) | __Sbox[i+1]
[block2b][block4b];
    }
}

```

```

uint32_t permutation(uint32_t block32b) {

```

```

    uint32_t new_block32b = 0;
    for (uint8_t i = 0 ; i < 32; ++i) {
        new_block32b |= ((block32b >> (32 - __P[i])) &
0x01) << (31 - i);
    }
    return new_block32b;
}

uint64_t initial_permutation(uint64_t block64b) {
    uint64_t new_block64b = 0;
    for (uint8_t i = 0 ; i < 64; ++i) {
        new_block64b |= ((block64b >> (64 - __IP[i])) &
0x01) << (63 - i);
    }
    return new_block64b;
}

uint64_t final_permutation(uint64_t block64b) {
    uint64_t new_block64b = 0;
    for (uint8_t i = 0 ; i < 64; ++i) {
        new_block64b |= ((block64b >> (64 - __FP[i])) &
0x01) << (63 - i);
    }
    return new_block64b;
}

uint8_t extreme_bits(uint8_t block6b) {
    return ((block6b >> 6) & 0x2) | ((block6b >> 2) & 0x1);
}

uint8_t middle_bits(uint8_t block6b) {
    return (block6b >> 3) & 0xF;
}

void split_64bits_to_32bits(uint64_t block64b, uint32_t *
block32b_1, uint32_t * block32b_2) {
    *block32b_1 = (uint32_t)(block64b >> 32);
    *block32b_2 = (uint32_t)(block64b);
}

```

```

void split_64bits_to_8bits(uint64_t block64b, uint8_t *
blocks8b) {
    for (size_t i = 0; i < 8; ++i) {
        blocks8b[i] = (uint8_t)(block64b >> ((7 - i) * 8));
    }
}

```

```

void split_48bits_to_6bits(uint64_t block48b, uint8_t *
blocks6b) {
    for (uint8_t i = 0; i < 8; ++i) {
        blocks6b[i] = (block48b >> (58 - (i * 6))) << 2;
    }
}

```

```

uint64_t join_32bits_to_64bits(uint32_t block32b_1,
uint32_t block32b_2) {
    uint64_t block64b;
    block64b = (uint64_t)block32b_1;
    block64b = (uint64_t)(block64b << 32) | block32b_2;
    return block64b;
}

```

```

uint64_t join_28bits_to_56bits(uint32_t block28b_1,
uint32_t block28b_2) {
    uint64_t block56b;
    block56b = (block28b_1 >> 4);
    block56b = ((block56b << 32) | block28b_2) << 4;
    return block56b;
}

```

```

uint64_t join_8bits_to_64bits(uint8_t * blocks8b) {
    uint64_t block64b;
    for (uint8_t *p = blocks8b; p < blocks8b + 8; ++p) {
        block64b = (block64b << 8) | *p;
    }
    return block64b;
}

```

```

uint32_t join_4bits_to_32bits(uint8_t * blocks4b) {
    uint32_t block32b;

```

```

        for (uint8_t *p = blocks4b; p < blocks4b + 4; ++p) {
            block32b = (block32b << 8) | *p;
        }
        return block32b;
    }

```

#endif

Листинг 4. keys.h

#ifndef KEY_H

#define KEY_H

#include <stdint.h>

#include "permutations.h"

#define LSHIFT_28BIT(x, L) (((x) << (L)) | ((x) >> (-(L) & 27))) & (((uint64_t)1 << 32) - 1))

void key_expansion(uint64_t key64b, uint64_t * keys48b);

void key_permutation_56bits_to_28bits(uint64_t block56b, uint32_t * block28b_1, uint32_t * block28b_2);

void key_expansion_to_48bits(uint32_t block28b_1, uint32_t block28b_2, uint64_t * keys48b);

uint64_t key_contraction_permutation(uint64_t block56b);

void key_expansion(uint64_t key64b, uint64_t * keys48b) {

uint32_t K1 = 0, K2 = 0;

key_permutation_56bits_to_28bits(key64b, &K1, &K2);

key_expansion_to_48bits(K1, K2, keys48b);

}

void key_permutation_56bits_to_28bits(uint64_t block56b, uint32_t * block28b_1, uint32_t * block28b_2) {

for (uint8_t i = 0; i < 28; ++i) {

*block28b_1 |= ((block56b >> (64 - __K1P[i])) & 0x01) << (31 - i);

*block28b_2 |= ((block56b >> (64 - __K2P[i])) & 0x01) << (31 - i);

}

}

```

void key_expansion_to_48bits(uint32_t block28b_1, uint32_t
block28b_2, uint64_t * keys48b) {
    uint64_t block56b;
    uint8_t n;

    for (uint8_t i = 0; i < 16; ++i) {
        switch(i) {
            case 0: case 1: case 8: case 15: n = 1; break;
            default: n = 2; break;
        }

        block28b_1 = LSHIFT_28BIT(block28b_1, n);
        block28b_2 = LSHIFT_28BIT(block28b_2, n);
        block56b = join_28bits_to_56bits(block28b_1,
block28b_2);
        keys48b[i] = key_contraction_permutation(block56b);
    }
}

uint64_t key_contraction_permutation(uint64_t block56b) {
    uint64_t block48b = 0;
    for (uint8_t i = 0 ; i < 48; ++i) {
        block48b |= ((block56b >> (64 - __CP[i])) & 0x01)
<< (63 - i);
    }
    return block48b;
}

#endif

```

Листинг 5. des.h

```

#include "messageio.h"
#include "key.h"

```

```

class DES{
public:
    DES(Message message, const char * key_file);
    ~DES() = default;

```



```
Message encrypt();
Message decrypt();
```

```
private:
```

```
    static void feistel_decrypt(uint32_t * N1, uint32_t *
N2, uint64_t * keys48b);
    static void feistel_encrypt(uint32_t * N1, uint32_t *
N2, uint64_t * keys48b);
    static void round_feistel_cipher(uint32_t * N1,
uint32_t * N2, uint64_t key48b);
    static void feistel_cipher(uint32_t * N1, uint32_t *
N2, uint64_t * keys48b);
    static void swap(uint32_t * N1, uint32_t * N2);
    static void feistel_round(uint32_t * N1, uint32_t * N2,
uint64_t key48b);
    static uint32_t magic_fn(uint32_t block32b, uint64_t
key48b);

    uint8_t * key;
    uint8_t * message;
    size_t cipher_len;

    uint64_t keys48b[16] = {0};
    uint32_t N1, N2;
};
```

```
DES :: DES(Message msg, char const * key_file){
    this->key = (uint8_t*)msg.read_txt(key_file).c_str();
    this->message = msg.data;
    this->cipher_len = msg.data_len;

    key_expansion(
        join_8bits_to_64bits(this->key),
        this->keys48b
    );
}
```

```
Message DES :: encrypt(){
    for (size_t i = 0; i < this->cipher_len; i += 8) {
        split_64bits_to_32bits(
```

```

        initial_permutation(
            join_8bits_to_64bits(this->message + i)
        ),
        &this->N1, &this->N2
    );
    feistel_encrypt(&this->N1, &this->N2, this->keys48b);
    split_64bits_to_8bits(
        final_permutation(
            join_32bits_to_64bits(this->N1, this->N2)
        ),
        (this->message + i)
    );
}
}

```

```

Message DES :: decrypt(){
    for (size_t i = 0; i < this->cipher_len; i += 8) {
        split_64bits_to_32bits(
            initial_permutation(
                join_8bits_to_64bits(this->message + i)
            ),
            &this->N1, &this->N2
        );
        feistel_decrypt(&this->N1, &this->N2, this->keys48b);
        split_64bits_to_8bits(
            final_permutation(
                join_32bits_to_64bits(this->N1, this->N2)
            ),
            (this->message + i)
        );
    }
}

```

```

void DES :: feistel_encrypt(uint32_t * N1, uint32_t * N2,
uint64_t * keys48b) {
    for (int8_t round = 0; round < 16; ++round) {
        round_feistel_cipher(N1, N2, keys48b[round]);
    }
    swap(N1, N2);
}

```

```

void DES :: feistel_decrypt(uint32_t * N1, uint32_t * N2,
uint64_t * keys48b) {
    for (int8_t round = 15; round >= 0; --round) {
        round_feistel_cipher(N1, N2, keys48b[round]);
    }
    swap(N1, N2);
}

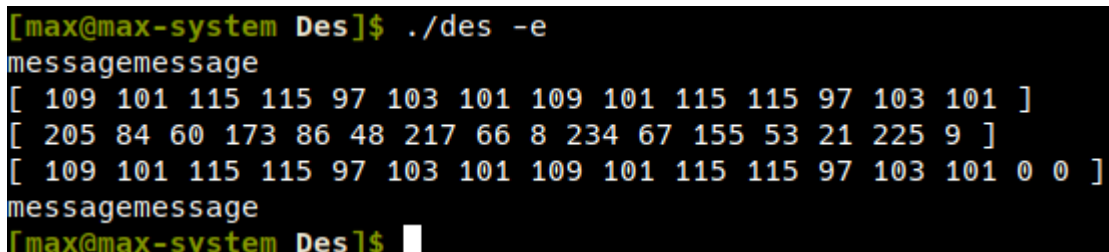
uint32_t DES :: magic_fn(uint32_t block32b, uint64_t
key48b) {
    uint64_t block48b = expansion_permutation(block32b);
    block48b ^= key48b;
    block32b = substitutions(block48b);
    return permutation(block32b);
}

void DES :: feistel_round(uint32_t * N1, uint32_t * N2,
uint64_t key48b) {
    uint32_t temp = *N2;
    *N2 = magic_fn(*N2, key48b) ^ *N1;
    *N1 = temp;
}

void DES :: swap(uint32_t * N1, uint32_t * N2) {
    uint32_t temp = *N1;
    *N1 = *N2;
    *N2 = temp;
}

```

Скомпилируем и выполним программу (рисунок 2).



```

[max@max-system Des]$ ./des -e
messagemessage
[ 109 101 115 115 97 103 101 109 101 115 115 97 103 101 ]
[ 205 84 60 173 86 48 217 66 8 234 67 155 53 21 225 9 ]
[ 109 101 115 115 97 103 101 109 101 115 115 97 103 101 0 0 ]
messagemessage
[max@max-system Des]$

```

Рисунок 2. Результат выполнения

Вывод. В лабораторной работе рассмотрен и релизован на языке программирования C++ алгоритм шифрования DES. Он является одним из самых первых и успешных алгоритмов шифрования, модификации которого применяются и сегодня (3DES). Однако одним из потенциально слабых мест алгоритма является использование короткого 64х битного ключа.