Министерство образования Республики Беларусь

Учреждение образования
БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
ИНФОРМАТИКИ И РАДИОЭЛЕКТРОНИКИ

Факультет компьютерных систем и сетей

Кафедра электронных вычислительных машин

Учебная практика

Программа для шифрования\дешифрования методом DES

Выполнил студент гр. 150502: Снитко Д.А

Проверил: Луцик Ю.А

Минск 2022

Содержание

1 Введение

Алгоритмы шифрования и дешифрования данных широко применяются в компьютерной технике в системах сокрытия конфиденциальной и коммерческой информации от злонамеренного использования сторонними лицами. Главным принципом в них является условие, что передатчик и приемник заранее знают алгоритм шифрования, а также ключ к сообщению, без которых информация представляет собой всего лишь набор символов, не имеющих смысла.

Симметричные криптосистемы (также симметричное шифрование, симметричные шифры) -- способ шифрования, в котором для (за)шифрования и расшифровывания применяется один и тот же криптографический ключ. До изобретения схемы асимметричного шифрования единственным существовавшим способом являлось симметричное шифрование. Ключ алгоритма должен сохраняться в секрете обеими сторонами. Ключ алгоритма выбирается сторонами до начала обмена сообщениями.

Полная утрата всех статистических закономерностей исходного сообщения является важным требованием к симметричному шифру. Для этого шифр должен иметь эффект лавины - должно происходить сильное изменение шифроблока при 1битном изменении входных данных.

Симметричные шифры: блочные, поточные.

Блочные шифры обрабатывают информацию блоками определённой длины (обычно 64, 128 бит), применяя к блоку ключ в установленном порядке, как правило, несколькими циклами перемешивания и подстановки, называемыми раундами. Результатом повторения раундов является лавинный эффект - нарастающая потеря соответствия битов между блоками открытых и зашифрованных данных.

Существенные параметры симметричных шифров: стойкость, длина ключа, число раундов, длина обрабатываемого блока, сложность аппаратной/программной реализации, сложность преобразования.

2 Алгоритм шифрования DES

Стандарт шифрования данных DES опубликован в 1977 г. Национальным бюро стандартом США. Он предназначен для защиты от несанкционированного доступа к важной, но несекретной информации. В 1980 г. был одобрен Национальным институтом стандартов и технологий США.

В настоящий момент DES является наиболее распространенным алгоритмом, используемым в системах защиты коммерческой информации.

Преимущества DES: DES использует алгоритм с симметричным ключом, поэтому шифрование и дешифрование могут выполняться одним ключом с использованием одного и того же алгоритма. DES был разработан для аппаратных средств, а не для программного обеспечения, и демонстрирует эффективность и быстрое внедрение в аппаратные средства.

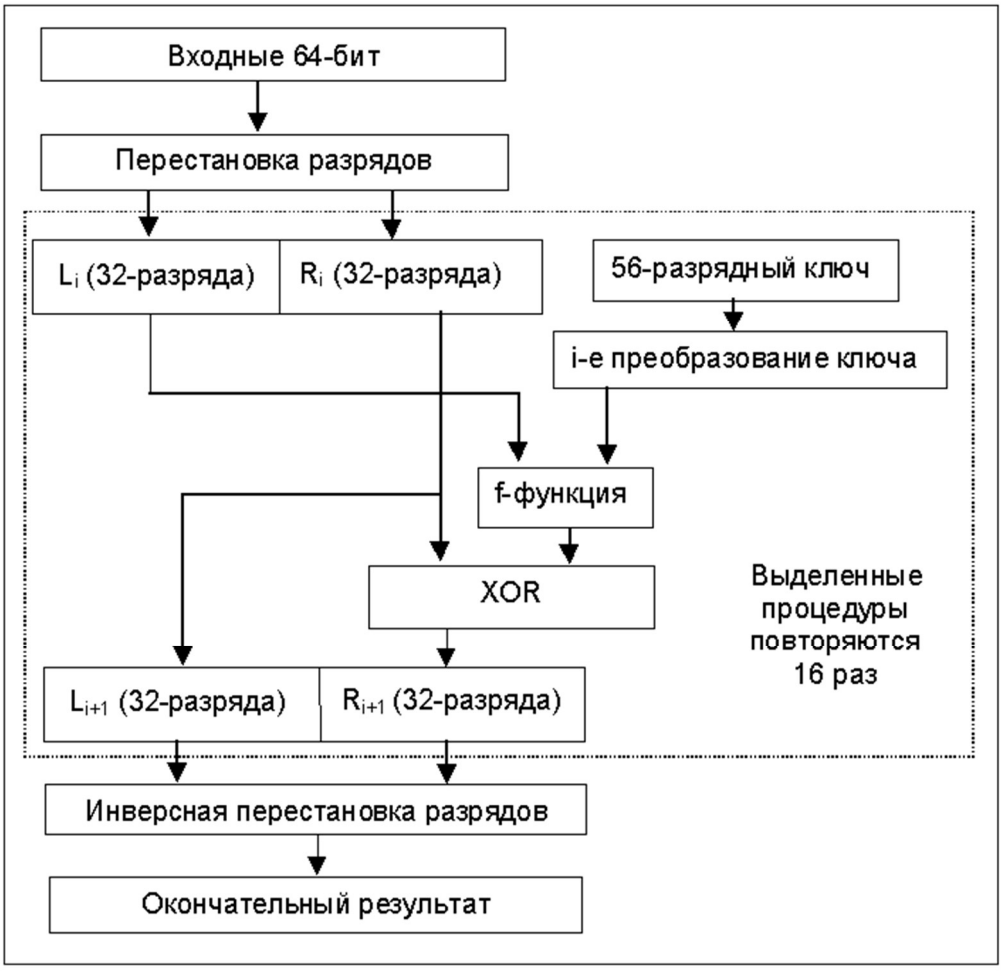
Недостатки DES: DES обеспечивает низкий уровень безопасности с точки зрения 56-битного ключа, поскольку брут форс атака может легко взломать его.

3 Описание алгоритма шифрования DES

Алгоритм DES использует комбинацию подстановок и перестановок. DES осуществляет шифрование 64-битных блоков данных с помощью 64-битового ключа, в котором значащими являются 56 бит. Дешифрование в DES является операцией, обратной шифрованию, и выполняется путем повторения операций шифрования в обратной последовательности.

3.1 Схема алгоритма

1. На первом этапе 64-битный блок открытого текста передается функции начальной перестановки (IP).
2. Начальная перестановка выполняется на обычном тексте.
3. Затем начальная перестановка (IP) создает две половины переставляемого блока; левый обычный текст (LPT) и правый обычный текст (RPT).
4. Теперь каждый LPT и RPT проходят 16 раундов процесса шифрования.
5. В конце LPT и RPT воссоединяются, и в объединенном блоке выполняется окончательная перестановка (FP).
6. Результатом этого процесса является 64-битный зашифрованный текст.



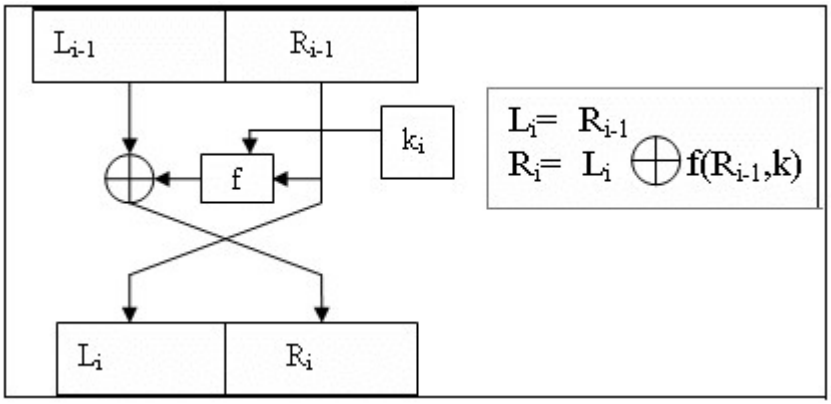
3.2 Начальная перестановка (IP)

Начальная перестановка (IP) происходит только один раз и происходит перед первым раундом. IP заменяет первый бит исходного блока открытого текста 58-м битом исходного открытого текста, второй бит 50-м битом исходного блока открытого текста и так далее исходя из таблицы IP (initial-perm).
(Все таблицы используемые в реализации являются рекомендуемыми).

3.3 Преобразование ключа

Для получения ключа размером 56 бит, каждый 8 бит отбрасывается исходя из таблицы (keyp).

3.4 Сеть Фейстеля



Сеть Фейстеля - это криптографический метод, используемый при построении алгоритмов и механизмов на основе блочного шифра.

3.4.1 Разделение начальной строки (PT)

После выполнения начальной перестановки 64-битный переставленный блок делится на два полублока по 32 бит каждый.

3.4.2 Перестановка ключа со сжатием

В каждом из 16 раундов 56-битный ключ преобразуется в 48-битный подключ. Для этого 56-битный ключ делится на две половины по 28 бит каждая. Эти половинки циклически сдвигаются влево на одну или две позиции в зависимости от таблицы сдвигов (shift_table) и раунда.

Процесс преобразования ключа включает перестановку, а также выбор 48-битного подмножества исходного 56-битного ключа.

После соответствующего сдвига выбираются 48 из 56 битов. Для выбора 48 из 56 бит используется таблица компрессии ключа (key-comp). Исходя из данных таблицы ниже, бит номер 14 перемещается на первую позицию, номер 17 на вторую и т.д. Для сведения (компрессии) 56-битного ключа к 48-битному, каждый восьмой бит отбрасывается, поэтому в таблице отсутствуют номера 8 бит.

```
int key_comp[48] = { [0]: 14,  [1]: 17,  [2]: 11,  [3]: 24,  [4]: 1,  [5]: 5,
                     [6]: 3,  [7]: 28,  [8]: 15,  [9]: 6,  [10]: 21,  [11]: 10,
                     [12]: 23,  [13]: 19,  [14]: 12,  [15]: 4,  [16]: 26,  [17]: 8,
                     [18]: 16,  [19]: 7,  [20]: 27,  [21]: 20,  [22]: 13,  [23]: 2,
                     [24]: 41,  [25]: 52,  [26]: 31,  [27]: 37,  [28]: 47,  [29]: 55,
                     [30]: 30,  [31]: 40,  [32]: 51,  [33]: 45,  [34]: 33,  [35]: 48,
                     [36]: 44,  [37]: 49,  [38]: 39,  [39]: 56,  [40]: 34,  [41]: 53,
                     [42]: 46,  [43]: 42,  [44]: 50,  [45]: 36,  [46]: 29,  [47]: 32 };
```

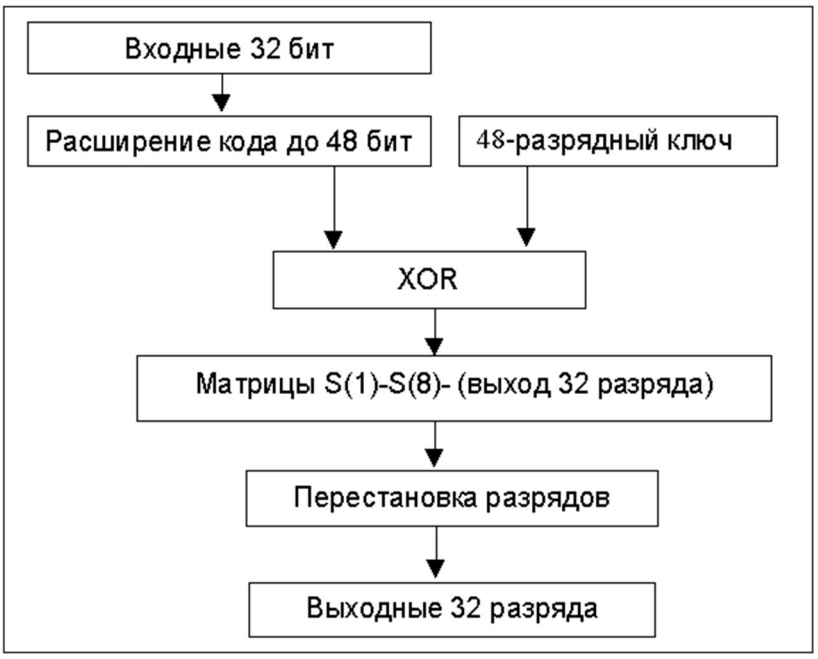
Такой метод компрессионной перестановки в каждом раунде используя разное подмножество битов ключа усложняет взлом шифрования.

3.4.3 Расширение правого обычного текста (RPT) с перестановкой

Во время перестановки с расширением RPT расширяется с 32 бит до 48 бит для дальнейшего преобразования с ключом той же длины. Биты также переставляются, поэтому это называется перестановкой с расширением.

Это происходит потому, что 32-битный RPT разделен на 8 блоков, каждый из которых состоит из 4 бит. Затем каждый 4-битный блок предыдущего шага затем расширяется до соответствующего 6-битного блока путем добавления к каждому 4-битному блоку еще 2 бит путем копирования соседнего бита.

3.4.4.1 f-функция



3.4.4.2 Перестановка с помощью S-box`ов:

Исходный 48-разрядный код делится на 8 групп по 6 разрядов. Первый и последний разряд в группе используется в качестве адреса строки, а средние 4 разряда - в качестве адреса столбца. В результате каждые 6 бит кода преобразуются в 4 бита, а весь 48-разрядный код в 32-разрядный.

3.4.4.3 Перестановка полученного кода с помощью P-box`ов:

Биты полученного блока переставляются исходя из таблицы (per).

3.5 Конечная перестановка (FP)

Конечная перестановка (Final Permutation) происходит только один раз после последнего (16) раунда.

Блок схема алгоритма в **Приложении А**. Исходный код - в **Приложении Б**.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

[1] Fabio Martignon; "System Securities - DES"
lri.fr/fmartignon/documenti/systemesecurite/4-DES.pdf

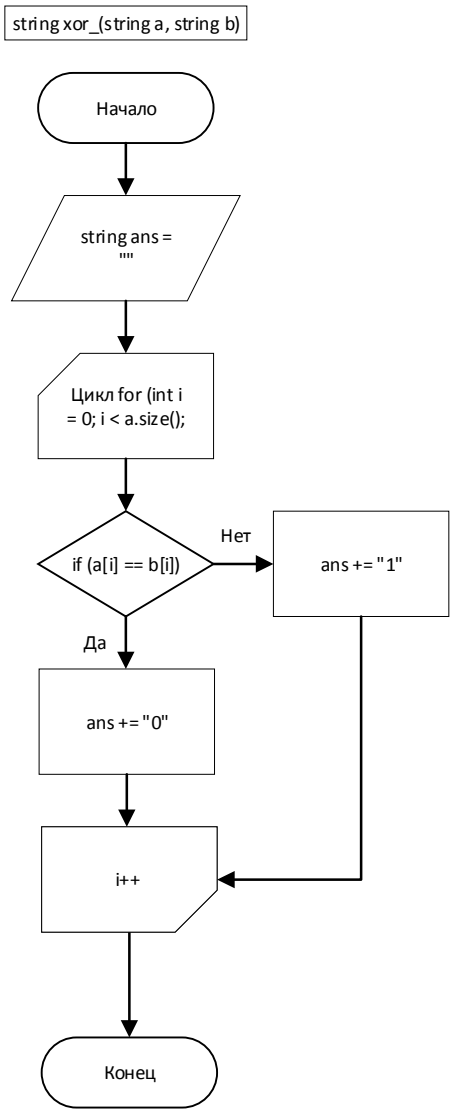
[2] Cleveland State University “Data Encryption Standard (DES)”
academic.csuohio.edu/yuc/security/Chapter_06_Data_Encription_Standard.pdf

[3] Семенов Ю.А. "Алгоритм DES"
opennet.ru/docs/RUS/inet_book/6/des-641.html

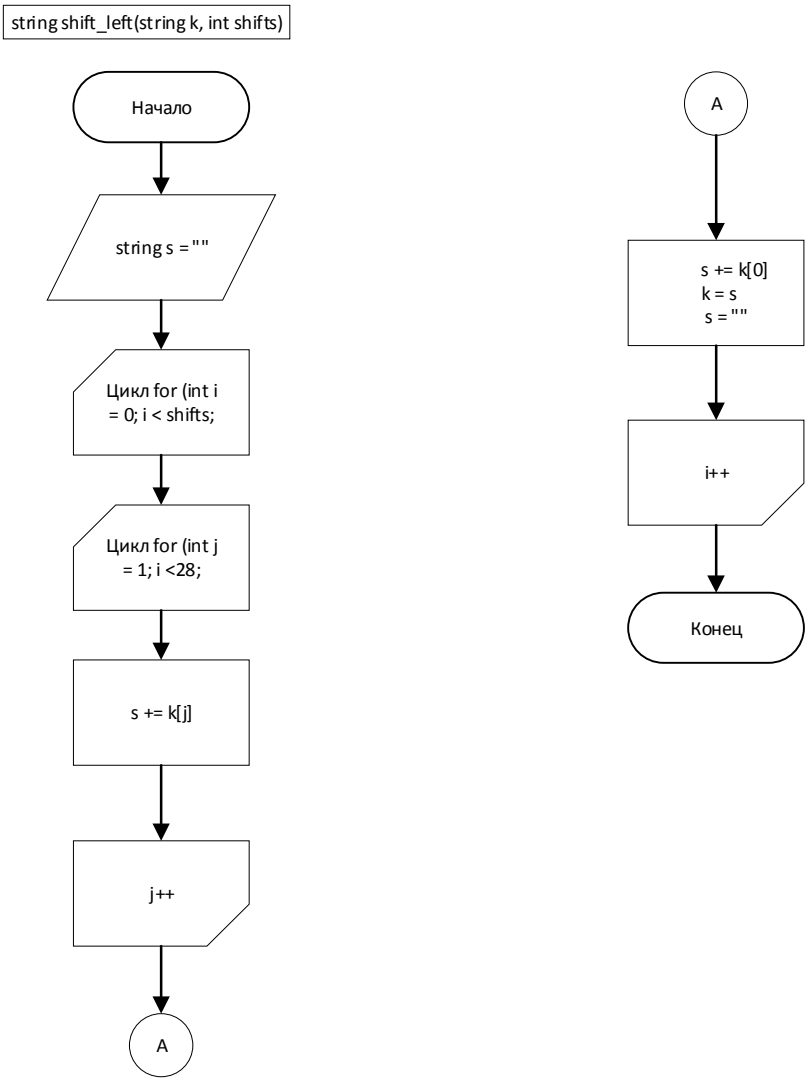
ПРИЛОЖЕНИЕ А
(обязательное)

Блок-схемы функций

Операция сложения по модулю два



Функция сдвига строки влево



ПРИЛОЖЕНИЕ Б
(обязательное)

Исходный код программы

Модуль *untitled.cpp*:

```
#include <iostream>
#include <bits/stdc++.h> // заголовочный файл, включающий все стандартные библиотеки
using namespace std;

string hex2bin(string s) // шестнадцатеричное преобразование в двоичное
{

    unordered_map<char, string> mp; // неупоряд словарь не хранит
    mp['0'] = "0000";               // элементы в отсортированном порядке
    mp['1'] = "0001";
    mp['2'] = "0010";
    mp['3'] = "0011";
    mp['4'] = "0100";
    mp['5'] = "0101";
    mp['6'] = "0110";
    mp['7'] = "0111";
    mp['8'] = "1000";
    mp['9'] = "1001";
    mp['A'] = "1010";
    mp['B'] = "1011";
    mp['C'] = "1100";
    mp['D'] = "1101";
    mp['E'] = "1110";
    mp['F'] = "1111";
    string bin = "";
    for (int i = 0; i < s.size(); i++) {
        bin += mp[s[i]];
    }
    return bin;
}
string bin2hex(string s) // двоичное преобразование в шестнадцатеричное
{
    // преобразование двоичного кода в шестнадцатеричный
    unordered_map<string, string> mp;
    mp["0000"] = "0";
    mp["0001"] = "1";
    mp["0010"] = "2";
    mp["0011"] = "3";
    mp["0100"] = "4";
    mp["0101"] = "5";
    mp["0110"] = "6";
    mp["0111"] = "7";
    mp["1000"] = "8";
```

```
    mp["1001"] = "9";
    mp["1010"] = "A";
    mp["1011"] = "B";
    mp["1100"] = "C";
    mp["1101"] = "D";
    mp["1110"] = "E";
    mp["1111"] = "F";
    string hex = "";
    for (int i = 0; i < s.length(); i += 4) {
        string ch = "";
        ch += s[i];
        ch += s[i + 1];
        ch += s[i + 2];
        ch += s[i + 3];
        hex += mp[ch];
    }
    return hex;
}

string permute(string k, int* arr, int n) // перестановка
{
    string per = "";
    for (int i = 0; i < n; i++) {
        per += k[arr[i] - 1];
    }
    return per;
}

string shift_left(string k, int shifts)
{
    string s = "";
    for (int i = 0; i < shifts; i++) {
        for (int j = 1; j < 28; j++) {
            s += k[j];
        }
        s += k[0];
        k = s;
        s = "";
    }
    return k;
}

string xor_(string a, string b)
{
    string ans = "";
    for (int i = 0; i < a.size(); i++) {
        if (a[i] == b[i]) {
            ans += "0";
        }
        else {
            ans += "1";
        }
    }
    return ans;
}
```

```
string encrypt(string pt, vector<string> rkb, vector<string> rk)
{
   // Шестнадцатеричный код в двоичный
   pt = hex2bin(pt);

   // Начальная таблица перестановок (IP) без ключа
   int initial_perm[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 };
   // Начальная перестановка (IP) без ключа
   pt = permute(pt, initial_perm, 64);
   cout << "After initial permutation: " << bin2hex(pt) << endl;

   // Разделение начальной строки (PT)
   string left = pt.substr(0, 32); // substr возвращает часть строки
   string right = pt.substr(32, 32);
   cout << "After splitting: L0=" << bin2hex(left)
       << " R0=" << bin2hex(right) << endl;

   // Таблица D-box для расширения правой части из 32->48 бит
   int exp_d[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 };

   // Таблица 8 S-Box`ов (substition boxes) для замены (функция f)
   int s[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 } };

    // Прямая таблица перестановок P-box для кода после S-box`ов (функция f)
    int per[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 };

    cout << endl;
    for (int i = 0; i < 16; i++) {

        // Расширение RPT с перестановкой по D-блоку (функция f)
        string right_expanded = permute(right, exp_d, 48);

        // XOR RoundKey[i] и right_expanded (функция f)
        string x = xor_(rkb[i], right_expanded);

        // S-boxes  (функция f)
        string op = "";
        for (int i = 0; i < 8; i++) {
            int row = 2 * int(x[i * 6] - '0') + int(x[i * 6 + 5] - '0');
            int col = 8 * int(x[i * 6 + 1] - '0') + 4 * int(x[i * 6 + 2] - '0') + 2 * int(x[i * 6 + 3] - '0') + int(x[i * 6 + 4] -
'0');
            int val = s[i][row][col]; // S-box
            op += char(val / 8 + '0');
            val = val % 8;
            op += char(val / 4 + '0');
            val = val % 4;
            op += char(val / 2 + '0');
            val = val % 2;
            op += char(val + '0');
        }
        // Перестановка полученного кода через S-box`ы с помощью P-box (функция f)
        op = permute(op, per, 32);

        // XOR L и op (op - R после преобразования S-box`ами и P-box`ами)
        x = xor_(op, left);

        left = x;
```

```
        // Замена
        if (i != 15) {
            swap(left, right);
        }
        cout << "Round " << i + 1 << " " << bin2hex(left) << " "
             << bin2hex(right) << " " << rk[i] << endl;
    }

    // Комбинация
    string combine = left + right;

    // Конечная таблица перестановок (FP) без ключа
    int final_perm[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 };

    // Конечная перестановка
    string cipher = bin2hex(permute(combine, final_perm, 64));
    return cipher;
}
int main()
{
    // pt — обычный текст
    string pt, key;

    /*cout<<"Введите обычный текст (в шестнадцатеричном формате): ";
    cin>>pt;
    cout<<"Введите ключ (в шестнадцатеричном формате): ";
    cin>>key;*/

    pt = "123456ABCD132536";
    key = "AABB09182736CCDD";
    // Генерация ключей

    // Шестнадцатеричный код в двоичный
    key = hex2bin(key);

    // Таблица получение 56-битного ключа из 64-битного
    int keyp[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 };

    // получение 56-битного ключа из 64-битного
    key = permute(key, keyp, 56); // ключ без четности
```

```
    // Количество битовых сдвигов половины ключа (28 бит)
    int shift_table[16] = { 1, 1, 2, 2,
                            2, 2, 2, 2,
                            1, 2, 2, 2,
                            2, 2, 2, 1 };

    // Таблица компрессии (прим. бит №14 становится первым и тд)
    int key_comp[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 };

    // Разделение
    string left = key.substr(0, 28);
    string right = key.substr(28, 28);

    vector<string> rkb; // RoundKeys в двоичном формате
    vector<string> rk; // RoundKeys в шестнадцатеричном формате

    for (int i = 0; i < 16; i++) {
        // Сдвиг половины ключа (28 бит)
        left = shift_left(left, shift_table[i]);
        right = shift_left(right, shift_table[i]);

        // Объединение ключец
        string combine = left + right;

        // Сжатие ключей и перестановки битов
        string RoundKey = permute(combine, key_comp, 48);

        rkb.push_back(RoundKey); // добавляем элемент в конец дин массива
        rk.push_back(bin2hex(RoundKey));
    }
    cout << "\nPlain Text: " << pt << endl;
    cout << "\nEncryption:\n\n";
    string cipher = encrypt(pt, rkb, rk);
    cout << "\nCipher Text: " << cipher << endl;

    cout << "\nDecryption\n\n";
    reverse(rkb.begin(), rkb.end());
    reverse(rk.begin(), rk.end());
    string text = encrypt(cipher, rkb, rk);
    cout << "\nPlain Text: " << text << endl;
}
```