БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

ИНФОРМАТИКИ И РАДИОЭЛЕКТРОНИКИ

Кафедра информатики

Факультет КСиС

Специальность ИиТП

Индивидуальная практическая работа

по дисциплине «Методы защиты информации»

Выполнил: Реут М.В.

группа 893551

Минск 2022

# **1. Постановка задачи**

**Цель**: реализовать на языке программирования C++ алгоритмы

симметричного шифрования DES и ГОСТ 28147.

**Результат**: Программа шифрования, осуществляющая криптографическое

преобразование введенного текста с помощью алгоритмов симметричного

шифрования DES и ГОСТ 28147.

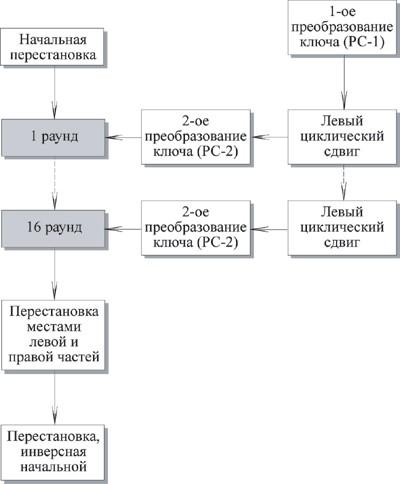
**Общая постановка задачи**: Создать программу, читающую данные из

файла и шифрующие их с помощью алгоритмов DES и ГОСТ 28147

# **2. Схема алгоритма**

## 2.1 Алгоритм DES

Самым распространенным и наиболее известным алгоритмом симметричного шифрования является *DES* (Data Encryption Standard). Алгоритм был разработан в 1977 году, в 1980 году был принят NIST (National Institute of Standards and Technology США) в качестве стандарта (FIPS PUB 46). *DES* является классической *сетью Фейстеля* с двумя ветвями. Данные шифруются 64-битными блоками, используя 56-битный ключ. Алгоритм преобразует за несколько *раундов* 64-битный вход в 64-битный выход. Длина ключа равна 56 битам. Процесс шифрования состоит из четырех этапов. На первом из них выполняется начальная перестановка (*IP*) 64-битного исходного текста (забеливание), во время которой биты переупорядочиваются в соответствии со стандартной таблицей. Следующий этап состоит из 16 *раундов* одной и той же функции, которая использует операции сдвига и подстановки. На третьем этапе левая и правая половины выхода последней (16-й) итерации меняются местами. Наконец, на четвертом этапе выполняется перестановка IP-1 результата, полученного на третьем этапе. Перестановка IP-1 инверсна начальной перестановке.



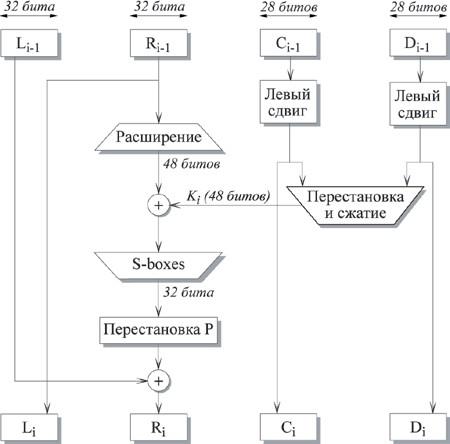
# **Рисунок 2.1 – Общая схема DES**

**Шифрование**

# **Начальная перестановка**

Начальная перестановка и ее инверсия определяются стандартной таблицей. Если М- это произвольные 64 бита, то X = IP (M)-переставленные 64 бита. Если применить обратную функцию перестановки Y = IP-1 (X) = IP-1 (IP(M)), то получится первоначальная последовательность бит.

# **Последовательность преобразований отдельного раунда**

Теперь рассмотрим последовательность преобразований, используемую в каждом *раунде*.

# **Рисунок 2.2 –I-ый раунд DES**

64-битный входной блок проходит через 16 *раундов*, при этом на каждой итерации получается промежуточное 64-битное значение. Левая и правая части каждого промежуточного значения трактуются как отдельные 32- битные значения, обозначенные *L* и *R*. Каждую итерацию можно описать следующим образом:

Li = Ri-1

Ri = Li-1 ⊕ F(Ri-1, Ki)

Где ⊕ обозначает операцию XOR.

Таким образом, выход левой половины *Li* равен входу правой половины *Ri-1*. Выход правой половины *Ri* является результатом применения операции XOR к *Li-1* и функции *F*, зависящей от *Ri-1* и *Ki*.

Рассмотрим функцию *F* более подробно.

*Ri*, которое подается на вход функции *F*, имеет длину 32 бита. Вначале Riрасширяется до 48 бит, используя таблицу, которая определяет перестановку плюс расширение на 16 бит. Расширение происходит следующим образом. 32 бита разбиваются на группы по 4 бита и затем расширяются до 6 бит, присоединяя крайние биты из двух соседних групп. Например, если часть входного сообщения

. . . efgh ijkl mnop . . .

то в результате расширения получается сообщение

. . . defghi hijklm lmnopq . .

После этого для полученного 48-битного значения выполняется операция XOR с 48-битным *подключом Ki*. Затем полученное 48-битное значение подается на вход функции подстановки, результатом которой является 32- битное значение.

Подстановка состоит из восьми *S-boxes,*каждый из которых на входе получает 6 бит, а на выходе создает 4 бита. Эти преобразования определяются специальными таблицами. Первый и последний биты входного значения *S-box* определяют номер строки в таблице, средние 4 бита определяют номер столбца. Пересечение строки и столбца определяет 4-битный выход. Например, если входом является 011011, то номер строки равен 01 (строка 1) и номер столбца равен 1101 (столбец 13). Значение в строке 1 и столбце 13 равно 5, т.е. выходом является 0101.

Далее полученное 32-битное значение обрабатывается с помощью перестановки *Р*, целью которой является максимальное переупорядочивание бит, чтобы в следующем *раунде* шифрования с большой вероятностью каждый бит обрабатывался другим *S-box*.

# **Создание подключей**

Ключ для отдельного *раунда Ki* состоит из 48 бит. Ключи *Ki* получаются по следующему алгоритму. Для 56-битного ключа,используемого на входе алгоритма, вначале выполняется перестановка в соответствии с таблицей Permuted Choice 1 (РС-1). Полученный 56-битный ключ разделяется на две 28- битные части, обозначаемые как C0 и D0 соответственно. На каждом *раунде Ci* и *Di* независимо циклически сдвигаются влево на 1 или 2 бита, в зависимости от номера *раунда*. Полученные значения являются входом следующего *раунда*. Они также представляют собой вход в Permuted Choice 2 (РС-2), который создает 48-битное выходное значение, являющееся входом функции *F*(*Ri-1*, *Ki*).

# **Дешифрование**

Процесс дешифрования аналогичен процессу шифрования. На входе алгоритма используется зашифрованный текст, но ключи *Ki* используются в обратной последовательности. *K16* используется на первом *раунде*, *K1* используется на последнем *раунде*.

**Проблемой DES** является малая длина ключа. Также без ответа пока остается вопрос, возможен ли криптоанализ с использованием существующих характеристик алгоритма *DES*. Основой алгоритма являются восемь таблиц подстановки, или *S-boxes*, которые применяются в каждой итерации. Существует опасность, что эти *S-boxes* конструировались таким образом, что криптоанализ возможен для взломщика, который знает слабые места *S-boxes*. В течение многих лет обсуждалось как стандартное, так и неожиданное поведение *S-boxes*, но все-таки никому не удалось обнаружить их фатально слабые места.

# **Двойной DES**

Простейший способ увеличить длину ключа состоит в повторном применении *DES* с двумя разными ключами. Используя незашифрованное

сообщение P и два ключа K1 и K2, зашифрованное сообщение С можно получить следующим образом:

C = Ek2 [Ek1 [P]]

При дешифрования два ключа применяются в обратном порядке: P = Dk1 [Dk2 [C]]

В этом случае длина ключа равна 56 \* 2 = 112 бит.

# **Атака "встреча посередине"**

Для приведенного выше алгоритма двойного *DES* существует так называемая атака "встреча посередине". Она основана на следующем свойстве алгоритма. Мы имеем

С = Ek2 [Ek1 [P]]

Тогда

X = Ek1 [P] = Dk2 [C].

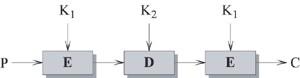
Атака состоит в следующем. Требуется, чтобы атакующий знал хотя бы одну пару незашифрованный текст и соответствующий ему зашифрованный текст: (Р, С). В этом случае, во-первых, шифруется Р для всех возможных 256 значений K1. Этот результат запоминается в таблице, и затем таблица упорядочивается по значению Х. Следующий шаг состоит в дешифровании С, с применением всех возможных 256 значений K2. Для каждого выполненного дешифрования ищется равное ему значение в первой таблице. Если соответствующее значение найдено, то считается, что эти ключи могут быть правильными, и они проверяются для следующей известной пары незашифрованный текст, зашифрованный текст.

# **Тройной DES с двумя ключами**

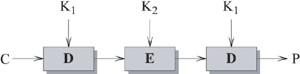
Очевидное противодействие атаке "встреча посередине" состоит в использовании третьей стадии шифрования с тремя различными ключами. Это поднимает стоимость лобовой атаки до 2168, которая на сегодняшний день считается выше практических возможностей. Но при этом длина ключа равна 56 \* 3 = 168 бит, что иногда бывает громоздко.

В качестве альтернативы предлагается метод тройного шифрования,использующий только два ключа. В этом случае выполняется последовательность зашифрование-расшифрование-зашифрование (EDE).

C = EK1 [DK2 [EK1 [P]]]



# **Рисунок 2.3 – Шифрование тройным DES**



**Рисунок 2.4 – Дешифрование тройным DES**

Не имеет большого значения, что используется на второй стадии: шифрование или дешифрование. В случае использования дешифрования существует только то преимущество, что можно *тройной DES* свести к обычному одиночному *DES*, используя K1 = K2:

C = EK1 [DK1 [EK1 [P]]] = EK1 [P]

Известных криптографических атак на *тройной DES* не существует. Цена подбора ключа в *тройном DES* равна 2112.

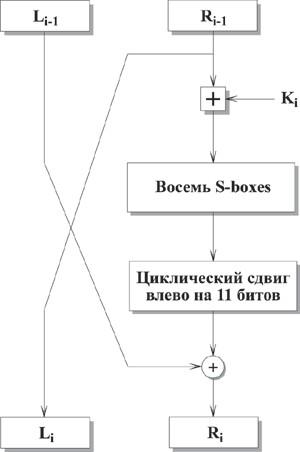
## 2.2 Алгоритм ГОСТ 28147

*Алгоритм ГОСТ 28147* является отечественным стандартом для алгоритмов симметричного шифрования. *ГОСТ 28147* разработан в 1989 году, является блочным алгоритмом шифрования, длина блока равна 64 битам, длина ключа равна 256 битам, количество *раундов* равно 32. Алгоритм представляет собой классическую сеть Фейштеля.

Li = Ri-1

Ri = Li ⊕ f (Ri-1, Ki)

Функция F проста. Сначала правая половина и i-ый *подключ* складываются по модулю 232. Затем результат разбивается на восемь 4- битовых значений, каждое из которых подается на вход *S-box*. *ГОСТ 28147* использует восемь различных *S-boxes*, каждый из которых имеет 4-битовый вход и 4-битовый выход. Выходы всех *S-boxes* объединяются в 32-битное слово, которое затем циклически сдвигается на 11 битов влево. Наконец, с помощью XOR результат объединяется с левой половиной, в результате чего получается новая правая половина.



**Рисунок 2.5 – I-ый раунд ГОСТ 28147**

Генерация ключей проста. 256-битный ключ разбивается на восемь 32- битных *подключей*. Алгоритм имеет 32 *раунда*, поэтому каждый *подключ* используется в четырех *раундах* по следующей схеме:

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Раунд** | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
| **Подключ** | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
| **Раунд** | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 |
| **Подключ** | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
| **Раунд** | 17 | 18 | 19 | 20 | 21 | 22 | 23 | 24 |
| **Подключ** | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |

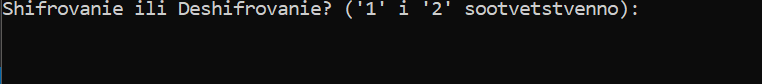
39

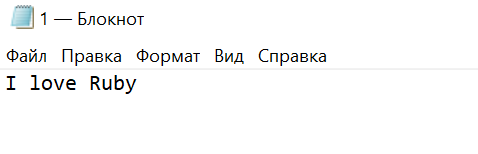
|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Раунд** | 25 | 26 | 27 | 28 | 29 | 30 | 31 | 32 |
| **Подключ** | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 |

Считается, что стойкость *алгоритма ГОСТ 28147* во многом определяется структурой *S-boxes*. Входом и выходом *S-box* являются 4- битные числа, поэтому каждый *S-box* может быть представлен в виде строки чисел от 0 до 15, расположенных в некотором порядке. Тогда порядковый номер числа будет являться входным значением *S-box*, а само число - выходным значением *S-box*.

# **3. Скриншоты ввода данных и результатов выполнения программы**

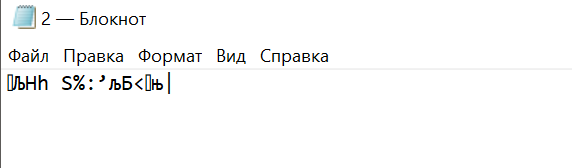
Порядок использования программы:





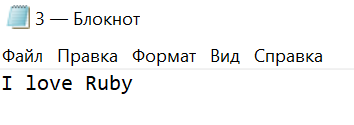
Если вы запустите gost программу то эта программа будет работать с шифрованием ГОСТ 28147. Изначально прежде чем шифровать у вас в файле должно быть написано сообщения для шифровки в файле проекта “1.txt”.

Для того чтобы зашифровать имеющееся сообщение в файле нужно нажать клавишу “1” на клавиатуре вашего устройства.



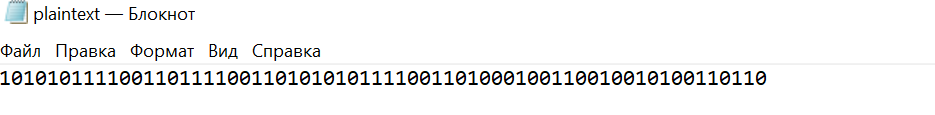
После выполнения шифрования программа выключается, и в файле “2.txt” оказывается зашифрованное сообщение.

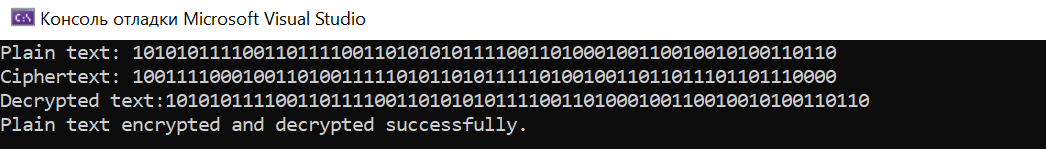
Чтобы расшифровать данное сообщение, нужно заново запустить программу и воспользоваться клавишей “2”.



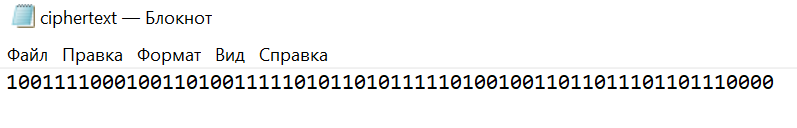
Расшифрованное сообщение после закрытия программы будет находиться в файле “3.txt”.

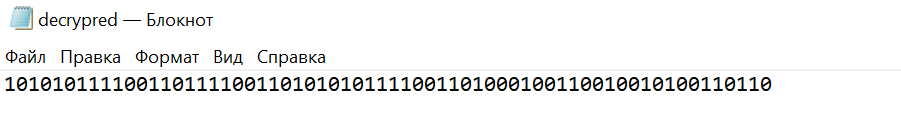
DES шифрование реализовано в проекте des. Перед запуском программы требуются записать в “plaintext.txt” бинарный код вашего сообщения, а после запустить программу.





В консоли программа показывает считанный код вашего сообщения, его шифровку записывающую в файл “ciphertext.txt” и расшифрование зашифровки в файл “decrypred.txt”.





# **4. Исходные файлы**

## gost.cpp

#pragma once

#define \_CRT\_SECURE\_NO\_WARNINGS

#include <stdio.h>

#include "conio.h"

void VivodSimvolov(FILE\* stream, int\* x); //Вывод символов

long Pow(int a, int b); //Возведение в степень

long FileSize(FILE\* stream); //Размер файла (в байтах)

int Zapros(FILE\*\* a, FILE\*\* b, FILE\*\* log); //Выбор режима работы

int NumbOfBloks(FILE\* a, long size); //Определение числа блоков

void ZapolnT(int\* x); //Заполнение блока T символами из входного файла (fi)

void ZapolnB(); //Заполнение блока B

void ZapolnA(); //Заполнение блока A

void ZapolnK(int\* t); //Заполнение блока K

void Sum232(); //Сумматор по модулю Pow(2,32)=2^32

void ZapolnN(); //Заполнение накопителя

void UzliZamen(); //Прохождение заполнителя через узлы замен

void ZapolnN1(); //Преобразование результатов функции UzliZamen() в накопитель N1

void Sdvig11(); //Сдвиг на 11 бит влево. Результат сдвига - результа функции Фейстеля F(Ai,Ki)

void XOR(int\* t); //Ai+1 = Bi XOR F(Ai,Ki)

void SohrBnext(int\* t); //Bi+1 = Ai

void SkleivanieAB(); //Склеивание блоков. T=AB

void ViviodTexta(int\* x); //Преобразование блока T в символы и вывод этих символов в выходной файл (fo)

FILE\* log; //Файл лога.

FILE\* fi; //Файловая переменная с открытым/расшифрованным текстом

FILE\* fo; //Файловая переменная с зашифрованным/расшифрованным текстом

int T[64], c[8], A[32], B[32], C[32], K[32], SUM32[32]; //Блоки для обработки битов

int N[8] = { 0,0,0,0,0,0,0,0 }, N1[32], F[32]; //Вспомогательные блоки (накопители)

unsigned long KEY[8] = //Ключ

{

123456,

654321,

162534,

111111,

222222,

333333,

444444,

555555

};

unsigned char ch; //Символьная переменная (для обработки символов)

int n; //Вспомогательная переменная

int blokN; //Число блоков

long size; //Размер файла в байтах

bool nekratno = 0; //Логическая переменная: 0 - если число символов кратно 8, 1 - если не кратно

int rejim; //Выбор режима работы (1 - шифрование, 2 -расшифрование)

int TABLE[8][16] = //Таблица замен

{

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

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

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

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

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

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

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

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

};

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

int main()

{

rejim = Zapros(&fi, &fo, &log); //Выбор режима работы (шифрование или расшифрование)

blokN = NumbOfBloks(fi, size); //Определение числа блоков

for (int p = 0; p < blokN; p++) //Заполнение всех блоков T:

{

VivodSimvolov(fi, &p);

ZapolnT(&p); //Заполнение (p+1)-ого блока Т

ZapolnB(); //Заполнение блока B

ZapolnA(); //Заполнение блока A

for (int t = 1; t <= 32; t++) //Начало основного цикла:

{

fprintf(log, "\n%-2d\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*", t);

ZapolnK(&t); //Заполнение ключа K

//Функция Фейстеля:

Sum232(); //1) Сумматор по модулю 2^32

ZapolnN(); //2) Заполнение накопителя

UzliZamen(); //3) Проход через узлы замен

ZapolnN1(); //4) Заполнение выходного накопителя

Sdvig11(); //5) Сдвиг на 11 битов влево

//Конец функции Фейстеля.

for (int i = 0; i < 32; i++) //Сохраняем блок A в C

C[i] = A[i];

XOR(&t); //Двоичное исключающее "или" для Bi и F. Результат сохраняем в At+1

SohrBnext(&t); //Сохраняем At в Bt+1

} //Конец основного цикла.

SkleivanieAB(); //Склеивание блоков A33+B33

ViviodTexta(&p); //Вывод зашифрованного текста в выходной файл

} //Конец алгоритма.

return 0;

}

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

long Pow(int a, int b) //Возведение в степень

{

long rez = 1;

for (int k = 1; k <= b; k++)

{

rez = rez \* a;

}

return rez;

}

long FileSize(FILE\* stream) //Размер файла (в байтах)

{

long curpos, length;

curpos = ftell(stream); //Сохраняем значение указателя на символ

fseek(stream, 0L, SEEK\_END);

length = ftell(stream);

fseek(stream, curpos, SEEK\_SET);//Возвращяемся на исходное место

return length;

}

void VivodSimvolov(FILE\* stream, int\* x) //Вывод символов

{

unsigned char cc;

long curpos;

int d;

curpos = ftell(stream); //Сохраняем значение указателя на символ

fprintf(log, " ");

if ((\*x == blokN - 1) && (nekratno == 1))

d = FileSize(stream) % 8;

else

d = 8;

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

{

cc = fgetc(stream);

fprintf(log, "%8c", cc);

}

if ((\*x == blokN - 1) && (nekratno == 1))

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

fprintf(log, "%8c", ' ');

fprintf(log, "\n");

fseek(stream, curpos, SEEK\_SET);//Возвращяемся на исходное место

}

int Zapros(FILE\*\* a, FILE\*\* b, FILE\*\* log) //Выбор режима работы

{

int vib;

char c;

printf("Shifrovanie ili Deshifrovanie? ('1' i '2' sootvetstvenno): ");

c = \_getch();

printf("%c", c);

if (c == '1') //Режим шифрования

{

\*a = fopen("1.txt", "rb");

\*b = fopen("2.txt", "wb");

\*log = fopen("LogShifr.txt", "wb");

vib = 1;

}

if (c == '2') //Режим расшифрования

{

\*a = fopen("2.txt", "rb");

\*b = fopen("3.txt", "wb");

\*log = fopen("LogRasshifr.txt", "wb");

vib = 2;

}

return vib;

}

int NumbOfBloks(FILE\* a, long size) //Определение числа блоков

{

int n;

size = FileSize(a);

n = size / 8; //Сохраняем число ПОЛНЫХ (8-символьных) блоков

if (size % 8 != 0) //Если число символов в файле не кратно 8, то увеличивыем число блоков

{

n++;

nekratno = 1;

}

fprintf(log, "%d %d\n", size, n);

return n;

}

void ZapolnT(int\* x) //Заполнение блока T символами из входного файла (fi)

{

int d = FileSize(fi) % 8;

if ((\*x == blokN - 1) && (nekratno == 1)) //Если идёт последний блок и число символов в файле не кратно 8 то:

{

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

{

ch = fgetc(fi); //Берем символ

n = ch; //Сохраняем его номер

for (int j = 0; j < 8; j++) //Заполняем блок битами символов (число символов N<8)

{

if (n / Pow(2, 7 - j) >= 1)

{

T[8 \* i + j] = 1;

n = n - Pow(2, 7 - j);

}

else

T[8 \* i + j] = 0;

}

}

for (int i = 8 \* (d % 8); i < 64; i++) //Заполняем оставшуюся часть блока нулями.

T[i] = 0;

}

else //Иначе

{

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

{

ch = fgetc(fi); //Берем символ

n = ch; //Сохраняем его номер

for (int j = 0; j < 8; j++) //Заполняем блок битами символов (число символов N=8)

{

if (n / Pow(2, 7 - j) >= 1)

{

T[8 \* i + j] = 1;

n = n - Pow(2, 7 - j);

}

else

T[8 \* i + j] = 0;

}

}

}

//Вывод блока Т

fprintf(log, "Tnach ");

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

fprintf(log, "%d", T[i]);

}

void ZapolnB() //Заполнение блока B

{

//Заполнение и вывод блока В1

fprintf(log, "\nB1 ");

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

{

B[i] = T[i];

fprintf(log, "%d", B[i]);

}

}

void ZapolnA() //Заполнение блока A

{

//Заполнение и вывод блока A1

fprintf(log, "\nA1 ");

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

{

A[i] = T[32 + i];

fprintf(log, "%d", A[i]);

}

}

void ZapolnK(int\* t) //Заполнение и вывод подключа K

{

fprintf(log, "\nK%-5d", \*t);

unsigned long x = 0;

if (rejim == 1) //Если режим шифрования, то

{ //последовательность подключей имеет следующий вид:

if ((\*t >= 25) && (\*t <= 32))

x = KEY[7 - (\*t - 1) % 8];

else

x = KEY[(\*t - 1) % 8];

}

if (rejim == 2) //Если режим шифрования, то

{ //последовательность подключей имеет следующий вид:

if ((\*t >= 1) && (\*t <= 8))

x = KEY[(\*t - 1) % 8];

else

x = KEY[7 - (\*t - 1) % 8];

}

for (int i = 0; i < 32; i++) //Переводим значение подключа в массив битов

{

if (x / Pow(2, 31 - i) >= 1)

{

K[i] = 1;

x = x - Pow(2, 31 - i);

}

else

K[i] = 0;

fprintf(log, "%d", K[i]); //Вывод подключа

}

}

void Sum232() //Заполнение блока K

{

fprintf(log, "\nSUM32 ");

for (int c = 0, i = 31; i >= 0; i--) //Поразрядное сложение блоков

{

if ((A[i] + K[i] + c) >= 2) //Если переполнение, то:

{

SUM32[i] = A[i] + K[i] + c - 2;

c = 1;

}

else //Иначе:

{

SUM32[i] = A[i] + K[i] + c;

c = 0;

}

}

for (int i = 0; i < 32; i++) //вывод сумматора

fprintf(log, "%d", SUM32[i]);

}

void ZapolnN() //Заполнение накопителя

{

fprintf(log, "\nN ");

for (int i = 0; i < 8; i++) //В накопителе 8 чисел

{

for (int j = 0; j < 4; j++) //Преобразуем 4 бита сумматора (начиная с конца, т.е. 4 посл бита, 4 предпосл бита и т.д.) в 10-чное число

if (SUM32[31 - 4 \* i - j] == 1)

N[i] = N[i] + Pow(2, j);

fprintf(log, "%d ", N[i]); //Выводим накопитель

}

}

void UzliZamen() //Прохождение заполнителя через узлы замен

{

fprintf(log, "%\nN ");

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

{

k = N[i];

N[i] = TABLE[i][k]; //Выход из i-ого узла

fprintf(log, "%d ", N[i]);

}

}

void ZapolnN1() //Преобразование результатов функции UzliZamen() в накопитель N1

{

for (int i = 0; i < 8; i++) //Все 8 чисел накопителя N

for (int j = 0; j < 4; j++) //переводим в двоичный вид

{

if (N[i] / Pow(2, 3 - j) >= 1)

{

N1[4 \* i + j] = 1;

N[i] = N[i] - Pow(2, 3 - j);

}

else

N1[4 \* i + j] = 0;

}

fprintf(log, "%\nN1 ");

for (int i = 0; i < 32; i++) //вывод выходного накопителя

fprintf(log, "%d", N1[i]);

}

void Sdvig11() //Сдвиг на 11 бит влево. Результат сдвига - результа функции Фейстеля F(Ai,Ki)

{

fprintf(log, "\nF ");

for (int i = 0; i < 21; i++)

{

F[i] = N1[i + 11]; //Сохраняем биты с 12 по 32

fprintf(log, "%d", F[i]);

}

for (int i = 0; i < 11; i++)

{

F[i + 21] = N1[i]; //Сохраняем биты с 1 по 11

fprintf(log, "%d", F[i + 21]);

}

}

void XOR(int\* t) //Ai+1 = Bi XOR F(Ai,Ki)

{

fprintf(log, "\nA%-5d", \*t + 1);

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

{

A[i] = F[i] ^ B[i]; //Двоичное исключающее "ИЛИ"

fprintf(log, "%d", A[i]);

}

}

void SohrBnext(int\* t) //Bi+1 = Ai

{

fprintf(log, "\nB%-5d", \*t + 1);

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

{

B[i] = C[i];

fprintf(log, "%d", B[i]);

}

}

void SkleivanieAB() //Склеивание блоков. T=AB

{

fprintf(log, "\nTkon ");

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

T[i] = A[i];

for (int i = 32; i < 64; i++)

T[i] = B[i - 32];

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

fprintf(log, "%d", T[i]);

}

void ViviodTexta(int\* x) //Преобразование блока T в символы и вывод этих символов в выходной файл (fo)

{

int d;

fprintf(log, "\n\nRezultat: ");

if ((\*x == blokN - 1) && (nekratno == 1) && (rejim == 2)) //Если последний блок, "..." - некратно и режим расшифрования, то

d = FileSize(fi) % 8; //выводим первые d=FileSize(fi)%8 символов.

else //Иначе

d = 8; //выводим 8 символов

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

{

n = 0;

for (int j = 0; j < 8; j++) //Преобразование из 8 бит в соответствующий символ

if (T[8 \* i + j] == 1)

n = n + Pow(2, 7 - j);

ch = n;

fprintf(log, "%c", ch);

fprintf(fo, "%c", ch);

}

fprintf(log, "\n");

}

## des.cpp

// Including dependancies

#include <iostream>

#include <string>

#include <cmath>

#include <fstream>

using namespace std;

// Array to hold 16 keys

string round\_keys[16];

// String to hold the plain text

string pt;

// Function to convert a number in decimal to binary

string convertDecimalToBinary(int decimal)

{

string binary;

while (decimal != 0) {

binary = (decimal % 2 == 0 ? "0" : "1") + binary;

decimal = decimal / 2;

}

while (binary.length() < 4) {

binary = "0" + binary;

}

return binary;

}

// Function to convert a number in binary to decimal

int convertBinaryToDecimal(string binary)

{

int decimal = 0;

int counter = 0;

int size = binary.length();

for (int i = size - 1; i >= 0; i--)

{

if (binary[i] == '1') {

decimal += pow(2, counter);

}

counter++;

}

return decimal;

}

// Function to do a circular left shift by 1

string shift\_left\_once(string key\_chunk) {

string shifted = "";

for (int i = 1; i < 28; i++) {

shifted += key\_chunk[i];

}

shifted += key\_chunk[0];

return shifted;

}

// Function to do a circular left shift by 2

string shift\_left\_twice(string key\_chunk) {

string shifted = "";

for (int i = 0; i < 2; i++) {

for (int j = 1; j < 28; j++) {

shifted += key\_chunk[j];

}

shifted += key\_chunk[0];

key\_chunk = shifted;

shifted = "";

}

return key\_chunk;

}

// Function to compute xor between two strings

string Xor(string a, string b) {

string result = "";

int size = b.size();

for (int i = 0; i < size; i++) {

if (a[i] != b[i]) {

result += "1";

}

else {

result += "0";

}

}

return result;

}

// Function to generate the 16 keys.

void generate\_keys(string key) {

// The PC1 table

int pc1[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

};

// The PC2 table

int pc2[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

};

// 1. Compressing the key using the PC1 table

string perm\_key = "";

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

perm\_key += key[pc1[i] - 1];

}

// 2. Dividing the key into two equal halves

string left = perm\_key.substr(0, 28);

string right = perm\_key.substr(28, 28);

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

// 3.1. For rounds 1, 2, 9, 16 the key\_chunks

// are shifted by one.

if (i == 0 || i == 1 || i == 8 || i == 15) {

left = shift\_left\_once(left);

right = shift\_left\_once(right);

}

// 3.2. For other rounds, the key\_chunks

// are shifted by two

else {

left = shift\_left\_twice(left);

right = shift\_left\_twice(right);

}

// Combining the two chunks

string combined\_key = left + right;

string round\_key = "";

// Finally, using the PC2 table to transpose the key bits

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

round\_key += combined\_key[pc2[i] - 1];

}

round\_keys[i] = round\_key;

}

}

// Implementing the algorithm

string DES() {

// The initial permutation table

int initial\_permutation[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

};

// The expansion table

int expansion\_table[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

};

// The substitution boxes. The should contain values

// from 0 to 15 in any order.

int substition\_boxes[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

} };

// The permutation table

int permutation\_tab[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

};

// The inverse permutation table

int inverse\_permutation[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

};

//1. Applying the initial permutation

string perm = "";

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

perm += pt[initial\_permutation[i] - 1];

}

// 2. Dividing the result into two equal halves

string left = perm.substr(0, 32);

string right = perm.substr(32, 32);

// The plain text is encrypted 16 times

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

string right\_expanded = "";

// 3.1. The right half of the plain text is expanded

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

right\_expanded += right[expansion\_table[i] - 1];

}; // 3.3. The result is xored with a key

string xored = Xor(round\_keys[i], right\_expanded);

string res = "";

// 3.4. The result is divided into 8 equal parts and passed

// through 8 substitution boxes. After passing through a

// substituion box, each box is reduces from 6 to 4 bits.

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

// Finding row and column indices to lookup the

// substituition box

string row1 = xored.substr(i \* 6, 1) + xored.substr(i \* 6 + 5, 1);

int row = convertBinaryToDecimal(row1);

string col1 = xored.substr(i \* 6 + 1, 1) + xored.substr(i \* 6 + 2, 1) + xored.substr(i \* 6 + 3, 1) + xored.substr(i \* 6 + 4, 1);;

int col = convertBinaryToDecimal(col1);

int val = substition\_boxes[i][row][col];

res += convertDecimalToBinary(val);

}

// 3.5. Another permutation is applied

string perm2 = "";

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

perm2 += res[permutation\_tab[i] - 1];

}

// 3.6. The result is xored with the left half

xored = Xor(perm2, left);

// 3.7. The left and the right parts of the plain text are swapped

left = xored;

if (i < 15) {

string temp = right;

right = xored;

left = temp;

}

}

// 4. The halves of the plain text are applied

string combined\_text = left + right;

string ciphertext = "";

// The inverse of the initial permuttaion is applied

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

ciphertext += combined\_text[inverse\_permutation[i] - 1];

}

//And we finally get the cipher text

return ciphertext;

}

int main() {

// A 64 bit key

ifstream fin("plaintext.txt");

string key = "1010101010111011000010010001100000100111001101101100110011011101";

// A block of plain text of 64 bits

fin >> pt;

string apt = pt;

// Calling the function to generate 16 keys

generate\_keys(key);

cout << "Plain text: " << pt << endl;

// Applying the algo

string ct = DES();

ofstream foutc("ciphertext.txt");

foutc << ct;

cout << "Ciphertext: " << ct << endl;

// Reversing the round\_keys array for decryption

int i = 15;

int j = 0;

while (i > j)

{

string temp = round\_keys[i];

round\_keys[i] = round\_keys[j];

round\_keys[j] = temp;

i--;

j++;

}

pt = ct;

string decrypted = DES();

ofstream foutd("decrypred.txt");

foutd << decrypted;

cout << "Decrypted text:" << decrypted << endl;

// Comapring the initial plain text with the decrypted text

if (decrypted == apt) {

cout << "Plain text encrypted and decrypted successfully." << endl;

}

return 0;

}