Министерство образования Республики Беларусь
Белорусский государственный университет информатики и радиоэлектроники
Кафедра информатики

ОТЧЕТ
по лабораторной работе №1

Симметричная криптография. Двойной и тройной DES

Выполнил:
студент гр. 653501
Шинкевич Г. С.

Проверил:
Артемьев В. С.

Минск 2019

ЗАДАНИЕ:

Реализовать программные средства шифрования и дешифрования текстовых файлов при помощи алгоритмов двойной и тройной DES.

Алгоритм DES

Одной из наиболее известных криптографических систем с закрытым ключом является DES – Data Encryption Standard. Эта система первой получила статус государственного стандарта в области шифрования данных. Она разработана специалистами фирмы IBM и вступила в действие в США 1977 году. Алгоритм DES по-прежнему широко применяется и заслуживает внимания при изучении блочных шифров с закрытым ключом.

Стандарт DES построен на комбинированном использовании перестановки, замены и гаммирования. Шифруемые данные должны быть представлены в двоичном виде.

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 1 Общая схема DES

Шифрование

Начальная перестановка

Начальная перестановка и ее инверсия определяются стандартной таблицей. Если M- 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.

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

Рисунок 2 - DES. Начальная перестановка

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

Рисунок 3 - DES. Заключительная перестановка

Последовательность преобразований отдельного раунда

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

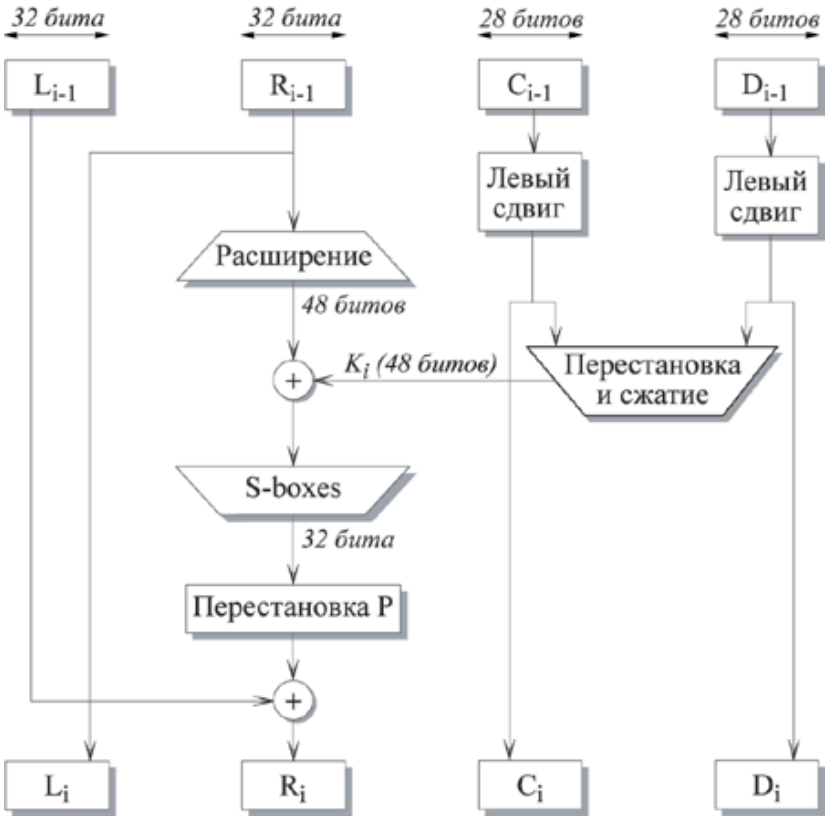

Рисунок 4 - I-ый раунд DES

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:

$L_i = R_{i-1}$

$R_i = L_{i-1} \oplus F(R_{i-1}, K_i)$

Где $\oplus$ обозначает операцию XOR.

Таким образом, выход левой половины L_i равен входу правой половины R_{i-1}. Выход правой половины R_i является результатом применения операции XOR к L_{i-1} и функции F, зависящей от R_{i-1} и K_i.

Рассмотрим функцию F более подробно.

R_i, которое подается на вход функции F, имеет длину 32 бита. Вначале R_i расширяется 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* K_i. 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.

שורה	מס' עמודה															
	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
	S_1															
0	14	4	13	1	2	15	11	8	3	10	6	12	5	9	0	7
1	0	15	7	3	14	2	13	1	10	6	12	11	9	5	3	8
2	4	1	14	8	13	6	2	11	15	12	9	7	13	10	5	0
3	15	12	8	2	4	9	1	7	5	11	3	14	10	0	6	13
	S_2															
0	15	1	8	14	6	11	3	4	9	7	2	13	12	0	5	10
1	3	13	4	7	15	2	8	14	12	0	1	10	6	9	11	5
2	0	14	7	11	10	4	13	1	5	8	12	6	9	3	2	15
3	13	8	10	1	3	15	4	2	11	6	7	12	0	5	14	9
	S_3															
0	10	0	9	14	6	3	15	5	1	13	12	7	11	4	2	8
1	13	7	0	9	3	4	6	10	2	8	5	14	12	11	15	1
2	13	6	4	9	8	15	3	0	11	1	2	12	5	10	14	7
3	1	10	13	0	6	9	8	7	4	15	14	3	11	5	2	12
	S_4															
0	7	13	14	3	0	6	9	10	1	2	8	5	11	12	4	15
1	13	8	11	5	6	15	0	3	4	7	2	12	1	10	14	9
2	10	6	9	0	12	11	7	13	15	1	3	14	5	2	8	4
3	3	15	0	6	10	1	13	8	9	4	5	11	12	7	2	14
	S_5															
0	2	12	4	1	7	10	11	6	8	5	3	15	13	0	14	9
1	14	11	2	12	4	7	13	1	5	0	15	10	3	9	8	6
2	4	2	1	11	10	13	7	8	15	9	12	5	6	3	0	14
3	11	8	12	7	1	14	2	13	6	15	0	9	10	4	5	3
	S_6															
0	12	1	10	15	9	2	6	8	0	13	3	4	14	7	5	11
1	10	15	4	2	7	12	9	5	6	1	13	14	0	11	3	8
2	9	14	15	5	2	8	12	3	7	0	4	10	1	13	11	6
3	4	3	2	12	9	5	15	10	11	14	1	7	6	0	8	13
	S_7															
0	4	11	2	14	15	0	8	13	3	12	9	7	5	10	6	1
1	13	0	11	7	4	9	1	10	14	3	5	12	2	15	8	6
2	1	4	11	13	12	3	7	14	10	15	6	8	0	5	9	2
3	6	11	13	8	1	4	10	7	9	5	0	15	14	2	3	12
	S_8															
0	13	2	8	4	6	15	11	1	10	9	3	14	5	0	12	7
1	1	15	13	8	10	3	7	4	12	5	6	11	0	14	9	2
2	7	11	4	1	9	12	14	2	0	6	10	13	15	3	5	8
3	2	1	14	7	4	10	8	13	15	12	9	0	3	5	6	11

Рисунок 5 - S-boxes

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

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

Рисунок 6 - Перестановка с помощью P-блоков

Создание подключей

Ключ для отдельного *раунда* K_i состоит из 48 бит. Ключи K_i получаются по следующему алгоритму. Для 56-битного ключа, используемого на входе алгоритма (если используется 64-битный ключ, то, как видно из рис. 5 убираются биты 64, 56, 48, 40, 32, 16, 8) , вначале выполняется перестановка в соответствии с таблицей Permuted Choice 1 (PC-1).

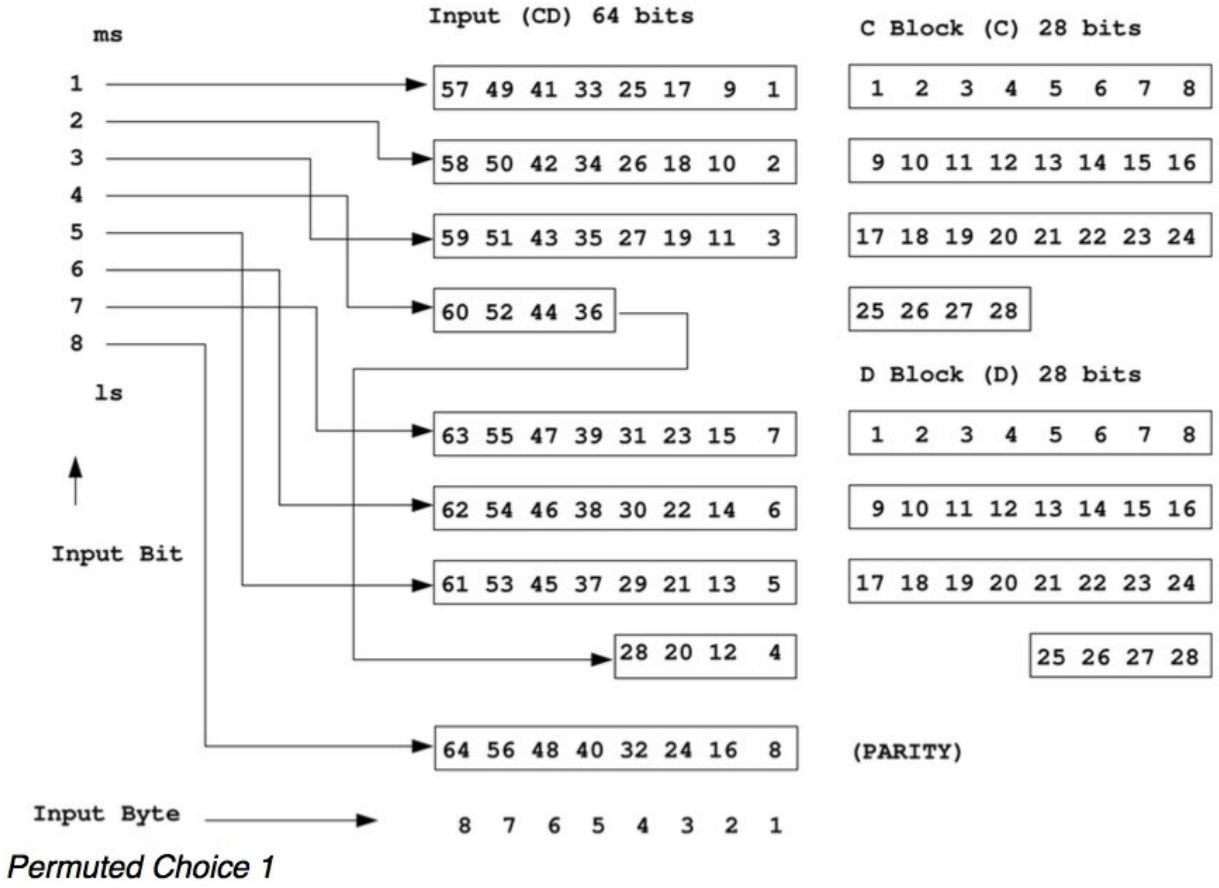

Рисунок 7 - Схема Permuted Choice

Полученный 56-битный ключ разделяется на две 28-битные части, обозначаемые как C_0 и D_0 соответственно. На каждом *раунде* C_i и D_i независимо циклически сдвигаются влево на 1 или 2 бита, в зависимости от номера *цикла*.

Номер цикла	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
Сдвиг (бит)	1	1	2	2	2	2	2	2	1	2	2	2	2	2	2	1

Рисунок 8 - Сдвиг ключа в зависимости от номера цикла

Полученные значения являются входом следующего *раунда*. Они также представляют собой вход в Permuted Choice 2 (PC-2),

который создает 48-битное выходное значение, являющееся входом функции $F(R_{i-1}, K_i)$

Дешифрование

Процесс дешифрования аналогичен процессу шифрования. На входе алгоритма используется зашифрованный текст, но ключи K_i используются в обратной последовательности. K_{16} используется на первом *раунде*, K_1 используется на последнем *раунде*.

3DES (Triple DES, TDES) - тройной DES. Усовершенствованный блочный алгоритм DES. Симметричный блочный криптографический алгоритм, созданный на основе алгоритма DES с целью устранения главного недостатка — малой длины ключа (56 бит), который может быть взломан методом перебора ключа.

Принцип работы 3DES не отличается от применяемого в DES; наращивание криптостойкости было достигнуто благодаря трехкратному (triple) шифрованию одного блока DES. Три 56-разрядных ключа, используемых в данном процессе, объединяются алгоритмом в один 168-разрядный ключ. Хотя время атаки перебором при обычных ресурсах компьютера составляет несколько миллиардов машинолет, что говорит о хорошей стойкости алгоритма, в некоторых публикациях описаны способы сокращения времени атаки — до уровня перебора 108-разрядного ключа.

В варианте, известном как EEE, данную операцию можно описать следующим образом: DES (k3;DES (k2;DES (k1; M))), где M— блок исходных данных, k1, k2, и k3— ключи DES. Более распространен вариант EDE, в котором серединное шифрование DES с ключом k2 заменяется операцией дешифрования с тем же ключом (k2): DES(k3;DES — 1(k2; DES (k1; M))).

Таким образом, длина ключа алгоритма 3DES равна 156 битам (3-х ключей DES). Сегодня также существуют варианты Triple DES с «двойным» DES ключом размером 112 бит, «тройным» DES-ключом, которые стали применяться чаще (3DES, TDES, TDEA, 2TDEA, 3TDEA и т.д.).

ДЕМОНСТРАЦИЯ РАБОТЫ

```
Initial text: bla-bla (626C612D626C61)

Encrypted text(DES): 60BE093A7E65868A
Decrypted text(DES): bla-bla (626C612D626C61)

Encrypted text(2DES): 499A96A878C5AB6CCB12A48C6B54C99E
Decrypted text(2DES): bla-bla (626C612D626C61)

Encrypted text(3DES): 6945B8FAF2B652FEB21BC3BE8308D0F4241BC91E5E55B3D6
Decrypted text(3DES): bla-bla (626C612D626C61)
```

ВЫВОД

В результате лабораторной работы была написана программа для шифрования и дешифрования текстовых файлов с помощью алгоритмов DES, 2DES и 3DES.

ЛИСТИНГ ПРОГРАММЫ

```
from enum import Enum

INITIAL_PERMUTATION = [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]

EXPAND_TABLE = [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]

P_TABLE = [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]

FINAL_PERMUTATION = [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]

KEY_TABLE_1 = [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]

KEY_SHIFT_TABLE = [1, 1, 2, 2, 2, 2, 2, 2, 1, 2, 2, 2, 2, 2, 2, 1]

KEY_TABLE_2 = [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]

S_BOX = [
   [[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],
    ]
]


class Action(Enum):
    DECRYPT = 0
    ENCRYPT = 1


def str_to_bin(text, size=8):
    return ''.join(format(ord(char), 'b').zfill(size) for char in text)


def bin_to_hex(binary_string):
    return '%0*X' % ((len(binary_string) + 3) // 4, int(binary_string, 2))


def str_to_hex(text):
    return bin_to_hex(str_to_bin(text))


def substitute(s):
    blocks = split(s, 6)
    result = []
    for i in range(len(blocks)):
        block = blocks[i]
        a = int(str(block[0]) + str(block[5]), 2)
        b = int(''.join([str(x) for x in block[1:][:-1]]), 2)
        val = S_BOX[i][a][b]
        val_bin = str_to_bin(chr(val), 4)
        result += [int(x) for x in val_bin]
    return result


def xor(a, b):
    return [x ^ y for x, y in zip(a, b)]
```

```
def shift(a, b, n):
    return a[n:] + a[:n], b[n:] + b[:n]


def split(s, n):
    return [s[i:i + n] for i in range(0, len(s), n)]


def permutate(block, table):
    return [block[x - 1] for x in table]


def string_to_bit_array(string):
    array = []
    for char in string:
        val_bin = str_to_bin(char)
        array.extend([int(x) for x in list(val_bin)])
    return array


def bit_array_to_string(array):
    return ''.join([chr(int(y, 2)) for y in [''.join([str(x) for x in _bytes]) for _bytes in
split(array, 8)]])


def generate_keys(key):
    keys = []
    key = string_to_bit_array(key)
    key = permutate(key, KEY_TABLE_1)  # 56 bit
    l, r = split(key, 28)
    for i in range(16):  # 16 keys
        l, r = shift(l, r, KEY_SHIFT_TABLE[i])
        key = l + r
        keys.append(permutate(key, KEY_TABLE_2))  # 48 bit
    return keys


def get_padding(text):
    padding_len = 8 - (len(text) % 8)
    return padding_len * chr(padding_len)
```

```
def remove_padding(text):
    padding_len = ord(text[-1])
    return text[:-padding_len]


def des(key, text, action):
    if len(key) < 8:
        raise Exception("Key is too small")
    elif len(key) > 8:
        key = key[:8]

    if action == Action.ENCRYPT:
        text += get_padding(text)

    keys = generate_keys(key)
    text_blocks = split(text, 8)
    result = []
    for block in text_blocks:
        block = string_to_bit_array(block)
        block = permutate(block, INITIAL_PERMUTATION)  # 64 bit
        l, r = split(block, 32)
        for i in range(16):  # 16 rounds
            r_expanded = permutate(r, EXPAND_TABLE)  # 48 bit
            if action == Action.ENCRYPT:
                r_xor = xor(keys[i], r_expanded)
            else:
                r_xor = xor(keys[15 - i], r_expanded)
            r_sbox = substitute(r_xor)
            r_p = permutate(r_sbox, P_TABLE)
            r_p = xor(l, r_p)
            l = r
            r = r_p
        result += permutate(r + l, FINAL_PERMUTATION)
    result = bit_array_to_string(result)
    if action == Action.DECRYPT:
        return remove_padding(result)
    else:
        return result


def encrypt(key, text):
    return des(key, text, Action.ENCRYPT)
```

```
def decrypt(key, text):
    return des(key, text, Action.DECRYPT)


def encrypt_2(key_1, key_2, text):
    return des(key_2, des(key_1, text, Action.ENCRYPT), Action.ENCRYPT)


def decrypt_2(key_1, key_2, text):
    return des(key_1, des(key_2, text, Action.DECRYPT), Action.DECRYPT)

def encrypt_3(key_1, key_2, key_3, text):
    return des(key_3, des(key_2, des(key_1, text, Action.ENCRYPT),
Action.ENCRYPT), Action.ENCRYPT)

def decrypt_3(key_1, key_2, key_3, text):
    return des(key_1, des(key_2, des(key_3, text, Action.DECRYPT),
Action.DECRYPT), Action.DECRYPT)

if __name__ == '__main__':
    with open("data.txt", 'r') as file:
        t = file.read()
    k = "12345678"
    k2 = "87654321"
    k3 = "abcdefgh"
    print("Initial text: {} ({})".format(t, str_to_hex(t)))
    print()
    e = encrypt(k, t)
    print("Encrypted text(DES): {}".format(str_to_hex(e)))
    d = decrypt(k, e)
    print("Decrypted text(DES): {} ({})".format(d, str_to_hex(d)))
    print()
    e2 = encrypt_2(k, k2, t)
    print("Encrypted text(2DES): {}".format(str_to_hex(e2)))
    d2 = decrypt_2(k, k2, e2)
    print("Decrypted text(2DES): {} ({})".format(d2, str_to_hex(d2)))
    print()

    e3 = encrypt_3(k, k2, k3, t)
    print("Encrypted text(3DES): {}".format(str_to_hex(e3)))
    d3 = decrypt_3(k, k2, k3, e3)
    print("Decrypted text(3DES): {} ({})".format(d3, str_to_hex(d3)))
    print()
```