МИНОБРНАУКИ РОССИИ

Изображение выглядит как текст

Автоматически созданное описаниеФедеральное государственное бюджетное образовательное учреждение высшего образования

НИЖЕГОРОДСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ

УНИВЕРСИТЕТ им. Р.Е.АЛЕКСЕЕВА

Институт радиоэлектроники и информационных технологий

Кафедра информатики и систем управления

Отчет

По лабораторной работе №4

«Режимы шифра DES»

По дисциплине

«Защита информации»

РУКОВОДИТЕЛЬ:

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Капранов С.Н.

(подпись) (фамилия, и.,о.)

СТУДЕНТ:

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Игнаков К

(подпись) (фамилия, и.,о.)

19-ВМ

(шифр группы)

Работа защищена «\_\_\_» \_\_\_\_\_\_\_\_\_\_\_\_

С оценкой \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

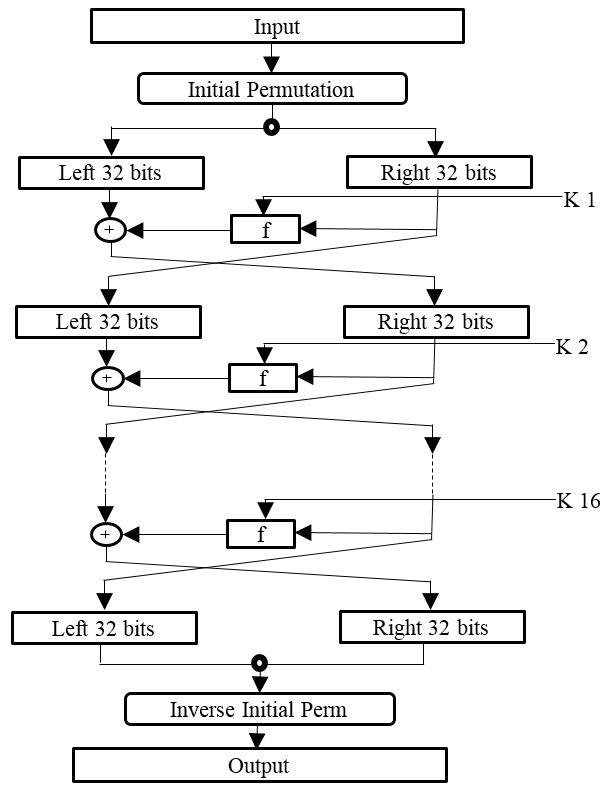
Нижний Новгород 2022

**Задание №4.1.**

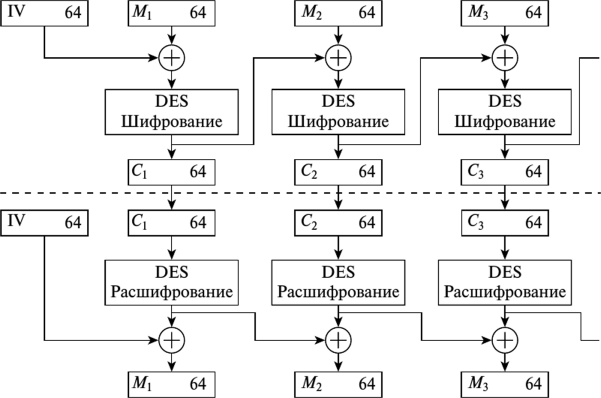
Реализовать стандарт шифрования данных DES в режиме «Сцепление блоков в шифре».

**DES в режиме «Сцепление блоков в шифре».**

Алгоритм DES использует комбинацию подстановок и перестановок. DES осуществляет шифрование 64-битовых блоков данных с помощью 64-битового ключа, в котором значащими являются 56 бит (остальные 8 бит — проверочные биты для контроля на четность). Дешифрование в DES является операцией, обратной шифрованию, и выполняется путем повторения операций шифрования в обратной последовательности. Обобщенная схема процесса:



В режиме «Сцепление блоков в шифре» исходный файл М разбивается на 64-битовые блоки. Первый блок M1 складывается по модулю 2 с 64-битовым начальным вектором IV. Полученная сумма затем шифруется с использованием ключа DES. Полученный 64-битовый шифр C1 складывается по модулю 2 со вторым блоком текста, результат шифруется и получается второй 64-битовый шифр C2 и т.д. Процедура повторяется до тех пор, пока не будут обработаны все блоки текста.



**Алгоритм.**

**Общий алгоритм**:

1. Вводим сообщение, которое будем кодировать/декодировать.

2. Вводим ключ, конвертируем в биты.

3. Генерируем случайную последовательность из 64 бит – вектор V.

4. Вычисляем количество блоков для кодирования.

5. Выбираем первые 8 символов сообщения.

6. Конвертируем в последовательность bits.

7. Выполняем исключающее ИЛИ между bits и вектора V.

8. Кодируем bits.

9. Получившуюся последовательность сохраняем, как новый вектор Vi для следующего блока информации.

10 Конвертируем bits в строку.

11 Вычисляем количество блоков для декодирования.

12. Выбираем первые 8 символов зашифрованного сообщения.

13. Конвертируем в последовательность bits.

14. Сохраняем bits, как новый вектор Vi для следующего блока информации.

15. Декодируем bits.

16. Выполняем исключающее ИЛИ между bits и вектором V.

17 Конвертируем получившуюся последовательность в строку.

**Кодирование**: получаем на вход последовательность бит сообщения и последовательность бит ключа.

1. Сначала производим первоначальную перестановку в соответствии с таблицей IP.

2. Разделяем последовательность бит на левую (L, 0-31) и правую (R,32-63).

3. Из ключа переставляем биты в две последовательности C и D по 28 бит.

4. Запускаем цикл на 16 итераций:

* Сохраняем последовательность R.
* Генерируем ключ Ki.
* Вычисляем последовательность f(R,Ki).
* Выполняем исключающее ИЛИ между L и f.
* Меняем местами L и R (кроме последней итерации).

5. Выполняем конечную перестановку в соответствии с таблицей IP-1.

**Декодирование**: получаем на вход последовательность бит зашифрованного сообщения и последовательность бит ключа.

1. Сначала производим перестановку обратную конечной перестановке в соответствии с таблицей IP-1.

2. Разделяем последовательность бит на левую (L, 0-31) и правую (R,32-63).

3. Из ключа переставляем биты в две последовательности C и D по 28 бит.

4. Генерируем все ключи Ki и сохраняем в вектор.

5. Производим 0 итерацию вне общего цикла:

* Вытаскиваем последний ключ Ki из вектора.
* Удаляем последний ключ Ki из вектора.
* Вычисляем последовательность f(R,Ki).
* Выполняем исключающее ИЛИ между L и f.

6. Запускаем цикл на оставшиеся 15 итераций:

* Сохраняем последовательность L.
* Вытаскиваем последний ключ Ki из вектора.
* Удаляем последний ключ Ki из вектора.
* Вычисляем последовательность f(L,Ki).
* Выполняем исключающее ИЛИ между R и f.
* Меняем местами L и R.

7. Выполняем перестановку обратную первичной перестановке в соответствии с таблицей IP.

**Функция шифрования f**: на вход получаем последовательность бит половины сообщения и ключ Ki.

1. Сначала производим расширение последовательности с 32 бит до 48 бит в соответствии с таблицей E.

2. Затем выполняем исключающее ИЛИ между получившейся последовательность E и ключом Ki.

3. Производим трансформацию S:

* Производим разбиение получившейся последовательности EK на 8 блоков по 6 битов.
* Каждый блок мы преобразуем из 6 бит в 4 бита в зависимости от таблиц Si.
* Собираем последовательность S из восьми четырех битных блоков Si.

4. Над последовательностью S выполняем перестановку бит P в соответствии с таблицей P.

**Генерация ключей Ki:** на вход поступают две последовательности C и D.

1. Производим сдвиг влево на 1 или 2 бита (в зависимости от итерации) в последовательностях C и D.

2. Затем собираем одну последовательность CD из 56 бит.

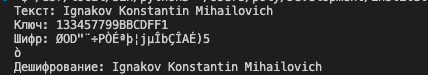
3. И переставляем биты в соответствии с таблицей H. Получается последовательность Ki из 48 бит (8 бит мы не задействуем).

**Примеры шифрования/расшифровывания.**

1. Нижегородский государственный технический университет:



2. Ignakov Konstantin Mihailovich:



**Текст программы.**

ip = [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,

56, 48, 40, 32, 24, 16, 8, 0,

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

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

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

]

expansion\_table = [

31, 0, 1, 2, 3, 4,

3, 4, 5, 6, 7, 8,

7, 8, 9, 10, 11, 12,

11, 12, 13, 14, 15, 16,

15, 16, 17, 18, 19, 20,

19, 20, 21, 22, 23, 24,

23, 24, 25, 26, 27, 28,

27, 28, 29, 30, 31, 0

]

pc1 = [56, 48, 40, 32, 24, 16, 8,

0, 57, 49, 41, 33, 25, 17,

9, 1, 58, 50, 42, 34, 26,

18, 10, 2, 59, 51, 43, 35,

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

6, 61, 53, 45, 37, 29, 21,

13, 5, 60, 52, 44, 36, 28,

20, 12, 4, 27, 19, 11, 3

]

left\_rotations = [

1, 1, 2, 2, 2, 2, 2, 2, 1, 2, 2, 2, 2, 2, 2, 1

]

pc2 = [13, 16, 10, 23, 0, 4,

2, 27, 14, 5, 20, 9,

22, 18, 11, 3, 25, 7,

15, 6, 26, 19, 12, 1,

40, 51, 30, 36, 46, 54,

29, 39, 50, 44, 32, 47,

43, 48, 38, 55, 33, 52,

45, 41, 49, 35, 28, 31

]

fp = [

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,

32, 0, 40, 8, 48, 16, 56, 24

]

sbox = [

# S1

[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],

# S2

[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],

# S3

[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],

# S4

[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],

# S5

[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],

# S6

[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],

# S7

[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],

# S8

[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],

]

p = [

15, 6, 19, 20, 28, 11,

27, 16, 0, 14, 22, 25,

4, 17, 30, 9, 1, 7,

23, 13, 31, 26, 2, 8,

18, 12, 29, 5, 21, 10,

3, 24

]

def key\_generation(key\_64):

subkeys = []

key\_draft = []

c0 = []

d0 = []

key\_64\_p = [key\_64[pc1[i]] for i in range(56)]

c0 = key\_64\_p[0:28]

d0 = key\_64\_p[28:56]

for j in range(0, 16):

for i in range(left\_rotations[j]):

c1 = c0[1:] + c0[:1]

d1 = d0[1:] + d0[:1]

c0 = c1[:]

d0 = d1[:]

tab\_pc2 = c1 + d1

res\_pc2 = [tab\_pc2[pc2[i]] for i in range(48)]

subkeys.append(res\_pc2)

return subkeys

# f applies on function R-1(32bits) and K1(48bits) ,its output is in 32bits

def f(R, K):

# Expansion D-box 32bits => 48 bits

R48 = list(range(48))

for i in range(48):

R48[i] = R[expansion\_table[i]]

# Whitener R48 XOR K

after\_XOR = [R48[j] ^ K[j] for j in range(48)]

# S-boxes 8 S-boxes of with 6bits as input

after\_sbox = []

for j in range(8):

Sixbits = after\_XOR[(j \* 6):(j + 1) \* 6]

bits\_1\_6 = int(str(Sixbits[0]) + str(Sixbits[5]), 2)

bits\_2\_5 = int(str(Sixbits[1]) + str(Sixbits[2]) + str(Sixbits[3]) + str(Sixbits[4]), 2)

found\_int = sbox[j][bits\_1\_6 \* 16 + bits\_2\_5]

after\_sbox += [int(i) for i in bin(found\_int)[2:].zfill(4)]

result = after\_sbox[:]

for i in range(32):

result[i] = after\_sbox[p[i]]

return result

def DES(plaintext\_64, key\_64):

subkeys = key\_generation(key\_64)

iptext = plaintext\_64[:]

for i in range(64):

iptext[i] = plaintext\_64[ip[i]]

L = iptext[:32]

R = iptext[32:64]

for i in range(16):

f\_result = f(R, subkeys[i])

C = R[:]

R = [L[q] ^ f\_result[q] for q in range(32)]

L = C[:]

res = R + L

fptext = res[

:] # becareful to the assignment here : fptext=res => it means passing by refernce and fptext=res[:] it means copy by value

for i in range(64):

fptext[i] = res[fp[i]]

return fptext

def DES\_decrypt(ciphertext, key\_64):

initial\_permut = [ciphertext[ip[i]] for i in range(64)]

L = initial\_permut[:32]

R = initial\_permut[32:64]

subkeys = key\_generation(key\_64)

for i in range(15, -1, -1):

f\_result = f(R, subkeys[i])

C = R[:]

R = [L[q] ^ f\_result[q] for q in range(32)]

L = C[:]

# ipdb.set\_trace()

res = R + L

final\_result = [res[fp[i]] for i in range(64)]

return final\_result

def CBC\_DES\_ENC(IV, text,

key\_hexa): # IV : is a 64 bits binary vector and text : is a string of characteres, key\_hexa string of HEX chars

key\_binary = [int(i, 2) for i in bin(int('0x' + key\_hexa, 16))[2:].zfill(64)]

binary\_ciphertext\_array = []

binary\_text\_array = []

binary\_array\_of\_char = [int(i, 2) for i in bin(ord('a'))[2:].zfill(8)]

for i in text:

binary\_text\_array += [int(i, 2) for i in bin(ord(i))[2:].zfill(8)]

block\_number = len(binary\_text\_array) // 64

last\_block\_size = len(binary\_text\_array) % 64

# if(last\_block\_size!=0) : block\_number+=1

init\_vect = IV[:]

cypher\_text = []

# ipdb.set\_trace() # debuging

for i in range(block\_number):

block = binary\_text\_array[i \* 64:(i + 1) \* 64]

XOR\_res = [int(init\_vect[j]) ^ block[j] for j in range(64)]

cipher\_block = DES(XOR\_res, key\_binary)

cypher\_text += cipher\_block

init\_vect = cipher\_block[:]

if (last\_block\_size != 0):

block\_number = block\_number + 1

last\_block = binary\_text\_array[-last\_block\_size:] + [0] \* (64 - last\_block\_size)

XOR\_res = [init\_vect[i] ^ last\_block[i] for i in range(64)]

cipher\_block = DES(XOR\_res, key\_binary)

cypher\_text += cipher\_block

# ipdb.set\_trace()

# cypher\_text\_hex = hex(int(''.join(map(str, cypher\_text)), 2))[2:].zfill(block\_number\*16)

cypher\_text\_ascii\_string = [chr(int(

str(cypher\_text[i]) + str(cypher\_text[i + 1]) + str(cypher\_text[i + 2]) + str(cypher\_text[i + 3]) + str(

cypher\_text[i + 4]) + str(cypher\_text[i + 5]) + str(cypher\_text[i + 6]) + str(cypher\_text[i + 7]), 2)) for i

in range(0, len(cypher\_text) - 1, 8)]

return cypher\_text\_ascii\_string # cypher\_text\_ascii\_string : string (a list) of ascii char

def CBC\_DES\_DEC(IV, cyphertext,

key\_hexa): # cyphertext : is string (or list of chars) ,IV 64 bits binary vector , key\_hexa a string of HEX chars

key\_binary = [int(i, 2) for i in bin(int('0x' + key\_hexa, 16))[2:].zfill(64)]

binary\_ciphertext\_array = []

for i in cyphertext:

binary\_ciphertext\_array += [int(i, 2) for i in bin(ord(i))[2:].zfill(8)]

block\_number = len(binary\_ciphertext\_array) // 64

last\_block\_size = len(binary\_ciphertext\_array) % 64

# ipdb.set\_trace()

init\_vect = IV[:]

text = []

for i in range(block\_number):

block = binary\_ciphertext\_array[i \* 64:(i + 1) \* 64]

text\_block = DES\_decrypt(block, key\_binary)

XOR\_res = [int(init\_vect[i]) ^ text\_block[i] for i in range(64)]

text += XOR\_res

init\_vect = block[:]

text\_ascii\_string = [chr(int(

str(text[i]) + str(text[i + 1]) + str(text[i + 2]) + str(text[i + 3]) + str(text[i + 4]) + str(

text[i + 5]) + str(text[i + 6]) + str(text[i + 7]), 2)) for i in range(0, len(text) - 1, 8)]

return text\_ascii\_string # text\_ascii\_string : decrypted string (a list of ASCII chars)

def main():

IV = [0, 1, 1, 0, 0, 0, 1, 1, 0, 1, 1, 0, 0, 0, 1, 1, 0, 1, 1, 0, 0, 0, 1, 1, 0, 1, 1, 0, 0, 0, 1, 1, 0, 1, 1, 0, 0, 0, 1,

1, 0, 1, 1, 0, 0, 0, 1, 1, 0, 1, 1, 0, 0, 0, 1, 1, 0, 1, 1, 0, 0, 0, 1, 1, 0, 1, 1, 0, 0, 0, 1, 1]

key = '133457799BBCDFF1'

text = 'Nizhegorodskiy Gosudarstvenniy Politehnicheskii Univarsitet'

print(f"Текст: {text}")

print(f"Ключ: {key}")

cyphertext = CBC\_DES\_ENC(IV, text, key)

print('Шифр:', ''.join(cyphertext))

print('Дешифрование:', ''.join(CBC\_DES\_DEC(IV, cyphertext, key)))

if \_\_name\_\_ == '\_\_main\_\_':

main()