

Министерство науки и высшего образования Российской Федерации
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
«Национальный исследовательский университет ИТМО»
(Университет ИТМО)

Факультет безопасности информационных технологий
Дисциплина: «Криптографические методы защиты информации»

О Т Ч Е Т
Лабораторная работа № 2
«Основные структурные элементы алгоритма DES»

Выполнил студент группы N3351:

Ярьсько С.А.



Проверил к.т.н., доцент ФБИТ:

Таранов С.В.

Дата:

Санкт-Петербург, 2020

Цель работы:

Изучить основные принципы работы алгоритмы DES.

Задачи работы:

1. Выполнение 1 цикла раундовой функции алгоритма DES вручную (т.е. выполнение всех функций, входящих в раундовую функцию DES, для фиксированного входного двоичного вектора с отображением промежуточных значений шифрования) или программная реализация 1 раунда (или полной системы) DES.
2. Анализ визуализации алгоритма DES и примитивных атак на шифр, используя Cryptool 2.

Ход работы:

Часть 1.

Демонстрация работы программной реализации полного процесса шифрования для 1 раунда DES с генерацией 64 битного блока, начальной перестановкой IP, выполнением 1 раунда функции расширения ключа.

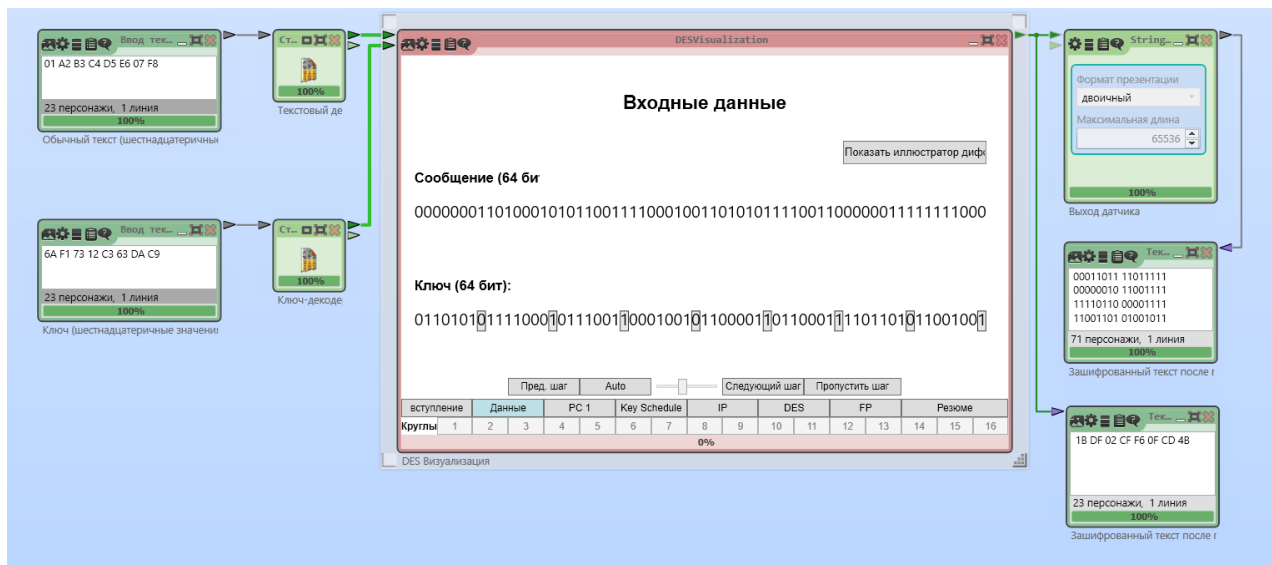
Алгоритм работы программы:

1. Генерирование 64-бит сообщения и 56-бит ключа.
2. Начальная перестановка IP.
3. Разделение сообщения на 2 части L0 и R0.
4. Расширение ключа до 64-бит путем добавления бит в позиции каратные 8.
5. Формирование блоков C0 и D0 из 64-бит ключа.
6. Циклический левый сдвиг блоков на 1 бит (для 1 раунда).
7. Формирование 48-бит ключа путем выборки из вектора C1D1.
8. Расширение “E” сообщения R0 до 48-бит сообщения E.
9. Сложение E с ключом по модулю 2.
10. Операция преобразования “S” при помощи S-box.
11. Перестановка P.
12. Получение сообщений L1 и R1 для 2 раунда DES.

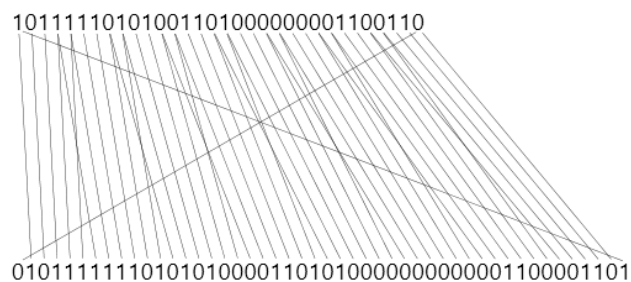
Демонстрация работы программного модуля:

Сообщение T (64 бит)	0000000110100010101100111100010011010101111001100000011111111000
Ключ k (56 бит)	01101011111000011100100010011100001011000111011011100100
Сообщение IP (64 бит)	10111000100101000111100001010110111110101001101000000001100110
Сообщение L0 (32 бит)	101110001001010001111000010101
Сообщение R0 (32 бит)	10111110101001101000000001100110
Ключ k0 (64 бит)	0110101011110000011100100001001011000010011000101101101011001000
Блок C0 (28 бит)	1101001011110111001001110100
Блок D0 (28 бит)	0111110100000000110000011110
Блок C1 (28 бит)	1010010111101110010011101001
Блок D1 (28 бит)	1111101000000001100000111100
Ключ k1 (48 бит)	10101011111100001000010011000101110100001000011
Расширение E (48 бит)	01011111101010100001101010000000000001100001101
XOR с ключом (48 бит)	['111101', '000010', '110101', '001111', '001000', '101110', '101101', '001110']
Операция S-box (32 бит)	['0110', '0001', '1110', '0011', '0111', '0011', '1010', '0001']
Перестановка P (32 бит)	10100000011001011110111110000101
Сообщение L1 (32 бит)	10111110101001101000000001100110
Сообщение R1 (32 бит)	00011000111100011001011111010000

Часть 2. Визуализация 1 раунда алгоритма DES.



Expansion



Побитовая операция XOR

K1: 101010111111100001000010011000101110100001000011



Exp: 01011111110101010000110101000000000001100001101

111101000010110101001111001000101110101101001110

S-коробки

111101 000010 110101 001111 001000 101110 101101 001110

S8	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
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

Входные данные: 001110

Ряд: 0__0 \triangleq 0

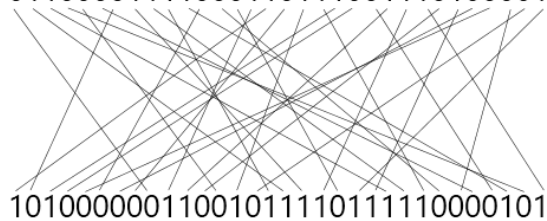
Колонка: _0111_ \triangleq 7

Вывод: 0001 \triangleq 1

Вывод всех S-блоков: 01100001111000110111001110100001

Функция перестановки

01100001111000110111001110100001

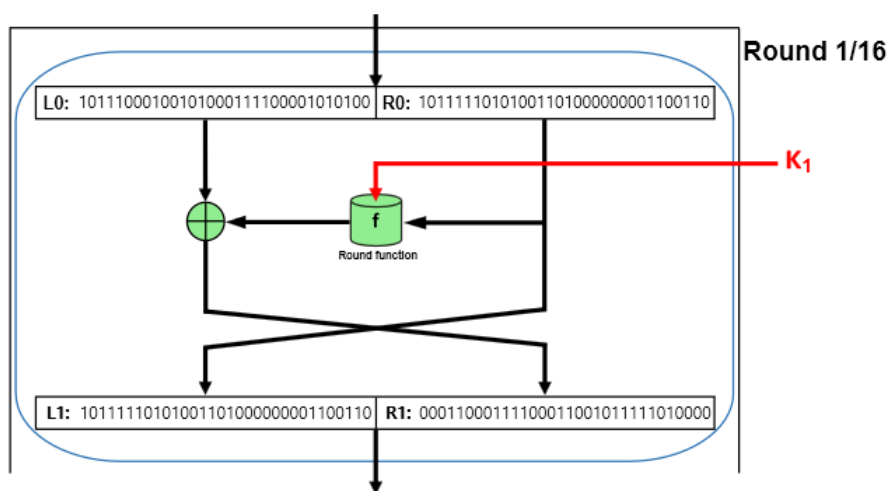


Bitwise XOR Operation

L0: 10111000100101000111100001010100

\oplus

f(R0): 10100000011001011110111110000101



Часть 2. Визуализация «лавинного эффекта» DES.

The screenshot displays the 'Avalanche Visualization' software interface. On the left, there are two input boxes: 'DES Plaintext' containing 'BEA68067' and 'DES Key (hex)' containing '6A F1 73 12 C3 63 DA C9'. Both are decoded into binary strings. The main window, titled 'Show Changed Input Data DES', shows the 'Initial message' and 'Initial key' in binary. The 'Modified message' and 'Modified key' are also shown. Below these, there are radio buttons for 'Display input as: Binary', 'Decimal', and 'Hexadecimal'. A progress bar at the bottom indicates '5%'. On the right, a 'Text Output' window shows the initial DES message and key, and the modified DES message and key. It also displays the avalanche effect statistics for the first round: 'After round 0: Flipped bits: 1. Avalanche effect: 1.6%, Length of longest identical bit sequence: 39, Offset: 25, Length of longest flipped bit sequence: 1, Offset: 24. After round 1: Flipped bits: 1. Avalanche effect: 1.6%, Length of longest identical bit sequence: 56, Offset: 0, Length of longest flipped bit sequence: 1, Offset: 56. After round 2: Flipped bits: 6. Avalanche effect: 9.4%, 3,262 characters, 99 lines'.

DES Plaintext: BEA68067 (8 characters, 1 line, 100%)

DES Key (hex): 6A F1 73 12 C3 63 DA C9 (23 characters, 1 line, 100%)

String Decode: 100%

Initial message: 01000010010001010100000100110110
00111000001100000011011000110110

Modified message: 01000010010001010100000100110110
00111000001100000011011000110111

Initial key: 01101010111100010111001100010010
11000011011000111101101011001001

Modified key: 01101010111100010111001100010010
11000011011000111101101011001001

Display input as: ☒ Binary ☐ Decimal ☐ Hexadecimal

Check avalanche effect after round ... 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 Instructions General Overview

5%

Text Output: Initial DES message: BEA68066, Initial DES key: 6A F1 73 12 C3 63 DA C9, Modified DES message: BEA68067, DES key not modified. After round 0: Flipped bits: 1. Avalanche effect: 1.6%, Length of longest identical bit sequence: 39, Offset: 25, Length of longest flipped bit sequence: 1, Offset: 24. After round 1: Flipped bits: 1. Avalanche effect: 1.6%, Length of longest identical bit sequence: 56, Offset: 0, Length of longest flipped bit sequence: 1, Offset: 56. After round 2: Flipped bits: 6. Avalanche effect: 9.4%, 3,262 characters, 99 lines. Generated Data

The screenshot displays the 'Avalanche Visualization' software interface showing the results after all DES rounds. The main window, titled 'Encryption Results after All DES Rounds', shows a table of results for each round. The table has columns for 'Round', 'Left half', 'Right half', and '% changed'. The results show that the avalanche effect increases significantly over the rounds, reaching 51.6% by round 16. The 'Text Output' window shows the final DES ciphertext derived from the initial input: 'B4 25 65 A5 AD E9 B4 CA' and the final DES ciphertext derived from the modified input: '72 1B 86 2B EB 95 9E 79'. It also displays the avalanche effect statistics for the final round: 'After round 16: Flipped bits: 33. Avalanche effect: 51.6%, Length of longest identical bit sequence: 3, Offset: 1, Length of longest flipped bit sequence: 4, Offset: 7. Final DES ciphertext derived from the initial input: B4 25 65 A5 AD E9 B4 CA, Final DES ciphertext derived from the modified input: 72 1B 86 2B EB 95 9E 79. 3,262 characters, 99 lines. Generated Data

DES Plaintext: BEA68067 (8 characters, 1 line, 100%)

DES Key (hex): 6A F1 73 12 C3 63 DA C9 (23 characters, 1 line, 100%)

String Decode: 100%

Encryption Results after All DES Rounds

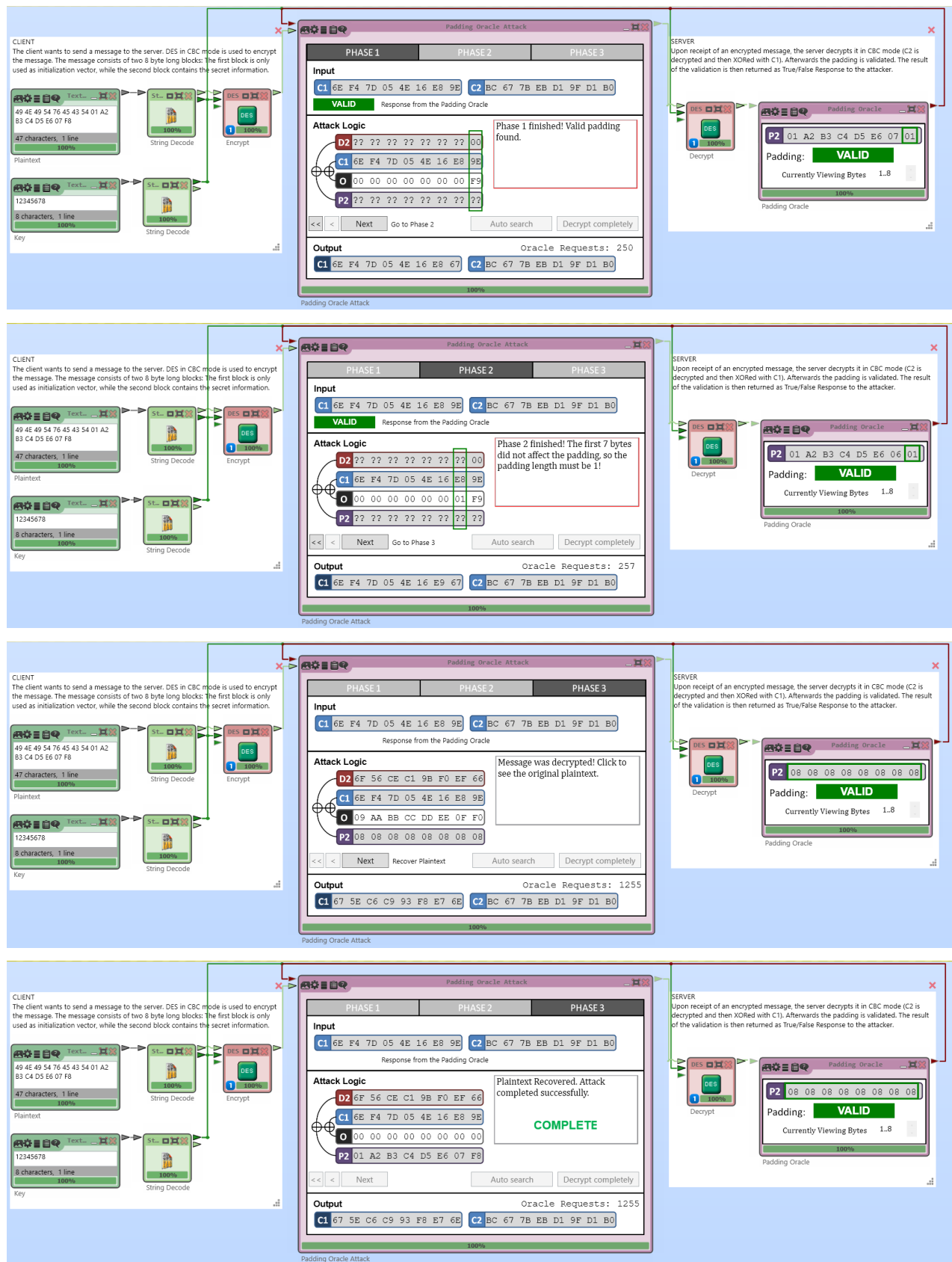
Round	Left half	Right half	% changed
0	00000111111110001001010100001110	0000000011111000001000011001000	1.6 %
1	0000000011111000001000011001000	110010110110001001000010111000	1.6 %
2	110010110110001001000010111000	001000010011110000111101010000	9.4 %
3	001000010011110000111010100000	100010011100000010000001011110	32.8 %
4	10001001110000001001000001011110	011001110101000000100100010000	50 %
5	011001101010000000100100010000	100100101010010010011010101000	48.4 %
6	1001001010100100001011010101000	00010010100100000101110011001100	46.9 %
7	00010010101000000101110011001100	11001001011100011001001000011000	50 %
8	11001001011100011001010000100100	11101011100011101000001010011110	53.1 %
9	11101011100011101000001010011110	001100001100100000101101010000	50 %
10	0011000011001000001010101000100	001101001111110010111100101010	45.3 %
11	001101001111111001011111001010	00111000111111000000001101000	53.1 %
12	001111000111110000000001101000	001001001010110000010011110110	56.3 %
13	001001001010110000010011110110	001011110100111101011000011100	50 %
14	001011110100111101010000111100	000000010100011101111101000010	40.6 %
15	000000010100011101110100001010	011101001001001101101001001110	43.8 %
16	011101001001001101101001001110	100100011110001101100100101010	51.6 %

Check avalanche effect after round ... 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 Instructions General Overview

100%

Text Output: After round 14: Flipped bits: 26. Avalanche effect: 40.6%, Length of longest identical bit sequence: 6, Offset: 54, Length of longest flipped bit sequence: 4, Offset: 24. After round 15: Flipped bits: 28. Avalanche effect: 43.8%, Length of longest identical bit sequence: 6, Offset: 22, Length of longest flipped bit sequence: 4, Offset: 39. After round 16: Flipped bits: 33. Avalanche effect: 51.6%, Length of longest identical bit sequence: 3, Offset: 1, Length of longest flipped bit sequence: 4, Offset: 7. Final DES ciphertext derived from the initial input: B4 25 65 A5 AD E9 B4 CA, Final DES ciphertext derived from the modified input: 72 1B 86 2B EB 95 9E 79. 3,262 characters, 99 lines. Generated Data

Часть 2. Проведение Padding Oracle атаки на DES.



Вывод:

В результате данной лабораторной работы были изучены основные принципы работы алгоритмы DES, программно реализован полный процесс шифрования для 1 раунда с генерацией 64 битного блока, начальной перестановкой IP, выполнением 1 раунда функции расширения ключа. Был выполнен анализ визуализации алгоритма DES, продемонстрирован лавинный эффект, проведена padding oracle атака на DES.

Приложение 1. Листинг программного кода.

```
import random

# msg_T = '000000011010001010110011110001001101010111100110000001111111000'
# key56 = '01101011111000011100100010011100001011000111011011100100'
msg_T = key56 = ''
for i in range(64):
    msg_T += str(round(random.random()))
for i in range(56):
    key56 += str(round(random.random()))
print('Сообщение T (64 бита) ', msg_T)
print('Ключ k (56 бит) ', key56, '\n')

# Начальная перестановка
msg_IP = ''
msg_IP_list = [
    (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)
]
for i in range(len(msg_IP_list)):
    msg_IP += msg_T[msg_IP_list[i]-1]
print('Сообщение IP (64 бита) ', msg_IP)

# Разделение на L0 и R0
L_list = []
R_list = []
L_list.append(msg_IP[:32])
R_list.append(msg_IP[32:])
print('Сообщение L0 (32 бита) ', L_list[0])
print('Сообщение R0 (32 бита) ', R_list[0], '\n')

# Расширение ключа
key64 = ''
for i in range(0, len(key56), 7):
    for n in range(7):
        key64 += key56[i+n]
    key64 += '0'
print('Ключ k0 (64 бита) ', key64)

# Формирование блоков C0 и D0
C_list = (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)
D_list = (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)
msg_C = ''
msg_D = ''
for i in range(len(C_list)):
    msg_C += key64[C_list[i]-1]
    msg_D += key64[D_list[i]-1]
print('Блок C0 (28 бита) ', msg_C)
print('Блок D0 (28 бита) ', msg_D)

# Циклический сдвиг
msg_C += msg_C[0]
msg_C = msg_C[1:]
msg_D += msg_D[0]
msg_D = msg_D[1:]
msg_CD = msg_C + msg_D
print('Блок C1 (28 бита) ', msg_C)
print('Блок D1 (28 бита) ', msg_D)

# Получение ключа 48 бит
key = ''
key_list = [
    (14,17,11,24,1,5,3,28,15,6,21,10,23,19,12,4,26,8,16,7,27,20,13,2,41,52,31,37,47,55,30,40,51,45,33,48,44,4,
    9,39,56,34,53,46,42,50,36,29,32)
]
for i in range(len(key_list)):
    key += msg_CD[key_list[i]-1]
print('Ключ k1 (48 бит) ', key, '\n')

# Расширение E
msg_E = ''
expansion_E_list = [
    (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)
]
for i in range(len(expansion_E_list)):
    msg_E += R_list[0][expansion_E_list[i]-1]
print('Расширение E (48 бита) ', msg_E)

# Сложение с ключом
msg_B = []
msg_temp = ''
for i in range(len(msg_E)):
    msg_temp += str((int(msg_E[i])+int(key[i]))%2)
    if (i+1)%6 == 0:
        msg_B.append(msg_temp)
        msg_temp = ''
print('XOR с ключом (48 бита) ', msg_B)

# Преобразование (Операция подстановки S-box)
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),
```

```

s2 = ((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, 14, 4, 1, 7, 10, 11, 6, 8, 5, 3, 15, 13, 0, 9),
      (4, 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))

a = b = 0
msg_Bs = []
s_list = (s1, s2, s3, s4, s5, s6, s7, s8)
for i in range(len(msg_B)):
    a = int(str(msg_B[i][0])+str(msg_B[i][5]), 2)
    b = int(str(msg_B[i][1])+str(msg_B[i][2])+str(msg_B[i][3])+str(msg_B[i][4]), 2)
    msg_Bs.append(format(s_list[i][a][b], '04b'))
print('Операция S-box (32 бит)', msg_Bs)
msg_temp = msg_Bs
msg_Bs = ''
for elem in msg_temp:
    msg_Bs += elem

# Перестановка P
msg_P = ''
permutation_P_list =
(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)
for i in range(len(permutation_P_list)):
    msg_P += msg_Bs[permutation_P_list[i]-1]
print('Перестановка P (32 бит)', msg_P, '\n')

# Получение L1 и R1
L_list.append(R_list[0])
msg_temp = ''
for i in range(len(R_list[0])):
    msg_temp += str((int(L_list[0][i])+int(msg_P[i]))%2)
R_list.append(msg_temp)
print('Сообщение L1 (32 бит)', L_list[1])
print('Сообщение R1 (32 бит)', R_list[1])

```