Учреждение образования

«БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНОЛОГИЧЕСКИЙ УНИВЕРСИТЕТ»

Факультет информационных технологий

Кафедра информационных систем и технологий

**Лабораторная работа № 7**

«ИССЛЕДОВАНИЕ БЛОЧНЫХ ШИФРОВ»

Выполнил:

Студент: Дащинский М.Л..

ФИТ 3 курс 4 группа

Преподаватель: Сазонова Д.В.

Минск 2023

1. **Описание приложения**

Приложение написано на языке программирования Python и позволяет провести шифрование блочным шифром.

1. **Методика выполнения расчетов**

В данной лабораторной работе была поставлена цель создания приложения, которое реализовывает алгоритм DES. На листинге 2.1 представлены функции, реализующие данную функциональность, а на листинге 2.2 предствавлены функции для преобразование ключа.

def ReadFile(name\_file):

    file\_for\_only\_read = open(name\_file, 'r', encoding="utf8")

    return file\_for\_only\_read.read()

def GetMessageInTheNumberSystem(message, number\_system):

    match number\_system:

        case 2:

            message\_in\_system = ''.join(['0' \* (8 - len(bin(ord(i))[2:])) + bin(ord(i))[2:].upper() for i in message])

        case 16:

            message\_in\_system = ''.join(['0' \* (2 - len(hex(ord(i))[2:])) + hex(ord(i))[2:].upper() for i in message])

        case \_:

            return message

    return message\_in\_system

def SplitMesssageIntoBlocks(message\_binary):

    blocks\_4x16 = []

    for i in range(len(message\_binary) // 64):

        blocks\_4x16.append(message\_binary[i \* 64:(i + 1) \* 64])

    return blocks\_4x16

def PermutationBlock(blocks, number\_permutation=0):

    table\_permutation = {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,

                             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],

                         1: [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]}

    blocks\_permutation = []

    for block in blocks:

        current\_block = ''

        for j in range(64):

            current\_block += block[table\_permutation[number\_permutation][j] - 1]

        blocks\_permutation.append(current\_block)

    return blocks\_permutation

def SplitBlockIntoBlocksLR(blocks\_4x16\_permutation):

    blocks\_l\_r = []

    for block in blocks\_4x16\_permutation:

        blocks\_l\_r.append((block[0:32], block[32:]))

    return blocks\_l\_r

def PBoxExtension(blocks\_l\_32\_r\_32, number\_extension=0):

    table\_extension = {0: [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]}

    blocks\_l\_32\_r\_48 = []

    for block in blocks\_l\_32\_r\_32:

        current\_block = ''

        for i in range(48):

            current\_block += block[1][table\_extension[number\_extension][i] - 1]

        blocks\_l\_32\_r\_48.append((block[0], current\_block))

    return blocks\_l\_32\_r\_48

def XORRoundKeys(blocks\_l\_32\_r\_48, round\_keys\_, round\_):

    xor\_block = ''

    for index, block in enumerate(blocks\_l\_32\_r\_48):

        for i in range(len(block[1])):

            if block[1][i] == round\_keys\_[round\_][i]:

                xor\_block += '0'

            else:

                xor\_block += '1'

        blocks\_l\_32\_r\_48[index] = (blocks\_l\_32\_r\_48[index][0], xor\_block)

        xor\_block = ''

    return blocks\_l\_32\_r\_48

def XOR(number\_list\_tuple):

    xor\_list\_number = []

    for block in number\_list\_tuple:

        current\_xor = ''

        for i in range(len(block[0])):

            if block[0][i] == block[1][i]:

                current\_xor += '0'

            else:

                current\_xor += '1'

        xor\_list\_number.append((block[0], current\_xor))

    return xor\_list\_number

def SBoxes(blocks\_l\_32\_r\_48\_XOR):

    table\_s\_boxes = {0: [[14, 4, 13, 1, 2, 15, 11, 8, 3, 10, 6, 12, 6, 7, 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]],

                     1: [[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]],

                     2: [[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]],

                     3: [[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]],

                     4: [[2, 12, 4, 1, 7, 10, 11, 6, 8, 5, 3, 14, 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]],

                     5: [[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]],

                     6: [[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, 13, 10, 15, 6, 8, 0, 5, 9, 2],

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

                     7: [[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]]}

    blocks\_split = []

    for index\_, block in enumerate(blocks\_l\_32\_r\_48\_XOR):

        for i in range(8):

            blocks\_split.append(block[1][i \* 6:(i+1) \* 6])

        block\_after\_permutation = ''

        for index, block\_after\_split in enumerate(blocks\_split):

            row = int(block\_after\_split[0] + block\_after\_split[5], 2)

            col = int(block\_after\_split[1:5], 2)

            current\_block = (4 - len(bin(table\_s\_boxes[index][row][col])[2:])) \* '0' + \

                            bin(table\_s\_boxes[index][row][col])[2:]

            block\_after\_permutation += current\_block

        blocks\_l\_32\_r\_48\_XOR[index\_] = (blocks\_l\_32\_r\_48\_XOR[index\_][0], block\_after\_permutation)

        blocks\_split = []

    return blocks\_l\_32\_r\_48\_XOR

def PBoxDirect(blocks\_l\_32\_r\_48\_XOR\_S, number\_permutation=0):

    table\_permutation = {0: [16, 7, 20, 21, 29, 12, 28, 17,

                             1, 15, 23, 26, 5, 18, 31, 19,

                             2, 8, 24, 14, 32, 27, 3, 9,

                             19, 13, 30, 6, 22, 11, 4, 25]}

    for index, block in enumerate(blocks\_l\_32\_r\_48\_XOR\_S):

        block\_after\_permutation = ''

        for i in range(len(block[1])):

            block\_after\_permutation += block[1][table\_permutation[number\_permutation][i] - 1]

        blocks\_l\_32\_r\_48\_XOR\_S[index] = (blocks\_l\_32\_r\_48\_XOR\_S[index][0], block\_after\_permutation)

    return blocks\_l\_32\_r\_48\_XOR\_S

def EncryptAndDecrypt(message\_2, round\_keys\_):

*# print(message\_2)*

    blocks\_4x16 = SplitMesssageIntoBlocks(message\_2)

*# print(blocks\_4x16)*

    blocks\_4x16\_permutation = PermutationBlock(blocks\_4x16, number\_permutation=0)

*# print(blocks\_4x16\_permutation)*

    blocks\_l\_32\_r\_32 = SplitBlockIntoBlocksLR(blocks\_4x16\_permutation)

*# print(blocks\_l\_32\_r\_32)*

*# Прохождение 16 раундов*

    for i in range(16):

        blocks\_r\_32 = [block\_[1] for block\_ in blocks\_l\_32\_r\_32]

*# print(blocks\_r\_32)*

        blocks\_l\_32\_r\_48\_P = PBoxExtension(blocks\_l\_32\_r\_32)

*# print(blocks\_l\_32\_r\_48\_P)*

        blocks\_l\_32\_r\_48\_P\_XOR = XORRoundKeys(blocks\_l\_32\_r\_48\_P, round\_keys\_, i)  *# Работает правильно*

*# print(blocks\_l\_32\_r\_48\_P\_XOR)*

        blocks\_l\_32\_r\_48\_P\_XOR\_S = SBoxes(blocks\_l\_32\_r\_48\_P\_XOR)  *# Работает правильно*

*# print(blocks\_l\_32\_r\_48\_P\_XOR\_S)*

        blocks\_l\_32\_r\_48\_P\_XOR\_S\_P = PBoxDirect(blocks\_l\_32\_r\_48\_P\_XOR\_S)  *# Работает правильно*

*# print(blocks\_l\_32\_r\_48\_P\_XOR\_S\_P)*

        blocks\_l\_32\_r\_48\_P\_XOR\_S\_P = XOR(blocks\_l\_32\_r\_48\_P\_XOR\_S\_P)

*# print(blocks\_l\_32\_r\_48\_P\_XOR\_S\_P)*

        for index, block in enumerate(blocks\_l\_32\_r\_48\_P\_XOR\_S\_P):  *# Работает правильно*

            blocks\_l\_32\_r\_32[index] = (blocks\_r\_32[index], block[1])

*# print(blocks\_l\_32\_r\_32)*

*# print()*

    blocks\_l\_32\_r\_32 = PermutationBlock([block[1] + block[0] for block in blocks\_l\_32\_r\_32], number\_permutation=1)

*# print(blocks\_l\_32\_r\_32)*

    encrypt\_message = ''.join([block for block in blocks\_l\_32\_r\_32])

    return encrypt\_message

Листинг 2.1 –код программы, реализующий алгоритм DES

ef MakeBinaryKey(key\_message):

    key\_64 = ''.join([(8 - len(bin(ord(i))[2:])) \* '0' + bin(ord(i))[2:] for i in key\_message])

    if len(key\_64) != 64:

        key\_64 = key\_64[0:64]

        key\_64 += '0' \* (64 - len(key\_64))

    return key\_64

def RearrangeKeyBits(key\_64, number\_permutation=0):

    table\_permutation = {0: [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\_rearranged\_56 = []

    for i in range(len(table\_permutation[number\_permutation])):

        key\_rearranged\_56.append(key\_64[table\_permutation[number\_permutation][i] - 1])

    return ''.join(key\_rearranged\_56)

def SplitBlockIntoTwoBlocks(key\_56):

    key\_c = key\_56[0:28]

    key\_d = key\_56[28:]

    return key\_c, key\_d

def GetRoundKeys(key\_c, key\_d):

    shifts = [1, 1, 2, 2, 2, 2, 2, 2, 1, 2, 2, 2, 2, 2, 2, 1]

    current\_key\_c = key\_c[:]

    current\_key\_d = key\_d[:]

    round\_keys = []

    for i in range(len(shifts)):

        current\_key\_c = current\_key\_c[shifts[i]:] + current\_key\_c[0:shifts[i]]

        current\_key\_d = current\_key\_d[shifts[i]:] + current\_key\_d[0:shifts[i]]

        round\_keys.append((current\_key\_c, current\_key\_d))

    return round\_keys

def RearrangingCompression(keys\_round\_c\_d, number\_permutation=0):

    table\_permutation = {0: [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]}

    keys\_round\_ = []

    for i in range(len(keys\_round\_c\_d)):

        current\_keys\_c\_d = keys\_round\_c\_d[i][0] + keys\_round\_c\_d[i][1]

        key\_48 = ''

        for j in range(48):

            key\_48 += current\_keys\_c\_d[table\_permutation[number\_permutation][j] - 1]

        keys\_round\_.append(key\_48)

    return keys\_round\_

def PrepareRoundKeys(key\_message):

    key\_64 = MakeBinaryKey(key\_message)

    key\_56 = RearrangeKeyBits(key\_64=key\_64)

    key\_c, key\_d = SplitBlockIntoTwoBlocks(key\_56)

    keys\_round\_c\_d = GetRoundKeys(key\_c, key\_d)

    keys\_round\_ = RearrangingCompression(keys\_round\_c\_d)

    print(f'Ключевое слово: {key\_message}\n'

          f'64 битовый ключ: {key\_64}\n'

          f'56-битовый ключ: {key\_56}\n'

          f'Левый 28-битный блок C0: {key\_c}\n'

          f'Правый 28-битный блок D0: {key\_d}\n'

          f'Список раундовых 28-битных C0 и D0 ключей: {keys\_round\_c\_d}\n'

          f'Список раундовых 48-битовых ключей: {keys\_round\_}\n')

    return keys\_round\_

Листинг 2.2 –код программы, реализующий преобразование ключа

**3. Результаты работы приложения**

Для выполнения расчетов достаточно необходимо запустить приложение и указать файл текста. Рисунок 3.1, 3.2, 3.3, 3.4 и 3.5 показывают требуемые в данной лабораторной работе результаты.



Рисунок 3.1 – Результат работы по алгоритму DES



Рисунок 3.2 – Результат работы по алгоритму DES (продолжение)



Рисунок 3.3 – Результат работы по алгоритму DES (продолжение)

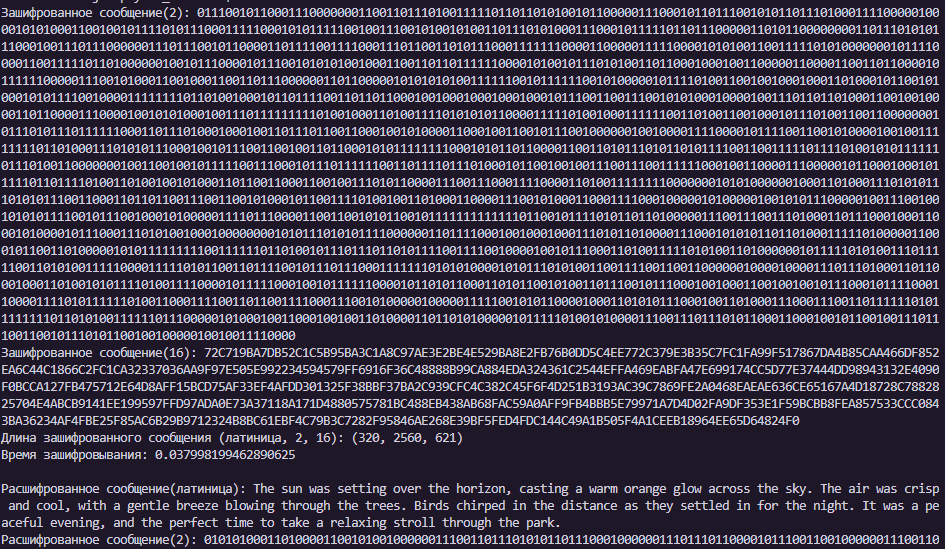


Рисунок 3.4 – Результат работы по алгоритму DES (продолжение)



Рисунок 3.5 – Результат работы по алгоритму DES (продолжение)

**Вывод**

В ходе лабораторной работы было разработано приложение для шифрование по алгоритму DES.