Министерство образования и науки Российской Федерации  
ФГАОУ ВПО «Уральский федеральный университет имени первого  
Президента России Б.Н. Ельцина»

Научно-учебный центр «Информационная Безопасность»

Отчет

По дисциплине «Криптографические методы защиты информации»

Лабораторная работа №4

Подпись ФИО

Преподаватель \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ А.В. Агафонов

Студенты \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Л.В. Родионов

И.А. Ибрагимов

Группа РИ-481220

Екатеринбург 2022

Цели: изучить метод построения шифров с использованием сети Фейстеля

1. Изучить главу 8 пособия.

2. Создать программу, которая зашифровывает и расшифровывает файлы с использованием алгоритма DES.

3. Зашифровать и расшифровать с ее помощью текстовый файл.

4. Выполнить криптоанализ на основе частот появления символов и известного текста. Сделать вывод о причинах неудачи.

Ход работы:

В ходе выполнения работы была необходимость работать с множеством массивов, содержащих биты информации. Чтобы не запутаться и проверить правильность кодируемых последовательностей на каждом из промежуточных этапов, была использована методичка в формате pdf, прикрепленная к отчету.

Сценарий работы DES на ЯП Python выглядит следующим образом:

# Словарь шестнадцатеричной системы счисления, нужен для проверки значений на промежуточных этапах  
HEX = {0: '0', 1: '1', 2: '2', 3: '3', 4: '4', 5: '5', 6: '6', 7: '7', 8: '8', 9: '9', 10: 'A', 11: 'B', 12: 'C',  
 13: 'D', 14: 'E', 15: 'F'}  
# Матрицы S-блоков для преобразования 48-битового вектора к 32-битовому  
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]]  
  
  
# Функция перевода десятичного числа в массив битов  
def dec2bin(num: int, q: int):  
 bits = []  
 while num / 2 != 0:  
 bits.append(num % 2)  
 num = int(num / 2)  
 bits.reverse()  
 if len(bits) < q:  
 bits = [0] \* (q - len(bits)) + bits  
 return bits  
  
  
# Функция перевода массива с битами в десятичное число  
def bin2dec(bits: list):  
 count = 0  
 for i in range(len(bits)):  
 count += bits[i] \* 2 \*\* (len(bits) - i - 1)  
 return count  
  
  
# Начальная перестановка  
def IP(a: list):  
 return [a[57], a[49], a[41], a[33], a[25], a[17], a[9], a[1], a[59], a[51], a[43], a[35], a[27], a[19], a[11], a[3],  
 a[61], a[53], a[45], a[37], a[29], a[21], a[13], a[5], a[63], a[55], a[47], a[39], a[31], a[23], a[15],  
 a[7], a[56], a[48], a[40], a[32], a[24], a[16], a[8], a[0], a[58], a[50], a[42], a[34], a[26], a[18], a[10],  
 a[2], a[60], a[52], a[44], a[36], a[28], a[20], a[12], a[4], a[62], a[54], a[46], a[38], a[30], a[22],  
 a[14], a[6]]  
  
  
# Конечная перестановка  
def IP\_back(a: list):  
 return [a[39], a[7], a[47], a[15], a[55], a[23], a[63], a[31], a[38], a[6], a[46], a[14], a[54], a[22], a[62],  
 a[30], a[37], a[5], a[45], a[13], a[53], a[21], a[61], a[29], a[36], a[4], a[44], a[12], a[52], a[20],  
 a[60], a[28], a[35], a[3], a[43], a[11], a[51], a[19], a[59], a[27], a[34], a[2], a[42], a[10], a[50],  
 a[18], a[58], a[26], a[33], a[1], a[41], a[9], a[49], a[17], a[57], a[25], a[32], a[0], a[40], a[8], a[48],  
 a[16], a[56], a[24]]  
  
  
# Функция перевода массива двоичных чисел в шестнадцатеричный формат, нужна для контроля  
def bin2hex(a: list):  
 h = ''  
 for i in range(0, len(a),  
 4): # Берет по 4 элемента массива, превращает в int и по словарю находит отображение в шестнадцатеричном формате  
 h += HEX[bin2dec(a[i:i + 4])] # Возвращает строку  
 return h  
  
  
# Операция XOR для двух массивов одинаковой длины  
def xor(a: list, b: list):  
 c = []  
 for i in range(len(a)):  
 c.append(a[i] ^ b[i])  
 return c  
  
  
# Функция получения раундовых ключей  
def get\_keys(k: list): # Получаем C0 и D0  
 C = [k[56], k[48], k[40], k[32], k[24], k[16], k[8], k[0], k[57], k[49], k[41], k[33], k[25], k[17], k[9], k[1],  
 k[58], k[50], k[42], k[34], k[26], k[18], k[10], k[2], k[59], k[51], k[43], k[35]]  
 D = [k[62], k[54], k[46], k[38], k[30], k[22], k[14], k[6], k[61], k[53], k[45], k[37], k[29], k[21], k[13], k[5],  
 k[60], k[52], k[44], k[36], k[28], k[20], k[12], k[4], k[27], k[19], k[11], k[3]]  
 s = [1, 1, 2, 2, 2, 2, 2, 2, 1, 2, 2, 2, 2, 2, 2, 1] # Массив сдвигов по раундам  
 keys = []  
 for i in s:  
 C = C[i:] + C[:i] # Циклический сдвиг половинок  
 D = D[i:] + D[:i]  
 CD = C + D  
 keys.append(  
 [CD[13], CD[16], CD[10], CD[23], CD[0], CD[4], CD[2], CD[27], CD[14], CD[5], CD[20], CD[9], CD[22], CD[18],  
 CD[11], CD[3], CD[25], CD[7], CD[15], CD[6], CD[26], CD[19], CD[12], CD[1], CD[40], CD[51], CD[30], CD[36],  
 CD[46], CD[54], CD[29], CD[39], CD[50], CD[44], CD[32], CD[47], CD[43], CD[48], CD[38], CD[55], CD[33],  
 CD[52], CD[45], CD[41], CD[49], CD[35], CD[28], CD[31]])  
 return keys # При каждой итерации сдвига добавляем в итоговый двумерный список keys получившийся раундовый ключ  
  
  
# Функция сжатия ветора B(b1b2...b8) к S-блоку(32 бит)  
def comp(b: list, S: list):  
 a = bin2dec([b[0], b[-1]]) # Получаем номер строки в S1-8  
 c = bin2dec(b[1:-1]) # Получаем номер столбца  
 return dec2bin(S[a][c], 4) # Возвращаем 4 бита, соответствующие позиции в матрице S1-8  
  
  
# Функция шифрования  
def F(R: list, key: list): # Е - расширение 32 до 48  
 E = [R[31], R[0], R[1], R[2], R[3], R[4], R[3], R[4], R[5], R[6], R[7], R[8], R[7], R[8], R[9], R[10], R[11], R[12],  
 R[11], R[12], R[13], R[14], R[15], R[16], R[15], R[16], R[17], R[18], R[19], R[20], R[19], R[20], R[21], R[22],  
 R[23], R[24], R[23], R[24], R[25], R[26], R[27], R[28], R[27], R[28], R[29], R[30], R[31], R[0]]  
 B = xor(E, key) # xor расширенного вектора и ключа раунда  
 S = []  
 S += comp(B[0:6], S1) # Сжатие B - блоков  
 S += comp(B[6:12], S2)  
 S += comp(B[12:18], S3)  
 S += comp(B[18:24], S4)  
 S += comp(B[24:30], S5)  
 S += comp(B[30:36], S6)  
 S += comp(B[36:42], S7)  
 S += comp(B[42:48], S8)  
 return [S[15], S[6], S[19], S[20], S[28], S[11], S[27], S[16], S[0], S[14], S[22], S[25], S[4], S[17], S[30], S[9],  
 S[1], S[7], S[23], S[13], S[31], S[26], S[2], S[8], S[18], S[12], S[29], S[5], S[21], S[10], S[3], S[24]]  
  
  
# Режим работы программы  
mode = int(input('Введите режим работы:\n1 - Зашифровать\t2 - Расшифровать\n'))  
  
fr = open(input('Имя изменяемого файла\n'), 'rb')  
fw = open(input('Имя файла с результатом\n'), 'wb')  
fk = open(input('Имя файла-ключа\n'), 'rb')  
t = fr.read()  
a, key = [], []  
for j in fk.read():  
 key = key + dec2bin(j, 8) # побайтово преобразуем ключ в бинарный формат  
keys = get\_keys(key)  
if mode == 2: # Если пользователь выбрал режим расшифровки, 16-й раундовый ключ становится 1-м  
 keys.reverse()  
hkeys = [] # Для проверки ключей по методичке  
for i in keys:  
 hkeys.append(bin2hex(i))  
for i in t:  
 a = a + dec2bin(i, 8) # побайтово преобразуем открытый текст в бинарный формат  
if len(a) % 64 != 0: # Заполняем нулями недостающее окно  
 a = a + [0] \* (64 - (len(a) % 64))  
for o in range(0, len(a), 64): # Делим исходную битовую последовательность по 64 бита  
 block = a[o:o + 64]  
 T = IP(block) # Применяем начальную перестановку  
 Th = bin2hex(T) # для контроля  
 L, R = T[:32], T[32:] # разбиваем на половины  
 Lh, Rh = bin2hex(L), bin2hex(R)  
 # Основной цикл программы шифрования\дешифрования  
 for i in keys: # Для каждого раундового ключа проделываем следующие действия  
 temp = R # Запоминаем значение Ri-1  
 R = xor(L, F(R, i)) # Присваиваем R значение по схеме  
 L = temp # Li = Ri-1  
 print('L:', bin2hex(L)) # Для контроля по методичке  
 print('R:', bin2hex(R), '\n\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_')  
 T16 = R + L # Сборка финального блока T16  
 print(bin2hex(T16))  
 Final = IP\_back(T16) # Обратная перестановка  
 print('После конечной перестановки:', bin2hex(Final))  
 for i in range(0, len(Final),  
 8): # Разбиваем зашифрованную последовательность по 8 бит и записываем соответствующий байт в файл  
 fw.write(bytes([bin2dec(Final[i:i + 8])]))  
fr.close()  
fw.close()  
fk.close()

Рассмотрим пример кодирования текстового файла:

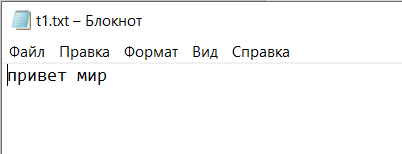


Рисунок – Файл с исходным текстом

Взаимодействие с программой выглядит следующим образом:

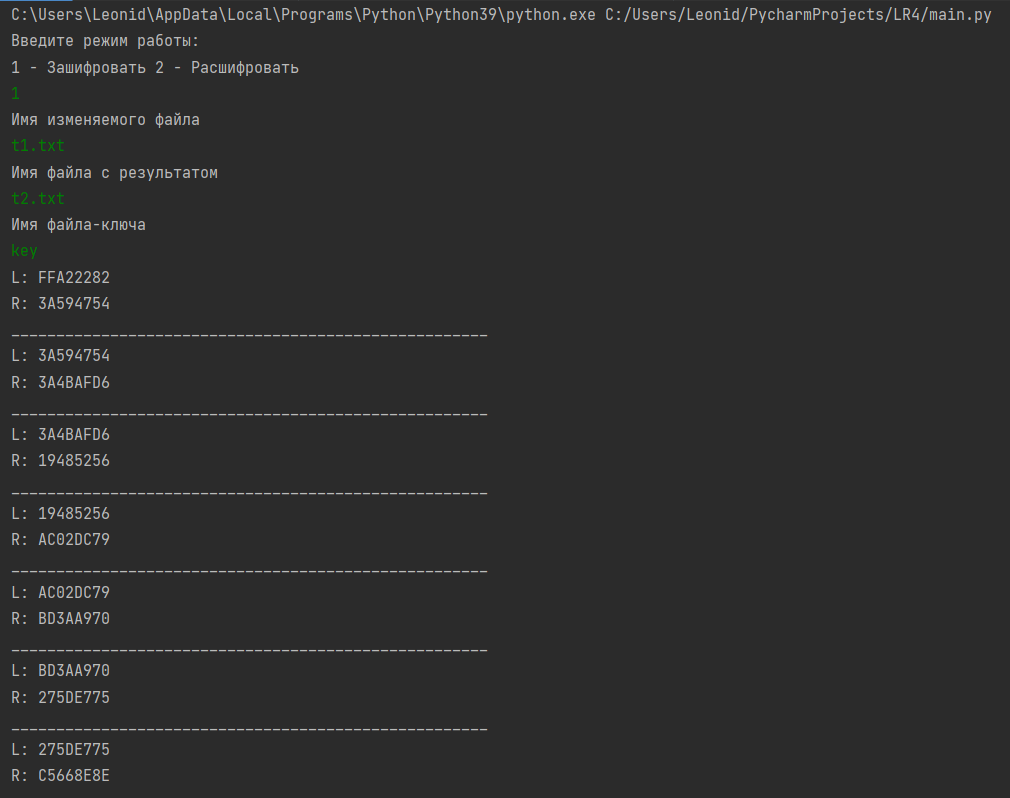


Рисунок – Взаимодействие 1

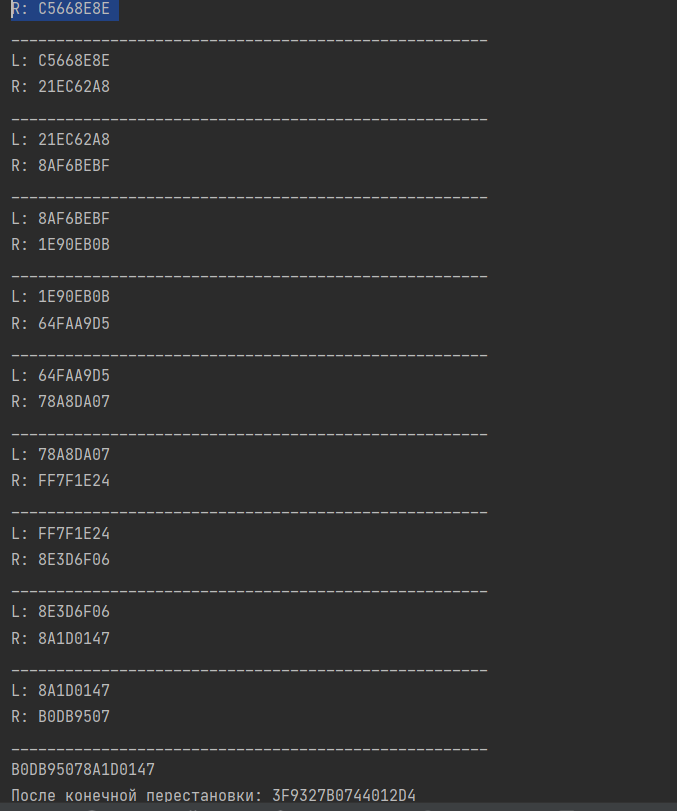


Рисунок – Взаимодействие 2

В результате работы DES зашифрованный текст выглядит так:

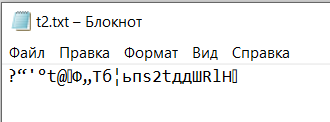


Рисунок – Зашифрованный текст

Взаимодействие с программой при расшифровке выглядит следующим образом:

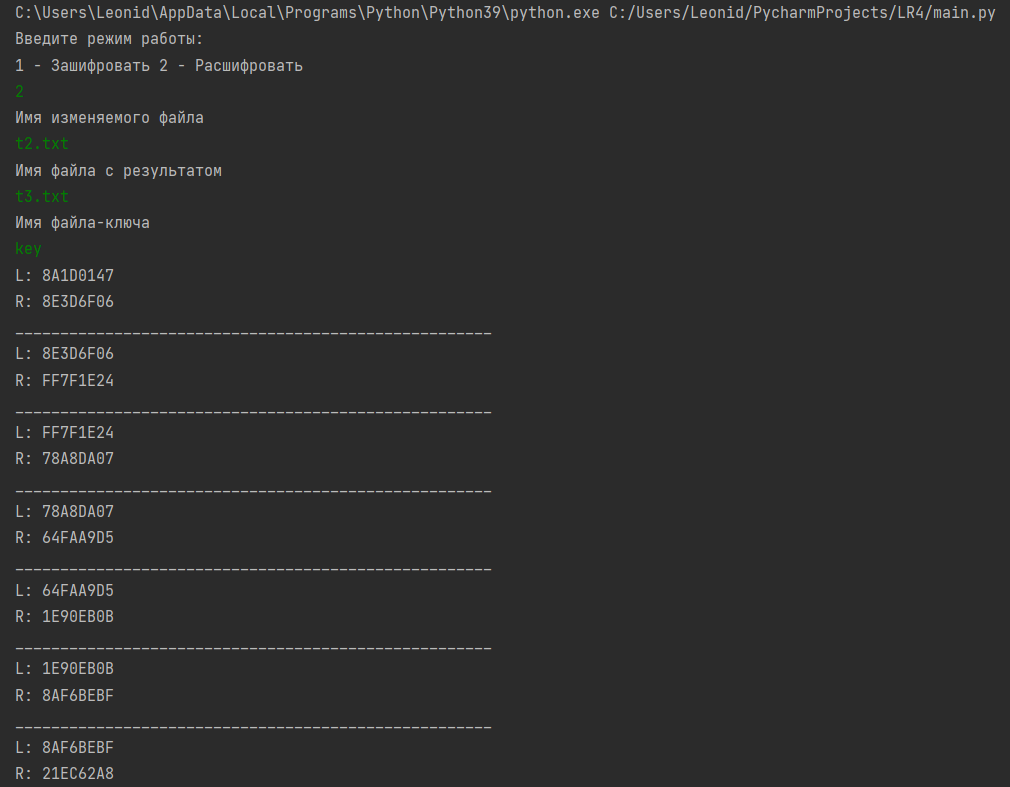


Рисунок – Взаимодействие при расшифровке

Файл key представляет собой бинарный файл, длиной 8 байт (64 бит), взятый из методички(приложение к отчету в электронном письме) на странице 8:

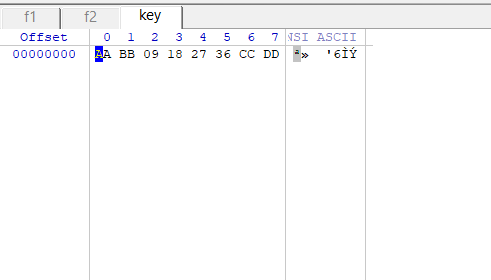


Рисунок – Файл ключ

В результате работы сценария в режиме расшифровки получился следующий результат:

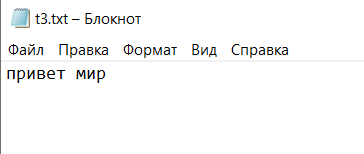


Рисунок – Расшифрованный текст

Калибровка и настройка алгоритма производилась по методичке. В качестве открытого текста использовалась последовательность байтов: 123456ABCD132536

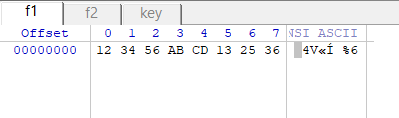


Рисунок – Открытый текст из примера

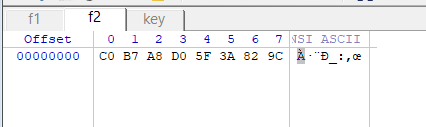


Рисунок – Шифротекст

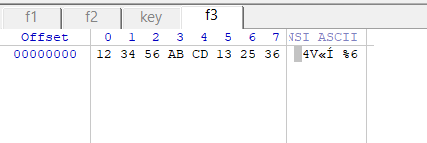


Рисунок – Расшифрованный текст

На двух примерах мы убедились, что программа работает корректно, теперь проведем криптоанализ на основе частоты появления символов:

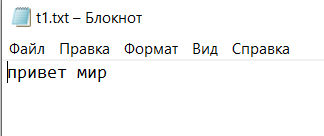


Рисунок – Открытый текст

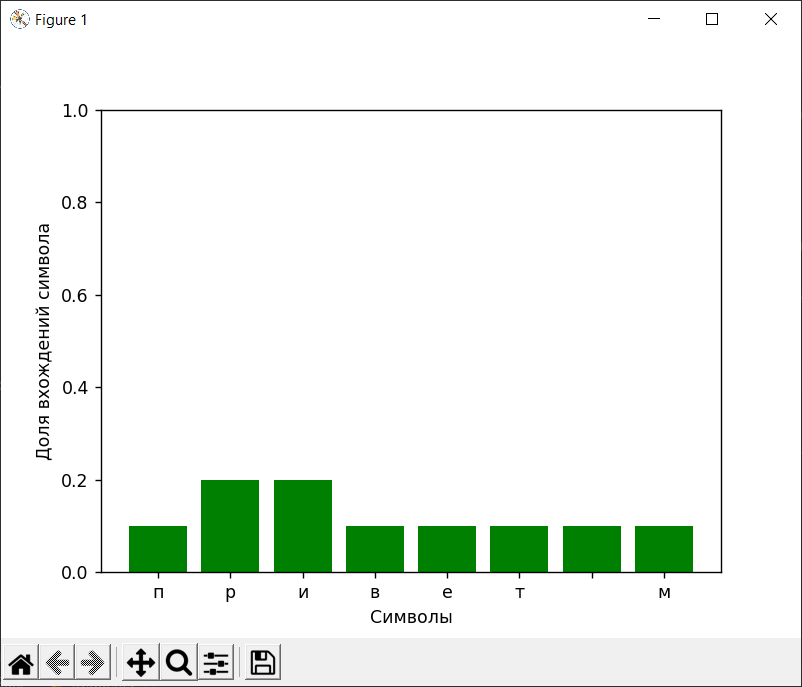


Рисунок – Частоты символов открытого текста

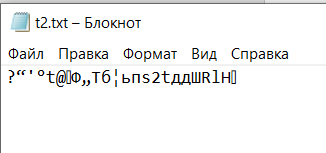


Рисунок - Шифротекст

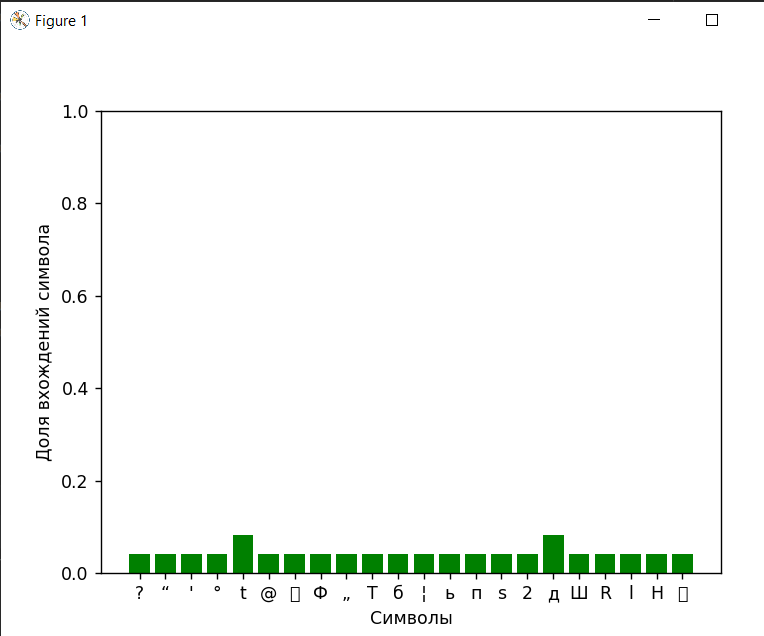
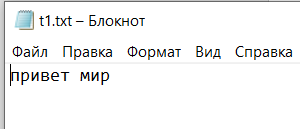


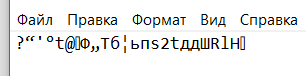
Рисунок – Частоты символов шифротекста

Как мы можем заметить шифротекст имеет равномерное распределение частот появления символов, поэтому данным способом раскрыть ключ шифрования DES не получится

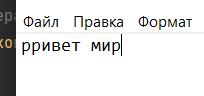
Анализ на основе открытого текста:

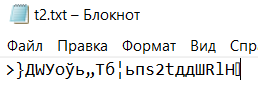
Для примера попробуем поменять букву «п» в фразе «привет мир» на букву «р»:



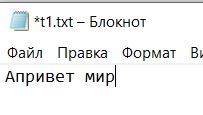


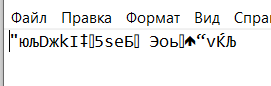
Поменяли букву:





Как мы можем заметить, изменились сразу 8 символов. Теперь попробуем из фразы «привет мир» сделать фразу «Апривет мир» и посмотрим на результат шифрования





Сравнив этот шифротекст с исходным, понимаем, что сходства нет ни в одном символе, следовательно по открытому тексту, когда алгоритм представляет собой «черный ящик» вычислить ключ не удастся.

Выводы

В результате выполнения данной лабораторной работы были улучшены навыки программирования на ЯП Python. В рамках изучения сетей Фейстеля мы научились реализовывать SP- блоки, производить побайтовое чтение и запись в файлы, а также производить блоковое шифрование открытого текста.

Алгоритм DES на практике показал себя как хороший способ симметричного шифрования. Для задач передачи информации по закрытому каналу, когда передающая и принимающая стороны знают ключ, он отлично подходит, поскольку абсолютно неуязвим к стандартным методам криптоанализа, а малейшее изменение битовой последовательности исходного текста лавинообразно меняет конечный результат.