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 №2

Подпись ФИО

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

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

И.А. Ибрагимов

Группа РИ-481220

Екатеринбург 2022

Цель работы: изучить исторически значимые методы шифрования на основе перестановок и их недостатки.

Задачи:

1. Создать программу, которая зашифровывает и расшифровывает файлы с использованием шифра вертикальной перестановки (с. 95 пособия). При этом имена входного и выходного файлов, а также режим работы задаются параметрами командной строки программы (подсказка: input()).

2. Зашифровать и расшифовать произвольный текстовый файл.

3. Выполнить криптоанализ на основе известного текста: заполнить файл последовательностью букв алфавита и зашифровать. Создать программу, которая на основе сформированного полученного шифротекста определяет размерность таблицы и ключ.

4. Ответить на вопрос: чем с точки зрения ИБ запрос ключа программой лучше ее ввода в качестве параметра командной строки?Ход работы:

Помимо шифров подстановки, широкое распространение также получили перестановочные шифры. В качестве примера опишем **Шифр вертикальной перестановки.**

В процессе шифрования сообщение записывается в виде таблицы. Количество колонок таблицы определяется размером ключа. Например, зашифруем сообщение: **это пример шифра вертикальной перестановки**. С использованием ключа [4 3 2 1]. В итоге создается таблица размерностью 4 столбца и 10 строк, после записывается строка в порядке, определяемом ключом:

**пефекнетвпомивиьпсоптишатлйениэрррраорак.**

Код сценария, решающего текущие задачи, приведен ниже:

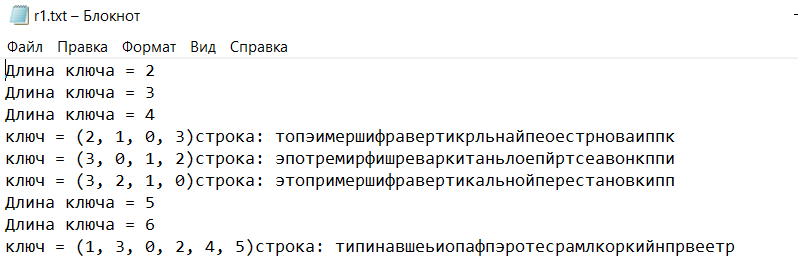
# Блок шифрования  
  
case 'en':  
 arr = []  
 print('Введите порядок шифрования (пустой ввод - конец ключа)')  
 while True: # Пока ввод не пустой считываем порядок номеров столбцов  
 a = input()  
 if a == '':  
 break  
 arr.append(int(a))  
  
 file = input('Введите имя файла, который хотите зашифровать?\n')  
 f = open(file, 'r', encoding='utf-8')  
 file\_out = input('Как будет называться зашифрованный файл?\n')  
 f2 = open(file\_out, 'w', encoding='utf-8')  
  
 str1 = f.read()  
 str1 = str1.replace(' ', '') # Убираем пробелы из строки  
 str1 = str1.lower() # Переводим строку в нижний регистр  
 n = len(arr)  
 if (int(len(str1) % n)) > 0:  
 m = int(len(str1) / n) + 1  
 str1 += (n - int(len(str1) % n)) \* 'п' # Заполняем строку до нужного размера буквами "п"  
 else:  
 m = int(len(str1) / n)  
 dv = zapoln(m, n, str1) # заполняем массив  
 final = shifr(m, dv, arr) # шифруем строку  
 f2.write(final) # записываем строку в файл  
 f2.close()  
 f.close()  
case \_:  
 print('Неверный ввод, введите en или de\n')  
 input('Введите режим работы программы: en - зашифровать, de - расшифровать\n')

Лучшим способом атаки шифра вертикальной перестановки будет полный перебор всех возможных ключей малой длины (до 9 включительно — около 400 000 вариантов). Для этого был написан код, который построен на методе Брутфорса — метод угадывания пароля (или ключа, используемого для шифрования), предполагающий систематический перебор всех возможных комбинаций символов до тех пор, пока не будет найдена правильная комбинация. Однако выбирать нужный вариант среди 400 000 вариантов – слишком долго и неэффективно, поэтому был реализован алгоритм фильтрации, считающий количество слов, являющихся частью расшифрованной строки. Для проверки строк был использован словарь, представленный обычным txt- файлом, содержащий более 300 000 слов, включая междометия, частицы союзы и предлоги. На основе значений, полученных при тестировании алгоритма на строках, длиной в 40 символов, было выяснено, что строки, содержащие обычный текст содержат не менее 27 слов по выбранному словарю. Таким образом, поставив ограничение в 25 слов, мы можем отфильтровать строки с потенциально полезным содержимым и вывести их в файл. Таким образом вместо 400 000 вариантов мы получаем намного более сокращенную выборку, позволяющую отследить правильный исходный ключ.

Код для дешифровки сообщения методом Брутфорса приведен ниже:

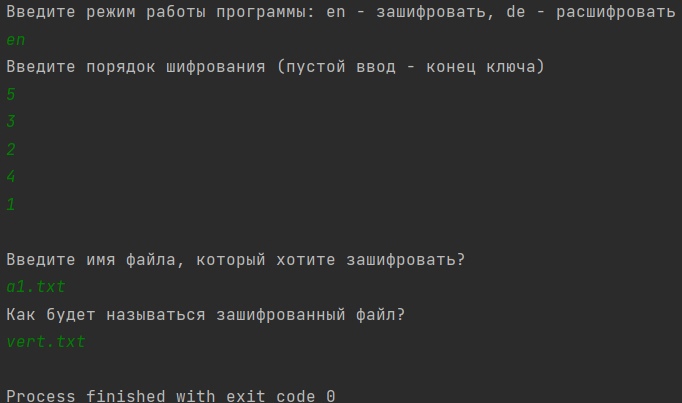
# Блок расшифрования  
match y:  
 case 'de':  
 slovar = open('RUS.txt', 'r', encoding='utf-8') # открываем скачанный файл со всеми русскими словами  
 rus = {}  
 h = 0  
 for line in slovar: # создаем словарь с русскими словами  
 if line == '':  
 break  
 line = line.replace('\n', '') # убираем символы конца строки  
 rus.update({line: h})  
 slovar.close()  
 file = open(input("Введите название файла, который нужно расшифровать\n"), 'r', encoding='utf-8')  
 outfile = open(input("Введите название файла, куда будут записаны варианты расшифровки\n"), 'w',  
 encoding='utf-8')  
 str1 = file.read() # считываем текст в одну строку  
  
 for d in range(2, 7): # Задаем диапазон длины ключа, в котором будет осуществляться поиск строки  
 step = d  
 outfile.write('Длина ключа = ' + str(step) + '\n')  
 keys = []  
 for i in permutations(range(step)): # Получаем массив всех комбинаций ключей методом перестановок  
 keys.append(i)  
 de\_sh = dl\_mas(step, str1) # Получаем массив с зашифрованными буквами  
 arg = len(de\_sh[0])  
 for j in range(len(keys)): # Для каждой комбинации строк(ключа) составляем расшифрованную строку  
 out\_str = decode\_text(arg, keys[j], de\_sh)  
 g = 0  
 for k, v in rus.items(): # ищем в словаре совпадения с элемнтами строки длиной не меньше 2 символов  
 if len(k) > 2:  
 if out\_str.find(k, 0) != -1: # если совпадение найдено, повышаем значение счетчика  
 g += 1  
 if g > 25: # если счетчик больше 25, считаем строку потенциально полезной и записываем в файл  
 outfile.write('ключ = ' + str(keys[j]) + 'строка: ' + out\_str + '\n')  
 file.close()  
 outfile.close()

Файл с вариантами расшифровки выглядит следующим образом:

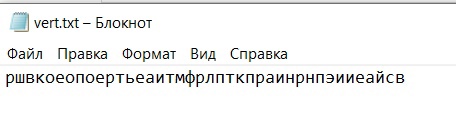


Как можно заметить, программа просканировала 2-х, 3-х,…, 6- символьные ключи и нашла исходный текст, помимо этого выдала ключ шифрования

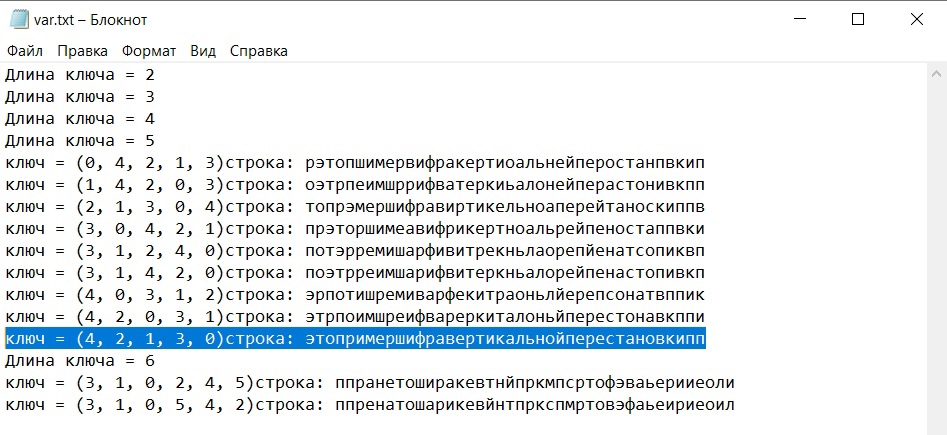
Так выглядит процесс шифрования и расшифровки с помощью 5- символьного ключа:



После этого, зашифрованный текст выглядит так:



А так выглядит файл с вариантами расшифрованного текста. Как видно на скриншоте программа не только расшифровала текст, но и вывела правильный ключ.



Код всего сценария выглядит так:

from itertools import \*  
  
  
def dl\_mas(step: int, text: str): # Функция создания таблицы по длине ключа и ее заполнение  
 sr = ''  
 # step = int(len(text) / (step + 1))  
 if len(text) % step > 0: # Задаем размерность массива по длине строки и ключа  
 m = int(len(text) / step) + 1  
 else:  
 m = int(len(text) / step)  
 sr = [[0] \* m for u in range(step)] # По высчитаным значениям инициализируем массив  
 for k in range(m):  
 it1 = list(range(k, len(text), m)) # Получаем список индексов и по нему заполняем массив буквами  
 for b in it1:  
 sr[int(b / m)][k] = text[b]  
 print(sr)  
 return sr  
  
  
def decode\_text(step: int, key: list, val: list): # Функция формирования расшифрованной строки по ключу  
 viv = ''  
 for i in range(step): # i - номер элемента в одномерном списке  
 for j in key: # j - номер строки  
 if val[j][i] == 0: # пропускаем нулевые символы  
 continue  
 else:  
 viv += val[j][i] # В итоговую строку записываем символы в порядке, установленном ключом  
  
 return viv  
  
  
def zapoln(m: int, n: int, p: str): # Заполнение двумерного массива символами строки  
 dv = [[0] \* n for i in range(m)] # Инициализируем массив  
 k = 0  
 for i in range(m):  
 for j in range(n):  
 if k < len(str1):  
 if str1[k] != '\n': # Проверка символов  
 dv[i][j] = str1[k]  
 k += 1  
 else:  
 k += 1  
 else:  
 k = 0  
 break  
 return dv  
  
  
def shifr(m: int, d: list, key: list): # Шифрование строки по заданному ключу  
 final\_str = ''  
 for i in key:  
 for j in range(m):  
 if d[j][i - 1] == 0: # Обработка незаполненных ячеек массива  
 final\_str = final\_str + ''  
 else:  
 final\_str = final\_str + str(d[j][i - 1])  
 return final\_str # Возвращает зашифрованную строку  
  
  
y = input('Введите режим работы программы: en - зашифровать, de - расшифровать\n')  
  
# Блок расшифрования  
match y:  
 case 'de':  
 slovar = open('RUS.txt', 'r', encoding='utf-8') # открываем скачанный файл со всеми русскими словами  
 rus = {}  
 h = 0  
 for line in slovar: # создаем словарь с русскими словами  
 if line == '':  
 break  
 line = line.replace('\n', '') # убираем символы конца строки  
 rus.update({line: h})  
 slovar.close()  
 file = open(input("Введите название файла, который нужно расшифровать\n"), 'r', encoding='utf-8')  
 outfile = open(input("Введите название файла, куда будут записаны варианты расшифровки\n"), 'w',  
 encoding='utf-8')  
 str1 = file.read() # считываем текст в одну строку  
  
 for d in range(2, 7): # Задаем диапазон длины ключа, в котором будет осуществляться поиск строки  
 step = d  
 outfile.write('Длина ключа = ' + str(step) + '\n')  
 keys = []  
 for i in permutations(range(step)): # Получаем массив всех комбинаций ключей методом перестановок  
 keys.append(i)  
 de\_sh = dl\_mas(step, str1) # Получаем массив с зашифрованными буквами  
 arg = len(de\_sh[0])  
 for j in range(len(keys)): # Для каждой комбинации строк(ключа) составляем расшифрованную строку  
 out\_str = decode\_text(arg, keys[j], de\_sh)  
 g = 0  
 for k, v in rus.items(): # ищем в словаре совпадения с элемнтами строки длиной не меньше 2 символов  
 if len(k) > 2:  
 if out\_str.find(k, 0) != -1: # если совпадение найдено, повышаем значение счетчика  
 g += 1  
 if g > 25: # если счетчик больше 25, считаем строку потенциально полезной и записываем в файл  
 outfile.write('ключ = ' + str(keys[j]) + 'строка: ' + out\_str + '\n')  
 file.close()  
 outfile.close()  
 # Блок шифрования  
  
 case 'en':  
 arr = []  
 print('Введите порядок шифрования (пустой ввод - конец ключа)')  
 while True: # Пока ввод не пустой считываем порядок номеров столбцов  
 a = input()  
 if a == '':  
 break  
 arr.append(int(a))  
  
 file = input('Введите имя файла, который хотите зашифровать?\n')  
 f = open(file, 'r', encoding='utf-8')  
 file\_out = input('Как будет называться зашифрованный файл?\n')  
 f2 = open(file\_out, 'w', encoding='utf-8')  
  
 str1 = f.read()  
 str1 = str1.replace(' ', '') # Убираем пробелы из строки  
 str1 = str1.lower() # Переводим строку в нижний регистр  
 n = len(arr)  
 if (int(len(str1) % n)) > 0:  
 m = int(len(str1) / n) + 1  
 str1 += (n - int(len(str1) % n)) \* 'п' # Заполняем строку до нужного размера буквами "п"  
 else:  
 m = int(len(str1) / n)  
 dv = zapoln(m, n, str1) # заполняем массив  
 final = shifr(m, dv, arr) # шифруем строку  
 f2.write(final) # записываем строку в файл  
 f2.close()  
 f.close()  
 case \_:  
 print('Неверный ввод, введите en или de\n')  
 input('Введите режим работы программы: en - зашифровать, de - расшифровать\n')

Скрипт работает таким образом, что пользователь может выбирать между режимами работы программы. Если выбран режим работы «Шифрование», то пользователю предлагается ввести ключ, после чего будет записан файл, содержащий зашифрованный текст. В режиме расшифровки от пользователя только требуется ввести называния входного и выходного файла, после чего подождать завершения работы программы.

В данной лабораторной работе имена файлов и другие входные данные поступают в программу с помощью функции input(). Такой подход не только дружелюбнее к пользователям, но и безопаснее, поскольку данные не сразу поступают в программу, а ввод контролируется пользователем.

Выводы

В результате выполнения данной лабораторной работы были улучшены навыки работы с ЯП Python. В ходе работы был написал сценарий, позволяющий шифровать содержимое файла методом вертикальной перестановки. Данного метод шифрования реализован благодаря двумерным массивам (спискам). Принцип работы блока шифрования состоит в следующем:

Длина строки делится на длину ключа, таким образом определяются размеры шифровальной таблицы. Далее таблица заполняется символами строки, после чего в итоговый файл записывается строка, содержащая все столбцы таблицы, выводимые в порядке, указанном ключом.

Основная проблема данного алгоритма заключается в том, что метод полного перебора прямо пропорционален факториалу длины ключа, поскольку для перебора всех комбинаций ключа, состоящего из n символов, существует n! вариантов. Это означает, что уже для 10 - значного ключа требуется перебрать 3 628 800 вариантов. Соответственно для ключей большой длины такой метод расшифровки не подходит.