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 №1

Подпись ФИО

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

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

И.А. Ибрагимов

Группа РИ-481220

Екатеринбург 2022

Цель работы: изучить метод шифрования на основе одноразового блокнота

Задачи: 1. Создать программу, которая зашифровывает и расшифровывает файлы с

использованием одноразового блокнота, хранящегося в отдельном файле (подсказка:

нужны 2 копии). Блокнот формируется с использованием криптографического

генератора случайных чисел (ГСЧ) языка Python (подсказка: в Debian установочный

пакет называется python3-crypto, модуль Crypto.Random, для учебных целей

можно также использовать штатный ГСЧ из модуля random, не предназначенный для

задач криптографии).

2. Зашифровать и расшифровать с ее помощью несколько файлов.

3. Выполнить криптоанализ на основе частот появления символов (см. задачу 1) и

известного текста (см. задачу 2). Сделать вывод о причинах неудачи.

Ход работы:

Метод шифрования на основе одноразового блокнота заключается в следующем:

На каждую пару символов ключа и открытого текста применяется операция XOR (исключающее ИЛИ). Данный метод шифрования является одним из наилучших способов зашифровать информацию, поскольку, не зная ключа, невозможно получить открытый текст. Принцип, следующий: a + b + b = a, где “+” – операция двоичного сложения чисел.

Соответственно, если точный ключ не известен, результат расшифровки будет совершенно случайным.

Согласно заданию на лабораторную работу, требуется реализовать на ЯП Python сценарий, позволяющий зашифровывать и расшифровывать файлы методом одноразового блокнота.

Логика сценария делится на следующие этапы:

\* Чтение файла

\* Создание ключа одной длины с шифруемым открытым текстом.

\* Побитовое применение операции xor к каждому символу сформированного ключа и открытого текста

\* Запись в отдельный файл ключа шифрования

\* Запись в файл зашифрованный текст

Код сценария разделен на две части: блок шифрования и блок расшифровки по заданному ключу. Программа дружелюбна к пользователю и просит взаимодействия с консолью для выбора режима работы и прочей информации, необходимой для работы программы.

Код программы:

import random  
  
  
def xor\_file(file: str, key: str):  
 # Функция записи результата XOR между символами словаря и открытого текста  
 rez = ''  
 for i in range(len(file)):  
 rez = rez + chr(ord(file[i]) ^ ord(key[i]))  
 return rez  
  
  
def get\_key(text: str):  
 # Функция генерации ключа по длине содержимого файла  
 final = ''  
 for i in range(len(open\_text)):  
 final = final + chr(random.randint(1040, 1103)) # Генерируем случайное число и вставляем в ключ- строку ascii- символ, соответствующий этому числу  
 return final  
  
  
mode = input('Введите режим работы программы:\n 1: Зашифровать \t 2: Расшифровать\n')  
# Конструкция match - case для выбора режима работы  
match mode:  
 # Начало блока шифрования  
 case '1':  
 p = input('Сформировать ключ?\n1: Сформировать\t2: Шифрование по имеющемуся ключу\n')  
 match p:  
 case '1':  
 file = open(input('Имя файла, который нужно зашифровать\n'), 'r', encoding='utf-8')  
 file\_crypt = open(input('Имя зашифрованного файла\n'), 'w', encoding='utf-8')  
 key = open('key.txt', 'w', encoding='utf-8') # В файл key.txt записываем копию использованного для шифрования ключа  
 open\_text = file.read()  
 final = get\_key(open\_text) # Получаем строку, содержащую ключ  
 key.write(final) # записываем строку- ключ в файл  
 crypt\_text = xor\_file(open\_text, final) # Применяем Xor для открытого текста  
 file\_crypt.write(crypt\_text)  
 file\_crypt.close()  
 file.close()  
 key.close()  
 case '2':  
 key = open(input('Введите имя файла-ключа\n'), 'r', encoding='utf-8')  
 file = open(input('Имя файла, который нужно зашифровать\n'), 'r', encoding='utf-8')  
 file\_crypt = open(input('Имя зашифрованного файла\n'), 'w', encoding='utf-8')  
 open\_text = file.read()  
 final = key.read()  
 crypt\_text = xor\_file(open\_text, final) # Применяем Xor для открытого текста  
 file\_crypt.write(crypt\_text)  
 file\_crypt.close()  
 file.close()  
 key.close()  
 case \_:  
 print('Неверно выбран режим работы блока шифрования\n')  
 exit(0)  
  
# Конец блока шифрования  
  
# Начало блока расшифровки  
 case '2':  
 file\_cr = open(input('Введите имя файла, который нужно расшифровать\n'), 'r', encoding='utf-8')  
 outfile = open(input('Введите имя файла, в который будет размещен расшифрованный текст\n'), 'w', encoding='utf-8')  
 key\_encrypt = open(input('Введите имя файла-ключа\n'), 'r', encoding='utf-8')  
 key = key\_encrypt.read() #  
 text = file\_cr.read() #  
 outfile.write(xor\_file(text, key)) #  
 outfile.close()  
 file\_cr.close()  
 key\_encrypt.close()  
 # Конец блока расшифровки  
  
 # Начало блока обработки ошибок ввода  
 case \_:  
 print('Некорректный ввод, попробуйте еще раз!')  
 exit(0) # Если ввод некорректен, завершаем работу программы с уведомлением об ошибке

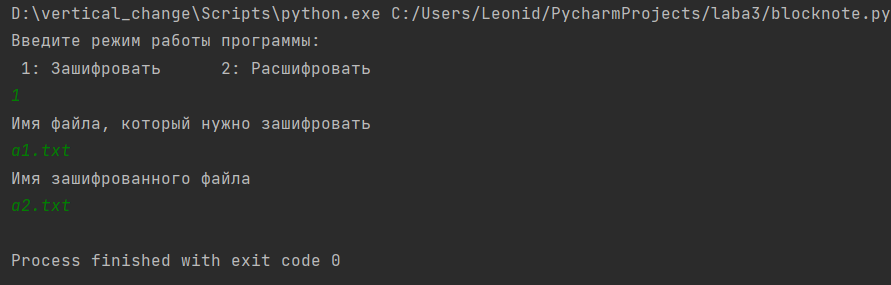


Рисунок 1 – Последовательность работы сценария в режиме шифрования

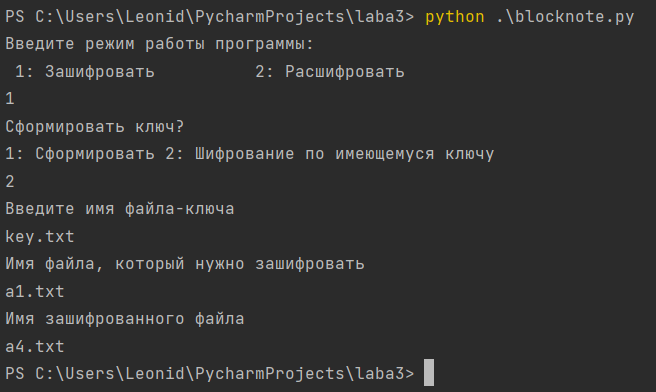


Рисунок 2 – Шифрование существующим ключом

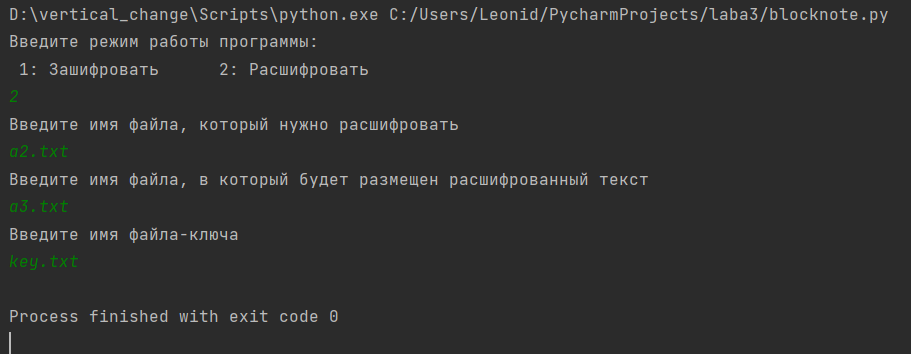


Рисунок 3 - Последовательность работы сценария в режиме расшифровки

**Криптоанализ**

В качестве криптоанализа в данной лабораторной работе предложены методы на основе частоты появления символов и анализ на основе открытого текста.

Начнем с криптоанализа по частотам появления символов.

Посмотрим на частоты появления символов открытого текста, потом файла-ключа, а затем – зашифрованного файла.

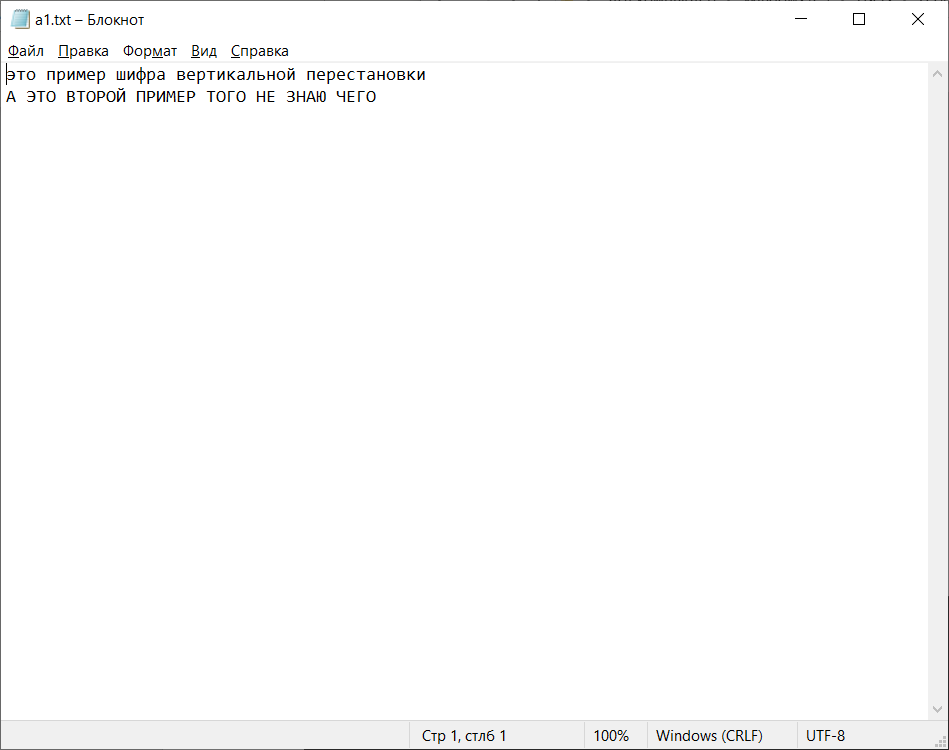


Рисунок 4 – Открытый текст

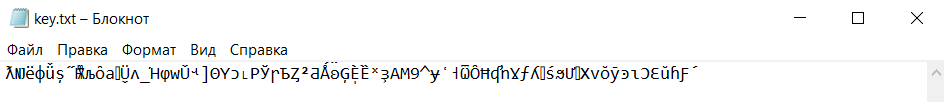


Рисунок 5 – Содержание файла – ключа шифрования

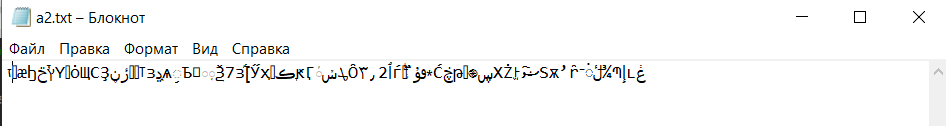


Рисунок 6 – Зашифрованный текст

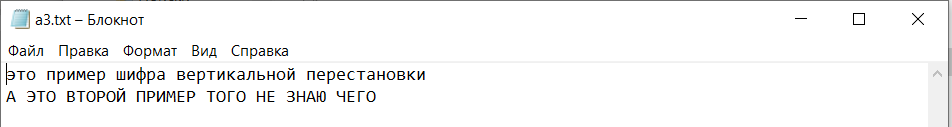


Рисунок 7 – Расшифрованный текст

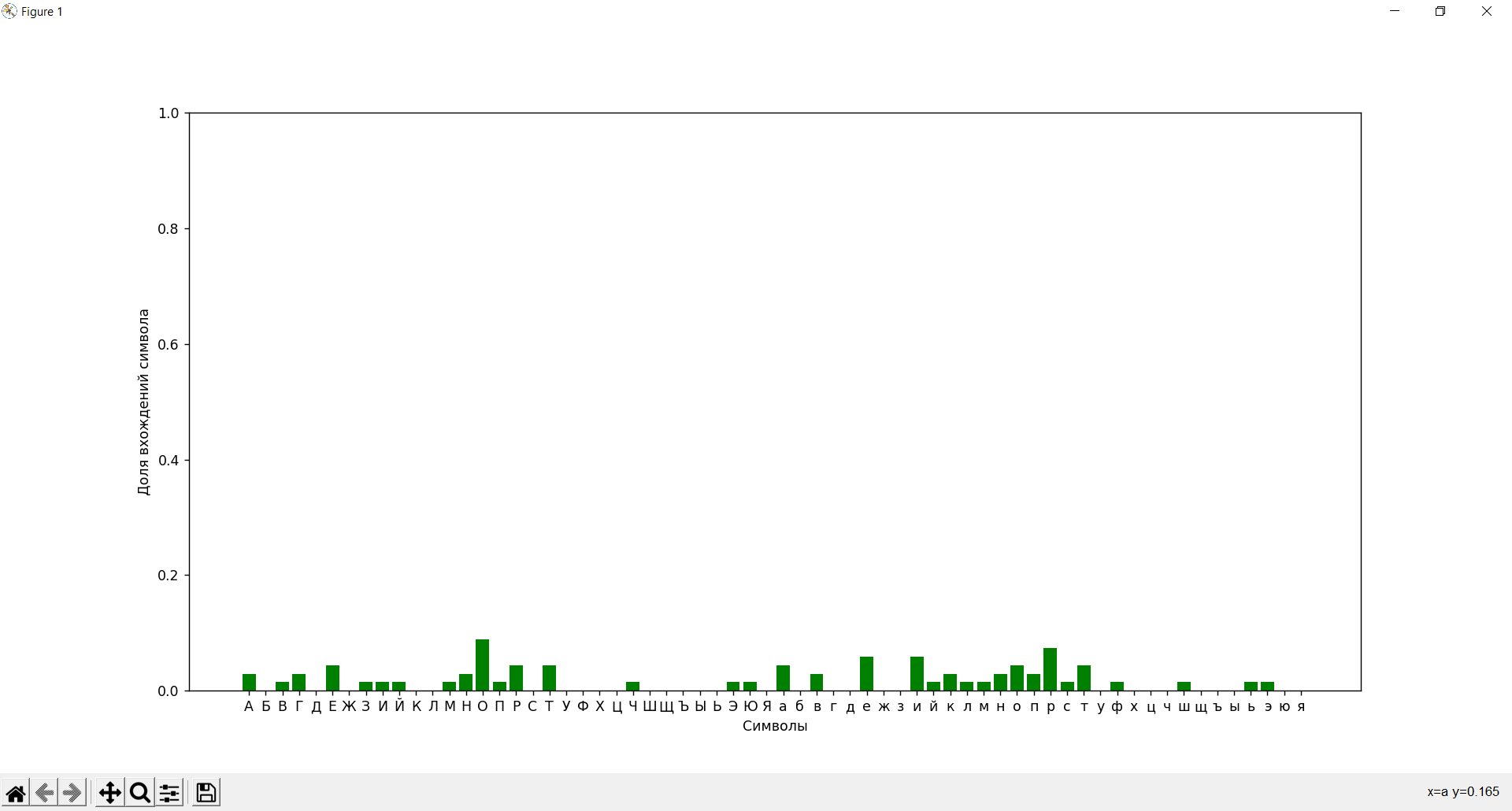


Рисунок 8 – Частота появления символов открытого текста

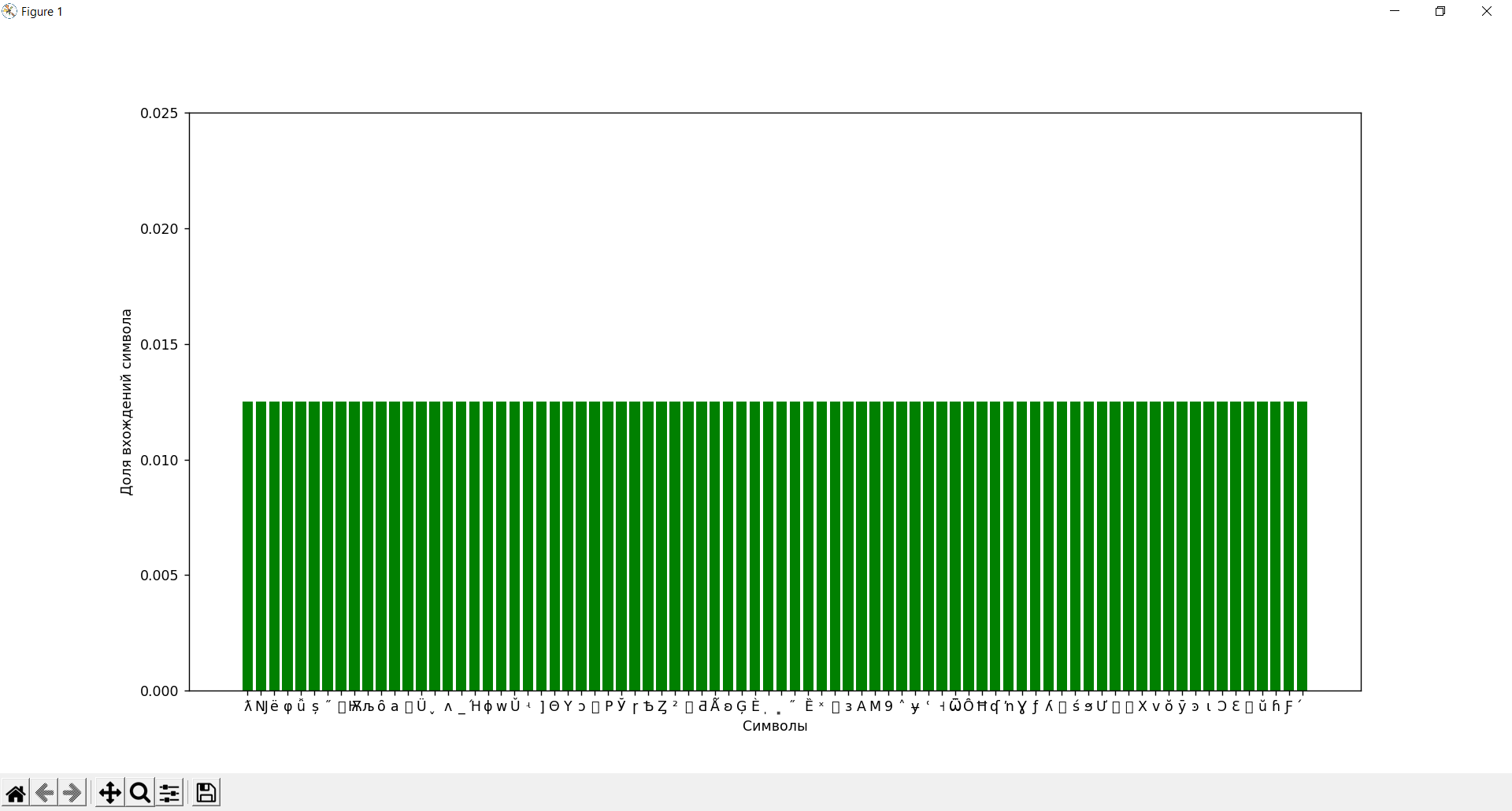


Рисунок 9 – Частота появления символов ключа

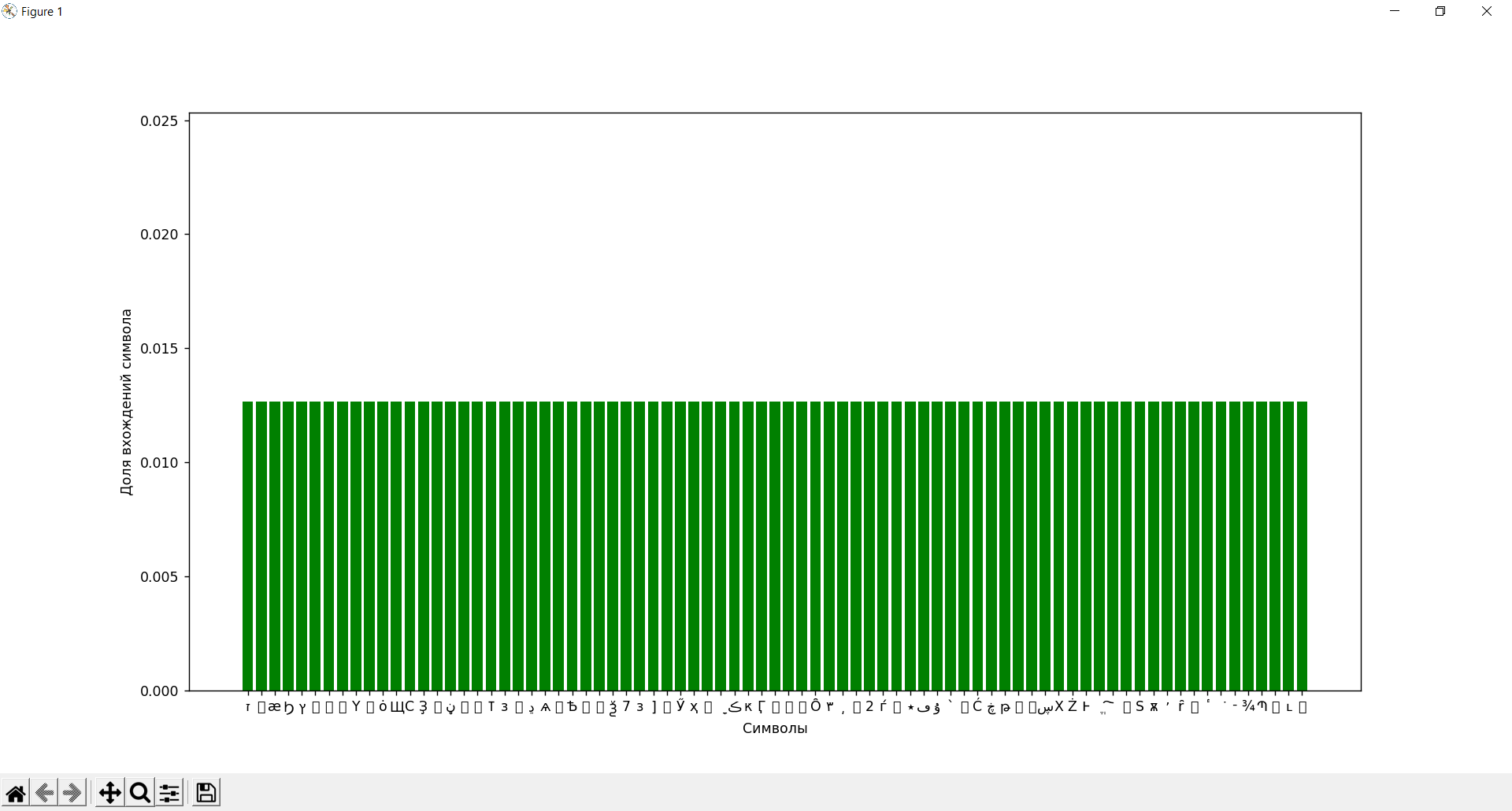


Рисунок 10 – Частота появления символов в зашифрованном файле

Как мы можем заметить, в осмысленном открытом тексте прослеживаются различия между частотой появления символов. Закон распределения такой случайной величины скорее похож на нормальный, тогда как для ключа и зашифрованного файла мы наблюдаем скорее равномерный закон распределения, характерный для шума. Три гистограммы, приведенные выше, показывают, что для файла, закодированного методом одноразового блокнота, не получится найти ключ путем вычисления частоты появления символов.

Код сценария для построения гистограмм:

import sys  
import matplotlib.pyplot as plt  
import numpy as np  
  
# Считываем файл из консоли  
file = input('Введите имя файла\n')  
  
# Задаем словарь и заполняем его буквами  
letters = {}  
f = open(file, 'r', encoding='utf-8') # Читаем файл в кодировке utf-8  
  
for line in f:  
 for i in line:  
 try:  
 letters[i] += 1  
 except KeyError:  
 letters.update({i: 0})  
  
for k, v in letters.items():  
 if v == 0:  
 letters[k] += 1  
keys = letters.keys()  
values = letters.values()  
plt.bar(keys, np.divide(list(values), sum(values)), color='green', label='Частотный анализ') # Строим гистограмму, np.divide находит отношение вхождения каждой буквы к общему числу букв  
plt.ylim(0, 2 \* max(np.divide(list(values), sum(values)))) # Нормируем вертикальную ось к единице  
plt.ylabel('Доля вхождений символа')  
plt.xlabel('Символы')  
plt.show()

Приступим к проверке шифротекста с помощью анализа открытого текста.

Следует сразу отметить, что способ умного перебора, использованный в качестве инструмента криптоанализа во второй лабораторной работе, в данном случае не подойдет, поскольку для шифра используются не только буквы латиницы и кириллицы, но и другие символы Unicode. В этой связи проверка строк по словарю не сработает, поскольку программа не найдет совпадений в символах.

Попробуем последовательно изменить несколько символов в исходном тексте, чтобы отследить изменения в шифротексте и попытаться выявить закономерность:

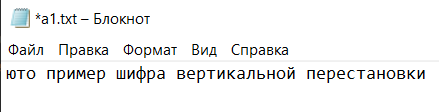
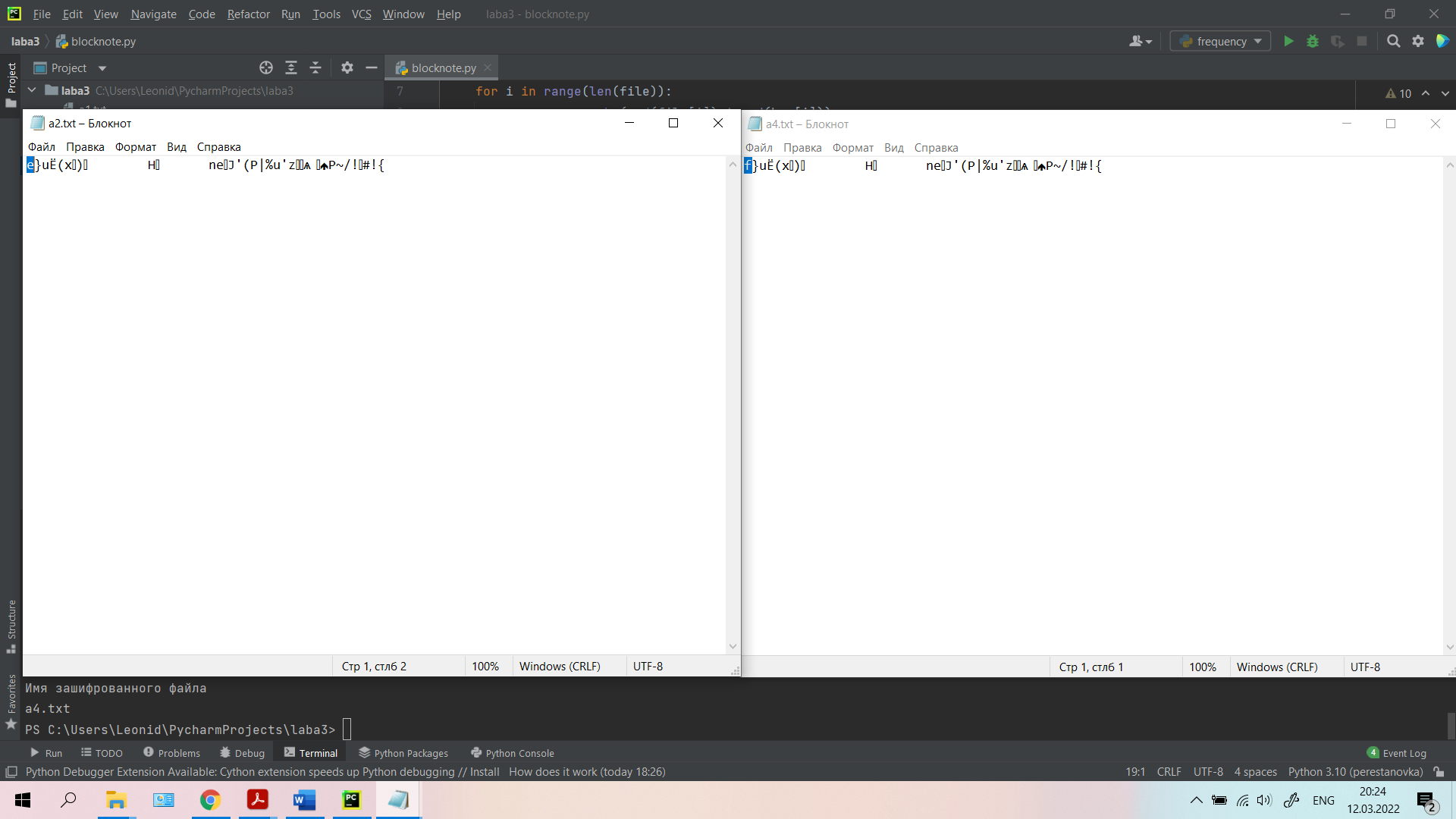
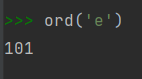
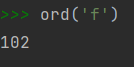


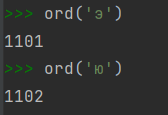
Рисунок 11 – Замена первого символа открытого текста



При замене буквы ‘э’ на букву «ю» получается линейный сдвиг шифра

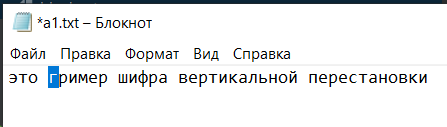


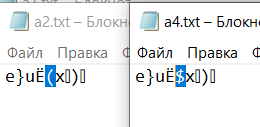


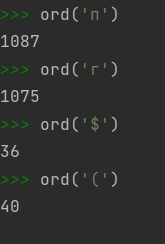


Как мы можем заметить, сдвиг ровно на 1000 позиций по таблице кодов.

Теперь сделаем ошибку в другом символе:







Как мы можем заметить, разница между значениями совершенно другая. Такая картина наблюдается для всех пар символов, уловить какую-либо закономерность невозможно, поэтому с помощью данного метода криптоанализа не получится подобрать однозначно распознать ключ.

Выводы:

В данной лабораторной работе мы закрепили навыки работы с файлами и строками в ЯП Python. Также мы убедились, что метод шифрования одноразовым блокнотом является очень надежным и защищенным, однако процесс составления ключа нельзя назвать однозначным. Также среди недостатков данного метода можно выделить то, что длина ключа должна совпадать с длиной открытого текста, который нужно зашифровать.