Лабораторная работа №3 ++

Программная реализация классических алгоритмов шифрования и их криптоанализа

Цель работы: ознакомление студентов с существующими криптографическими алгоритмами, используемые для защиты информации, формирование умений применять основные криптографические стандарты, протоколы и алгоритмы, формирование навыков прикладной и программной реализации криптоалгоритмов, а также владения криптографической терминологией.

Указания по выполнению лабораторной работы:

1. Ознакомиться с **Теоретическими сведениями** о криптографических алгоритмах и стеганографии.
2. Ознакомиться с примерами реализации криптографических алгоритмов на языке программирования Python.
3. На основе примеров разработать консольные приложения, реализующие указанные в задании алгоритмы шифрования. Разработанные программы должны иметь следующий функционал:
    a. Запрос вида криптографического преобразования – шифрование или расшифрование
    b. Ввод исходного/зашифрованного текста (в зависимости от выбранного вида криптографического преобразования) с помощью консоли либо при чтении из внешнего файла
    c. Ввод ключа шифрования
    d. Вычисление шифротекста/открытого текста согласно используемому алгоритму
    e. Вывод результата на экран либо запись в файл
4. Отчет о лабораторной работе должен содержать:
    a. Листинги разработанных программ с комментариями
    b. Скриншоты, иллюстрирующие работу программ как при шифровании, так и при расшифровании
    c. Ответы на контрольные вопросы

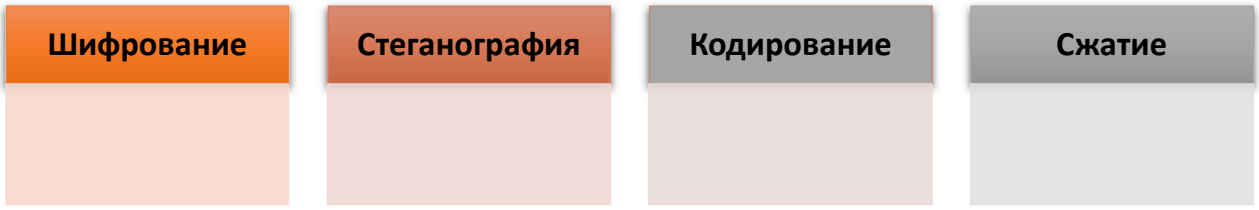
Теоретические сведения

Криптография представляет собой совокупность методов преобразования данных, направленных на то, чтобы сделать эти данные бесполезными для злоумышленника. Такие преобразования позволяют решить два главных вопроса, касающихся безопасности информации:

- защиту конфиденциальности;
- защиту целостности.

Проблемы защиты конфиденциальности и целостности информации тесно связаны между собой, поэтому методы решения одной из них часто применимы для решения другой.

Методы криптографического преобразования информации
(по виду воздействия на исходную информацию)



Сжатие информации может быть отнесено к методам криптографического преобразования информации с определенными оговорками. Целью сжатия является сокращение объема информации. В то же время сжатая информация не может быть прочитана или использована без обратного преобразования.

Учитывая доступность средств сжатия и обратного преобразования, эти методы нельзя рассматривать как надежные средства криптографического преобразования информации. Даже если держать в секрете алгоритмы, то они могут быть сравнительно легко раскрыты статистическими методами обработки.

Поэтому сжатые файлы конфиденциальной информации подвергаются последующему шифрованию. Для сокращения времени передачи данных целесообразно совмещать процесс сжатия и шифрования информации.

Содержанием процесса кодирования информации является замена исходного смысла сообщения (слов, предложений) кодами. В качестве кодов могут использоваться сочетания букв, цифр, знаков.

При кодировании и обратном преобразовании используются специальные таблицы или словари. В информационных сетях кодирование исходного сообщения (или сигнала) программно-аппаратными средствами применяется для повышения достоверности передаваемой информации.

Часто кодирование и шифрование ошибочно принимают за одно и тоже, забыв о том, что для восстановления закодированного сообщения, достаточно знать правило замены, в то время как для расшифровки сообщения помимо знания правил шифрования, требуется ключ к шифру.

Методы стеганографии позволяют скрыть не только смысл хранящейся или передаваемой информации, но и сам факт хранения или передачи закрытой информации.

В основе всех методов стеганографии лежит маскирование закрытой информации среди открытых файлов.

Скрытый файл также может быть зашифрован. Если кто-то случайно обнаружит скрытый файл, то зашифрованная информация будет воспринята как сбой в работе системы.

Комплексное использование стеганографии и шифрования многократно повышает сложность решения задачи обнаружения и раскрытия конфиденциальной информации.

Процесс шифрования заключается в проведении обратимых математических, логических, комбинаторных и других преобразований исходной информации, в результате которых зашифрованная информация представляет собой хаотический набор букв, цифр, других символов и двоичных кодов.

Для шифрования информации используются алгоритм преобразования и ключ.

Исходными данными для алгоритма шифрования служит информация, подлежащая зашифрованию, и ключ шифрования.

Ключ содержит управляющую информацию, которая определяет выбор преобразования на определенных шагах алгоритма и величины операндов, используемых при реализации алгоритма шифрования.

Шифрование является основным видом криптографического преобразования информации в компьютерных сетях.

Процесс преобразования открытой информации в закрытую получил название зашифрование, а процесс преобразования закрытой информации в открытую – расшифрование.

Атака на шифр (криптоанализ, криптоатака) – это процесс расшифрования закрытой информации без знания ключа и, возможно, при отсутствии сведений об алгоритме шифрования.

Современные методы шифрования должны отвечать следующим требованиям:

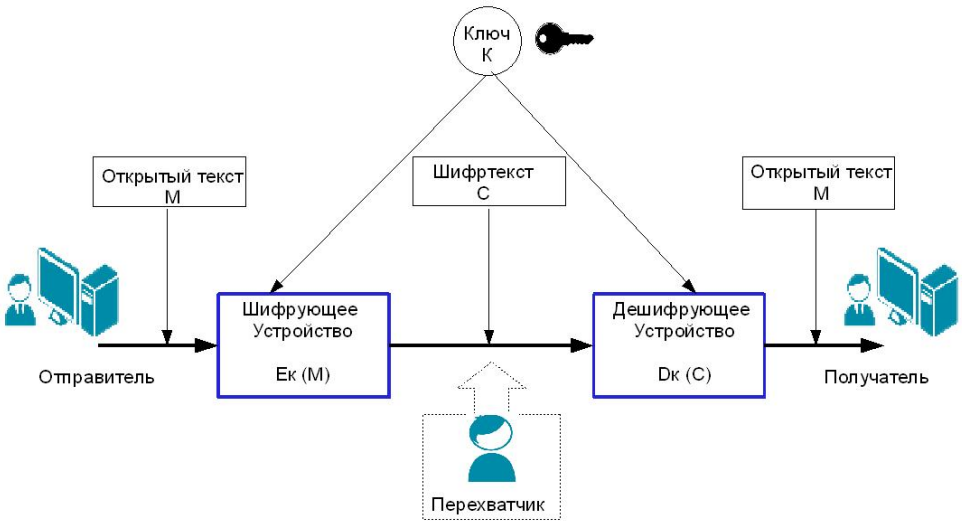
- стойкость шифра противостоять криптоанализу (криптостойкость) должна быть такой, чтобы вскрытие его могло быть осуществлено только путем решения задачи полного перебора ключей;
- криптостойкость обеспечивается не секретностью алгоритма шифрования, а секретностью ключа;
- шифртекст не должен существенно превосходить по объему исходную информацию;
- ошибки, возникающие при шифровании, не должны приводить к искажениям и потерям информации;
- время шифрования не должно быть большим;
- стоимость шифрования должна быть согласована со стоимостью закрываемой информации.

Преобразование шифрования может быть симметричным и асимметричным относительно преобразования расшифрования.

Это важное свойство определяет два класса криптосистем:

- симметричные (одноключевые) криптосистемы;
- асимметричные (двухключевые) криптосистемы (с открытым ключом).

Симметричное шифрование, которое часто называют шифрованием с помощью секретных ключей, в основном используется для обеспечения конфиденциальности данных.

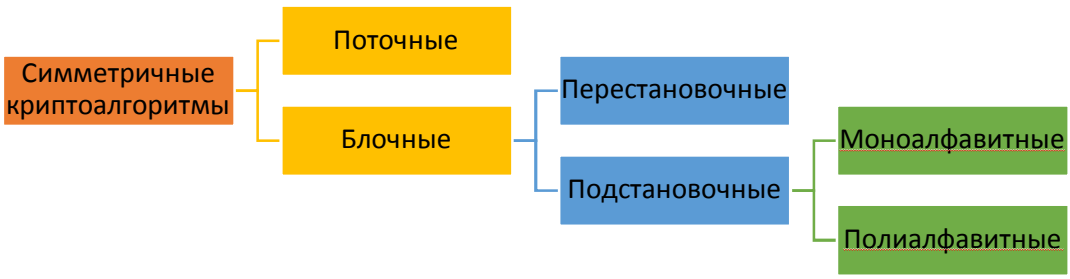


Для этого пользователи должны совместно выбрать единый математический алгоритм, который будет использоваться для шифрования и расшифровки данных. Кроме того, им нужно выбрать общий (секретный) ключ, который будет использоваться с принятым ими алгоритмом зашифрования и расшифрования.

С методом симметричного шифрования связаны следующие проблемы:

- необходимо часто менять секретные ключи, поскольку всегда существует риск их случайного раскрытия (компрометации);
- достаточно сложно обеспечить безопасность секретных ключей при их генерировании, распространении и хранении.

Классификация симметричных криптоалгоритмов



Предварительное задание для выполнения лабораторной работы

1. Изучить принцип действия классических методов шифрования:
    a. Шифр Цезаря
    b. Шифр перестановки
    c. Метод гаммирования
    d. Шифр Виженера
    e. Шифр Полибия
    f. Шифр Плейфера
    g. Аффинный шифр
2. Установить среду разработки для программирования на языке Python версии 3.7 и выше
   https://www.python.org/downloads/windows/

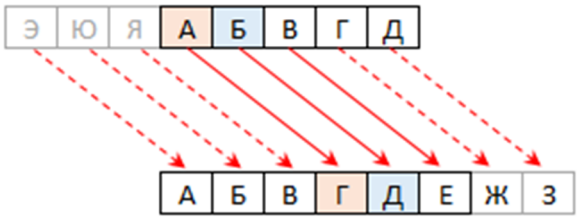
Пример реализации криптографических алгоритмов на языке программирования Python.

В качестве примера программной исполнения криптографических алгоритмов рассмотрим реализацию в двух вариантах шифра Цезаря на языке программирования Python.

Программная реализация шифра Цезаря (вариант 1)

Для составления программы, исполняющей криптографические преобразования с текстом, напомним принцип действия шифра Цезаря.

Шифр Цезаря является классическим примером шифра подстановки (замены). Он задается подстановкой следующего вида (применительно к русскому алфавиту):



А	Б	В	Г	Д	Е	Ё	Ж	З	И	Й	К	Л	М	Н	О	П	Р	С	Т	У	Ф	Х	Ц	Ч	Ш	Щ	Ы	Ь	Ъ	Э	Ю	Я
Г	Д	Е	Ё	Ж	З	И	И	К	Л	М	Н	О	П	Р	С	Т	У	Ф	Х	Ц	Ч	Ш	Щ	Ы	Ь	Ъ	Э	Ю	Я	А	Б	В

Верхний алфавит – это алфавит отрытого (исходного) текста. Нижний алфавит, представляющий собой циклический сдвиг верхнего алфавита влево на 3 шага, есть алфавит шифрованного текста. Буква А открытого текста

заменяется на букву Г в шифрованном, буква Б - на букву Д и т.д. В наши дни под шифром Цезаря понимаются все шифры, в которых нижняя строка является циклическим сдвигом верхней на произвольное число шагов. Но поскольку этот сдвиг не меняется в процессе шифрования, то шифр остается шифром простой (одноалфавитной) замены.

Для того, чтобы текст программы разработанного приложения в дальнейшем было удобно читать, редактировать и совершенствовать, мы будем применять модульный подход. Разрабатываемое нами консольное приложение будет состоять из следующих модулей:

- основной модуль **main.py**

- вспомогательный модуль с описанием функций шифрования и расшифрования **functions.py**

Файлы данных модулей необходимо поместить в заранее созданную папку **lab_rabota**.

В основном модуле будут осуществляться следующие действия:

- определение алфавита, из букв которого будут состоять исходные и зашифрованные тексты

- ввод ключа шифрования

- выбор пользователем вида криптографического преобразования

- ввод исходного/зашифрованного текста в зависимости от выбранного вида преобразования (если выбрано шифрование, то пользователю нужно будет ввести исходный текст, в противном случае пользователь введет зашифрованный текст)

- вызов функции шифрования или расшифрования, в зависимости от выбора пользователя.

Модуль с функциями будет содержать описание функций шифрования и расшифрования. Очевидно, что данные функции будут иметь сходную структуру и выполнять обратные друг другу криптографические преобразования. В рамках выполнения каждой из функций, приложение будет запрашивать у пользователя исходный/зашифрованный текст, выполнять шифрование/расшифрование и выводить результат на экран.

Начнем разработку приложения с написания модуля **main.py.** Для того, чтобы внутри него использовать функции модуля **functions.py**, необходимо их импортировать. Для этого напишем:

```
from functions import encryption, decryption
```

Как было указано выше, для шифрования и расшифрования необходимо осуществлять циклический сдвиг алфавита на определенное количество символов. В базовом варианте программной реализации шифра Цезаря будем

использовать только русский алфавит, поэтому первую строчку в нашей программе запишем следующим образом:

```
alpha = 'абвгдеёжзийклмнопрстуфхцчшщьыъэюя'
```

Записав в переменную **alpha** строку, состоящую из последовательности букв английского алфавита, мы далее можем производить сдвиг на количество символов, указанных пользователем в качестве ключа шифрования.

Для ввода в консоль ключа шифрования **Key** используем команду **input()**. Но нужно иметь в виду, что, если даже пользователь введет число, тип введенных данных будет строковым <str>. А для шифра Цезаря ключом шифрования является количество букв для циклического сдвига, то есть только целое число. Поэтому мы дополнительно применяем функцию **int()**, которая преобразует строку в целое число. Кроме того, функция **int()** «не пропустит» дальше и дробные числа (отбросит дробную часть).

```
Key = int(input("Enter key: "))
```

Следующее действие по плану – выбор вида криптографического преобразования, то есть мы должны спросить пользователя, что он намерен сделать – зашифровать или расшифровать текст. Для этого пользователь вводит букву, соответствующую необходимому виду преобразования:

```
operation = input("Введите 'ш' для шифрования, 'р' для расшифрования: ")
```

Теперь используя оператор условного перехода в зависимости от введенной буквы попросим пользователя ввести исходный/зашифрованный текст и выведем на экран результат выполнения соответствующей функции:

```
if operation == 'ш':
    PlainText = input("Введите текст для шифрования: ").strip()
    CipherText = encryption(Key, PlainText, alpha)
    print("Шифротекст: " + CipherText)
elif operation == 'р':
    CipherText = input("Введите текст для расшифрования: ").strip()
    PlainText = decryption(Key, CipherText, alpha)
    print("Исходный текст: " + PlainText)
else:
    print("Выбрана неверная операция")
```

В случае, если пользователь введет неверную букву, будет выведено соответствующее сообщение.

Как вы могли заметить, при вводе пользователем исходного или зашифрованного текста использована дополнительная функция **strip()**, которая удаляет лишние пробелы в начале и конце строки.

На этом написание программы основного модуля завершим и приступим к описанию функций шифрования и расшифрования в модуле **functions.py.**

Функция шифрования **encryption** принимает на вход три обязательных аргумента (ключ шифрования (**Key**), исходный текст (**PlainText**), используемый алфавит **alpha**) и возвращает зашифрованный текст (**CipherText**).

Для начала в теле функции **encryption** объявим переменную **CipherText**, в которую будет записываться зашифрованный текст, и присвоим ей значение, состоящее из пустой строки.

```
def encryption(Key, PlainText, alpha):
    CipherText = ''
```

Далее с помощью цикла будем последовательно зашифровывать каждую букву исходного текста и конкатенировать результат шифрования с переменной **CipherText**:

```
for p in PlainText:
    CipherText += alpha[(alpha.index(p) + Key) % len(alpha)]
```

Поясним, какие операции включает в себя строка, находящаяся в теле цикла:

1) **p** – очередная буква исходного текста

2) получение индекса (порядкового номера) буквы исходного текста в алфавите с помощью функции **alpha.index(p)**

3) сложение полученного индекса со смещением (ключом шифрования) - **alpha.index(p) + key**

4) **len(alpha)** – определение мощности алфавита (общее количество букв в алфавите)

5) получение остатка от деления полученного индекса на мощность алфавита - **(alpha.index(p) + key) % len(alpha)**, так как результат сложения на предыдущем этапе может оказаться больше, чем мощность алфавита

6) получение буквы с вычисленным индексом:

alpha[(alpha.index(p) + key) % len(alpha)]

7) конкатенация полученной буквы с переменной **CipherText:**

CipherText += alpha[(alpha.index(p) + Key) % len(alpha)]

После того, как мы получили зашифрованный текст, нам необходимо вывести результат выполнения функции **encryption**:

```
return CipherText
```

Задание.

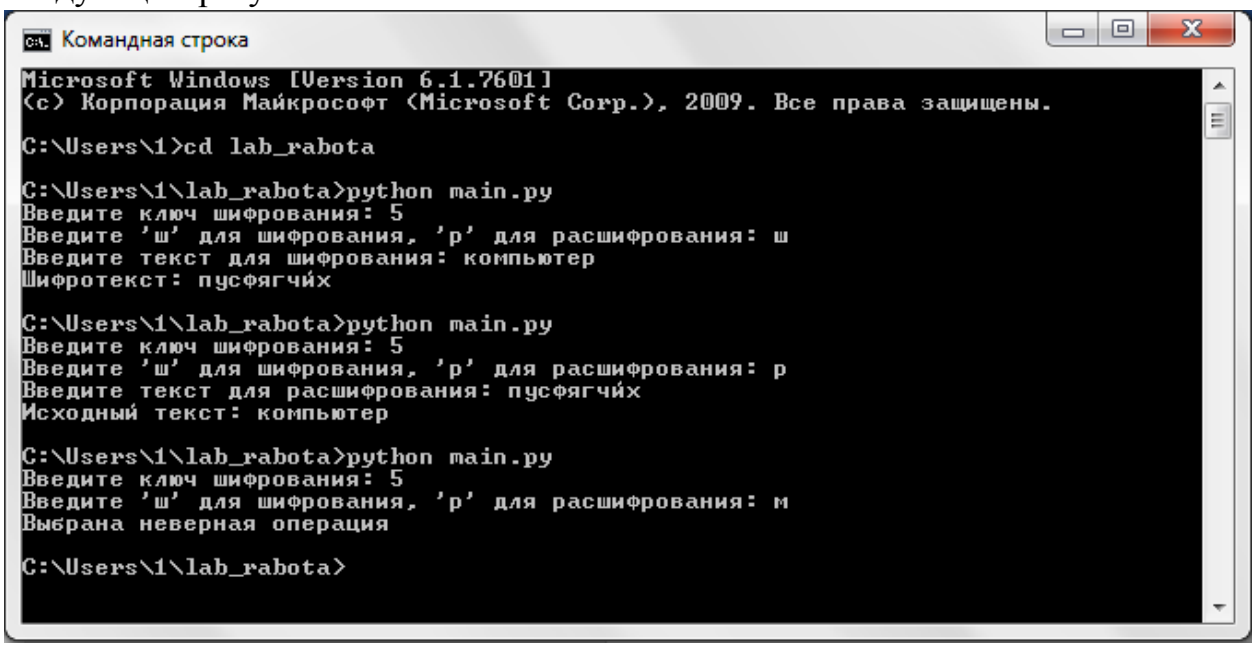
По аналогии с описанием функции шифрования, напишите самостоятельно обратную ей функцию - функцию дешифрования **decryption**, принимающую на вход три аргумента (ключ шифрования **Key**, зашифрованный текст **CipherText,** используемый алфавит **alpha**) и возвращающей исходный текст **PlainText**.

Тестирование программы

Для тестирования разработанной программы запустим командную строку **cmd.exe**, зайдем в папку **lab_rabota** с файлом программы и выполним следующую команду:

```
python main.py
```

Если программа была написана правильно, то мы должны получить следующий результат:



Задание.

1. Внесите в программу следующие изменения:
    a. Измените язык используемого алфавита на английский
    b. Добавьте цифры, знаки препинания.
2. Протестируйте работу программы.

Программная реализация шифра Цезаря (вариант 2)

Очевидно, что предыдущий вариант реализации шифра Цезаря имеет ограниченный функционал и следующие недостатки:

1) Необходимо вручную задавать алфавит, в рамках которого происходит шифрование и расшифрование.
2) Ввод и вывод пользовательской информации происходит только в консоли

Для того, чтобы исправить перечисленные недостатки, добавим возможность шифрования и расшифрования текстов, состоящих из символов без привязки к конкретному алфавиту и регистру, а также учитывающая наличие других специальных символов. В качестве алфавита здесь будет выступать таблица кодов символов ASCII.

Таблица ASCII определяет коды для символов:

- десятичных цифр;
- латинского алфавита;
- национального алфавита (в нашем случае русского)
- знаков препинания;
- управляющих символов.

Модификация предыдущей программы коснется главным образом модуля **`functions.py`**, основной модуль будет изменен незначительно. Создадим копию модуля **`functions.py`** и сохраним его под именем **`functions_v2.py`**.

Так как вместо задаваемого вручную алфавита мы будем использовать таблицу кодов ASCII, то надобность в третьем аргументе **`alpha`** для функций **`encryption()`** и **`decryption ()`** исчезает.

Для того, чтобы получить код какого-либо символа из таблицы ASCII, необходимо в нужную переменную записать результат выполнения функции **`ord(letter)`**, где **`letter`** – это символ. Обратное преобразование – получение символа из кода – происходит с помощью функции **`chr(code)`**, где **`code`** – это код искомого символа.

По аналогии с предыдущим вариантом шифрование будет происходить путем сложения индекса очередного символа шифруемого сообщения со смещением **`Key`**:

```
def encryption(Key, PlainText):
    CipherText = ''
    for p in PlainText:
        code = ord(p)+Key
        CipherText += chr(code)
```

```
    return CipherText
```

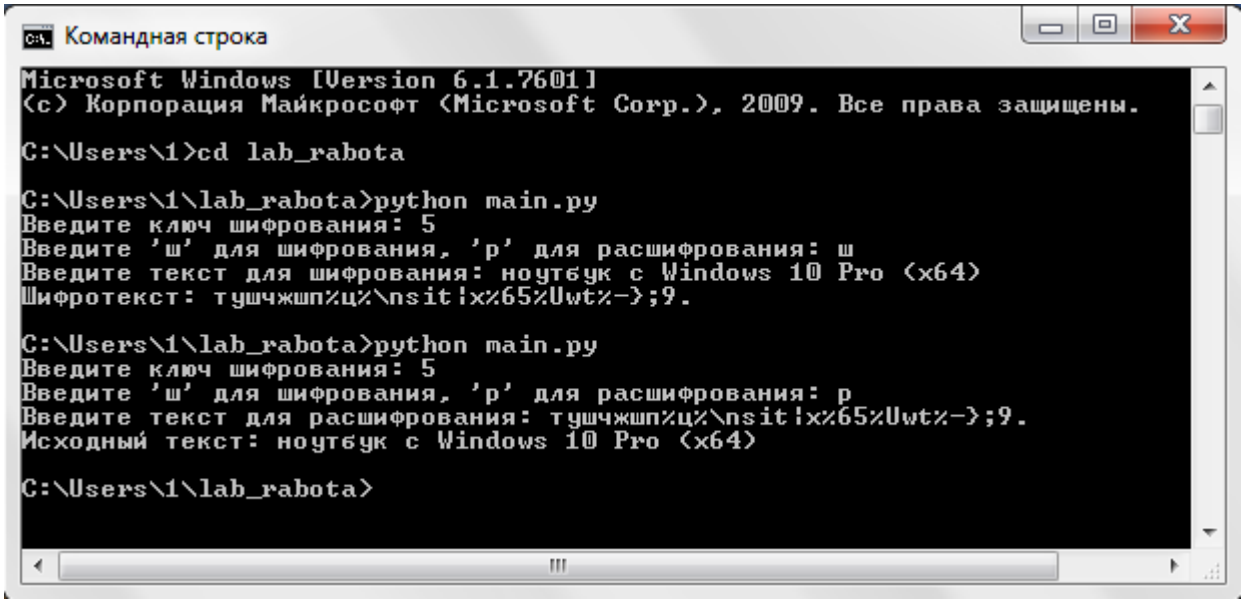
Соответственно функция расшифрования будет использовать вычитание из кода символа шифротекста значение смещения **Key** и дальнейшего преобразования кода в символ исходного текста:

Задание.

По аналогии с описанием функции шифрования, напишите самостоятельно обратную ей функцию - функцию дешифрования **decryption**, принимающую на вход два аргумента (ключ шифрования **Key**, зашифрованный текст **CipherText**) и возвращающей исходный текст **PlainText**.

Теперь, когда готовы к вызову функции шифрования и расшифрования, скорректируем основной модуль **main.py**. При вызове функций **encryption()** и **decryption()** из числа аргументов необходимо исключить **alpha**, а также в первой строке вместо модуля **functions.py** прописать **functions_v2.py**.

Теперь все готово для тестирования программы. В командной строке запускаем скрипт **main.py**. Текст для шифрования должен содержать русские и латинские буквы, имеющие разный регистр, а также числа и знаки препинания. В указанном на рис. X примере выбран исходный текст «ноутбук с Windows 10 Pro (x64)». Повторите действия согласно данному примеру. В результате расшифрования должен получиться исходный текст. Для того, чтобы скопировать из консоли шифротекст необходимо в поле консоли нажать правую кнопку мыши, из контекстного меню выбрать команду «Пометить», выделить левой кнопкой мыши нужный фрагмент текста и нажать «Enter». Затем установить курсор в том месте, где нужно вставить текст, снова вызвать контекстное меню и выбрать команду «Вставить».



Таким образом, мы избавились от необходимости вручную задавать алфавит и расширили возможности программы в части используемых символов для шифрования и расшифрования.

Но по-прежнему текст вводили вручную с консоли. Добавим программе возможность шифрования текста из какого-либо файла с расширением .txt и запись результата шифрования и расшифрования в этот же или другой файл, при этом сохранив возможность работы с консолью. Для этого нам необходимо создать дополнительные функции **reading() и writing().**

Функция **reading()** будет сначала запрашивать у пользователя способ получения исходного текста (из консоли или файла). Если пользователь выберет чтение из файла, то функция дополнительно запросит у пользователя имя файла с расширением .txt, из которого она получит текст вернет его для дальнейших преобразований. Если пользователь выберет чтение с консоли, то функция вернет текст, введенный с консоли.

Функция **writing(data, operation)** принимает на вход текст и вид операции, который предшествовал вызову функции, т.е. шифрование или расшифрование. Подобно функции **reading()** здесь также будет предусмотрена возможность работы с консолью (вывод данных в консоль).

Приведем код функции **reading()**

```
def reading():
	source = input("Источник данных: \n 'к' - консоль \t 'ф' - файл \n")
	if source == 'к':
		data = input("Введите текст: ").strip()
```

```
            return data
      elif source == 'ф':
            filename = input("Название файла с данными в формате '*.txt': \n")
            with open(filename, 'r') as f:
                  data = f.read()
            data = data.encode('cp1251').decode('cp1251')
            return data
      else:
            print("[-] Выбрана неверная операция")
```

Выбор способа получения данных реализован в конструкции условного перехода после того, как пользователь ввел в консоль букву, соответствующую нужному источнику.

Первая часть приведенной здесь конструкции условного перехода, начинающаяся после команды **if**, необходима для предоставления возможности ввода данных в консоль. После выполнения строки кода «**return data**» введенные пользователем данные возвращаются для дальнейших преобразований.

Во второй части конструкции условного перехода, начинающейся после команды **elif** и заканчивающейся также строкой «**return data**», реализована возможность чтения данных из файла. Сперва нам необходимо получить название файла, из которого будет происходит чтение, и записать его в переменную **filename**:

```
filename = input("Название файла с данными в формате '*.txt': \n")
```

Далее необходимо открыть данный файл, прочитать из него текст, и записать текст в переменную **data**, чтобы функция **reading()** могла передать его далее в качестве аргумента функциям шифрования или дешифрования.

Наиболее подходящей конструкцией для чтения данных из файла будет контекстный менеджер **with … as**, в котором задействована встроенная функция **open()**.

```
with open(filename, 'r') as f:
      data = f.read()
```

У функции **open()** много параметров, нам пока важны 3 аргумента: первый, это имя файла. Путь к файлу может быть относительным или абсолютным. Второй аргумент, это режим, в котором мы будем открывать файл.

Режим	Расшифровка
`'r'`	открытие на чтение (является значением по умолчанию)
`'w'`	открытие на запись, содержимое файла удаляется, если файла не существует, создается новый
`'x'`	открытие на запись, если файла не существует, иначе исключение.
`'a'`	открытие на дозапись, информация добавляется в конец файла
`'b'`	открытие в двоичном режиме
`'t'`	открытие в текстовом режиме (является значением по умолчанию)
`'+'`	открытие на чтение и запись

Режимы могут быть объединены, то есть, к примеру, **`'rb'`** - чтение в двоичном режиме. По умолчанию режим равен **`'r'`**.

И последний аргумент, **`encoding`**, нужен только в текстовом режиме чтения файла. Этот аргумент задает кодировку.

После открытия файла необходимо прочитать из него информацию. Для этого будем использовать метод **`read()`**, читающий весь файл целиком, если был вызван без аргументов, и n символов, если был вызван с аргументом (целым числом n).

Строка

```
data = data.encode('cp1251').decode('cp1251')
```

предназначена для представления данных из файла в кодировке Windows 1251.

Теперь рассмотрим функцию записи в файл **`writing(data, operation)`**. Ниже представлен код данной функции.

```
def writing(data, operation):
	op = operationtype(operation)
	exit = input("Вид вывода данных: \n 'к' - консоль \t 'ф' - файл: \n")
	if exit == 'к':
		print("Результат: ", data)
	elif exit == 'ф':
		string = f"Файл для вывода результата {op} в формате 'имя.txt': \n"
		filename = input(string)
		with open(filename, 'w') as f:
			for simbol in data:
				try:
					f.write(i.encode("cp1251").decode("cp1251"))
				except:
```

```
                        print(i)
                        pass
            print("\n[+] Результат %s записан в файл %s" %(operationtype(operation), filename))
      else:
            print("[-] Выбрана неверная операция")
```

Функция **writing(data, operation)** принимает на вход два аргумента:

- **data –** данные для записи в файл, например после шифрования

- operation – вид криптографического преобразования (шифрование или расшифрование)

Аналогично функции **reading()** здесь используется условная конструкция, позволяющая выбрать способ вывода данных (в консоль или в файл). Во второй части условной конструкции в контекстном менеджере **with … as** функция **open()** работает в режиме записи.

```
with open(filename, 'w') as f:
```

В отличие от метода **read()**, где файл считывался целиком, запись будем производить посимвольно с помощью цикла **for**:

```
for symbol in data:
```

Это позволяет избежать остановки программы из-за ошибок декодирования некоторых символов. Для того, чтобы пропустить символы, декодирование которых сопровождается исключением *UnicodeDecodeError* в программе использован обработчик исключений **try…except…** В блоке **try** мы выполняем инструкцию, которая может породить исключение, а в блоке **except** мы перехватываем их и на месте пропущенного символа ставим пробел или любой другой символ, далее переходим к следующей итерации, т.е. к следующему символу.

```
try:
      f.write(i.encode("cp1251").decode("cp1251"))
except:
      f.write(‘ ‘)
```

В процессе выполнения программы пользователю нужно вводить имя файла, в который будет произведена запись результата шифрования или расшифрования. Чтобы пользователь по ошибке не перепутал файлы,

приглашение для ввода их имен нужно формулировать по-разному. Например, когда нужно произвести запись данных после шифрования, сообщение для пользователя будет сформулировано следующим образом:

```
Файл для вывода результата шифрования в формате 'имя.txt':
```

А после расшифрования:

```
Файл для вывода результата расшифрования в формате 'имя.txt':
```

В данной строке изменяется только одно слово. Напишем вместо него переменную **op** в круглых скобках:

```
string = f"Файл для вывода результата {op} в формате 'имя.txt': \n"
```

Буква f впереди строки позволяет подставлять значение строковой переменной **op** чтобы, строка имела нужную нам формулировку.

Как вы могли заметить, в теле функции **writing()** используется вспомогательная функция **operationtype(operation)**, которая возвращает строку «шифрования» если была выбрана операция шифрования (пользователь ввел в консоль «ш»), и «расшифрования» - когда пользователь ввел «р». Результат функции записывается в переменную **op**. Код данной функции приведен ниже:

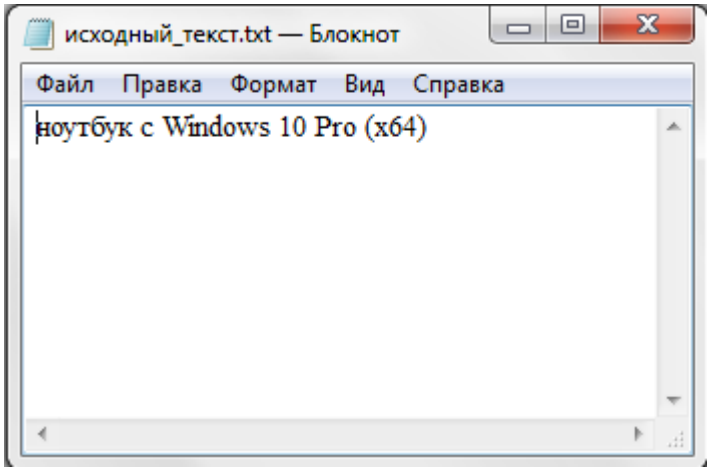
```
def operationtype(operation):
      sh = "шифрования"
      rsh = "расшифрования"
      if operation == "ш":
            return sh
      elif operation == "p":
            return rsh
```

Теперь необходимо внести изменения в основной модуль **main.py.** Код данного модуля приведен ниже:

```
from functions_v2 import encryption, decryption, reading, writing,
Key = int(input("Введите ключ шифрования: "))
operation = str(input("Введите 'ш' для шифрования, 'p' для расшифрования: "))
if operation == 'ш':
      PlainText = reading()
      CipherText = encryption(Key, PlainText)
      print("\n[+] Шифрование завершено\n")
      writing(CipherText, operation)
elif operation == 'p':  # если необходимо расшифровать текст
      CipherText = reading()
      PlainText = decryption(Key, CipherText)
      print("\n[+] Расшифрование завершено\n")
      writing(PlainText, operation)
```

```
else:
      print("[-] Выбрана неверная операция")  # если введена неверная операция
```

Протестируем созданную программу. Текст «ноутбук с Windows 10 Pro (x64)». Для этого создадим в папке **lab_rabota** текстовый файл **исходный текст.txt** и запишем в него текст «ноутбук с Windows 10 Pro (x64)».

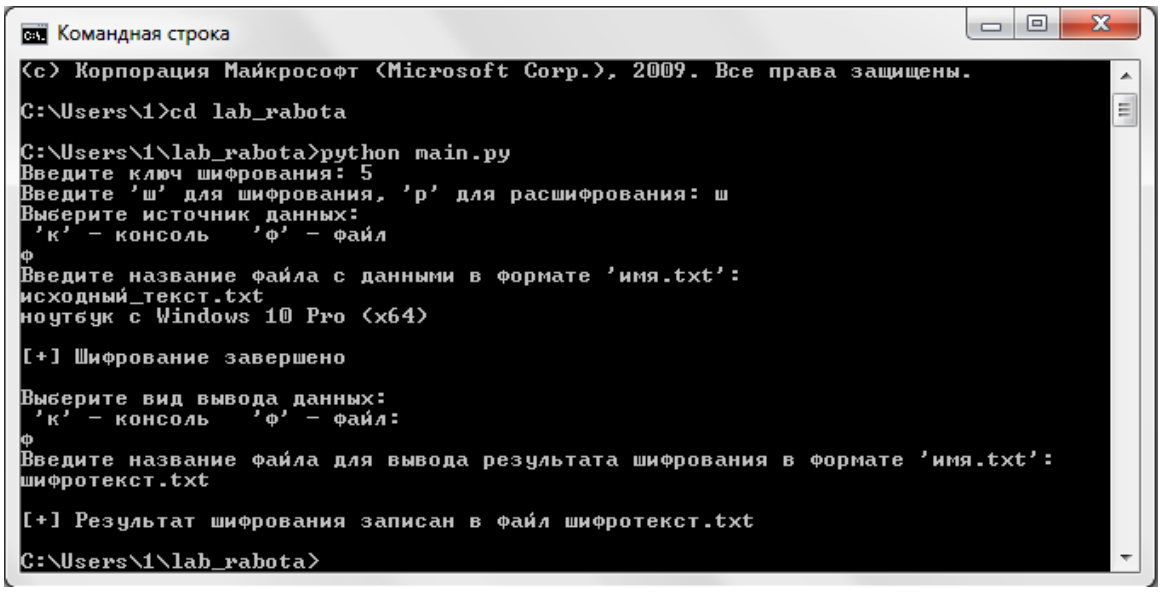


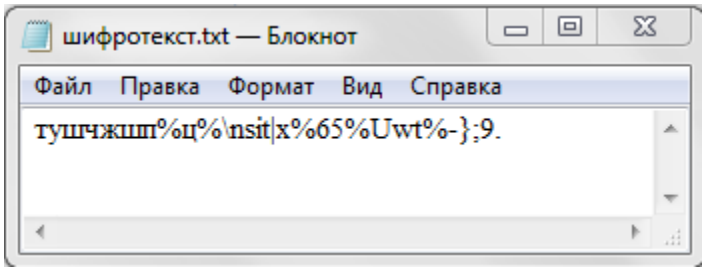
В командной строке запускаем скрипт **main.py**. Вводим ключ шифрования, выбираем тип преобразования – шифрование, в качестве источника данных выбираем файл и указываем его имя с расширением.

Далее нам необходимо ввести имя файла для записи зашифрованных данных.

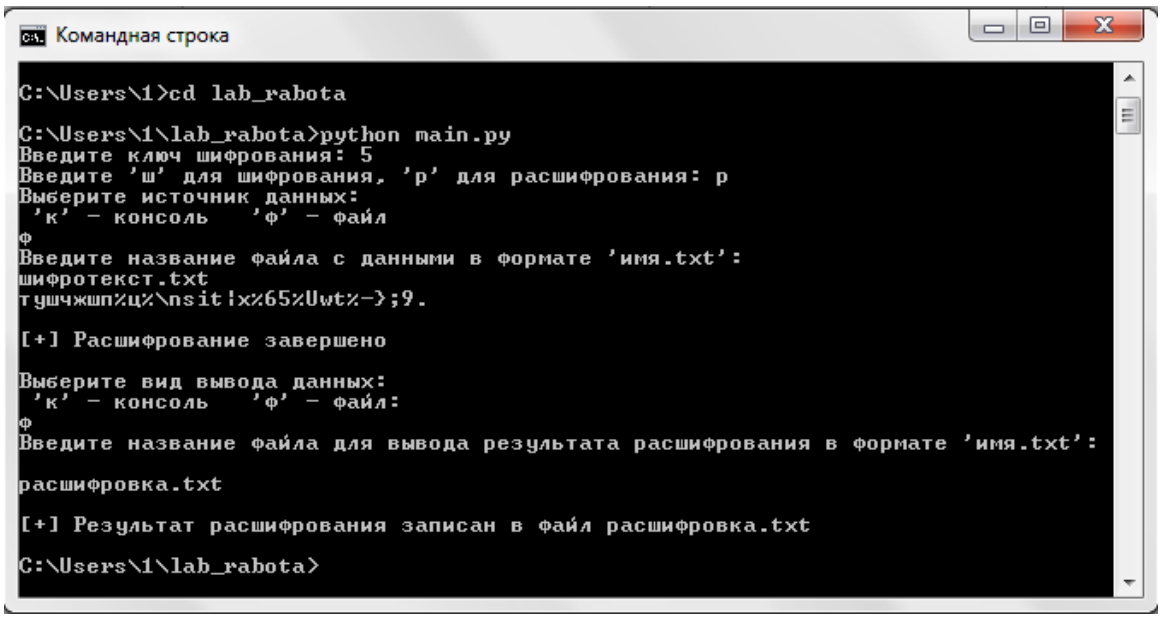
Введем шифротекст.txt.

Результат выполнения программы представлен на рис. X. Также откроем файл шифротекст.txt для просмотра его содержимого.

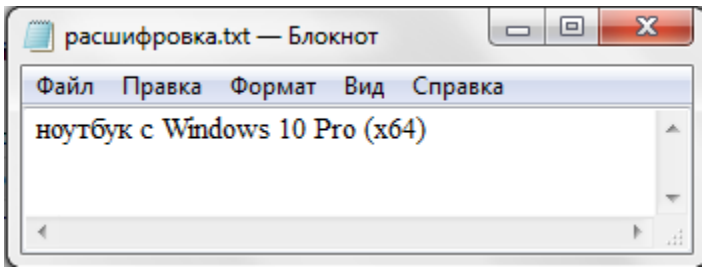




Далее протестируем обратную операцию – расшифрование. Результат расшифрования запишем в файл расшифровка.txt



Файл расшифровка.txt будет иметь следующее содержимое:



Таким образом, была разработана и протестирована программа для шифрования и расшифрования методом Цезаря, предусматривающая возможность работы с консолью и содержимым файлов.

Пример программной реализации криптоанализа шифра Цезаря методом протяжки вероятного слова на языке программирования Python.

В данном разделе напишем программу для реализации атаки на шифр Цезаря методом протяжки вероятного слова. Данный метод заключается в том, что на основании дополнительных данных, предполагается наличие в одном из открытых текстов конкретного слова. После фиксации слова, последовательно начиная с первой позиции оно подставляется в текст, при этом однозначно восстанавливается отрезок второго текста, который (в случае правильного позиционирования слова в первом тексте) можно продолжить по соображениям читаемости, в этом случае в первом тексте восстанавливается новый отрезок, который также можно продлить и т.д. Проиллюстрируем вышесказанное с помощью следующей схемы:

Рис. X

В качестве вероятных слов могут быть наиболее распространенные слова языка, на котором был написан исходный текст (такими словами как правило являются предлоги), а также термины или понятия, соответствующие

и другие служебные части речи

вероятной тематике исходного текста. Вычисляя разность между индексами символов (позициями в алфавите) шифротекста и вероятного слова, необходимо находить закономерности для нескольких подряд идущих символов. Если для двух и более символов смещение одинаково, можно сделать предположение о размере ключа и применить его для расшифрования. Очевидно, что если предположение сделано верно, то в результате операции расшифрования мы получим исходный текст. В противном случае необходимо перейти к следующему фрагменту шифротекста или новому вероятному слову.

Модульный подход, использованный нами при разработке предыдущих программ, позволит использовать готовые функции расшифрования текстов **decryption()**, чтения **reading()** и записи **writing()** их в файл. Поэтому в первую очередь импортируем их в новый модуль **attack.py**:

```
from functions import decryption, reading, writing
```

Получим из внешнего файла зашифрованный текст:

```
ciphertext = reading()
```

Составим список наиболее распространенных слов:

```
voc = [' и ',' в ',' не ',' на ',' что ',' по ',' к ',' но ',' у ',' как ']
```

Этот список можно значительно расширить для повышения эффективности атаки на шифр, но для данной работы мы ограничимся только самыми распространенными служебными частями речи.

Далее нам нужен алгоритм, который перебирает слова из списка **voc** для передачи их в качестве аргумента в некоторую функцию **finding()**(в программном коде ниже вместо описания функции пока поставим команду-заглушку **pass**), возвращающей значение предполагаемого смещения, а после запускает функцию расшифрования. Данный алгоритм можно реализовать следующим образом:

```
def finding(word, ciphertext):
        pass
for word in voc:
        key = finding(word, ciphertext)
        if key != None:
         print(decryption(key, ciphertext))
         break
```

Возвращаемое функцией **finding()** значение записывается в переменную **key** и используется далее для расшифрования текста с помощью функции **decryption(key, ciphertext)**. Команда **break** останавливает цикл **for…in** при наличии хотя бы одного полученного ключа шифрования.

Рассмотрим подробнее функцию **finding(word, ciphertext)**.

Данная функция принимает на вход два аргумента:

word – вероятное слово из списка **voc**,

ciphertext – шифротекст, полученный путем шифрования исходного текста методом одноалфавитной замены (шифр Цезаря)

Программный код функции:

```
1 def finding(word, ciphertext):
2    for i in range(len(ciphertext)-len(word)):
3        g = [ord(ciphertext[i+j])-ord(word[j]) for j in range(len(word))]
4        s = 0
5        for k in range(len(g)):
6            if g[k] == g[0]:
7                s +=1
8        if s == len(g):
9            return g[0]
```

В строке №2 с помощью цикла **for…in…** происходит перебор индексов символов в шифротексте, начиная с нулевого и заканчивая индексом, полученным в результате разности количества символов шифротекста **len(ciphertext)** и количества символов вероятного слова **len(word)**. Значение индекса **i** в каждой итерации данного цикла будет использовано далее как начало очередного фрагмента шифротекста, сравниваемого с вероятным словом.

В строке №3 происходит генерация списка разностей между позициями в алфавите (таблице кодировки) каждой пары символов шифротекста и вероятного слова. Например, согласно схеме на рис. X для первых пяти символов начиная с первого символа (но индекс у него равен 0), так как нумерация по умолчанию начинается с 0) шифротекста будет сгенерирован следующий список разностей – [5,5,5,5,5]. В таблице X показано каким образом данный список был получен.

Символ шифротекста	Индекс символа шифротекста	Символ вероятного слова	Индекс символа вероятного слова	Разность
ц	24	с	19	5

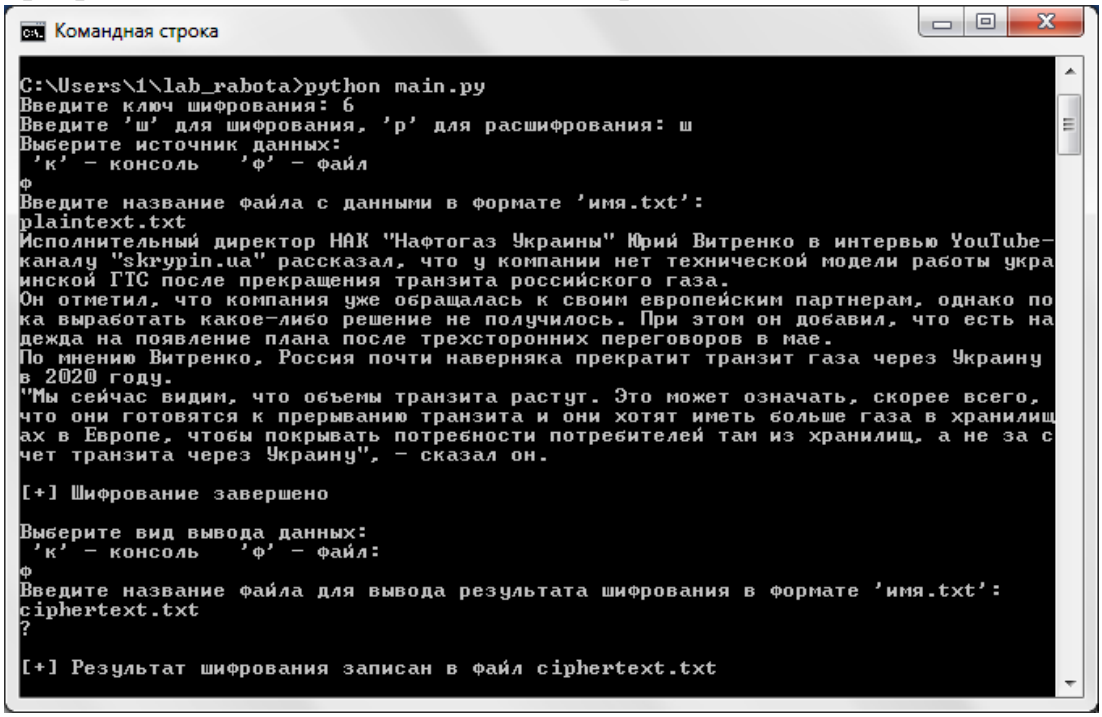
р	18	л	13	5
у	21	о	16	5
ж	8	в	3	5
у	21	о	16	5

Для пятерки символов шифротекста, начинающихся со второго (индекс равен 1) символа список разностей будет другим – [5,5,5,5,1], соответственно для пятерки, начинающейся с третьего – [5,5,5,1,-1] и т.д.

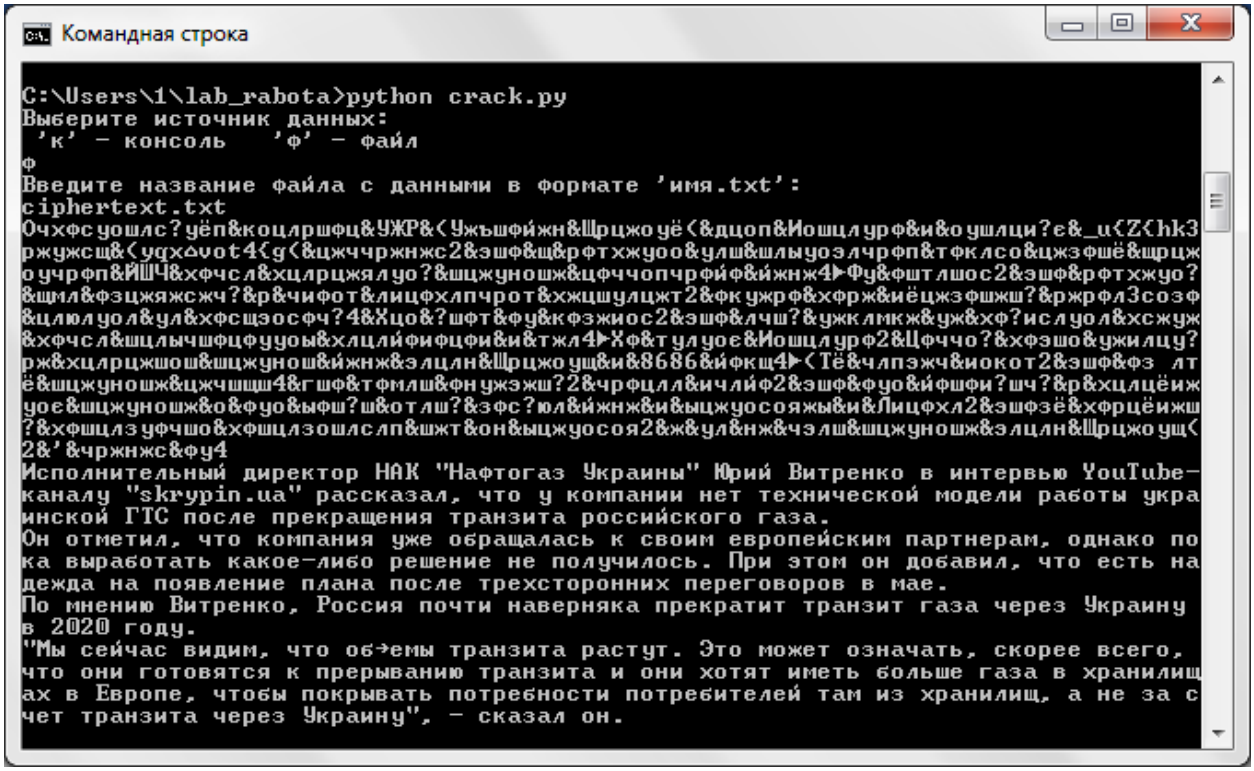
После того, как мы получили список разностей, необходимо проверить, равны ли друг другу все элементы данного списка, то есть прослеживается ли одинаковое смещение между символами проверяемого фрагмента зашифрованного текста и вероятного слова. Если данное условие выполняется, то можно возвратить предполагаемый ключ шифрования для дальнейшего расшифрования. Проверка на наличие одинакового смещения происходит с 4 по 9 строку описания функции **`finding()`**. С помощью цикла **`for…in…`** (строка 5) сравниваем (строка 6) каждый элемент текущего списка разностей **`g[k]`** с нулевым элементом **`g[0]`**. Если они равны, счетчик **`s`** инкрементируется (строка 7). Равенство значения счетчика и количества элементов списка (то есть количества символов вероятного слова) является условием (строка 8) возвращения функцией **`g[0]`** в качестве предполагаемого ключа шифрования (строка 9). Если данное условие не выполняется, то функция возвращает **`None`**.

Тестирование программы

Протестируем программу на примере достаточно большого текста. Для начала зашифруем текст из файла **`plaintext.txt`** с помощью разработанной ранее программы **`main.py`** и запишем его в файл **`cipertext.txt`**.



Теперь реализуем атаку на зашифрованный текст с помощью программы **crack.py**



Программа сначала выводит в консоль зашифрованный текст, затем предполагаемый исходный текст. Как можете заметить, атака прошла успешно и мы получили исходный текст, который был зашифрован шифром Цезаря.

Также программа выводит ключ (смещение), которое она определила.

Задание для самостоятельной проработки

1. Добавить в имеющиеся программы дополнительный функционал:

a) вывод статистики по тексту (частота символов, встречаемых как в исходном, так и зашифрованном тексте).

b) вывод названия метода шифрования, автора и дату разработки программы

c) модуль защиты интеллектуальных прав на программу (запрашивает ключ на использование программы, отсчитывает срок действия лицензии)

используйте модуль datetime

2. Разработать и протестировать программу на языке программирования Python для шифрования, расшифрования и криптоанализа текста:

a) методом перестановки символов (размер блока от 4 до 8 символов)

b) методом гаммирования (сложения с гаммой, размер гаммы от 4 до 8 символов).

c) методом Виженера (ключевое слово от 4 до 8 символов).

d) шифром Полибия

e) методом Playfair.

(отличаются методами взлома)

f) методом аффинного шифра.

Контрольные вопросы

1. Что представляет собой криптография?
2. Какие методы криптографического преобразования по виду воздействия на исходную информацию вам известны?
3. Каким требованиям должны отвечать современные методы шифрования?
4. В чем заключаются преимущества и недостатки симметричных криптосистем?
5. Какие подвиды симметричных алгоритмов шифрования вы знаете? Дайте им краткую характеристику.
6. Почему криптостойкость должна обеспечиваться секретностью ключа, а не секретностью алгоритма шифрования?
7. В чем заключается назначение криптоанализа?
8. В чем заключается криптоанализ методом протяжки вероятного слова?