Оглавление

**[1.](#_Toc913937)****[Введение](#_Toc913937)** [3](#_Toc913937)

**[2.](#_Toc913938)****[Теоретическая часть](#_Toc913938)** [4](#_Toc913938)

**[3.](#_Toc913939)****[Блок-схема алгоритма](#_Toc913939)** [6](#_Toc913939)

**[3.1. Шифр Цезаря](#_Toc913940)** [6](#_Toc913940)

**[3.2. Шифр Виженера](#_Toc913941)** [7](#_Toc913941)

**[4.](#_Toc913942)****[Результаты работы программы](#_Toc913942)** [8](#_Toc913942)

**[5.](#_Toc913943)****[Исходный код программы](#_Toc913943)** [9](#_Toc913943)

**[6.](#_Toc913944)****[Вывод](#_Toc913944)** [1](#_Toc913944)2

# **Введение**

В рамках данной лабораторной работы планируется изучить такие перестановочные шифры как шифр Цезаря и шифр Виженера, а также создать программный продукт, реализующий шифрование и дешифрование текстовой информации, записанной в файле с помощью данных шифров.

# **Теоретическая часть**

**Шифр Цезаря** — один из древнейших шифров. При шифровании каждый символ заменяется другим, отстоящим от него в алфавите на фиксированное число позиций. Шифр Цезаря можно классифицировать как шифр подстановки, при более узкой классификации — шифр простой замены.

Шифр назван в честь римского императора Гая Юлия Цезаря, использовавшего его для секретной переписки. Естественным развитием шифра Цезаря стал шифр Виженера. С точки зрения современного криптоанализа, шифр Цезаря не имеет приемлемой стойкости.

**Математическая модель**

Если сопоставить каждому символу алфавита его порядковый номер (нумеруя с 0), то шифрование и дешифрование можно выразить формулами модульной арифметики:

*y = (x + (k mod n))) mod n{\displaystyle y=(x+k) mod n ~\,\! }*

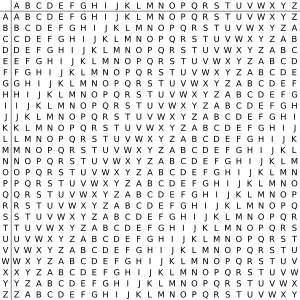
*x = (y − (k mod n)) mod n,*{\displaystyle x=(y+n-(k\ mod\ n)) mod n ~\,\! }

где *x* — символ открытого текста, *y* — символ шифрованного текста, *n* — мощность алфавита, а *k* — сдвиг(ключ).

**Шифр Виженера:**

В шифре Цезаря каждая буква алфавита сдвигается на несколько позиций; например в шифре Цезаря при сдвиге +3, A стало бы D, B стало бы E и так далее. Шифр Виженера состоит из последовательности нескольких шифров Цезаря с различными значениями сдвига. Для зашифровывания может использоваться таблица алфавитов, называемая tabula recta или квадрат (таблица) Виженера. Применительно к латинскому алфавиту таблица Виженера составляется из строк по 26 символов, причём каждая следующая строка сдвигается на несколько позиций. Таким образом, в таблице получается 26 различных шифров Цезаря.

Пример квадрата Виженера:



Например, предположим, что исходный текст имеет вид:

*ATTACKATDAWN*

Человек, посылающий сообщение, записывает ключевое слово («LEMON») циклически до тех пор, пока его длина не будет соответствовать длине исходного текста:

*LEMONLEMONLE*

Первый символ исходного текста A зашифрован последовательностью L, которая является первым символом ключа. Первый символ L шифрованного текста находится на пересечении строки L и столбца A в таблице Виженера. Точно так же для второго символа исходного текста используется второй символ ключа; то есть второй символ шифрованного текста X получается на пересечении строки E и столбца T. Остальная часть исходного текста шифруется подобным способом.

Исходный текст: ATTACKATDAWN

Ключ: *LEMONLEMONLE*

Зашифрованный текст: LXFOPVEFRNHR

**Дешифрование:**

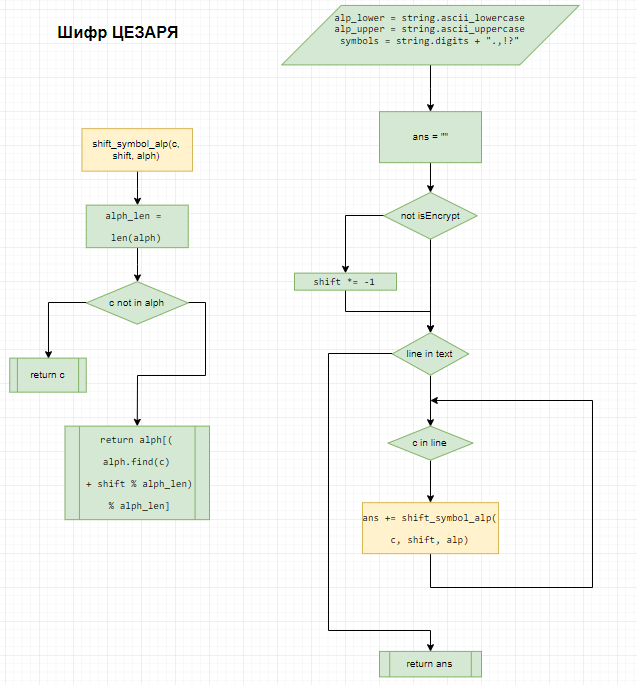
Зашифрованный текст: LXFOPVEFRNHR

Ключ: *LEMONLEMONLE*

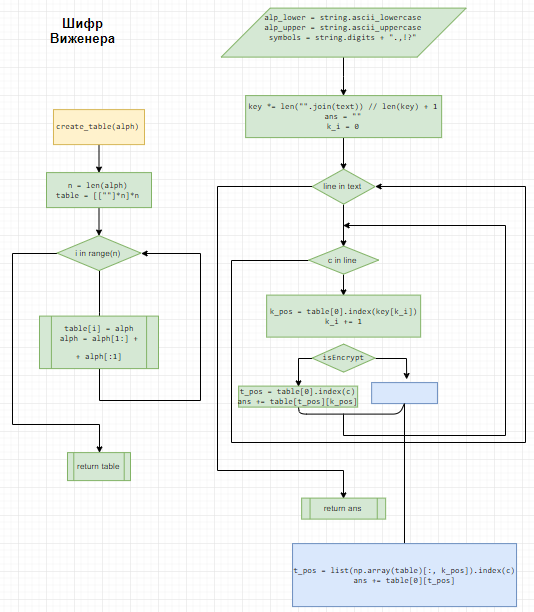
Дешифрованный текст: ATTACKATDAWN

# **Блок-схема алгоритма**

## **3.1. Шифр Цезаря**



## **3.2. Шифр Виженера**



# **Результаты работы программы**

*Результат работы шифра Цезаря:*

python caesar.py file 22 0

Check file file\_caesar\_0

Check file file\_caesar\_0\_caesar\_1

*Результат работы шифра Виженера:*

python vigenere.py file pavel 0

Check file file\_vigenere\_0

Check file file\_vigenere\_0\_vigenere\_1

# **Исходный код программы**

*Шифр Цезаря:*

*class* Caesar:

    @staticmethod

*def* read\_file(*file\_path*: str) -> str:

        with open(*file\_path*, 'r') as read\_file:

            text = read\_file.read()

        return text

    @staticmethod

    def write\_file(*file\_path*: str, *text*: str) -> None:

        with open(*file\_path*, 'w') as write\_file:

            write\_file.write(*text*)

    @staticmethod

    def \_define\_alphabet(*letter*: str) -> tuple:

        eng\_alphabet = string.ascii\_lowercase

        rus\_alphabet = 'абвгдеёжзийклмнопрстуфхцчшщъыьэюя'

        if *letter* in eng\_alphabet or *letter* in rus\_alphabet:

            curr\_alphabet = eng\_alphabet if *letter* in eng\_alphabet else rus\_alphabet

            return curr\_alphabet, len(curr\_alphabet)

        else:

            return *None*, *None*

    @classmethod

    def caesar\_code(*cls*, *text*: str, *key*: int, *mode*: int) -> str:

        cipher\_text = []

        for letter in *text*:

            alphabet, power\_alphabet = *cls*.\_define\_alphabet(letter.lower())

            if alphabet is None:

                cipher\_text.append(letter)

*continue*

            if *mode*:

                index = (alphabet.find(letter.lower()) - *key* + power\_alphabet) % power\_alphabet

            else:

                index = (alphabet.find(letter.lower()) + *key*) % power\_alphabet

            ciph\_let = alphabet[index]

            cipher\_text.append(ciph\_let if letter.islower() else ciph\_let.upper())

        return "".join(cipher\_text)

def main(*file\_path*: str, *key*: str, *mode*: str):

    caesar = Caesar()

    if *key*.isdigit() and *mode*.isdigit():

        try:

            text = caesar.read\_file(*file\_path*)

            cipher\_text = caesar.caesar\_code(text, int(*key*), int(*mode*))

            new\_file\_path = *file\_path* + "\_caesar\_" + *mode*

            caesar.write\_file(new\_file\_path, cipher\_text)

            print(f"Check file {new\_file\_path}")

            return new\_file\_path

        except FileNotFoundError:

            print("File not found")

    else:

        print("The key or mode must be a number!")

if \_\_name\_\_ == "\_\_main\_\_":

    args = sys.argv[1:]

    final\_path = None

    try:

        \_file\_path = args[0]

        \_key = args[1]

        \_mode = args[2]

    except IndexError:

        \_file\_path = input("Enter filepath: ")

        \_key = input("Enter key: ")

        \_mode = input("Enter mode:\n 0 - encrypt\n 1 - decrypt\n")

    finally:

        final\_path = main(\_file\_path, \_key, \_mode)

    if final\_path is None:

        pass

    else:

        main(final\_path, \_key, str(1 - int(\_mode)))

*Шифр Виженера:*

*class* Vigenere:

    @classmethod

*def* read\_file(*cls*, *file\_path*: str) -> str:

        with open(*file\_path*, 'r') as read\_file:

            text = read\_file.read()

        return text

    @classmethod

    def write\_file(*cls*, *file\_path*: str, *text*: str) -> None:

        with open(*file\_path*, 'w') as write\_file:

            write\_file.write(*text*)

    @staticmethod

    def generate\_key(*key*: str, *text*: str) -> str:

        new\_key = list(*key*)

        while len(new\_key) < len(*text*):

            new\_key.extend(*key*)

        return "".join(new\_key)[:len(*text*) + 1]

    @staticmethod

*def* \_define\_alphabet(*letter*: str) -> tuple:

        eng\_alphabet = ascii\_lowercase

        rus\_alphabet = 'абвгдеёжзийклмнопрстуфхцчшщъыьэюя'

        if *letter* in eng\_alphabet or *letter* in rus\_alphabet:

            curr\_alphabet = eng\_alphabet if *letter* in eng\_alphabet else rus\_alphabet

            return curr\_alphabet, len(curr\_alphabet)

        else:

            return *None*, *None*

    @staticmethod

    def check\_key(*key*: str) -> bool:

        rus\_alphabet = 'абвгдеёжзийклмнопрстуфхцчшщъыьэюя'

        if not len(set(*key*.lower()) - set(ascii\_lowercase)):

            return True

        elif not len(set(*key*.lower()) - set(rus\_alphabet)):

            return True

        else:

            return False

    @classmethod

*def* vigenere\_code(*cls*, *text*: str, *key*: str, *mode*: int) -> str:

        cipher\_text = []

        i = 0

        for letter in *text*:

            alph, power\_alph = *cls*.\_define\_alphabet(letter.lower())

            if alph is None:

                cipher\_text.append(letter)

*continue*

            kv = alph.find(*key*[i].lower())

            if *mode*:

                index = (alph.find(letter.lower()) - kv + power\_alph) % power\_alph

            else:

                index = (alph.find(letter.lower()) + kv) % power\_alph

            ciph\_let = alph[index]

            cipher\_text.append(ciph\_let if letter.islower() else ciph\_let.upper())

            i += 1

        return "".join(cipher\_text)

def main(*file\_path*, *key*, *mode*):

    vigenere = Vigenere()

    if *mode*.isdigit():

        try:

            if not vigenere.check\_key(*key*):

                raise KeyError

            text = vigenere.read\_file(*file\_path*)

*key* = vigenere.generate\_key(*key*, text)

            cipher\_text = vigenere.vigenere\_code(text, *key*, int(*mode*))

            new\_file\_path = *file\_path* + "\_vigenere\_" + *mode*

            vigenere.write\_file(new\_file\_path, cipher\_text)

            print(f"Check file {new\_file\_path}")

            return new\_file\_path

        except FileNotFoundError:

            print("File not found")

        except KeyError:

            print("The key is not only letters")

    else:

        print("The mode must be a number!")

if \_\_name\_\_ == "\_\_main\_\_":

    # print('args->', sys.argv[1:])

    args = sys.argv[1:]

    final\_path = None

    try:

        \_file\_path = args[0]

        \_key = args[1]

        \_mode = args[2]

    except IndexError:

        \_file\_path = input("Enter filepath: ")

        \_key = input("Enter key: ")

        \_mode = input("Enter mode:\n 0 - encrypt\n 1 - decrypt\n")

    finally:

        final\_path = main(\_file\_path, \_key, \_mode)

    if final\_path is None:

        pass

    else:

        main(final\_path, \_key, str(1 - int(\_mode)))

# **Вывод**

В данной лабораторной работе были рассмотрены шифры Цезаря и Виженера. Были построены алгоритмы, выполняющие шифрование и дешифрование текстового файла с помощью выбранного алгоритма.