федеральное государственное автономное образовательное учреждение
высшего образования

«НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИТМО»

ОТЧЕТ

по лабораторной работе Ожиганов №1

«Основы шифрования данных»

по дисциплине «**Информационная безопасность**»

Вариант 12

Автор: Кулаков Н. В.

Факультет: ПИиКТ

Группа: P34312

Преподаватель: Маркина Т. А.

Санкт-Петербург 2023

1. Цель работы

Изучить основные принципы шифрования информации, познакомиться с широко известными алгоритмами шифрования, приобрести навыки их программной реализации.

2. Описание

Реализовать шифрование и дешифрацию файла по методу Виженера. Ключевая фраза вводится. Реализовать в программе частотный криптоанализ зашифрованного текста.

3. Выполнение

Вариант 2. Реализовать шифрование и дешифрацию файла по методу Виженера. Ключевая фраза вводится. Реализовать в программе частотный криптоанализ зашифрованного текста.

3.1. Описание программы

Принцип работы алгоритма следующий: Если n — количество букв в алфавите, mi — номер буквы открытого текста, ki— номер буквы ключа в алфавите, то шифрование Виженера можно записать следующим образом:

ri = (si + ki) % n

И расшифровывание:

ri = (si - ki) % n

Ниже в листинге кода алгоритм реализован в методах encrypt и decrypt аналогично описанному выше, но используя индексы массива, поскольку интовое представление букв в алфавите является последовательным. Почти весь остальной код — обработка пользовательского ввода.

3.2. Листинг

```
#!/usr/bin/python3

import argparse
from typing import List, Tuple


ALPHABET = (
    [chr(i) for i in range(ord('A'), ord('z') + 1)] +
    [" ", "!", ";", ":", "?", ",", "-", ".", "—"] +
    [chr(i) for i in range(ord('A'), ord('я') + 1)]
)


def parse_args():
    parser = argparse.ArgumentParser()
    parser.add_argument("-k", "--key", required=True)
    parser.add_argument("ifile")
    parser.add_argument("ofile")
    parser.add_argument("-c", "--encrypt", action="store_true")
    parser.add_argument("-z", "--decrypt", action="store_true")
    parser.add_argument("-a", "--analyze", action="store_true")
    parser.add_argument("-v", "--verbose", action="store_true")
    return parser.parse_args()


def validate_args(args) -> Tuple[bool, str]:
    if args.encrypt == args.decrypt:
        if args.encrypt == False and args.analyze == False:
            return (True, "oneof encrypt or decrypt should be present")
        if args.encrypt == True:
            return (True, "oneof encrypt or decrypt should be present")

    return (False, "")


def encrypt(istr, key):
    n = len(ALPHABET)
    rs = ""

    for i, s in enumerate(istr):
        sai, kai = ALPHABET.index(s), ALPHABET.index(key[i % len(key)])
```

```
        ri = (sai + kai) % n
        r = ALPHABET[ri]
        rs += r

    return rs


def decrypt(istr, key):
    n = len(ALPHABET)
    rs = ""

    for i, s in enumerate(istr):
        sai, kai = ALPHABET.index(s), ALPHABET.index(key[i % len(key)])
        ri = (sai - kai) % n
        r = ALPHABET[ri]
        rs += r

    return rs


def analyze(istr) -> List[float]:
    flo = ord(ALPHABET[0])
    cnts = [0] * len(ALPHABET)
    for a in istr:
        cnts[ord(a) - flo] += 1

    freqs = []
    for c in cnts:
        freqs.append(c / len(istr))

    return freqs


def display_analysis(freqs):
    for i, a in enumerate(ALPHABET):
        print(f"({a}, {round(freqs[i], 3)})")

    # kasiski match index
    ksum = 0.0
    for f in freqs:
        ksum += f * f
    print("Kasiski match index: ", round(ksum, 4))
```

```
def process(args):
    ifile = open(args.ifile, "r")
    ofile = open(args.ofile, "w")

    istr = "".join(ifile.readlines()).strip()
    ostr = None
    key = args.key

    if args.verbose:
        print(f"read from {args.ifile}:", istr)

    if args.encrypt:
        ostr = encrypt(istr, key)

    if args.decrypt:
        ostr = decrypt(istr, key)

        if args.analyze:
            freqs = analyze(istr)
            display_analysis(freqs)

    if ostr is not None:
        ofile.write(ostr)

    if args.verbose:
        print(f"written to {args.ofile}:", ostr)

    ofile.close()
    ifile.close()


def main():
    args = parse_args()

    err, msg = validate_args(args)
    if err:
        print("error:", msg)
        exit(-1)

    process(args)
```

```
if __name__ == "__main__":
    main()
```

3.3. Результат работы

```
(.venv) nikit@host lab-oziganov-1-1 % python main.py -k key -c -v -a data/in.txt
data/out.txt
read from data/in.txt: abcthisisinputtext
written to data/out.txt: kfadlgcmqsrnexrobr
(.venv) nikit@host lab-oziganov-1-1 % python main.py -k key -z -v -a
data/out.txt data/in.txt
read from data/out.txt: kfadlgcmqsrnexrobr
(a, 0.056)
(b, 0.056)
(c, 0.056)
(d, 0.056)
(e, 0.056)
(f, 0.056)
(g, 0.056)
(h, 0.0)
(i, 0.0)
(j, 0.0)
(k, 0.056)
(l, 0.056)
(m, 0.056)
(n, 0.056)
(o, 0.056)
(p, 0.0)
(q, 0.056)
(r, 0.167)
(s, 0.056)
(t, 0.0)
(u, 0.0)
(v, 0.0)
(w, 0.0)
(x, 0.056)
(y, 0.0)
(z, 0.0)
Kasiski match index:  0.0741
written to data/in.txt: abcthisisinputtext
```

4. Выводы

В ходе выполнения лабораторной работы был реализован алгоритм Виженера, который является одним из возможных методов симметричного шифрования полиалфавитного буквенного текста. Алгоритм в отличие от шифра Цезаря является более криптостойким, однако существуют различные тесты, например тест Касиски, которые позволяют найти размер ключа, а после с помощью частотного анализа может быть расшифрован и исходный текст.