МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ
Федеральное государственное автономное образовательное учреждение
высшего профессионального образования
«САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
АЭРОКОСМИЧЕСКОГО ПРИБОРОСТРОЕНИЯ»

Кафедра №33

ОТЧЕТ
ЗАЩИЩЕН С ОЦЕНКОЙ

ПРЕПОДАВАТЕЛЬ

ст. преподаватель		А.В. Афанасьева
должность, уч. степень, звание	подпись, дата	инициалы, фамилия

ОТЧЕТ О ЛАБОРАТОРНОЙ РАБОТЕ

Текстовая стеганография

по курсу: Технологии стеганографии в системах инфокоммуникаций

РАБОТУ ВЫПОЛНИЛ

СТУДЕНТ ГР. № 3931		В.С. Кулешов
	подпись, дата	инициалы, фамилия

Санкт-Петербург
2022

1 Цель работы

Целью данной лабораторной работы является разработка стегосистемы на основе замены символов кириллицы на аналогичные по написанию символов латинского алфавита.

2 Ход выполнения работы

В основе реализации лежит следующий алгоритм сокрытия:

1. Данные преобразуются в бинарный поток BS;
2. Информация о размере BS так же переводится поток бит MetaInformationBS;
3. Полезная информация Payload есть результат конкатенации MetaInformationBS и BS;
4. Последовательно проходя по содержимому контейнера, и каждый раз встречая подходящий символ для замены, необходимо брать очередной бит из потока Payload и в соответствии с его значением принимать решение о замене.

Для распаковки закрытого контейнера применяется следующий алгоритм:

1. Последовательно пройтись по содержимому закрытого контейнера и регистрировать потенциальные символы для замены и их латинские-аналоги. Полученную информацию заносить в контейнер типа FIFO Message;
2. По достижению Message размера MetaInformationBS интерпретировать содержимое контейнера Message как информацию о размере BS. Очистить содержимое Message и продолжить проход по контейнеру с места остановки;
3. Аналогично п. 1 регестрируются все подходящие символы и их аналоги, до тех пор пока размер контейнера не достигнет размера BS;
4. Интерпретировать данные контейнера как поток байт и вывести в стандартный поток вывода.

2. 1 О контейнерах

Для закрытия сообщения длиной Size необходимо чтобы контейнер содержал:

$$8*Size + MetaInformationSize$$

потенциальных символов для замены; где MetaInformationSize в текущей реализации — константа целочисленного типа равная 32.

При использовании контейнера все оригинальные символы-аналоги будут заменены на соответствующие символы кириллицы, что негативно скажется на функциональных характеристиках следующих типов контейнеров: .html, .rtf, .cpp. Это необходимо для избежания искажений при раскрытии запакованного сообщения.

3 Вывод

В результате выполнения лабораторной работы была реализована стегосистема в соответствии с предъявленными требованиями.

4 Листинг кода

```
import os.path
import argparse
import concurrent.futures as concur


twinks = {"A": "A",
        "а": "a",
        "В": "B",
        "г": "r",
        "Е": "E",
        "е": "e",
        "К": "K",
        "к": "k",
        "М": "M",
        "м": "m",
        "О": "O",
        "о": "o",
        "Р": "P",
        "р": "p",
        "С": "C",
        "с": "c",
        "Т": "T",
        "у": "Y",
        "у": "y",
        "Х": "X",
        "х": "x",
        "Я": "R"}



class Container(object):
    data : list = list()
    cells : list = list()

    def __init__(self, fn : str):
        with open(fn, "r") as f:
            self.data = list(f.read())

        for symbol in range(len(self.data)):
            for var in twinks:
                if twinks[var] == self.data[symbol]:
                    self.data[symbol] = var

            if self.data[symbol] in twinks:
                self.cells.append(symbol)
```

```
def payload_into_bs(fn : str) -> list:
    result = list()

    with open(fn, "rb") as f:
        data = f.read()

    size = len(data) * 8
    for i in range(31, -1, -1):
        result.append(((size & (1 << i)) != 0))

    for b in data:
        for i in range(7, -1, -1):
            result.append(((b & (1 << i)) != 0))

    return result


def from_bs(bs : list) -> list:
    tmp = 0
    result = list()
    for i in range(len(bs)):
        if (i % 8) == 0:
            result.append(tmp)
            tmp = 0

        if bs[i]:
            tmp += 1 << (7 - (i % 8))
    return result


def pack(c : Container, p : list) -> list:
    for idx in range(len(p)):
        if p[idx]:
            c.data[c.cells[idx]] = twinks[c.data[c.cells[idx]]]
    return c
```

```python
def unpack(fn : str) -> list:
    with open(fn, "r") as f:
        data = f.read()

    index = 0
    counter = 0
    result = list()
    while index < len(data) and counter != 32:
        if data[index] in twinks:
            result.append(False)
            counter += 1

        elif data[index] in twinks.values():
            result.append(True)
            counter += 1
        index += 1

    size = 0
    for i in range(len(result)):
        if result[i]:
            size += (1 << (31 - i))

    result = list()
    counter = 0
    while index < len(data) and counter < size:
        if data[index] in twinks:
            result.append(False)
            counter += 1

        elif data[index] in twinks.values():
            result.append(True)
            counter += 1
        index += 1

    return from_bs(result)

with concur.ProcessPoolExecutor(max_workers=1) as executor:
```

```
with concur.ProcessPoolExecutor(max_workers=1) as executor:
    if __name__ == "__main__":
        parser = argparse.ArgumentParser("")
        parser.add_argument("mode", choices=["pack", "unpack"], help="Processing mode")
        parser.add_argument("container", help="Container for packing/unpacking")
        parser.add_argument("--payload", "-p", help="Message that will be packed. Only for
        parser.add_argument("--closed-container", help="Path to closed container.")

        args = parser.parse_args()

        if args.mode == "pack":
            if args.payload is None:
                print("Not defined message for packing.")
                parser.print_help()
                exit(1)
            elif not os.path.isfile(args.payload):
                print("File with payload doesnt exists: ", args.payload)
                exit(1)
            elif not os.path.isfile(args.container):
                print("File with container doesnt exists: ", args.container)
                exit(1)

            future_bs_payload = executor.submit(payload_into_bs, args.payload)
            container = Container(args.container)

            bs_payload = future_bs_payload.result()
            if len(bs_payload) > len(container.cells):
                print("Not enough avaible symbols for packing:\n\tAvaible: {}\n\tNeed: {}"
                exit(2)

            closed_container = pack(container, bs_payload)

            path = "result.txt"
            if not args.closed_container is None:
                path = args.closed_container

            with open(path, "wb") as f:
                for b in closed_container.data:
                    f.write(b.encode())
        else:
            if not os.path.isfile(args.container):
                print("File with container doesnt exists: ", args.container)
                exit(1)

            payload = unpack(args.container)
```