**Министерство науки и высшего образования РФ**

Федеральное государственное автономное

образовательное учреждение высшего образования

«Национальный исследовательский университет ИТМО»

факультет БЕЗОПАСНОСТИ ИНФОРМАЦИОННЫХ

ТЕХНОЛОГИЙ

Управление мобильными устройствами

**ОТЧЕТ ПО ЛАБОРАТОРНОЙ РАБОТЕ №1**

«Обработка и тарификация CDR (Call Detail Record)»

Выполнил студент

группы N3352

\_\_\_\_\_\_ Гарницкий Даниил Игоревич



Проверил: инженер ФБИТ,

Университет ИТМО,

\_\_\_\_\_\_ Федоров Иван Романович

Санкт-Петербург

2020

**Цель работы:** научиться работать с текстовыми алгоритмами стеганографии.

**Теоретическая часть:**

Стеганография служит для скрытия секретных сообщений в других сообщениях, причём скрывается даже само существование секрета. Как правило, отправитель пишет какое-нибудь невинное сообщение (контейнер), а затем скрывает конфиденциальную информацию на этом же листе бумаги. Данное направление защиты информации очень актуально в настоящее время, о чём свидетельствуют ресурсы, инвестируемые в развитие стеганографии правительствами ведущих стран мира и террористическими организациями.

В ходе выполнения данной лабораторной работы были рассмотрены следующие методы:

1. Метод прямой замены символов, когда один символ русского алфавита заменяется латинским того же начертания. Основная проблема этого метода в том, если будут использоваться программы с автоматической проверкой на орфографию.
2. Метод с использованием дополнительных пробелов. В конец строки дописываются пробелы. Для кодирования «0» используется единичный пробел, а для «1» используется двойной пробел.
3. Метод добавления служебных символов, когда один служебный символ заменяется аналогичным по таблице utf-8 либо добавляются невидимые символы.

**Практическая часть:**

Все алгоритмы были написаны на языке программирования Python.

На вход подается слово, которое мы хотим передать, и путь текстового файла, в который будет происходить запись. Преобразуем слово в бинарный код.

**Листинг кода:**

**Метод 1:**

word = 'Здравствуйте'

file\_name = 'file.txt'

#преобразование текста в бинарный код

def text\_to\_bits(word, encoding='utf-8'):

bits = bin(int.from\_bytes(word.encode(encoding), 'big'))[2:]

return bits.zfill(8 \* ((len(bits) + 7) // 8))

#декодирование текста

def text\_from\_bits(bits, encoding='utf-8'):

n = int(bits, 2)

return n.to\_bytes((n.bit\_length() + 7) // 8, 'big').decode(encoding) or '\0'

def language\_change(text):

count\_text = 0

count\_bits = 0

while(count\_text < len(text) and count\_bits < len(bits)):

if((text[count\_text] == ('о' if bits[count\_bits] == '0' else 'а')) and (text[count\_text-1] != 'а') and (text[count\_text+1] != 'а') and (text[count\_text-1] != 'о') and (text[count\_text+1] != 'о')):

text = text[:count\_text] + ('o' if bits[count\_bits] == '0' else 'a') + text[(count\_text+1):]

count\_bits +=1

count\_text +=1

return text

def language\_find(text\_after):

wrd = ''

count\_text = 0

count\_bits = 0

while(count\_text < len(text\_after)):

if(text\_after[count\_text] == 'o' and ((ord(text\_after[count\_text-1]) > 122 or ord(text\_after[count\_text-1]) < 97) and (ord(text\_after[count\_text-1]) > 90 or ord(text\_after[count\_text-1]) < 65)) and (ord(text\_after[count\_text+1]) > 122 or ord(text\_after[count\_text+1]) < 97)):

wrd +='0'

elif(text\_after[count\_text] == 'a' and ((ord(text\_after[count\_text-1]) > 122 or ord(text\_after[count\_text-1]) < 97) and (ord(text\_after[count\_text-1]) > 90 or ord(text\_after[count\_text-1]) < 65)) and (ord(text\_after[count\_text+1]) > 122 or ord(text\_after[count\_text+1]) < 97)):

wrd +='1'

count\_text +=1

return wrd

file = open(file\_name, 'r')

text\_before = file.read()

file.close()

bits = text\_to\_bits(word)

text\_after = language\_change(text\_before)

file = open(file\_name, 'w')

file.write(text\_after)

file.close()

word\_encode = language\_find(text\_after)

word\_decode = text\_from\_bits(word\_encode)

print(word\_decode)

**Метод 2:**

word = 'Здравствуйте'

file\_name = 'file.txt'

#преобразование текста в бинарный код

def text\_to\_bits(word, encoding='utf-8'):

bits = bin(int.from\_bytes(word.encode(encoding), 'big'))[2:]

return bits.zfill(8 \* ((len(bits) + 7) // 8))

#декодирование текста

def text\_from\_bits(bits, encoding='utf-8'):

n = int(bits, 2)

return n.to\_bytes((n.bit\_length() + 7) // 8, 'big').decode(encoding) or '\0'

def space\_change():

line\_out = ''

count\_bits = 0

f = open(file\_name, 'r')

for line in f.readlines():

line = line.rstrip() + '\n'

if line[len(line) - 1] == '\n' and count\_bits < len(bits):

if bits[count\_bits] == '1':

line\_out += line[:(len(line)-1)] + ' \n'

elif bits[count\_bits] == '0':

line\_out += line[:(len(line)-1)] + ' \n'

elif count\_bits >= len(bits):

line\_out += line

else: line\_out += line

count\_bits +=1

f.close()

return line\_out

def space\_find():

count\_text = 0;

count\_bits = 0

bits\_encode = ''

f = open(file\_name, 'r')

for line in f.readlines():

if line[len(line)-1] == '\n' and count\_bits < len(bits):

if line[(len(line)-3):(len(line)-1)] == ' ':

bits\_encode += '1'

count\_bits +=1

elif line[(len(line)-2):(len(line)-1)] == ' ':

bits\_encode += '0'

count\_bits +=1

else: break

f.close()

return bits\_encode

bits = text\_to\_bits(word)

text\_after = space\_change()

file = open(file\_name, 'w')

file.write(text\_after)

file.close()

bits\_encode = space\_find()

bits\_decode = text\_from\_bits(bits\_encode)

print(bits\_decode)

**Метод 3:**

word = 'Здравствуйте'

file\_name = 'file.txt'

#преобразование текста в бинарный код

def text\_to\_bits(word, encoding='utf-8'):

bits = bin(int.from\_bytes(word.encode(encoding), 'big'))[2:]

return bits.zfill(8 \* ((len(bits) + 7) // 8))

#декодирование текста

def text\_from\_bits(bits, encoding='utf-8'):

n = int(bits, 2)

return n.to\_bytes((n.bit\_length() + 7) // 8, 'big').decode(encoding) or '\0'

def symbols\_change(text):

text\_count = 0

bits\_count = 0

while(text\_count < len(text) and bits\_count < len(bits)):

if bits[bits\_count] == '0':

text = text[:text\_count] + chr(0) + text[(text\_count):]

bits\_count +=1

elif bits[bits\_count] == '1':

text = text[:text\_count] + chr(1) + text[(text\_count):]

bits\_count +=1

text\_count +=25

return text

def symbols\_find(text):

text\_count = 0

bits\_count = 0

wrd = ''

while(text\_count < len(text) and bits\_count < len(bits)):

if text[text\_count] == chr(0):

wrd += '0'

bits\_count +=1

elif text[text\_count] == chr(1):

wrd += '1'

bits\_count +=1

text\_count +=1

return wrd

file = open(file\_name, 'r')

text = file.read()

file.close()

bits = text\_to\_bits(word)

text = symbols\_change(text)

bits\_encode = symbols\_find(text)

bits\_decode = text\_from\_bits(bits\_encode)

print(bits\_decode)

В методе 1 чтение текстового файла происходит посимвольно. Меняем русскую букву «о» на английскую “o” для кодирования «0», русскую букву «а» на английскую “a” для кодирования «1».

В методе 2 чтение текстового файла происходит построчно. В конец строки добавляем два пробела, если в двоичной кодировке необходимо записать «1», либо один пробел для кодирования «0».

В методе 3 чтение текстового файла происходит посимвольно. Добавляем служебные символы «пустой символ» для кодирования «0», «начало загаловка» для кодирования «1».

Происходит посимвольное чтение последовательности 0 и 1.

В конце полученный бинарный код преобразуется в текст, который передавался.

**Анализ:**

Как основа для контейнера было взят отрывок из произведения А. П. Чехова «Палата №6».

Оценка размера контейнера до встраивания и после:

1. Используя первый метод со встраиванием слова «Здравствуйте», получаем контейнер с меньшим размером после встраивания. Это происходит из-за того, что латинские символы кодируются 1 байтом, а символы кириллицы – 2 байтами. (185105/185297) \* 100% = 99.9% сходства.
2. Используя второй метод со встраиванием слова «Здравствуйте», получаем контейнер с большим размером после встраивания, так как происходит добавление пробелов. (185297/185569) \* 100% = 99.85% сходства.
3. Используя третий метод со встраиванием слова «Здравствуйте», контейнер будет иметь тот же размер. (185297/185297) \* 100% = 100% сходства.

Таким образом, можно утверждать, что данные методы эффективны, но лишь при использовании достаточно большого текстового контейнера.

Частотный анализ не выявит серьезных расхождений, если анализ будет применяться исключительно для символов кириллицы. При частотном анализе всех символов в тексте:

1. Метод 1 окажется неэффективным, так как будут найдены латинские символы, которых в тексте быть не должно;
2. Метод 2 окажется эффективным, но только при использовании достаточно большого текстового контейнера;
3. Метод 3 окажется неэффективным, так как будут найдены спец. символы, которых в тексте быть не должно.

**Вывод:** в ходе данной лабораторной работы были изучены следующие методы текстовой стеганографии: метод прямой замены символов, метод с использованием дополнительный пробелов и метод с добавлением служебных символов. **Список использованной литературы:**

1. Текстовая стеганография [Электронный ресурс]: URL <https://www.osp.ru/pcworld/2004/11/169154/>
2. Стеганографическое сокрытие информации в текстовом файле при работе в вычислительных сетях [Электронный ресурс]: URL <http://network-journal.mpei.ac.ru/cgi-bin/main.pl?l=ru&n=3&pa=13&ar=4>
3. Таблица символов юникода [Электронный ресурс]: URL <https://unicode-table.com/ru/>