**Министерство науки и высшего образования РФ**

Федеральное государственное автономное

образовательное учреждение высшего образования

«Национальный исследовательский университет ИТМО»

# факультет БЕЗОПАСНОСТИ ИНФОРМАЦИОННЫХ

**ТЕХНОЛОГИЙ**

Основы стеганографии

**ОТЧЕТ ПО ЛАБОРАТОРНОЙ РАБОТЕ №1**

«Основы текстовой стеганографии»

Работу выполнил студент

Нгуен Тхай Хынг, N3352, ФБИТ



Работу проверил: ассистент ФБИТ,

Университет ИТМО,

Давыдов Вадим Валерьевич

Санкт-Петербург

2020

## Цель работы

Целью данной лабораторной работы :

▪ Применение текстовых стеганографических методов для сокрытия

▪ Извлечение сообщения из стегоконтейнера

▪ Анализ исходного текста и стегоконтейнера.

## Теория

**Стеганогра́фия** - это наука о скрытой передаче [информации](https://dic.academic.ru/dic.nsf/ruwiki/4939) путём сохранения в тайне самого факта передачи. Этот термин ввел в [1499 году](https://dic.academic.ru/dic.nsf/ruwiki/2672) [Иоганн Тритемий](https://dic.academic.ru/dic.nsf/ruwiki/1507064) в своем трактате «Стеганография» (*Steganographia*), зашифрованном под магическую книгу.

В отличие от [криптографии](https://dic.academic.ru/dic.nsf/ruwiki/17909), которая скрывает содержимое секретного сообщения, стеганография скрывает сам факт его существования. Как правило, сообщение будет выглядеть как что-либо иное, например, как изображение, статья, список покупок, письмо или судоку. Стеганографию обычно используют совместно с методами криптографии, таким образом, дополняя её.

Преимущество стеганографии над чистой криптографией состоит в том, что сообщения непривлекают к себе внимания. Сообщения, факт шифрования которых не скрыт, вызывают подозрение и могут быть сами по себе уличающими в тех странах, в которых запрещена криптография.Таким образом, криптография защищает содержание сообщения, а стеганография защищает сам факт наличия каких-либо скрытых посланий.

**Классификация стеганографии**

Выделилось несколько направлений стеганографии:

* Классическая стеганография
* Компьютерная стеганография
* Цифровая стеганография

**Классическая стеганография**

Одним из наиболее распространенных методов **классической стеганографии** является использование [симпатических (невидимых) чернил](https://dic.academic.ru/dic.nsf/ruwiki/299684). Текст, записанный такими [чернилами](https://dic.academic.ru/dic.nsf/ruwiki/212591), проявляется только при определенных условиях (нагрев, освещение, химический проявитель и т. д.)

Существуют также чернила с химически нестабильным пигментом. Написанное этими чернилами выглядит как написанное обычной ручкой, но через определенное время нестабильный пигмент разлагается, и от текста не остается и следа. Хотя при использовании обычной шариковой ручки текст можно восстановить по деформации бумаги, этот недостаток можно устранить с помощью мягкого пишущего узла, наподобие фломастера.

**Компьютерная стеганография** - направление классической стеганографии, основанное на особенностях компьютерной платформы. Примеры - стеганографическая файловая система StegFS для [Linux](https://dic.academic.ru/dic.nsf/ruwiki/7344), скрытие данных в неиспользуемых областях форматов [файлов](https://dic.academic.ru/dic.nsf/ruwiki/6904), подмена символов в названиях [файлов](https://dic.academic.ru/dic.nsf/ruwiki/6904), текстовая стеганография и т. Д

**Цифровая стеганография** - направление классической стеганографии, основанное на сокрытии или внедрении дополнительной информации в цифровые объекты, вызывая при этом некоторые искажения этих объектов. Но, как правило, данные объекты являются мультимедиа-объектами (изображения, видео, аудио, текстуры 3D-объектов) и внесение искажений, которые находятся ниже порога чувствительности среднестатистического человека, не приводит к заметным изменениям этих объектов. Кроме того, в оцифрованных объектах, изначально имеющих аналоговую природу, всегда присутствует шум квантования; далее, при воспроизведении этих объектов появляется дополнительный аналоговый шум и нелинейные искажения аппаратуры, все это способствует большей незаметности сокрытой информации.

**Другие стеганографические методы:**

Во время Второй мировой войны активно использовались микроточки - микроскопические фотоснимки, вклеиваемые в текст писем.

Также существует ряд альтернативных методов сокрытия информации:

* запись на боковой стороне колоды карт, расположенных в условленном порядке;
* запись внутри варёного яйца;
* «жаргонные шифры», где слова имеют другое обусловленное значение;
* трафареты, которые, будучи положенными на текст, оставляют видимыми только значащие буквы;
* узелки на нитках и т. д.

В настоящее время под стеганографией чаще всего понимают скрытие информации в текстовых, графических либо аудиофайлах путём использования специального программного обеспечения.

<https://dic.academic.ru/dic.nsf/ruwiki/30097>

## Практика

Для того, чтобы реализовать вышеописанные методы, мною было написана программа “stegano.py”, листинг которых можно найти в приложении к данному отчету, которое приведено после списка литературы.

Для написания программы использовался язык Python 3 с помощью Visual Studio Code, так как мне нравится работа со строками в данном языке.

Программа требует выбрать метод для скрытия инфармации, у нас есть три метода:

1. Метод замены символов, этот метод заменяет буквы “o” и “p” на английском языке на символы “o” и “p” на русском языке, с условием, что символ “o” (RU) соответствует биту 0, а символ “p” (RU) соответствует биту 1.

2. Метод с использованием допольнительных хвостовых пробелов. Этот метод добавляет 1 пробел сразу до символа “\ n”, соответствующей биту 1, добавляет 2 пробела сразу до символа “\ n”, соответствующего биту 0.

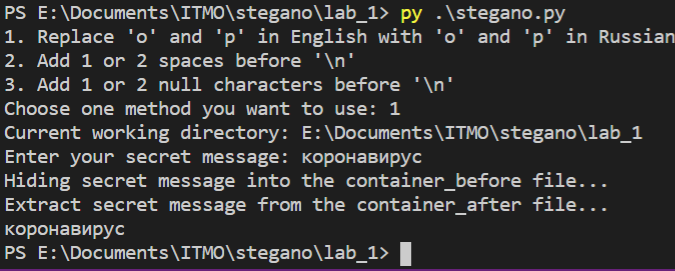
3. Метод с добавлением служебных символов. Этот метод добавляет 1 нулевой символ “\0” сразу до символа “\ n”, соответствующего биту 0, добавляет 2 нулевых символа сразу до символа “\ n”, соответствующего биту 0.

После ввода метода программа запрашивает текст, который мы хотим скрыть.

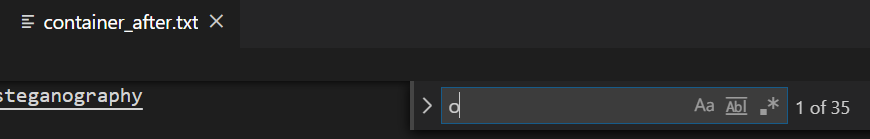
Эта программа читает файл “container\_before.txt” потом выводит файл “container\_after.txt”, содержащий скрытую информацию.

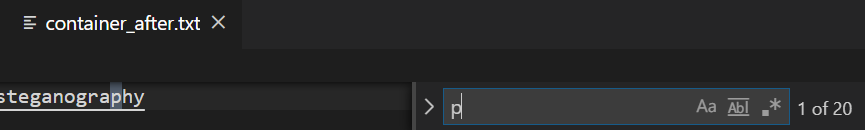
**Метод 1: Метод замены символов**

Запустить программу



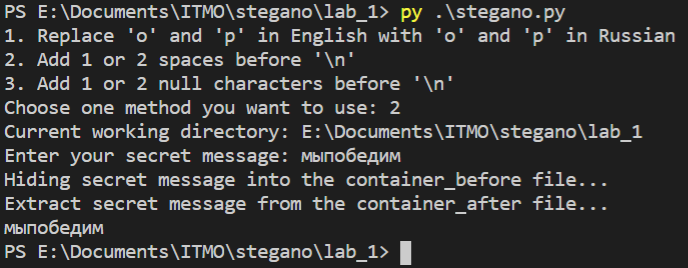
В файле “container\_after.txt” содержится 35 символов “o” и 20 символов “p”



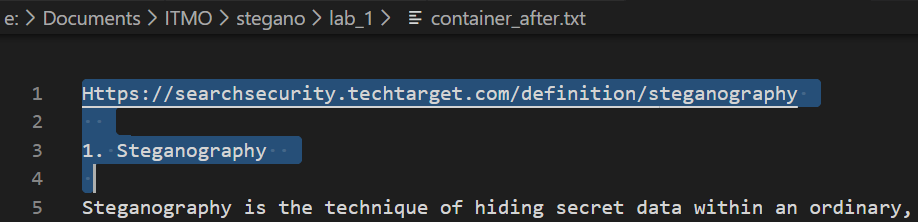


**Метод 2: Метод с использованием допольнительных хвостовых пробелов**

Запустить программу

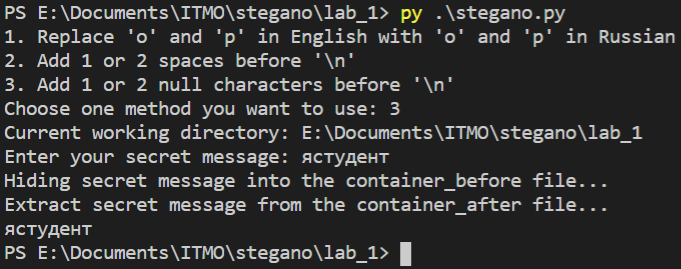


Открываем файл “container\_after.txt” для просмотра. Мы увидим на конец каждой строки есть один или два пробела.

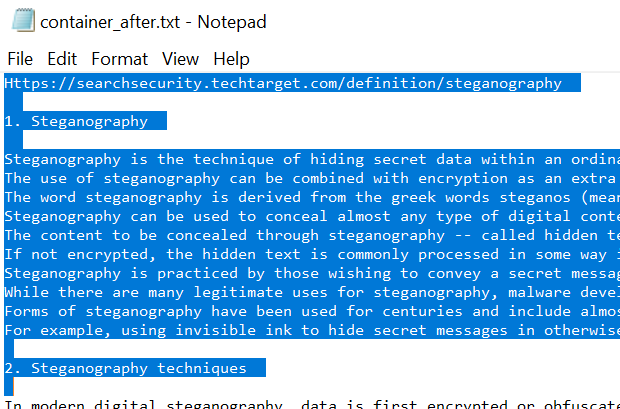


### Метод 3: Метод с добавлением служебных символов

Запустить программу

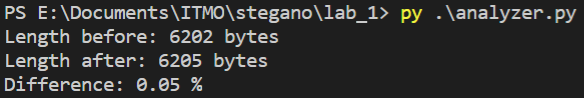


Открываем файл “container\_after.txt” для просмотра. Мы увидим на конец каждой строки есть один или два пробела. На самом деле это нулевой символ “\0”, отображаемый как пробел

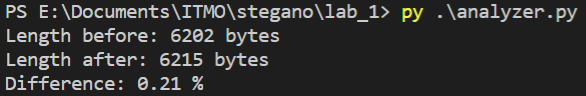


Мы видим, что программа работает корректно, обеспечивая встраивание стего в текст. Теперь необходимо проанализировать результат файл до и после встраивания. Для того, я написал программа “analyzer.py”, запускаем программу для анализа файлов по каждому методу, получаем результаты:

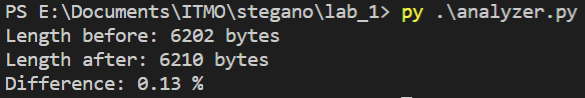
Результат первого метода:



Результат второго метода:



Результат третьего метода:



Мы видим, что первый метод имеет самое низкое изменение данных, поэтому этот метод очень сложно обнаружить, с помощью первого метода мы также скрываем больше данных, потому что в английском тексте есть много букв “o” и “p”.

Второй и третий методы менее эффективны, мы скрываем данные в тексте добавлением символов до символа "\ n", поэтому скрываем меньше данных, потому что в тексте очень мало символов “\n”. Эти методы также увеличивают изменение текстового содержимого, потому что они «добавляют» вместо «заменят» символы, поэтому этот метод НЕ очень сложно обнаружить.

## Вывод

После выполнения методов в первой лаборатории, и в то же время я исследовал и оценил надежность и безопасность методов, я сделал следующие выводы:

Метод замены символов одного типа весьма полезен и его трудно обнаружить обычным текстовым редакторам. Метод добавления пробела перед переносом строки легко обнаружить, приведенное выше изображение является примером того, когда его можно узнать в Word. Метод добавления специальных символов увеличивает объем текста, поэтому его легко обнаружить.

Поэтому, на мой взгляд, самый эффективный метод - это первый, который не меняет размер файла и его трудно обнаружить.

## Список Литературы

1. What is steganography? - <https://searchsecurity.techtarget.com/definition/steganography>

2. Wikipedia - <https://en.wikipedia.org/wiki/Steganography>

3. AH4S: an algorithm of text in text steganography using the structure of omega network - <https://onlinelibrary.wiley.com/doi/full/10.1002/sec.1752>

**Приложение**

Программа для всех методов: stegano.py

import os

import codecs # For handling unicode characters when reading file containing unicode characters to list

# Encode text (our secret message in clear text) to binary string, using the defined dictionary in the main program

def encode(secret\_message):

    binary\_secret\_message = ''

    for character in secret\_message:

        # Method dict.get(key) returns a value for a given key in the dictionary 'dict'

        binary\_secret\_message += dictionary.get(character)

    return binary\_secret\_message

# Decode binary string to text (our secret message in clear text), using the defined dictionary in the main program

def decode(binary\_secret\_message):

    secret\_message = ''

    for i in range(0, len(binary\_secret\_message), 5):

        for key, value in dictionary.items():

            if value == binary\_secret\_message[i: i+5]:

                secret\_message += key

    return secret\_message

# Hide secret message into the container file, output to another file

def hide\_secret\_message(container\_file\_in, secret\_message, container\_file\_out):

    # Read all characters from file 'container\_file\_in' into a list 'character\_list'

    character\_list = [char for char in open(container\_file\_in).read()]

    binary\_secret\_message = encode(secret\_message)

    number\_of\_replacement = 0

    j = 0

    for i in range(0, len(character\_list), 1):

        if character\_list[i] == pair[0][0] and binary\_secret\_message[j] == '0':

            character\_list[i] = pair[0][1]

            number\_of\_replacement += 1

            j += 1

        if character\_list[i] == pair[1][0] and binary\_secret\_message[j] == '1':

            character\_list[i] = pair[1][1]

            number\_of\_replacement += 1

            j += 1

        # If number\_of\_replacement == len(secret\_message\_in\_binary), then break the loop

        if number\_of\_replacement == len(binary\_secret\_message):

            break

    # Write all character list elements after hiding secret message

    f = open(container\_file\_out, "wb")

    for character in character\_list:

        f.write(character.encode('utf8'))

    f.close()

# Unhide secret message from a container file, return secret message in clear text

def unhide\_secret\_message(container\_file, method):

    # Read all characters from the container file into a list

    character\_list = []

    with codecs.open(container\_file, encoding='utf-8') as f:

        for line in f:

            for character in line:

                character\_list.append(character)

    binary\_secret\_message = ''

    if method == '1':

        for i in range(0, len(character\_list), 1):

            if character\_list[i] == pair[0][1]:

                binary\_secret\_message += '0'

            elif character\_list[i] == pair[1][1]:

                binary\_secret\_message += '1'

    elif method == '2' or method == '3':

        for i in range(0, len(character\_list), 1):

            if character\_list[i] == '\n':

                if (character\_list[i - 2] + character\_list[i - 1]).count(pattern) == 1:

                    binary\_secret\_message += '0'

                elif (character\_list[i - 2] + character\_list[i - 1]).count(pattern) == 2:

                    binary\_secret\_message += '1'

    return decode(binary\_secret\_message)

# The main program

if \_\_name\_\_ == "\_\_main\_\_":

    # Create a dictionary for storing {character:binary} pairs

    dictionary = {}

    for i in range(1072, 1104, 1):

        # Add a new key:value pair

        dictionary.update( {chr(i) : bin(i-1072)[2:].zfill(5)} )

        # Method dict.update({key: value}) appends a key-value pair to the dictionary 'dict'

        # Method chr(int) returns a character from an integer 'int'

        # Method bin(int)[2:] return the binary equivalent string of a given integer 'int'

        # Method binary\_string.zfill(int) pads binary\_string on the left with zeros to fill width 'int'

    print("1. Replace 'o' and 'p' in English with 'о' and 'р' in Russian")

    print("2. Add 1 or 2 spaces before '\\n'")

    print("3. Add 1 or 2 null characters before '\\n'")

    method = input("Choose one method you want to use: ")

    # The pair dictionary containing 2 tuples as its values

    pair = {}

    if method == '1':

        pair[0] = ('o', chr(1086))  # Function chr(1086) returns character 'o' in Russian, bit 0

        pair[1] = ('p', chr(1088))  # Function chr(1088) returns character 'p' in Russian, bit 1

    elif method == '2':

        pair[0] = ('\n', ' \n')     # Add 1 space right before \n character, bit 0

        pair[1] = ('\n', '  \n')    # Add 2 spaces right before \n character, bit 1

        # The following variable is used for function unhide\_secret\_message()

        pattern = ' '

    elif method == '3':

        pair[0] = ('\n', '\x00\n')      # Add 1 NULL character right before \n character, bit 0

        pair[1] = ('\n', '\x00\x00\n')  # Add 2 NULL characters right before \n character, bit 1

        # The following variable is used for function unhide\_secret\_message()

        pattern = '\x00'

    print("Current working directory:", os.getcwd())

    secret\_message = input("Enter your secret message: ")

    print("Hiding secret message into the container\_before file...")

    hide\_secret\_message('container\_before.txt', secret\_message, 'container\_after.txt')

    print("Extract secret message from the container\_after file...")

    print(unhide\_secret\_message('container\_after.txt', method))

Программа для анализа: analyzer.py

# Function to count how many bytes in a file

def count\_bytes(file):

    count = 0

    with open(file, 'rb') as f:

        while True:

            byte = f.read(1)

            if not byte:

                break

            count += 1

        return count

# The main program

if \_\_name\_\_ == "\_\_main\_\_":

    length\_before = count\_bytes('container\_before.txt')

    length\_after = count\_bytes('container\_after.txt')

    print("Length before: {} bytes".format(length\_before))

    print("Length after: {} bytes".format(length\_after))

    print("Difference: {:.2f} %". format((abs(length\_after - length\_before) / length\_before) \* 100))