**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»

Выполнила студентка

группы N3350

Нгуен Ван Куанг



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>

Метод хвостовых пробелов предполагает дописывание в конце каждой строки файла-контейнера одного или нескольких пробелов для кодирования единичного или нулевого бита стеганосообщения. Пропускная способность этого метода сильно зависит от используемого файла-контейнера: чем больше в файле строк, тем большее количество информации в нем можно скрыть. Если принять вышеописанный формат файла-контейнера, можно вычислить, что пропускная способность стеганосистемы составляет 0.15%. Важно отметить, что при распечатке файла-результата скрытая информация будет потеряна: на бумаге или ином носителе нельзя будет выявить пробелы в конце строк. В целом можно признать метод возможным для применения, особенно если использовать предварительное сжатие и шифрование стеганосообщения. Пропускная способность такой криптосистемы довольно низкая. (<http://www.nestego.ru/2012/05/blog-post_03.html>)

Добавление служебных символов же предполагает встраивание в текст символов, которые мы используем в повседневной жизни, например запятые, тире, двоеточия и так далее, а также некоторых непечатных символов, которые невозможно увидеть.

Каждый из данных методов имеет свои преимущества и недостатки.

**Практика**

Для того, чтобы реализовать вышеописанные методы, мною было написано 3 программы, листинг которых можно найти в приложении к данному отчету, которое приведено после списка литературы.

Для написания программ использовался язык Python v.3.8.1, так как мне нравится работа со строками в данном языке. Питон медленнее C/C++, но в данном случае мы не заинтересованы в космической скорости работы алгоритма, сотые доли секунд никак не влияют на эффективность программ.

Все программы построены по схожему принципу: в начале мы переводим наше сообщение, которое хотим спрятать, в двоичный код, и сохраняем его в строке, после чего начинаем посимвольно копировать исходный текст в новый файл, который станет стегоконтейнером. Как только в исходном тексте встречается символ, который следует заменить, программа вставляет вместо него в стегоконтейнер нужный нам символ, означающий единичный или нулевой бит, после чего продолжает свою работу. Так, в первой программе русская «о» заменяется английской, и это означает 0. Русская «р» заменяется английской, и это означает 1. Во второй программе 1 tab после знака абзаца равняется 1, 2 tabs равняются 0. В третьей программе знак “пробел” заменяется знак, который имеет типе ACSII = 127 означает 0, и = 160 означает 1 .

Я использовал функцию random для определения порядка вставки (вставить короткий до длинного или же после). Сделано это для того, чтобы при визуальном осмотре стегоконтейнера не бросалось в глаза то, что в начале текста все пробелов в конце текста. Для того, чтобы программа понимала, где заканчивается сообщение и начинается случайная абракадабра, я использовал символ длинного пробела, вставляя его после последнего измененного знака тире. Таким образом, программа извлекает из стегоконтейнера стего верно. Далее мы можем видеть примеры работы моих программ:

**Метод 1: Замены символов одинакового начертания.**

A screenshot of a cell phone

Description automatically generated

A screenshot of a social media post

Description automatically generated

A screenshot of a social media post

Description automatically generated

**Метод 2:** **Хвостовых пробелов.**

A screenshot of a cell phone

Description automatically generated

A screenshot of a cell phone

Description automatically generated

A screenshot of a social media post

Description automatically generated

**Метод 3:** **Добавление служебных символов.**

A screenshot of a cell phone

Description automatically generated

A screenshot of a social media post

Description automatically generated

A screenshot of a social media post

Description automatically generated

Мы видим, что программы работают корректно, обеспечивая встраивание стего в текст. Теперь необходимо выяснить целесообразность данного встраивания. Изначальный объем текста равен 80 КБ.

В первом случае в текст было встроено 40 бит информации, на первый взгляд размер не изменился – 80 КБ (рисунок 1)

A screenshot of a computer screen

Description automatically generated

Рисунок 1. Размеры файлов.

Во втором случае в текст было встроено также 40 бит, размер также на первый взгляд не изменился – 80КБ (рисунок 2)

A screen shot of a computer

Description automatically generated

Рисунок 2. Размеры файлов.

В третьем случае было встроено 80 бит полезной информации, но размер файла увеличился на 2 КБ (рисунок 3).

A screenshot of a computer screen

Description automatically generated

Рисунок 3. Размеры файлов.

Можно сделать вывод, что в первых двух случаях встраивание целесообразно, так как текст сам по себе весит внушительное количество байт, из-за чего сообщение скрыто, а вот в третьем случае – нет. Это произошло потому, что изменяются не только необходимые символы, но и все последующие, чтобы обеспечить визуальную незаметность сообщения в стегоконтейнере. К сожалению, размер файла при этом сильно меняется.

Если для просмотра сообщения пользоваться программой, которая не поддерживает отображение всех знаков, то во всех случаях обнаружить сообщение крайне сложно. Если не знать наверняка, что в тексте что-то содержится, то неподготовленный человек ничего не обнаружит. Если же использовать программу, которая позволит отображать все символы, то первый случай сразу же вызовет подозрения, во втором, если приглядываться, тоже можно понять, что что-то не так, а вот в третьем все не так просто. Создается впечатление, что это просто такой стиль письма. Но если иметь исходный текст и стегоконтейнер, полученный с помощью третьего метода, то можно с легкостью сделать вывод, что что-то было встроено, посмотрев на размеры файлов. Этот факт стал для меня решающим при проведении экспертной оценки – данный метод скрытия я не могу назвать успешным.

**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 использованной литературы**

1. Текстовая стеганография: Метод знаков одинакового начертания – [Электронный ресурс]. – Электрон. дан. – URL: <http://www.nestego.ru/2012/05/blog-post_05.html> (дата обращения: 09.04.2020)
2. Стеганография & путешествия <http://www.nestego.ru/2012/05/blog-post_05.html> суббота, 5 мая 2012 г.
3. СТЕГАНОГРАФИЯ - Криптографические методы защиты информации <https://www.sites.google.com/site/anisimovkhv/learning/kripto/lecture/tema19>
4. Текстовая стеганография: Метод хвостовых пробелов [Электронный ресурс]. – Электрон. дан. – URL: <http://www.nestego.ru/2012/05/blog-post_11.html> (дата обращения: 09.04.2020)

**Приложение**

Программа 1. Метод замены символов одинакового начертания.

import codecs

def transforms(text, key):

bina = ''

out\_text = ''

j = 0

for i in range(len(key)):

bina += bin(ord(key[i])-848)[2::]

#русская p = 1, русская o = 0

i = 0

while j < len(bina):

if text[i] == 'р' and bina[j] == '1':

out\_text += 'p'

j += 1

elif text[i] == 'о' and bina[j] == '0':

out\_text += 'o'

j += 1

else:

out\_text += text[i]

i += 1

out\_text += text[i::]

fout = codecs.open('out\_text.txt', mode = 'w',encoding='UTF-8')

fout.write(out\_text)

fout.close()

def decodes():

fout = codecs.open('out\_text.txt', mode = 'r',encoding='UTF-8')

text = fout.read()

fout.close()

code = ''

key = ''

for i in range(len(text)):

if text[i] == 'p':

code += '1'

if text[i] == 'o':

code += '0'

arr = [code[x:x+8] for x in range(0, len(code), 8)]

for i in arr:

key += chr(int(i, 2)+848)

fdecode = codecs.open('decoded.txt', mode = 'w',encoding='UTF-8')

fdecode.write(key)

fdecode.close()

fin = codecs.open('in\_text.txt', mode = 'r',encoding='UTF-8')

text = fin.read()

fin.close()

options = input('Options:\n1: For encode\n2: For decode\nYour option: ')

if options == '1':

key = input('Input your key: ')

transforms(text, key)

print("Set key!")

elif options == '2':

decodes()

print("Check your decode.text!")

Программа 2. Метод хвостовых пробелов.

import codecs

def tabs(text, key):

bina = ''

out\_text = ''

j = 0

for i in range(len(key)):

bina += bin(ord(key[i])-848)[2::]

i = 0

#1 tab = 1, 2 tabs пробел = 0

while j < len(bina):

if ord(text[i]) == 10 and bina[j] == '1':

out\_text += '\t\n'

j += 1

elif ord(text[i]) == 10 and bina[j] == '0':

out\_text += '\t\t\n'

j += 1

elif ord(text[i]) != 13:

out\_text += text[i]

i += 1

out\_text += text[i::]

fout = codecs.open('out\_text.txt', mode = 'w',encoding='UTF-8')

fout.write(out\_text)

fout.close()

def decodes():

fout = codecs.open('out\_text.txt', mode = 'r',encoding='UTF-8')

text = fout.read()

fout.close()

code = ''

key = ''

for i in range(len(text)):

if text[i] == '\n':

if text[i-1] == '\t':

if text[i-2] == '\t':

code += '0'

else:

code += '1'

arr = [code[x:x+8] for x in range(0, len(code), 8)]

for i in arr:

key += chr(int(i, 2)+848)

fdecode = codecs.open('decoded.txt', mode = 'w',encoding='UTF-8')

fdecode.write(key)

fdecode.close()

fin = codecs.open('in\_text.txt', mode = 'r',encoding='UTF-8')

text = fin.read()

fin.close()

options = input('Options:\n1: For encode\n2: For decode\nYour option: ')

if options == '1':

key = input('Input your key: ')

tabs(text, key)

print("Set key!")

elif options == '2':

decodes()

print("Check your decode.text")

Программа 3. Добавление служебных символов.

import codecs

import random

a = chr(160)

b = chr(127)

def my(text, key):

bina = ''

out\_text = ''

for i in range(len(key)):

bina += bin(ord(key[i])-848)[2::]

i = 0

j = 0

while j < len(bina):

if text[i] == ' ' and bina[j] == '1':

out\_text += a

j += 1

elif text[i] == ' ' and bina[j] == '0':

out\_text += b

j += 1

elif ord(text[i]) != 13:

out\_text += text[i]

i += 1

out\_text += chr(8195)

k = i

for k in range(len(text)):

if text[k] == ' ' and random.getrandbits(1) == 0:

out\_text += b

elif text[k] == ' ' and random.getrandbits(1) == 1:

out\_text += a

else:

out\_text += text[k]

fout = codecs.open('out\_text.txt', mode = 'w',encoding='UTF-8')

fout.write(out\_text)

fout.close()

def decodes():

fout = codecs.open('out\_text.txt', mode = 'r',encoding='UTF-8')

text = fout.read()

fout.close()

code = ''

key = ''

i = 0

s = text.find(chr(8195))

while i < s:

if text[i] == a:

code += '1'

i += 2

elif text[i] == b:

code += '0'

i += 2

else:

i += 1

arr = [code[x:x+8] for x in range(0, len(code), 8)]

for i in arr:

key += chr(int(i, 2)+848)

fdecode = codecs.open('decoded.txt', mode = 'w',encoding='UTF-8')

fdecode.write(key)

fdecode.close()

fin = codecs.open('in\_text.txt', mode = 'r',encoding='UTF-8')

text = fin.read()

fin.close()

options = input('Options:\n1: For encode\n2: For decode\nYour option: ')

if options == '1':

key = input('Input your key: ')

my(text, key)

print("Set key!")

elif options == '2':

decodes()

print("Check your decode.text")