|  |
| --- |
|  |
| МИНОБРНАУКИ РОССИИ |
| Федеральное государственное бюджетное образовательное  учреждение высшего образования  **«МИРЭА - Российский технологический университет»**  **РТУ МИРЭА** |
| Институт кибернетики |
| Кафедра информационной безопасности |
|  |

**КУРСОВАЯ РАБОТА**

по дисциплине

«Методы программирования»

Тема курсовой работы

**«Стеганография в формате .bmp»**

Студент группы ККСО-04-19: Бадаев Р.И. \_\_\_\_\_\_\_\_\_\_\_\_

Руководитель курсовой работы: Кирюхин В.А. \_\_\_\_\_\_\_\_\_\_\_\_

Работа представлена к защите «\_\_» \_\_\_\_\_\_\_\_\_\_ 2021 г.

Допущен к защите «\_\_» \_\_\_\_\_\_\_\_\_\_ 2021 г.

Москва 2021

Оглавление

1. [Цифровые отпечатки 3](#_Toc91445726)
2. [Стеганографические водяные знаки 4](#_Toc91445727)
3. [Скрытая передача данных (СПД) 4](#_Toc91445728)

[Практическое применение стеганографии 6](#_Toc91445729)

1. [1. Незаметная передача информации (СПД) 6](#_Toc91445730)
2. [2. Скрытое хранение информации (СПД) 6](#_Toc91445731)
3. [3. Недекларированное хранения информации (СПД) 7](#_Toc91445732)
4. [5. Защита авторского права 9](#_Toc91445733)

Теория

Стеганография — способ передачи или хранения информации с учётом сохранения в тайне самого факта такой передачи (хранения). Этот термин ввёл в 1499 году аббат бенедиктинского монастыря Св. Мартина в Шпонгейме Иоганн Тритемий(он считается одним из основоположников криптографии, которой посвящен его незавершенный труд «Стеганография» )

Стеганография в ХХI веке

 Я думаю каждый хоть раз слышал о стеганографии.Своими словами хочу сказать, что это некая междисциплинарная наука и искусство передавать сокрытые данные, внутри других ,не сокрытых данных. Скрываемые данные обычно называют стегосообщением, а данные, внутри которых находится стегосообщение называют контейнером.

Цели стеганографии

В стеганографии существуют три цели.

## Цифровые отпечатки

Данный вид стеганографии подразумевает наличие различных стеганографических меток-сообщений, для каждой копии контейнера. Например ЦО могут быть применимы для защиты исключительного права . Если с помощью какого-либо алгоритма противник сможет извлечь ЦО из контейнера, то идентифицировать противника невозможно, но до тех пор, пока противник не научится подделывать ЦО, он не сможет без обнаружения распространять защищаемый контейнер.  
  
Таким образом, при извлечении ЦО третья сторона (т.е. противник) может преследовать две цели:

1. извлечение ЦО из контейнера («слабая цель»);
2. подмена одного ЦО другим ЦО («сильная цель»).

В качестве примера ЦО можно привести продажу электронных книг (например в формате \*.PDF). При оплате книги и отправки её получателю можно в \*.pdf вкраплять информацию о e-mail; IP; данных, введенные пользователем и т.д. Конечно это не отпечатки пальцев и не анализ по ДНК, но, согласитесь, это лучше, чем ничего. Возможно в России, по причине иной культуры и иного, исторически сложившегося, отношения к исключительному праву данное применение стеганографии неактуально; но, например, в Японии, где за скачивание torrent-файлов могут посадить, применение стеганографических ЦО более вероятно.

## 

## Стеганографические водяные знаки

В отличие от ЦО, СВЗ подразумевает наличие одинаковых меток для каждой копии контейнера. В частности СВЗ можно использовать для подтверждения авторского права. Например, при записи на видеокамеру можно в каждый кадр вкраплять информацию о времени записи, модели видеокамеры и/или имени оператора видеокамеры.  
В случае если отснятый материал попадет в руки конкурирующей компании, вы можете попытаться использовать СВЗ для подтверждения авторства записи. Если ключ держать в секрете от владельца камеры, то с помощью СВЗ можно подтверждать подлинность фото и/или видео снимков. Кстати, наш коллега по цеху, Дмитрий Витальевич Скляров, успешно поломал стеганографию на некоторых моделях камеры Canon. Проблема правда была аппаратной, саму стеганку Дмитрий Витальевич не трогал, тем не менее он стеганографически «доказал» подлинность Сталина с iPhone'ом.

## 

## Скрытая передача данных (СПД)

Это «классическая» цель стеганографии, известная со времен Энея Тактика Задача — передать данные так, чтобы противник не догадался о самом факте появления сообщения.  
  
В современных русскоязычных работах, посвященных стеганографии, часто используется термин ЦВЗ (Цифровые водяные знаки). Под этим термином подразумевают то СВЗ, то ЦО. (А иногда СВЗ и ЦО одновременно, да еще в одной статье!) Тем не менее при реализации ЦО и СВЗ возникающие проблемы и задачи принципиально различные! Действительно, СВЗ на всех копиях электронного документа одинаков, а ЦО на всех копиях документов различен. По этой причине, например, атака сговором принципиально невозможна в СВЗ! Хотя бы по этой причине следует различать СВЗ и ЦО. Всех, кто собирается работать в области стеганографии, настоятельно советую не употреблять термин ЦВЗ в своей речи.  
  
Данная, казалось бы очевидная мысль, до сих пор у многих вызывает недоумение. Аналогичную точку зрения о необходимости различать СВЗ и ЦО высказывали такие небезызвестные в узких кругах «стеганографы», как Кашен (Cachin), Петикола (Petitcolas), Каценбейзер (Katzenbeisser).  
  
Для каждой из этих трех целей следует разрабатывать свои собственные критерии стойкости стеганографической системы и формальные информационно-теоретические модели для их достижения, т.к. смысл применения стеганографии различен. Про фундаментальное отличие СВЗ и ЦО написано выше. Но может быть имеет смысл объединить СПД с ЦО или с СВЗ? Нет! Дело в том, что смыслом СПД является сама скрытая передача данных, а ЦО и СВЗ предназначены для защиты самого контейнера. Более того, сам факт наличия ЦО или СВЗ может не быть тайным, в отличие от большинства задач для СПД. В частности, по этой причине говорить о возможности построения совершенной стегосистемы для реализации ЦО или СВЗ для большинства практических задач не имеет никакого практического смысла.  
  
Международный журнал Springer Information Hidding вообще предлагает под стеганографией называть только СПД, так как ЦО и СВЗ не требуют сокрытия самого факта передачи данных. Например, вы знаете, что купюра в 100 рублей защищена. Некоторые механизмы вам известны, некоторые известны только специалистам, а некоторые содержат государственную тайну. Но сам факт о том, что существуют некие секретные механизмы всем известны; неизвестно только каким образом и какие именно технологии дополнительно защищают бумажную купюру…  
  
Называние стеганографией только цель СПД в журнале Springer кажется разумным, но русскоязычный термин "Информационное сокрытие" в отечественных статьях и диссертациях пока не прижился. Поэтому в данной работе под термином Springer'а Information Hiding будем подразумевать всю стеганографию, а под термином Springer'а steganography будем подразумевать только одну цель стеганографии — СПД.

Практическое применение стеганографии

# Практическое применение стеганографии

Обсудив цели, перейдем к практическим применениям. Я нашел 15 задач, для которых может быть актуальна стеганография. Я расскажу вам о 5 из них.

## 1. Незаметная передача информации (СПД)

Самое очевидное, что первое приходит на ум. В отличие от криптографических методов (которые тайны, но не скрытны), стеганография может применяться как метод незаметной передачи информации. Это составляет «классическое практическое применение» стеганографии, поэтому данная цель — на первом месте.

## 2. Скрытое хранение информации (СПД)

Данная цель стеганографии во многом похожа на предыдущую. Только в данном случае стеганография используется не для передачи, а для хранения какой-либо информации, обнаружение самого факта наличия которой (пускай хоть даже в зашифрованном виде) пользователю нежелательно. Очевидно, что данная задача реализуема на носителях данных, но не в каналах связи. Причем избыточность на многих носителях может быть невероятно большой. Например общий объем данных (с учетом кодов RLL), которые можно записать на CD диск составляют 1828 Мб данных. Это огромная избыточность, которую можно использовать для сокрытия данных!  
  
Если бы Гена Рыжов из фильма ХОТТАБЫЧ подумал бы об этом и не поленился бы немножко попаять и попрогать, то он вряд ли хранил бы CD'шки с «компрометирующим» ПО в горшке из под кактуса. Я думаю хакер Геннадий просто бы вкраплял данные в ECC оптических дисков, а сами диски с фотографиями котиков хранил бы в открытую! Согласитесь, это гораздо лучше горшка из под кактуса!

## 3. Недекларированное хранения информации (СПД)

Многие информационные ресурсы позволяют хранить данные только определенного вида. Например портал YouTube позволяет хранить только видеоинформацию в форматах MOV, MPEG4, AVI, WMV, MPEG-PS, FLV, 3GPP, WebM. Однако можно использовать стеганографию для хранения данных в других форматах. Не спорю, что в условиях существования различных ресурсов наподобие Yandex Disk, данная цель может показаться странной. Скорее всего практической значимости и нет; только just4fun и забавная курсовая для студента.  
  
Однако сайт hid.im позволяет пользователям скрывать файлы .torrent внутри изображений PNG. Вот как прокомментировал Майкл Наттом (Michael Nutt), создатель проекта:

Это попытка сделать торренты более живучими. Разница в том, что нет больше нужды именно в индексирующем сайте, чтобы хранить ваш торрентовый файл. Многие форумы дозволяют закачку картинок, но более никаких других типов файлов

Есть так же проектStegTorrent, который в отличие от онлайн-сервиса hid требует установки.  
  
В феврале 2015 года в Хакере опубликовали заметку "Десятки тысяч баз MongoDB доступны через интернет". В принципе данные «косяки», помимо прочего, можно использовать для недекларированного хранения данных.

4. Защита исключительного права (ЦО)

В качестве возможного применения можно привести голографический многоцелевой диск. (Правда есть точка зрения, что данная технология изначально «мертворожденная» ) Разрабатываемые нынеHVB могут содержать до 200 Гб данных на один cartridge. Эти технологии предполагают использовать компаниями теле и радиовещания для хранения видео и аудио информации. Наличие ЦО внутри корректирующих кодов этих дисков может использоваться в качестве основного или дополнительного средства для защиты лицензионного права.  
  
В качестве другого примера, как я уже писал ранее, можно привести интернет-продажу информационных ресурсов. Это могут быть книги, фильмы, музыка и т.д. Каждая копия должна содержать ЦО для идентификации личности (хотя бы косвенной) или специальную метку для проверки лицензионная это копия или не лицензионная.  
  
Данную цель попыталась воплотить в 2007-2011 годах компания amazon.com. Цитата artty из статьи ”Защита” mp3 файлов на «Защита» mp3 файлов на amazon.com:

Если по-русски: скачанный файл будет содержать уникальный идентификатор покупки, дату/время покупки и др. информацию (...).  
  
Скачать в лоб данные композиции не получилось (амазон ругается и говорит, что может их продать только на территории США). Пришлось попросить американских знакомых и через некоторое время у меня на руках была одна и та же песня, но скачанная независимо двумя разными людьми с разных аккаунтов в амазоне. По виду файлы были абсолютно одинаковы, размер совпадал до байта.  
  
Но т.к. амазон писал, что включает в каждый мп3 идентификатор загрузки и еще кое-какие данные решил проверить два имеющихся файла побитово и сразу нашел различия.

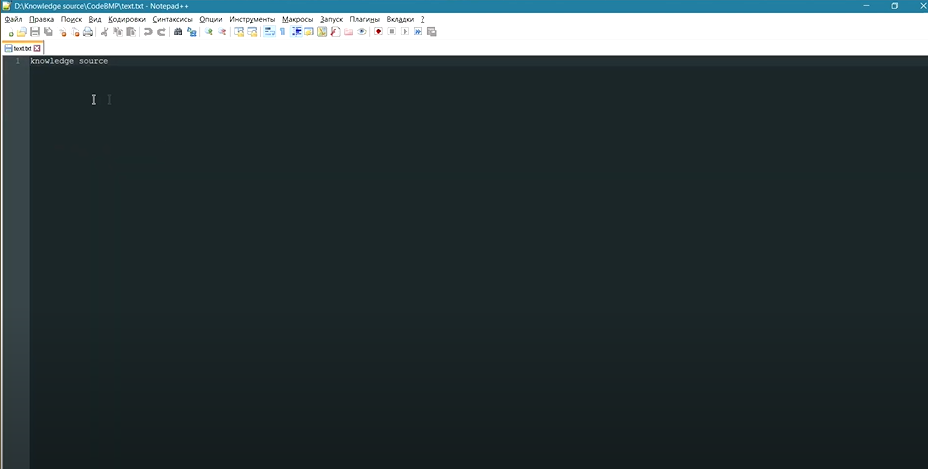
## 5. Защита авторского права

В данном случае одним знаком защищается каждая копия контента. Например это может быть фотография. В случае если фотографию опубликуют без разрешения фотографа, сказав, что якобы не он автор данной работы, фотограф может попытаться доказать свое авторство с помощью стеганографии. В данном случае в каждую фотографию должна вкрапляется информация о серийном номере фотоаппарата и/или какие либо иные данные, позволяющие «привязать» фотографию к одному единственному фотоаппарату; и через фотоаппарат фотограф может попытаться косвенно доказать, что именно он является автором снимка.

Реализация

Для начала мы подготовим текст, который мы будем шифровать

Я буду шифровать сообщение “knowledge source”



Далее подготовим изображение формата .bmp



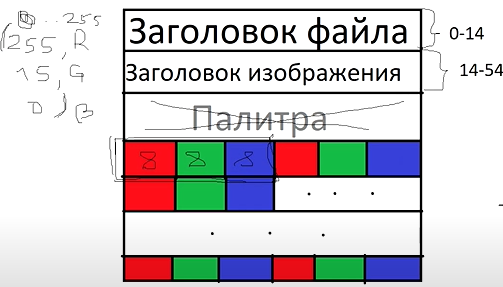
Сейчас я объясню, почему мы используем .bmp разрешением 24 бита.

Мы задаем цвет тремя параметрами. Каждое число у нас могло быть от 0 до 255.Этим мы кодируем три цвета Красный Зеленый Синий, если у нас числа занимают позиции от 0 до 255 это значит, что они занимают 8 бит или 1 байт.

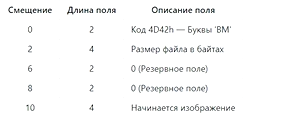
8 Бит,3 цвета значит 24 бита на кодировку.

Итак, у нас один пиксель задается смешиванием трех цветов: Красный Зеленый Синий, каждый из этих цветов занимает 8 бит(в сумме это 24 бита)

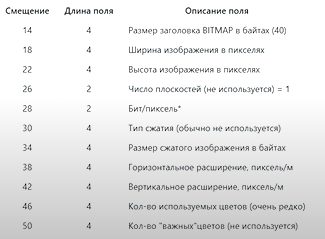
Справочка: почему мы используем именно формат bmp, а не какой-нибудь png или jpeg, потому что bmp формат не использует никакого сжатия шифрования и всего такого.



Итак, какая у нас структура bmp. У нас есть Заголовок файла.

Ниже у нас описано каждое поле заголовка файла

Также позже объясню информационные поля изображения



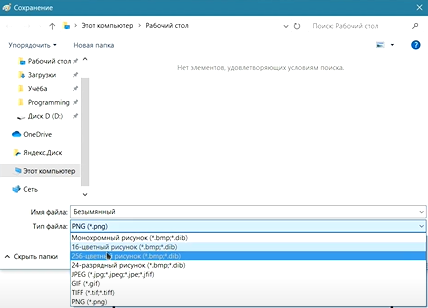
И также особенность формата bmp 24 бита –это то, что у нас отсутствует палитра. Потому что иначе как мы видим на 50 позиции «Количество «важных» цветов» у нас не используется, но все равно это все еще системная часть программы. После 54 поля идет палитра, но здесь у нас палитры не будет, поэтому кодировать мы будем с 54 поля🡪 на 54 позиции у нас стоит первый красный бит.

**Далее приступим к практике.**

Итак с чего же нам начать.

Создадим для начала python-файл в среде разработки PyCharm.

Назовем файл main.py.

Давайте для начала разберемся в стеганографии: тот объект, в котором мы будем прятать наше сообщение, при помощи которого будем скрывать факт передачи сообщения—называется стегоконтейнером, в нашем случае стегоконтейнером будет выступать изображение .bmp. Чтобы нам получить изображение bmp в формате 24 бит, мы можем открыть Paint, что-нибудь нарисовать, затем «Сохранить как»

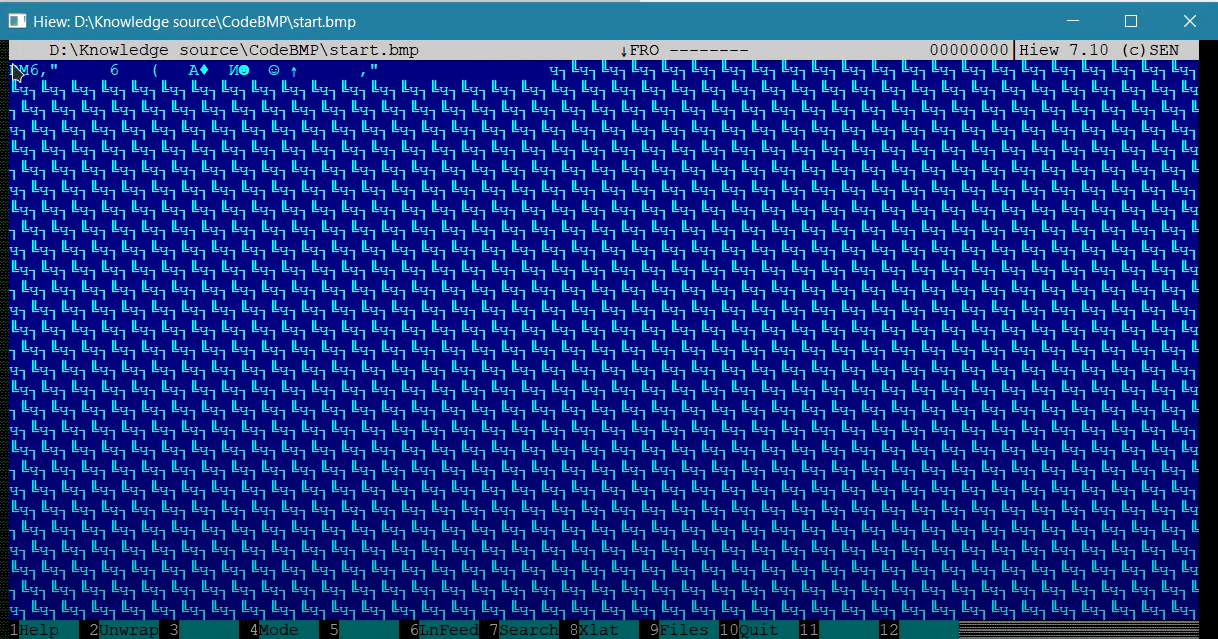
И сохраняем под типом файла «24-разрядный рисунок(\*.bmp;\*.dib)».

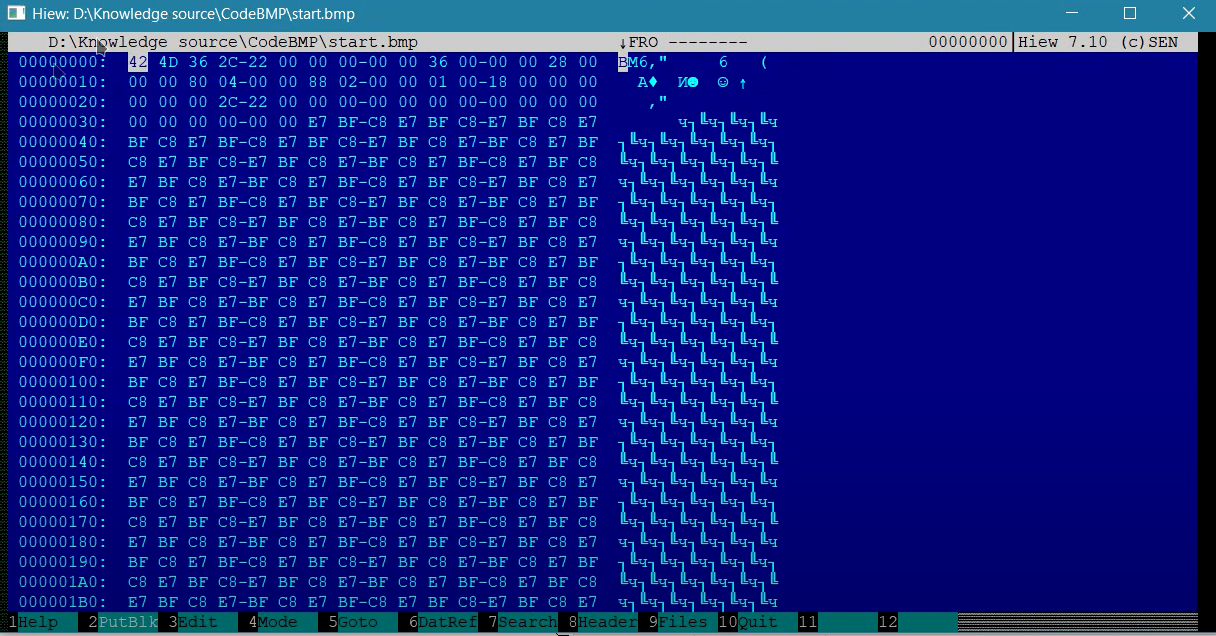
Я уже подготовил файл, его показал выше.

Еще одна утилита нужна будет нам это hiew32. 

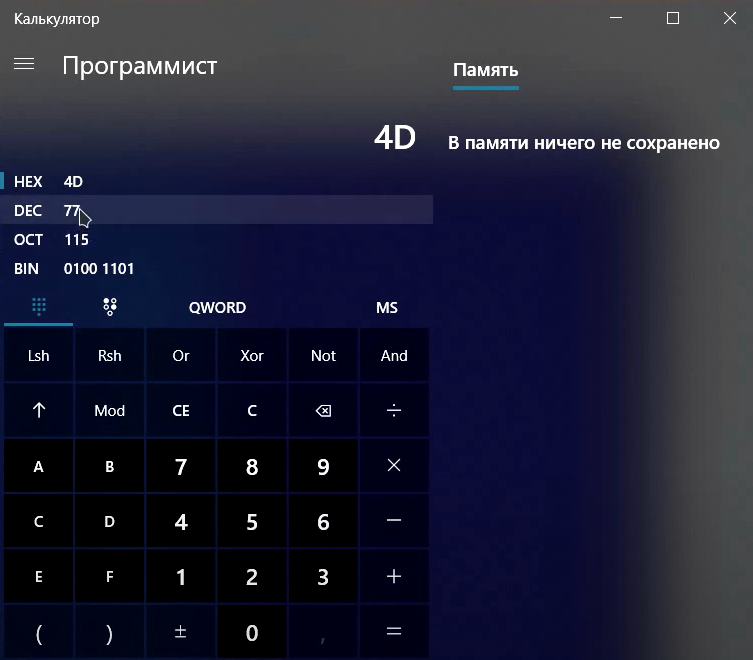
Что это такое? Это приложение дизассемблер (Дизассемблер — транслятор, преобразующий машинный код, объектный код или библиотечные модули в текст программы на языке ассемблера. Дизассемблирование – преобразование программы на машинном языке к ее ассемблерному представлению.) оно еще параллельно при этом позволяет посмотреть коды нашего файла (Это приложение не хочет работать с фалами, у которых русское название , поэтому выбирайте название на английском языке)

Давайте запустим наш файл start.bmp



Вот сейчас мы видим представление в формате ASCII символов(ASCII - это 7-битный набор символов, содержащий 128 символов. Он содержит цифры от 0 до 9, заглавные и строчные буквы английского алфавита от A до Z и некоторые специальные символы.). Здесь у нас указаны байты и их значения, первые два байта- это обозначение формата .bmp «BM», следующие 4 символа это размер файла в байтах «6,” »-эти 4 символа записаны как ASCII,а чтобы переключиться на режим чтения самих 16-ричных данных—нажимаем на Enter(если мы еще раз нажмем на Enter, то нам покажет ассемблерный код)

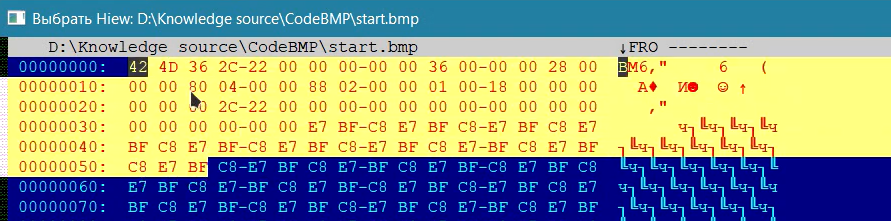
Вот теперь у нас видно Адрес от начала файла(небольшой столбик слева )—Шестнадцатиричное представление(основная часть посередине). Если мы переведем «4D» из 16-ричной в 10-чную получится 77



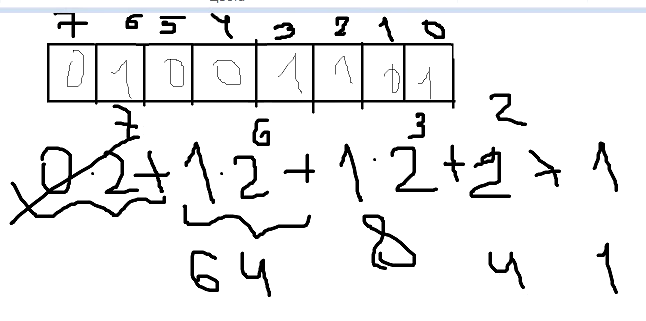
А если взглянуть на таблицу ASCII,то мы увидим, что 77 это заглавная буква М, точно также работают все остальные символы.



Как мы видим первые 54 бита у нас будет основная информация



Теперь мы поняли как прочитать что-нибудь в нашем .bmp.

А сейчас мы приступаем непосредственно к работе. Я, пожалуй, вам расскажу алгоритм, как мы вообще будем шифровать данные. Вспоминаем, что каждый цвет нашего пикселя, то есть либо красный, либо зеленый, либо синий(в сумме эти цвета дают один пиксель на экране),кодируется 8 битами или 1 байтом. В байте может лежать какая-угодно информация, допустим там лежал байт М (01001101),как работать с двоичной системой счисления я думаю уже все знают

Мы получаем число 77(то есть М).

Если мы перекроем последний бит этого числа и запишем туда 0,то у нас будет не 77 а 76. Ну и допустим у нас был какой-то цвет, выберем любой случайный



Если мы у каждого этого числа поменяем последнюю цифру в байте, 213 🡪212, 77 🡪76 , 142 🡪143 и мы получаем такой цвет



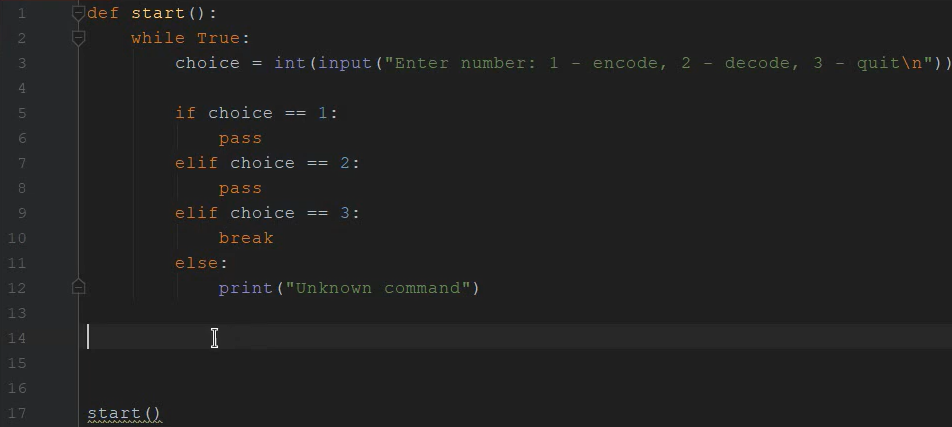
Они выглядят абсолютно одинаково, но тем не менее мы смогли последний бит из байта собрать под свою нужду. Если мы возьмем два бита из байта, максимальное отличие у нас в конце байта будет стоять два нуля и мы заменим их на две единички. То есть если мы берем то же число 77 и меняем два последних бита 0 1🡪 1 0, то число у нас уже будет не 77 78, попрежнему для человеческого глаза это будет совсем незаметно. И вот так, по мере «отхватывания куска» битов от исходного байта, мы постепенно портим качество нашего изображения, но кодируем больше информации.

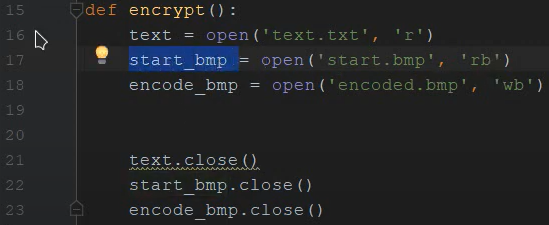
Давайте я лучше покажу это на реальном примере изображения .bmp.

Значит мы взяли какой то определенный байт из .bmp. Теперь представляем наш исходный текст, допустим «code». Мы хотим спрятать сообщение «code» в байтах изображения .bmp,берем первую букву нашего текста “c”-представляем ее в кодировке ASCII, ищем букву “c” в таблице ASCII, она обозначается цифрой 67. Мы берем это число в десятичной системе счисления (01000011 в двоичной). Мы задаем количество бит, которые мы отсекаем от поля каждого байта, допустим мы отсекаем 2 байта, «стираем» их значения, берем из нашего исходного кода из символа ASII первые два бита (это 01 из 01000011) и записываем их в последние два байта, которые мы отсекали и сохраняем этот бит в картинку. Потом берем следующий байт, опять «отсекаем» из этого байта последние два бита, и вместо них записываем следующие два бита из нашего символа «с»- это 00,сохраняем байт в картинку. Делаем аналогичные операции с оставшимися битами 0011. В итоге мы получаем практически ту же картинку с точностью до 2 бит, которую не отличить человеческому глазу, но при этом мы храним в последних битах нашего цвета закодированное сообщение. Теперь пользователь, которому мы скинули закодированное сообщение, знает об этом и он может прочитать сообщение. Мы использовали по последним двум битам 4 байтов из .bmp, он считывает первые 4 байта .bmp (пропуская заголовки). В них он извлекает по два последних символа, складывает их и получает букву «с» (01+00+00+11=01000011=«с»)

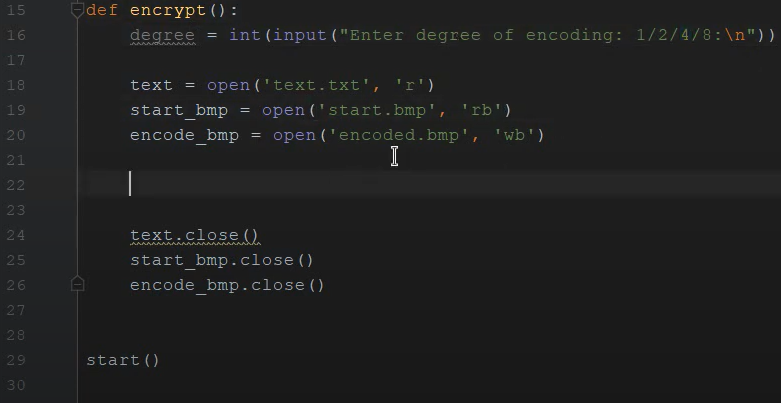
Приступим к коду.

Напишем стартовую функцию которая будет спрашивать у пользователя, что нам делать(если 1- то мы шифруем сообщение, 2- расшифровываем,3-выход из программы). Добавим бесконечный цикл чтобы пользователь вышел тогда когда ему нужно будет. И делаем ветвление на каждую из команд.



Далее займемся функциями кодирования и декодирования.Для того, чтобы нам кодировать и декодировать, нам для начала нужно открыть наше стартовое изображение, наш контейнер и наш текстовый файл, из которого мы будем считывать то, что нам нужно зашифровать,в конце не забудем, что их обязательно нужно закрыть. 

Пометочка: Открываем мы побайтно «rb»-read byte и записываем побайтно «wb»-write byte, а текст будем считывать посимвольно «r»

Далее (16 строка) мы запросим степень шифровки-это сколько бит от исходного байта .bmp мы будем шифровать,причем чем сильнее степень шифровки,тем сильнее будет пиксель отличаться от того каким он был

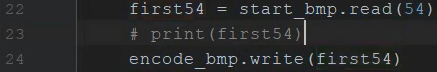
Посмотрим что лежит в первых 54 байтах нашего файла стартового



Как мы видим все тоже самое что мы видели в приложении hiew32

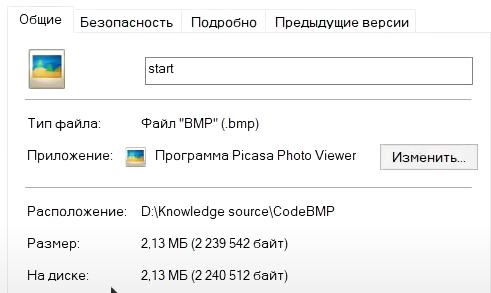


Нам нужно эти 54 байта скопировать в новую картинку



Теперь нам нужно читать следующие , после 54 байтов, данные из исходного изображения, изменять и сохранять в наш контейнер.

Мы не можем записать большой текст в изображение.Допустим наш файл весит 2 239 542 байт. Мы занимаем 1 бит из байта,то есть каждый 8 бит из байтов является битом нашего сообщения, и помним что 54 байта мы закодировать не можем,их мы вычитаем. Если наш текст меньше, чем эта величина (2 239 542\*(1/8)-54) , значит мы можем запихнуть этот текст в картинку



Чтобы извлечь размер файла импортируем в начале модуль os (import os). Сделаем проверку на длину текста сразу, как мы введем степень шифровки



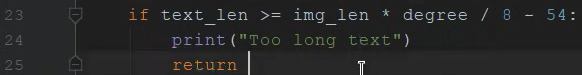
Запустим программу и увидим много всяких данных,из них нам интересно этот параметр



Сделаем тоже самое для картинки



Теперь пропишем условие, если длина текста больше чем эта величина (размер изображения \* (1/(степень шифровки))-54),то мы выйдем из этой программы, тк текст не влезет в изображение

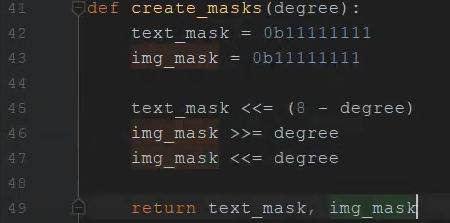


После того как мы записали первые 54 байта нам нужно записать остальные.

Пометка необязательна к прочтению

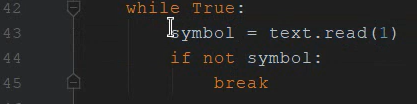
(((((Перед тем как мы к этому приступим напомню вам о логических операциях. При написании моей курсовой я пользовался некоторыми логическими операциями И(&),ИЛИ(|),ОТРИЦАНИЕ(~),СДВИГИ( << , >> ).

Создадим новую функцию которая будет в себя принимать степень упаковки в исходный .bmp, работаем исключительно в двоичной логике, поэтому давайте создадим изначально две маски.



К работе с текстом мы уже подготовились, теперь будем шифровать саму информацию.

Сначала мы пройдемся по всем символам текста. Делаем бесконечный цикл.



На этот символ нам придется некоторое кол-во байт из исходной картинки. Соответственно при степени упаковки 2 должны прочитать 4 байта. Пропишем такую строчку



Цикл нам обеспечит одинаковое количество работы с этимсимволом (43 строка) для каждого символа.

Здесь мы читаем байт изображения и преобразовываем int.from\_byte и применяем к нему маску.



Так мы получаем байт обнуленными degree битами. Теперь нам нужно получить эти же degree бит из символа. Можем это сделать следующим образом



Теперь вытаскиваем количество бит из этого символа. Что нам с этими битами нужно сделать? Мы их «сдвигаем вправо». И при помощи логического сложения мы приписываем эти биты



И теперь нам img\_byte нужно запихнуть в картинку,а это мы делаем при помощи следующей функции ”encode\_bmp.write(img\_byte.to\_bytes(1, sys.byteorder))”. Напомню вам, что если мы работаем с битовым представлением, а в текст нам нужно записать байт, то есть если мы в 48 строке преобразовываем из byte в integer, то сейчас нам нужно сделать наоборот



После того, как мы это все сделали нам нужно символ который мы обрабатываем (symbol = text.read(1)) сдвинуть влево на degree



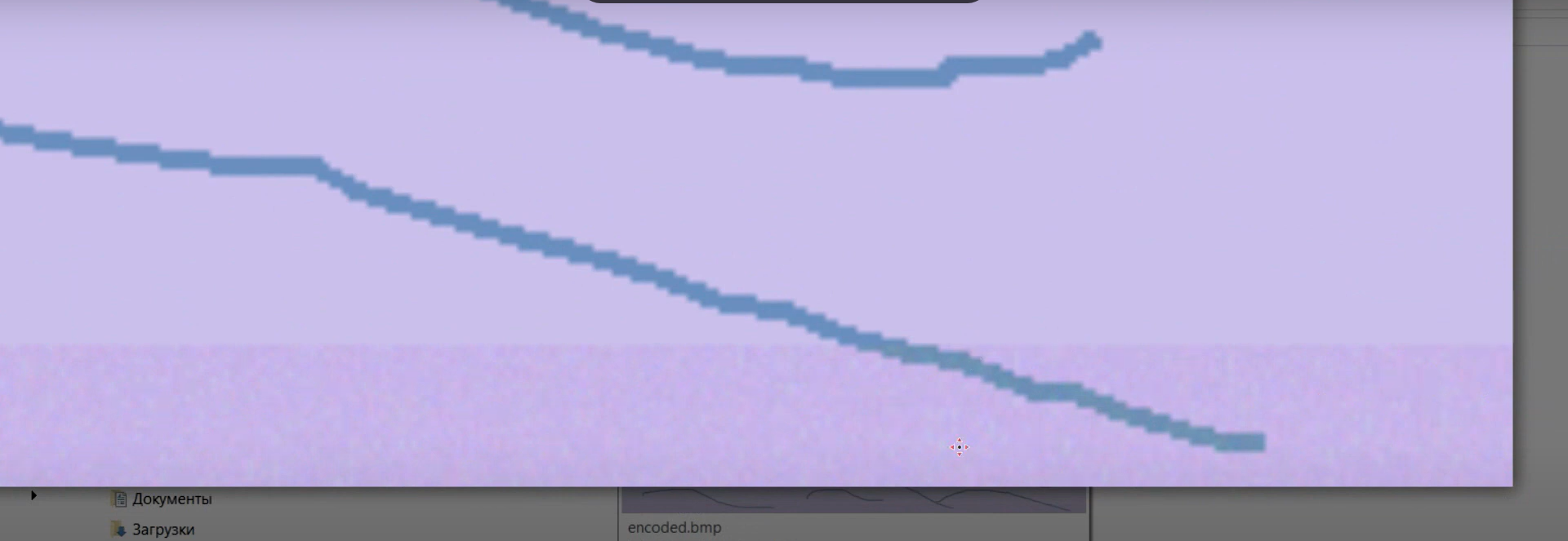
Потом как только мы записали все символы из текста конечно же этот текст будет намного короче чем наша картинка, нам нужно скопировать все оставшиеся символы, поэтому из нашего исходного файла мы берем все оставшееся, но во-первых давайте проверим на какой позиции мы сейчас находимся в исходном файле



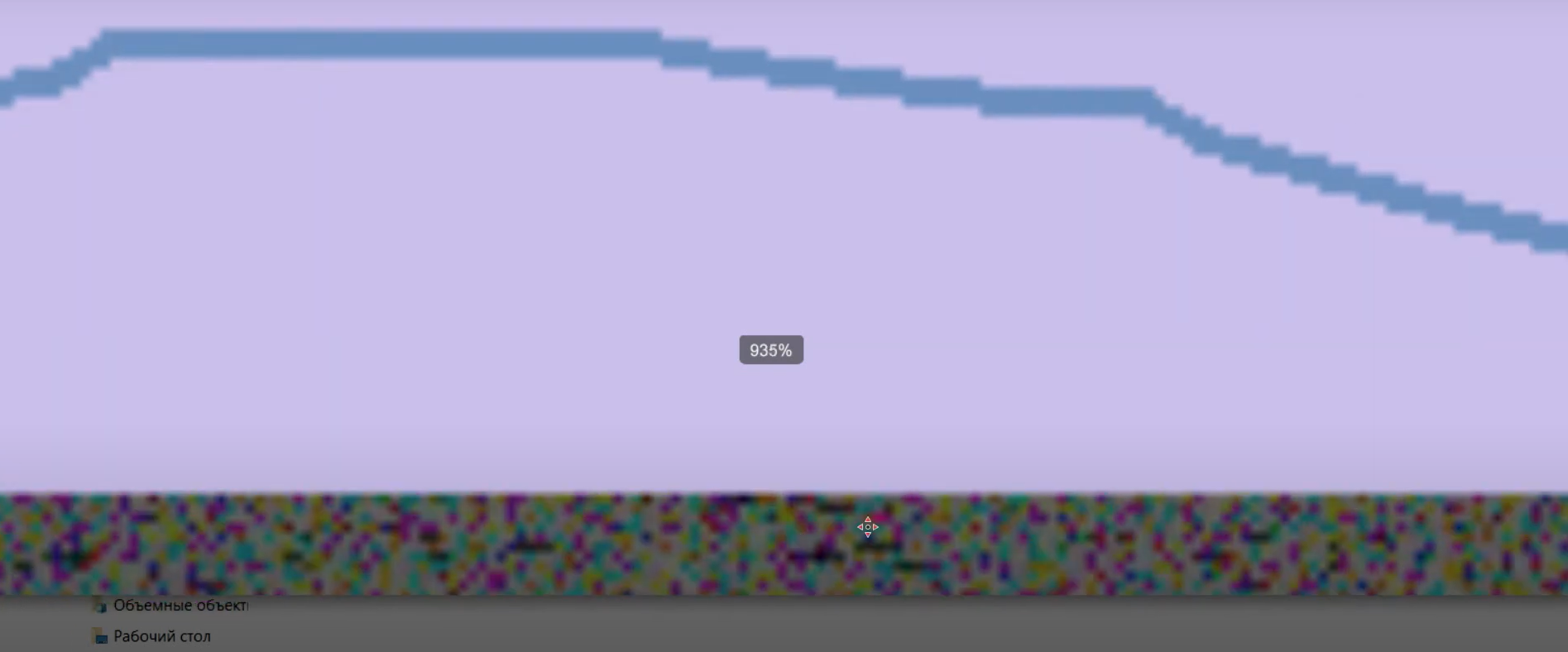
Мы получились на 118 позиции. Мы начали работать с 54 позиции, в тексте 16 символов, каждый символ мы кодируем двумя битами. Теперь мы можем посмотреть что произошло с фалом encoded, здесь будет лежать 54 символа и сверху 64 символа которые мы закодировали, и нам нужно закодировать все оставшиеся символы, то есть мы просто берем наше итоговое сообщение это у нас получается encoded\_bmp.write(start\_bmp.read()) в него мы записываем. Запускаем программу со степенью кодирования 2 и смотрим. Encoded такого же размера как и start и визуально мы их отличить никак не сможем



Как же нам вообще узнать, что у нас что-то закодировалось? Для этого я создал один большой текст text1. Открываем этот файл и будем кодировать различными степенями кодировки, здесь мы будем кодировать 1 бит из байта, в результате мы изменения визуальные тоже не увидим, если кодируем 2 битами, по прежнему ничего не видно. Попробуем 4 бита и теперь если взглянуть на наш encoded то увидим какие-то помехи уже

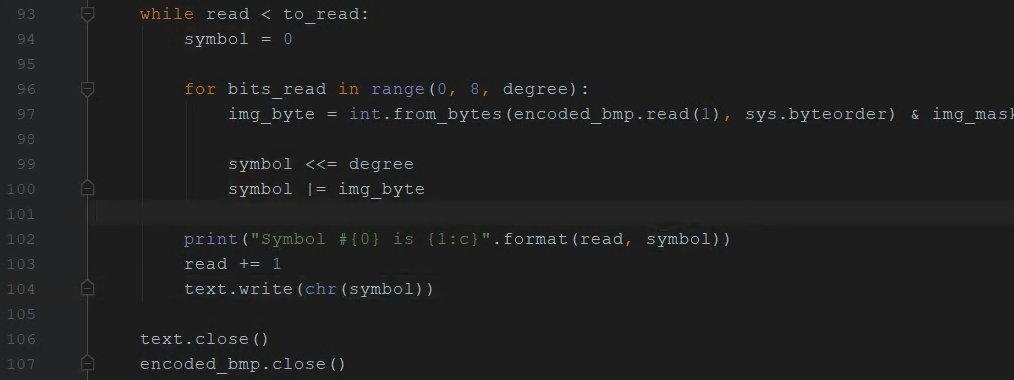
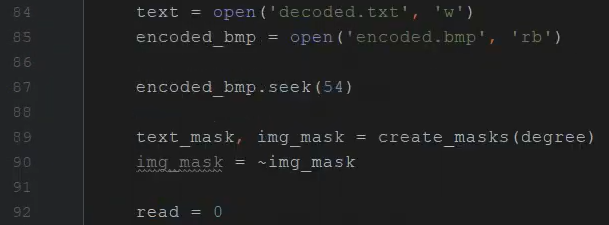
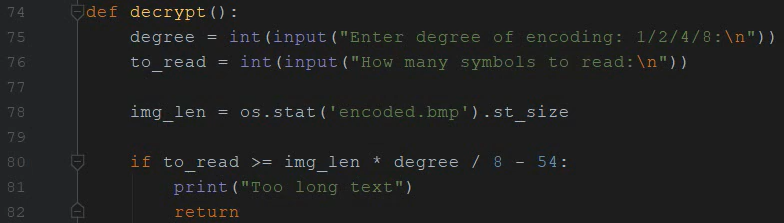


Это наши закодированные биты, которые мы меняли в палитре. А если мы возьмем степень кодировки 8, что значит, что мы будем заменять каждый из цветов палитры на абсолютно другой, то мы увидим следующее



Абсолютно случайное размещение цветов. Мы наглядно увидели шифровку текста в котором 40 000 символов.

Давайте закодируем туда наше обычное сообщение и займемся уже расшифровкой. Пишем следующую функцию



Попробуем запустить это все и мы увидим наше сообщение «knowledge source»!.

**Заключение**

Я очень надеюсь, что я смог понятно объяснить такую непростую тему, для меня это было впервое и было очень интересно изучать эту тему. В данной курсовой работе мы разобрали методы стеганографии, реализовали шифрование, показали примеры разных степеней кодировок и произвели дешифровку сообщения в изображении.

**Приложение**

import os #чтобы извлечь размер файла

import sys #системный порядок байт

def start ():

    choice = int(input("Enter number 1-encode 2-decode 3-quit\n"))

    if choise == 1:

        encrypt()

    elif choice == 2:

        decrypt()

    elif choice == 3:

        break

    else:

        print("Unknown command")

def encrypt():

    degree = int (input("Enter degree of encoding: 1/2/4/8:\n"))#степень шифровки/кодирования

    text\_len = os.stat('text.txt').st\_size #чтобы узнать размер файла

    img\_len = os.stat('start.bmp').st\_size #чтобы узнать размер файла

    if text\_len >- img\_len \* degree /8 - 54; #проверка количества символов на вместимость

        print ("Toolong text")

        return

    text = open ('text.txt', 'r') #открываем файлы; читаем их посимвольно

    start\_bmp = open ('start.bmp', 'rb') #...читаем побайтно

    encode\_bmp = open ('encoded\_bmp', 'wb') #...записываем побайтно

    first54 = start\_bmp.read(54) #посмотрим что лежит в первых 54 байтах и перепишем их в first54

    encode\_bmp.write(first54)

    text\_mask, img\_mask = create\_mask(degree)

    print("text: (0:b); image: (1:b)". format( text\_mask, img\_mask))

    print(bin(0b11111111 & text\_mask))

    print(bin(0b11111111 & img\_mask))

    while True:

        symbol = text.read(1)

        if not symbol:

            break

    print("\nSymbol (0:b), bin (1:b)". format(symbol, ord(symbol)))

    symbol = ord(symbol)

     for byte\_amount in range(0, 8, degree) #byte\_amount проходит диапазон от 0 до 8 с шагом degree

        img\_byte = int.from\_bytes(start\_bmp.read(1), sys.byteorder) & img\_mask

        bits = symbol & text\_mask

        bits>>= (8 - degree)

        print ("img (0), bits (1:b), num (1:d). format(img\_byte, bits)")

        img\_byte != bits

        print ('Encoded' \* str(img\_byte))

        print ('Writing: ' str(img\_byte.to\_bytes(1, sys.byteorder)))

        encode\_bmp.write(img\_byte.to\_bytes(1, sys.byteorder))

        symbol <<= degree

    print (start\_bmp.tell()) #чтобы узнать на какой позиции в файле мы находимся

    encoded\_bmp.write(start\_bmp.read())

    text.close()

    start\_bmp.close()

    encode\_bmp.close()

def decrypt ():

    degree = int (input("Enter degree of encoding: 1/2/4/8:\n")) #степень расшифровки

    to\_read = int(input("How many simbols to read:\n "))  #сколько символов нужно прочитать получателю

    img\_len = os.stat('encoded.bmp').st\_size #чтобы узнать размер файла

    if to\_read >- img\_len \* degree /8 - 54; #проверка количества символов на вместимость

        print ("Toolong text")

        return

    text = open ('text.txt', 'w') #откроем text.txt

    encoded\_bmp = open('encoded.bmp', 'rb') #откроем картинку

    encoded\_bmp.seek(54) #У encoded.bmp нужно пропустить первые 54 бита

    text\_mask, img\_mask = create\_mask(degree)

    img\_mask = ~img\_mask #мы можем сделать обратную самой себе и получить нужную нам маску

    read = 0 #счетчик прочитанных символов

    while read < to\_read:

        symbol = 0

    for bits\_read in range(0, 8, degree) #прочитать символы

        img\_byte = int.from\_bytes(encoded\_bmp.read(1), sys.byteorder) & img\_mask#читаем байты в сообщении

        symbol <<=degree

        symbol != img\_byte

        print("Symbol #{0} is {1:c}".format(read, symbol))

        read += 1

        text.write(chr(symbol))

    text.close()

    encoded\_bmp.close()

def create\_mask(degree):

    text\_mask = 0b11111111

    img\_mask = 0

    text\_mask<<= (8 - degree)#

    text\_mask %= 256

    img\_mask >>= (degree)

    img\_mask >>= (degree)

    return text\_mask, img\_mask

    start()

**Список литературы**

1. <https://www.youtube.com/watch?v=ZO615FnGsQk>
2. <https://habr.com/ru/post/253045/>