

# Assignment 6: Steganography

---

**Points** 100

---

## Background

Steganography is the practice of hiding one message within another. It is related to, but differs from cryptography. In steganography, the primary purpose is to sneak a message past a potential adversary. In cryptography, the primary purpose is secure communication. The precise dividing line between the fields is not completely clear and some schemes utilize components from both fields. Generally steganography does not have any shared secret component, a key, and instead relies upon some cleverness to evade detection. In contrast, cryptography requires a shared secret.

Historically the earliest versions of steganography were documented by Herodotus in his Histories. In one, a message was transmitted by one having the messenger shave their head, have the message put onto the scalp and let the hair re-grow. Once the messenger got to the recipient they were instructed to shave their head again to transmit the message. In the second case, wax tablets concealed the message. Wooden boards were covered in beeswax upon which a message could be carved as a reusable writing surface. The secret message was put upon the board prior to the wax layer being applied. To superficial checks of both the messenger and the wax tablet they would appear normal, but there was no security to these systems beyond the method of concealment being kept secret.

More modern forms of steganography do things like conceal messages in images, video files and so forth.

## Your Program

You will write a program that can both encode and decode messages into an image file.

It needs to be split into appropriate files and be built with CMake. It needs to be in at least three source / header files, but more should be used if it logically makes sense.

At its most simple, an image format is nothing more than a listing of pixels of the image with their corresponding color. This works, but it is not efficient in terms of storage size. Modern image formats use various techniques to compress the data into a smaller storage format. The complexity of these more modern formats is well beyond the scope of what we can easily deal with. So we will use a very simple format known as PPM.

## PPM Image Format

The PPM format is a lowest common denominator image format. It is designed to be simple to read and write at the expense of file size. There are six main variations. The images can be black and

white, gray scale, or color. For each of those the pixels can be specified in either binary or ASCII decimal.

To keep things simple, we will restrict our attention to the color images with ASCII decimal formatting. It supports 24-bit pixel colors, without any alpha channels. In other words, each pixel is specified in RGB format with 8-bits per color.

The file begins with a header. It specifies a width, height and color depth. For example

```
P3
10 20
255
```

The P3 specifies which PPM variation is being used. The 10 is how many pixels wide and the 20 is how many pixels high. The 255 is the maximum value for each color. We will assume this is always 255 because that corresponds to one byte per color.

After the header, there is a sequence of space separated decimal ASCII values that correspond to the RGB values of the pixels in row major order.

A full example may be:

```
P3
3 2
255
255 0 0
0 255 0
0 0 255
255 255 0
255 255 255
0 0 0
```

While there will be whitespace separating all of the numbers, there is no importance to it beyond its existence. In particular line breaks have no meaning.

If you were to research the PPM format, you'd find that comments are permitted. In particular they use a `#` to indicate a single line comment in the same manner as Python. To further keep things simple, you may assume no comments occur in the PPM for this assignment.

## Your Program Interface

Your program needs to work when invoked in the following two ways

```
$ ./steganography encode input.ppm payload.ext output.ppm
```

and

```
$ ./steganography decode input.ppm output.ext
```

In encode mode, you will hide the contents of `payload.ext` into `input.ppm` and write the result to `output.ppm`.

In decode mode, you will read `input.ppm`, extract the message and write it to `output.ext`.

Note that `payload.ext` can be an arbitrary file, binary or plain text.

## Hiding the Payload

Recall that each pixel is stored as a 3-tuple of 8-bit values for each of Red, Green, and Blue. Also it is worth knowing that two very close colors are effectively indistinguishable visually to most people under most circumstances. That is the difference between (255, 255, 255) and (255, 255, 254) will not be perceptible most of the time. We will exploit this to hide our payload.

For each pixel we will look at the least significant bit (LSB) of the green and blue components. The exclusive-or of these will be one bit of the payload.

Since our payloads are potentially binary files, we need to also encode the file size. The first 16 bits will be a unsigned 16-bit integer of the payload size in bytes. The remainder will be the actual payload.

So to encode a payload, go through the stuff to encode bit by bit. For each bit, take one pixel of the PPM, and modify the least significant bit of the blue component so that so that the exclusive-or of the LSB of the green and blue components match what you want. Meaning you might flip the LSB of blue or potentially not.

To decode a payload, read the pixels and use the exclusive-or to determine the payload bit by bit in the exact inverse of the encoding step.

## Error Conditions


There are several error conditions you will need to handle

- When encoding, ensure that the three filenames are all unique. Print an error if they are not and halt the program.
- When decoding, ensure the two filenames are unique. Print an error if they are not and halt the program.
- If the size of the payload exceeds what can be encoded in the PPM, print an error. It is acceptable to create an incorrect / partial PPM in this case, just make sure the error is printed.
- If the size encoded at the beginning of the PPM is larger than what the input PPM contains, print an error. It is acceptable to create a partial payload file in this case.
- Only PPMs in the P3 format with 255 maximum value per color need to be supported. Print an error if the PPM is in a different format.

## Test PPMs

You can create your own PPM files by using the program `netpbm` from PNG files. Just run the command

```
$ pngtopnm -plain file.png > file.ppm
```

A copy of the DU Logo without anything hidden in it is at [du\\_logo\\_ppm.zip](#) 

Forthcoming example with hidden file.



## Hints

- Read and write the PPM files using `fscanf` and `fprintf`.
- Read and write the payload file using `fread` and `fwrite`.
- It will be useful to wrap the logic of testing a bit, setting a bit, and toggling a bit as separate functions.