Steganography

Steganography is the idea of hiding a message inside another message. Steganography differs from cryptography in that there is no key required. To find the message, you simply need to know where to look. Here is an example. A message is hiding in this text.

```
Steganography is frequently
examined in popular
culture as a way to
reflect upon how
easily one can
trick others.
```

In hidden message is the word "secret," which can be found by looking at the first letter of each line. There are thousands of ways to hide messages. We will look at a few common ones. One simple trick to hide a message in a document is to change the font color to match the background color. This is the digital analog of invisible ink.

Hiding data in images

A very common technique is hiding data in an image. To understand how it works, we first need to know how colors work in computer images. Every color is broken into three components: red, green, and blue. Each of those components is commonly 8 bits, ranging from 0 to 255. In total, there are $256^3 \approx 16.8$ million colors. The key to hiding things in images is that it is very difficult for the human eye to distinguish between all of the 16.8 million colors, especially if there is only a change in the least significant bits of a pixel's components, like if we go from RGB components of $(160, 200, 80)$ to $(160, 200, 79)$.

It will help to think of the 8-bit components in binary form, like 10001010 or 00101110. The leftmost bits are the most significant ones, and a change in those will have a dramatic difference on the color. For instance, the difference between 00000000 and 10000000 is the difference between 0 and 128. The rightmost bits are the least significant, and a change in those will be likely be imperceptible to the human eye. Changing 00000000 to 00000001 changes 0 to 1.

We can actually hide an entire image inside another. In order to do that, we will do some bitwise operations. There are four operations that we will need:

- `&` — This is the bitwise AND operation. The rule is that `1 & 1 = 0` and all three other possibilities come out to 0. The most common use for `&` is as a *mask*. It allows us to pick out certain bits from a number and zero out everything else. Here is an example:

  ```
    11010101
  & 11100000
  ----------
    11000000
  ```

  To pick out the top three bits, we AND with 11100000. This acts as a window that lets the first three bits pass through unchanged and zeroes out the rest. If we want to pick out the bottom 5 bits, we would AND with 00011111.

- `^` — This is the bitwise XOR operation. The rules are that `0 ^ 1 = 1`, `1 ^ 0 = 1`, `0 ^ 0 = 0`, and `1 ^ 1 = 0`. The XOR operation acts a little like addition, and is used to combine things.

- `>>` — This is a right shift. It moves all the bits over a specified amount to the right, adding in 0s on the left as needed. For instance, `10110011 >> 3` becomes 00010110. The bits that get shifted off the end disappear.

- `<<` — This is a left shift. It works similarly to the right shift, but in the opposite direction. In Python, one thing to be careful of is if we start with an 8 bit number and shift left, Python will create a larger value. For instance, `10110011 << 3` will become 1011011000.

To hide an image, we will need an image to hide it in. We'll call that the *carrier image*. Let's assume that `r1` is the red component of a pixel of the carrier image, and `r2` is the red component in the message we want to hide. The following operation is part of hiding the image in the carrier:

```
r1 = (r1 & 0b11111000) ^ (r2 >> 5)
```

This operation first zeros out the bottom 3 bits from the carrier, leaving the top 5 bits alone. Then it shifts the most significant bits from the image we're hiding so that they are in the 3 least significant positions. Finally, we add the two together. In the result, the top 5 bits will be from the carrier and the bottom 3 will be from the hidden image. We do a similar thing for the green and blue components, and we do that for all the pixels in the image. Here is some Python code that will do the whole process.

```python
from tkinter import *
from PIL import Image, ImageTk

def change():
    global image1, photo1, image2, photo2
    pix1 = image1.load()
    for i in range(photo1.width()):
        for j in range(photo1.height()):
            r1,g1,b1 = pix1[i,j]
            r2,g2,b2 = pix2[i,j]
            r1 = (r1 & 0b11111000) ^ (r2 >> 5)
            g1 = (g1 & 0b11111000) ^ (g2 >> 5)
            b1 = (b1 & 0b11111000) ^ (b2 >> 5)
            pix1[i,j] = r1,g1,b1
    photo1=ImageTk.PhotoImage(image1)
    canvas.create_image(0,0,image=photo1,anchor=NW)

def load_files():
    global image1, photo1, image2, photo2
    image1=Image.open('img1.jpg').convert('RGB')
    photo1=ImageTk.PhotoImage(image1)
    image2=Image.open('img2.jpg').convert('RGB')
    photo2=ImageTk.PhotoImage(image2)
    canvas.configure(width=photo1.width(), height=photo1.height())
    canvas.create_image(0,0,image=photo1,anchor=NW)

root = Tk()
button = Button(text='Change', font=('Verdana', 18), command=change)
canvas = Canvas()
canvas.grid(row=0)
button.grid(row=1)
load_files()
mainloop()
```

To run the program, you will first need to install the Python Imaging Library. Use `pip install pillow` at the command prompt to do that. Next, you will need two image files, preferably both the same size. They are called `img1.jpg` and `img2.jpg` in the code above. When you run the program and click the "Change" button, the second image will be hidden in the first. I haven't provided the code to extract that image, but you should be able to work out what it is based on the discussion above about how to hide the image.

You might try experimenting with the number of bits to use for the carrier and the number for the hidden image. The more bits for the carrier, the better the hidden image will be concealed, but the worse its quality will be when revealed. It also helps to make sure the carrier image is busy. A plain image, especially one with a sky or a lot of solid colors, will make it easier for human eyes to spot the hidden image.

One way to hide text in other text

To hide data, it often helps to convert it to a binary format. Here is a helper function that will convert a string of text into a binary string of 0s and 1s:

```python
def to_binary(message):
    m = message.encode()
    h = bytes.hex(m)
    return ''.join('{:04b}'.format(int(x,16)) for x in h)
```

It starts by converting the string into a Python byte string. We then use `bytes.hex` to turn the byte string into a hex string. The last line converts the hex to binary. The purpose of using the call to the `format` method is to make sure leading 0s are preserved. For instance, we want hex number 6 (0110) to stay as 0110, and not become 110, which is what Python's `bin` function would do. Here is a helper function that takes a binary string and turns it back into text.

```python
def from_binary(message):
    s = int(message, 2)
    h = hex(s)[2:]
    return bytes.fromhex(h)
```

Now we can look at a neat trick for hiding a message. The letter *a* has ASCII/Unicode value 97. Farther up in the Unicode table, at character 1072, there is another character that is pixel-for-pixel identical to the *a* at character 97. It's the letter *a* in the Cyrillic alphabet. We can use this fact to hide a message inside some carrier text.

Let's use "aardvarks are animals any day" as our carrier text. Let's try to hide the letter *Z* (ASCII/Unicode value 90). The value 90 translates to 01011010 in binary (and we can use the code above to do that). Let's overlay this on the carrier message, like below:

```
01   0    1   1   0   1    0
aardvarks are animals any day
```

Specifically, we've lined up the binary with the *a*'s of the message. Every *a* that is lined up with a 0 will stay as an ordinary *a*. Every *a* that is lined up with a 1 will get changed to a Cyrillic (character code 1072) *a*. Here is code that will hidea message using this technique:

```python
def hide_message(carrier, message):
    s = ''
    b = to_binary(message)
    i = 0

    for c in carrier:
        if c != 'a':
            s += c
        elif i < len(b):
            if b[i] == '0':
                s += 'a'
            else:
                s += chr(1072)
            i += 1
        else:
            s += 'a'
    return s
```

We have to be a little careful in the likely event that the carrier message has more *a*'s than we have bits in the hidden message. That's what the `i < len(b)` part is for. To show a hidden message, we do things in reverse, like below:

```python
def find_message(carrier):
    s = ''
    for c in carrier:
        if c == 'a':
            s += '0'
        elif c == chr(1072):
            s += '1'
    return from_binary(s)
```

Here is a little code to test this out:

```
message = 'Zebra'
carrier = 'aardvarks are animals any day '*20
h = hide_message(carrier, message)
print(h)
print(find_message(h))
```

Note that there will be some extra `\x00` characters at the end of the message. If you want to, you can write a little code to remove them.

More about steganography

We have looked at a few simple ways of hiding data. There are so many more. One nice approach to hiding data used by the popular software *steghide* is as follows: Start with a passphrase. This is used to seed a PRNG that selects random pixels in a carrier image. The message to be hidden is converted to binary and hidden in the least significant digits of those pixels. The steghide program does more than just this to avoid detection, but that's the basic idea. And there's nothing special about images. Data can be hidden in audio and other types of files in a similar way.

Steganography can be detected by statistical and machine learning techniques. These techniques look at ordinary images and text and build up some statistics about what ordinary images and text looks like. If an image is hiding a significant amount of data, these tools will notice that the patterns of bits in the least significant bit portions of the pixels don't look like they would for an ordinary image. If you are only storing a very small amount of data or if you are clever about how you hide things, then you can get around these detection tools.