Python Steganography

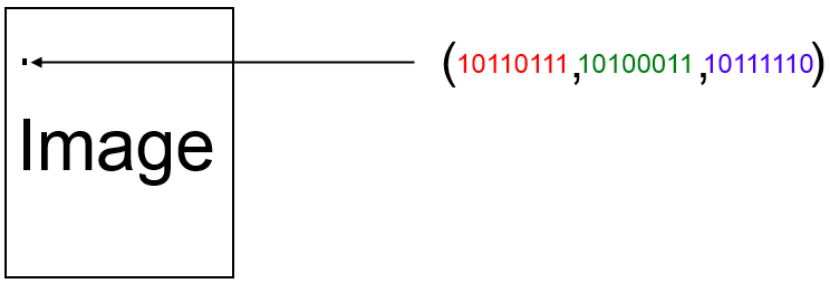
Steganography is the art of hiding data in multiple formats such as texts, images and sound before concealing this information in another piece of data. The name steganography comes from the Greek word covered writing. It has evolved from famous techniques such as invisible ink and hiding a message in every second letter of every word in a large body of text. Nowadays software such as Python can be used to design programs that can specifically hide data in plain sight.

The main advantage that steganography has over cryptography is that detecting a piece of information that has been encrypted is easy. Once an encrypted message is detected, only the encrypting key is needed to reveal this message, this presents a major weakness of this method. Whereas or steganography, unless the hidden message is known, it remains undetectable.

For example, person A wishes to deliver a secret message to person B without person C knowing. If person A uses cryptography and person C intercepts the message, person C knows that a secret message has been sent. If steganography had been used, then person C wouldn't know a message was hidden and it could therefore be delivered safely to person B.

This will be achieved using the least significant bit embedding method. In an image, each pixel is coded by 3 values which when superposed result in the colour of the pixel. These 3 values represent the quantity of each additive primary colour (red, green and blue) these values can represent any colour when combined. Each value is coded on a byte so ranges from 0 to 255.

The content of the secret message will be hidden in the 4 least significant bits of the RGB values of the pixel. As modifying the 4 LSBs represent a maximum change of 15 in each RGB value, the hidden information will not make a visible change to the colour of the pixel.



Hiding an image in another image

Two images of the same format (the loaded images may not be the same type of image, but both will be saved as bitmaps) and of the same dimensions will be selected, one to be hidden and the other one to hide the hidden image. The 4 most significant bits of each pixel from the secret image will replace the 4 least significant bits of each pixel from the carrier image. The main drawback of this method is that part of the secret image will be lost in the process.

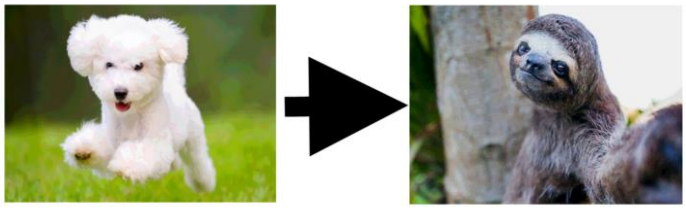
Example:

As an example, I decided to hide Image A into Image B, which gives Image C.

Image A Image B Image C

Finding the image inside the image

The process for finding the image consists of loading the Image created by the hiding program. Every RGB component of each pixel has its 4 MSB removed and replaces them with the 4 LSB of each component. This reveals the image that was hidden in the carrier image.



When comparing this new Image with the one originally hidden, the differences are negligible.

Original Found Image