MODULE 2

STEGANOGRAPHY

BHARATH S
KAVIMANI S
BADHRINATHAN S B
HARIPRASAD N M
AKSHATHAA A S

01 What is steganograpy

Steganography is the practice of concealing a file, message, image, or video within another file, message, image, or video.

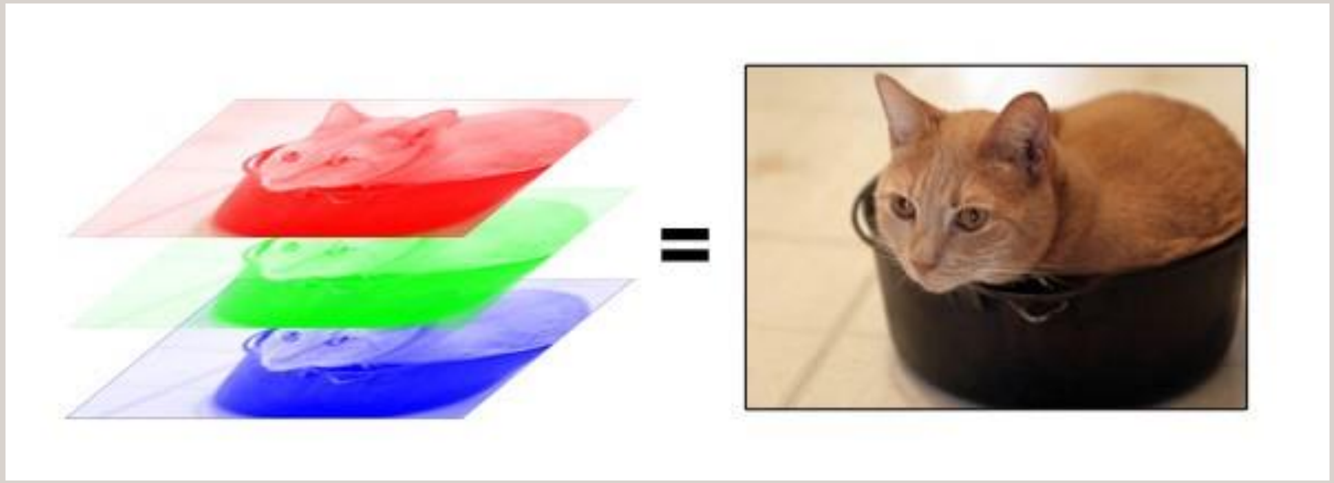
In other words, steganography is more discreet than cryptography when we want to send a secret information. On the other hand, the hidden message is easier to extract.

02

Why steganograpy

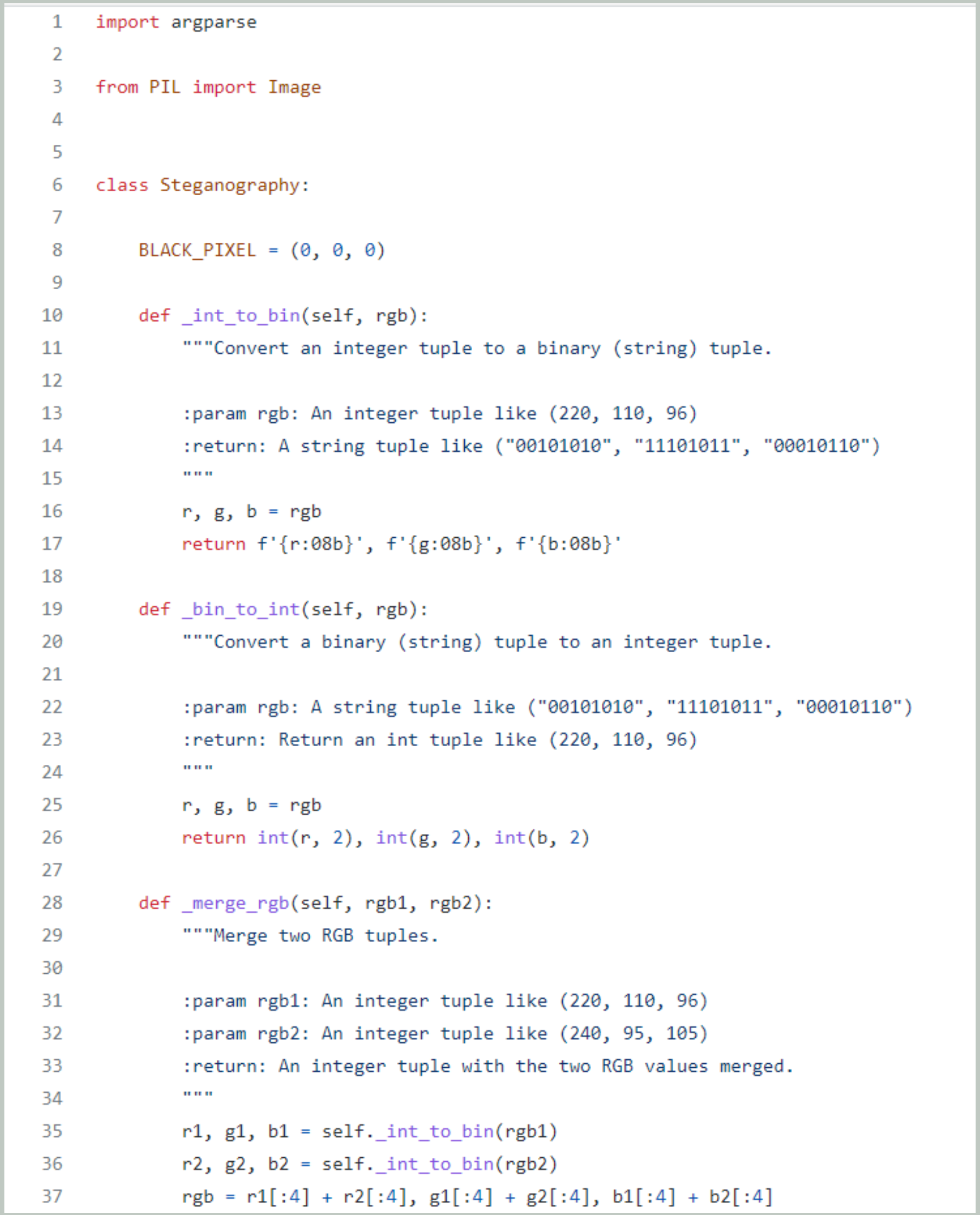
Since we used simple and less complex rubix cube method for the encryption it has a high possibility to be get cracked by attack.

So we need better additional option to make it more complex and using steganography is more easy and effect way which hides the encrypted image using cover image.

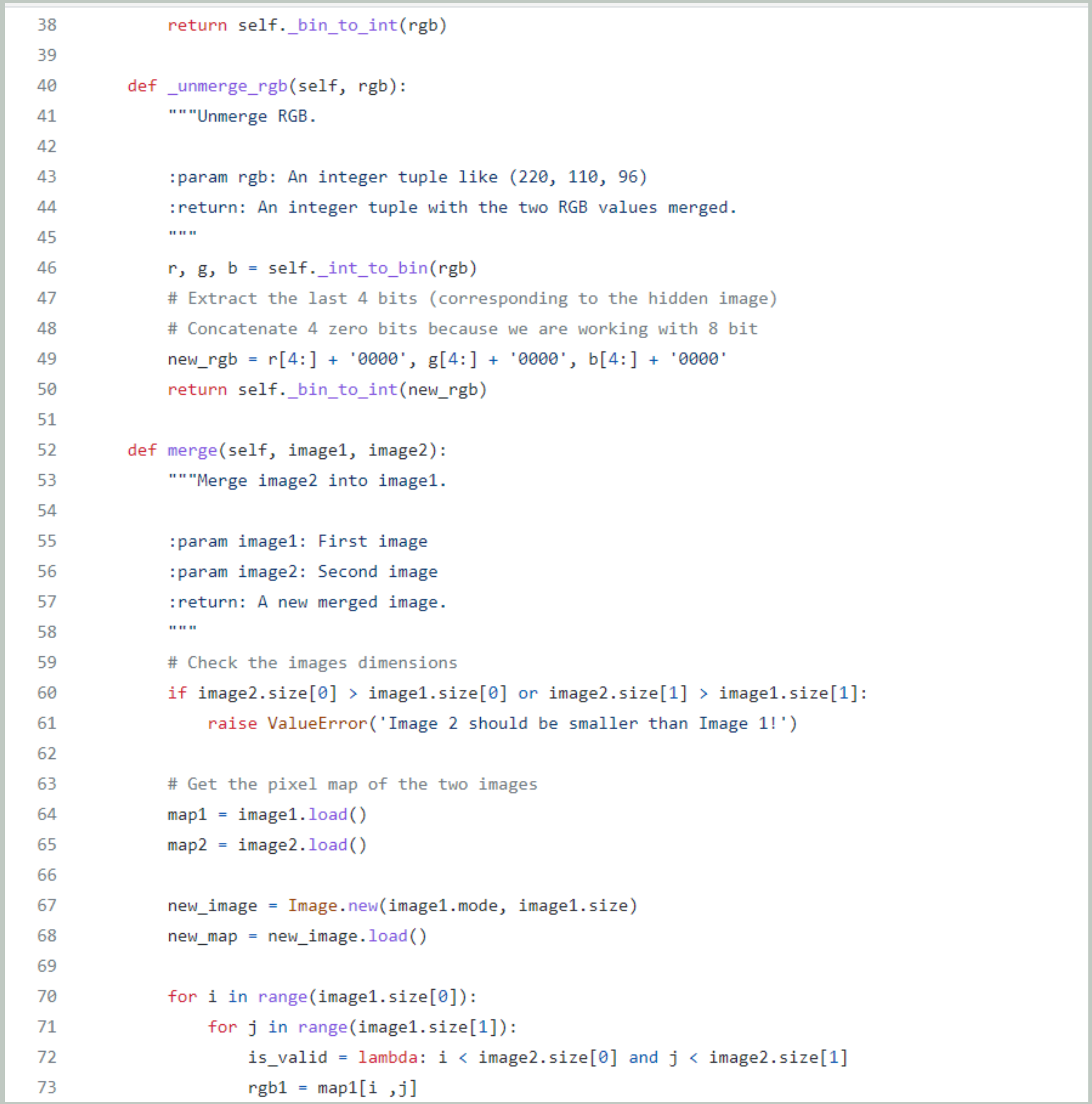


03 Image steganography methodology

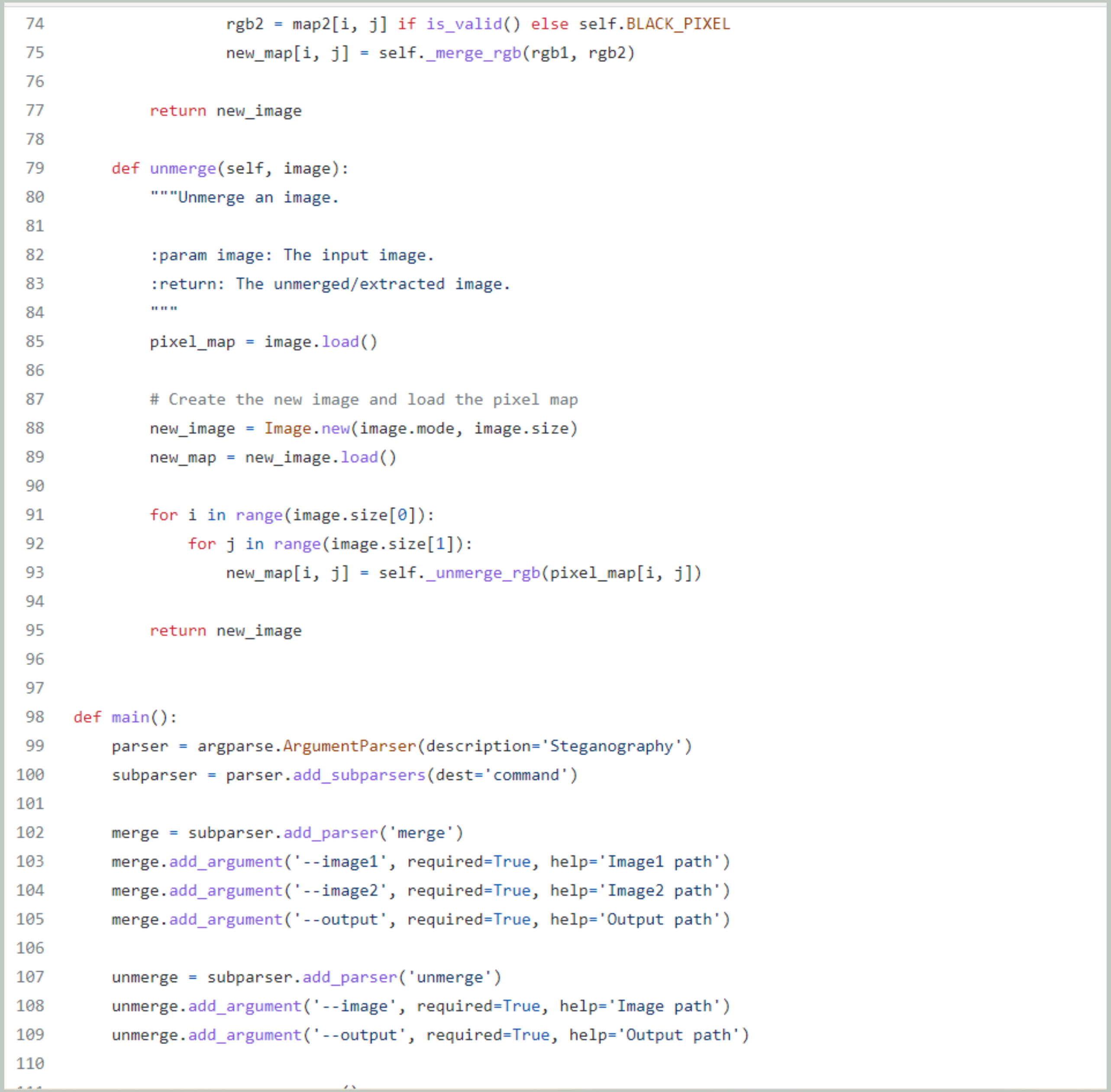
- In simple words steganography covers a image using cover image.
- This is done by interchanging of most significant bits of image to be covered with the least significant bit of cover image
- Pixels of image are stored as bytes where first four bits of each byte is the most significant which mostly contribute to the appearance of a image and last four bits are least significant bit and changing of this least significant bits doesnot affect the image much, so we replace the least significant bits of cover image with most significant bit of encrypted image.
- This method is performed in reverse to unmerge the images



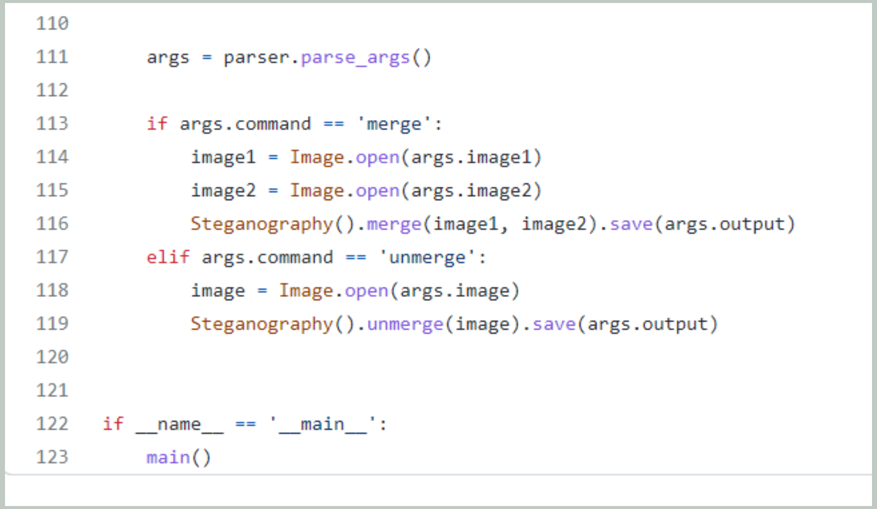
```python
import argparse

from PIL import Image


class Steganography:

    BLACK_PIXEL = (0, 0, 0)

    def _int_to_bin(self, rgb):
        """Convert an integer tuple to a binary (string) tuple.

        :param rgb: An integer tuple like (220, 110, 96)
        :return: A string tuple like ("00101010", "11101011", "00010110")
        """
        r, g, b = rgb
        return f'{r:08b}', f'{g:08b}', f'{b:08b}'

    def _bin_to_int(self, rgb):
        """Convert a binary (string) tuple to an integer tuple.

        :param rgb: A string tuple like ("00101010", "11101011", "00010110")
        :return: Return an int tuple like (220, 110, 96)
        """
        r, g, b = rgb
        return int(r, 2), int(g, 2), int(b, 2)

    def _merge_rgb(self, rgb1, rgb2):
        """Merge two RGB tuples.

        :param rgb1: An integer tuple like (220, 110, 96)
        :param rgb2: An integer tuple like (240, 95, 105)
        :return: An integer tuple with the two RGB values merged.
        """
        r1, g1, b1 = self._int_to_bin(rgb1)
        r2, g2, b2 = self._int_to_bin(rgb2)
        rgb = r1[:4] + r2[:4], g1[:4] + g2[:4], b1[:4] + b2[:4]
```



```python
        return self._bin_to_int(rgb)

    def _unmerge_rgb(self, rgb):
        """Unmerge RGB.

        :param rgb: An integer tuple like (220, 110, 96)
        :return: An integer tuple with the two RGB values merged.
        """
        r, g, b = self._int_to_bin(rgb)
        # Extract the last 4 bits (corresponding to the hidden image)
        # Concatenate 4 zero bits because we are working with 8 bit
        new_rgb = r[4:] + '0000', g[4:] + '0000', b[4:] + '0000'
        return self._bin_to_int(new_rgb)

    def merge(self, image1, image2):
        """Merge image2 into image1.

        :param image1: First image
        :param image2: Second image
        :return: A new merged image.
        """
        # Check the images dimensions
        if image2.size[0] > image1.size[0] or image2.size[1] > image1.size[1]:
            raise ValueError('Image 2 should be smaller than Image 1!')

        # Get the pixel map of the two images
        map1 = image1.load()
        map2 = image2.load()

        new_image = Image.new(image1.mode, image1.size)
        new_map = new_image.load()

        for i in range(image1.size[0]):
            for j in range(image1.size[1]):
                is_valid = lambda: i < image2.size[0] and j < image2.size[1]
                rgb1 = map1[i ,j]
```



```python
                rgb2 = map2[i, j] if is_valid() else self.BLACK_PIXEL
                new_map[i, j] = self._merge_rgb(rgb1, rgb2)

        return new_image

    def unmerge(self, image):
        """Unmerge an image.

        :param image: The input image.
        :return: The unmerged/extracted image.
        """
        pixel_map = image.load()

        # Create the new image and load the pixel map
        new_image = Image.new(image.mode, image.size)
        new_map = new_image.load()

        for i in range(image.size[0]):
            for j in range(image.size[1]):
                new_map[i, j] = self._unmerge_rgb(pixel_map[i, j])

        return new_image


def main():
    parser = argparse.ArgumentParser(description='Steganography')
    subparser = parser.add_subparsers(dest='command')

    merge = subparser.add_parser('merge')
    merge.add_argument('--image1', required=True, help='Image1 path')
    merge.add_argument('--image2', required=True, help='Image2 path')
    merge.add_argument('--output', required=True, help='Output path')

    unmerge = subparser.add_parser('unmerge')
    unmerge.add_argument('--image', required=True, help='Image path')
    unmerge.add_argument('--output', required=True, help='Output path')

```



```python

    args = parser.parse_args()

    if args.command == 'merge':
        image1 = Image.open(args.image1)
        image2 = Image.open(args.image2)
        Steganography().merge(image1, image2).save(args.output)
    elif args.command == 'unmerge':
        image = Image.open(args.image)
        Steganography().unmerge(image).save(args.output)


if __name__ == '__main__':
    main()
```

Thank you