Steganography Using LSB Method

TianxiaoHu 14300240007

October 29, 2016

1 Experiment Background

In computing, the least significant bit(LSB) is the bit position in a binary integer giving the units value, that is, determining whether the number is even or odd. The LSB is sometimes referred to as the right-most bit, due to the convention in positional notation of writing less significant digits further to the right.
Least significant bits are frequently employed in steganography. For example, RGB graphs have 3 color columns: red(R), green(G) and blue(B), which range from 0~255. Vector (0, 0, 0) indicates that a pixel is totally black and (255, 255, 255) represents purely white. Hence, if we only change a pixel's least significant bit so that the graph contains data, the slight difference can't be distinguished by naked eyes.
Python Image Library is a perfect tool for dealing with graphs with python. The following report will describe the encryption and decryption steps in details. A logo of Fudan University in BMP format(150×150) is taken as an example for its lossless compression and the ciphertext is "FudanUniversity".

2 Experiment Enviornment

Processor: Intel(R) Core(TM) i5-4590 CPU@3.30GHz 3.30GHz
RAM: 8.00GB
System: Windows 10
Python interpreter: Anaconda 2.7.12

3 Simulation of LSB Encryption

3.1 Encryption

Import packages first

```
In [1]: from PIL import Image
        import binascii
        import numpy as np
```

Load the image

```
In [2]: img = Image.open(r"E:/Programming/LSB_encrypt/fudan_logo.bmp")
        width, height = img.size
```

Preview the image

```
In [3]: img
```

```
Out[3]:
```

Preprocess the ciphertext: mark the end of string

```
In [4]: infostr = "FudanUniversity"
        infostr = infostr + '\0'
```

Convert ASCII string into binary code

```
In [5]: def asc2bin(ascstr):
            binstr = bin(int(binascii.hexlify(ascstr), 16)).replace('0b', '')
            return '0' * (8 - len(binstr) % 8) + binstr

        binstr = asc2bin(infostr)
```

Get enough pixels to contain the binary data

```
In [6]: def get_RGB_col(img, binstrlen):
            pixnum = binstrlen / 3 + 1
            R, G, B = [], [], []
            for w in range(pixnum / height):
                for h in range(height):
                    pixelr, pixelg, pixelb = img.getpixel((w, h))
                    R.append(pixelr)
                    G.append(pixelg)
                    B.append(pixelb)
            for h in range(pixnum % height):
                pixelr, pixelg, pixelb = img.getpixel((pixnum / height, h))
                R.append(pixelr)
                G.append(pixelg)
                B.append(pixelb)
```

```
        return R, G, B

    Rcol, Gcol, Bcol = get_RGB_col(img, len(binstr))

    encrarr = list()
    for tmp in zip(Rcol, Gcol, Bcol):
        encrarr.extend(list(tmp))
    encrarr = np.array(encrarr)
```

Check the length

```
In [7]: len(binstr)
```

```
Out[7]: 128
```

```
In [8]: len(encrarr)
```

```
Out[8]: 129
```

Turning string into a binary array

```
In [9]: infoarr = []
        for i in range(len(binstr)):
            infoarr.append(int(binstr[i]))
        # infoarr shoule be of same length as encrarr
        infoarr.extend([0] * (len(encrarr) - len(infoarr)))
        infoarr = np.array(infoarr)
```

Change encrarr's least significant bits into binary data

```
In [10]: encrarr = encrarr / 2 * 2 + infoarr
```

Put the encrypted pixels back to graph

```
In [11]: l = len(encrarr)
         encrarr = encrarr.reshape((l/3, 3))
         for i in range(encrarr.shape[0]):
             img.putpixel((i/height, i%height), tuple(encrarr[i]))
```

Save the graph

```
In [12]: img.save(r"E:/Programming/LSB_encrypt/fudan_logo_LSB.bmp")
```

Show the encrypted graph

```
In [13]: img
```

```
Out[13]:
```

It turns out that we can hardly recognize the difference

3.2 Decryption

Load the image

```
In [14]: img = Image.open(r"E:/Programming/LSB_encrypt/fudan_logo_LSB.bmp")
         width, height = img.size
```

Extract binary code until meet the string end mark

```
In [15]: binarr = []
         end = [0] * 8
         for w in range(width):
             for h in range(height):
                 tmp = np.array(img.getpixel((w, h)))
                 binarr.extend(tmp - tmp / 2 * 2)
                 if binarr[(len(binarr) - 8) / 8 * 8: len(binarr) / 8 * 8]==end:
                         break
             else:
                 continue
             break
```

Check the length

```
In [16]: len(binarr)
```

```
Out[16]: 129
```

Drop the string end mark

```
In [17]: binarr = binarr[0: (len(binarr) - 8) / 8 * 8]
```

```
In [18]: len(binarr)
```

```
Out[18]: 120
```

Convert binary array into binary string

```
In [19]: binstr = ''
         for i in range(len(binarr)):
             binstr = binstr + str(binarr[i])
```

Decode binary string to ASCII

```
In [20]: def bin2asc(binstr):
             hexstr = hex(int(binstr, 2)).replace('0x', '').replace('L', '')
             return binascii.unhexlify(hexstr)

         bin2asc(binstr)
```

```
Out[20]: 'FudanUniversity'
```

We have successfully extracted the ciphertext