LSB隐写(最低有效位隐写)

原创

Floating Snow 于 2019-12-22 14:41:19 发布

分类专栏： 图像隐写 文章标签： LSB 图像隐写

本文链接：https://blog.csdn.net/qq1437715969/article/details/103617581

LSB隐写(最低有效位隐写)

我们先思考如下几个问题，然后再去实现

1、图片在计算机中存储的方式

2、什么原因可以是实现隐写

3、为什么选择最低有效位？

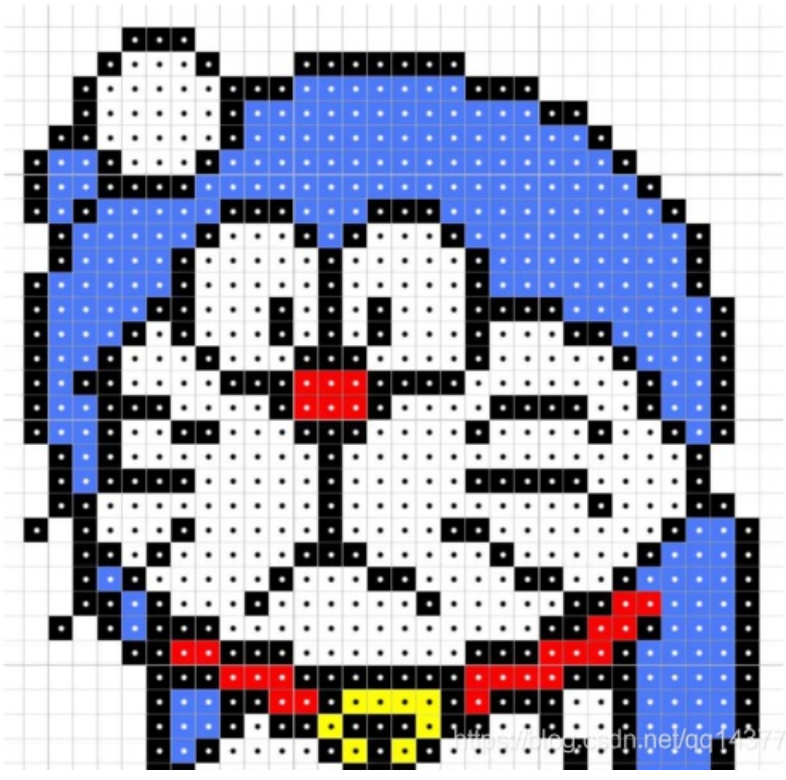
4、具体实现思路

5、如果用代码实现LSB隐写

1、图片在计算机中存储的方式

如果将一幅图像放大，我们可以看到它是由一个个的小格子组成的，每个小格子就是一个色块。如果我们用不同的数字来表示不同的颜色，图像就可以表示为一个由数字组成的矩阵（matrix），这样就可以在计算机中存储。这个小格子就是像素（pixel），矩阵的行数与列数，就是分辨率（resolution）。

我们常说某张图像的分辨率是1280*720，指的就是这张图像是由1280行，720列的像素组成。反过来，如果我们有一个矩阵，将矩阵中的每个数值都转换为颜色，并在计算机中显示出来，就可以复现这张图像



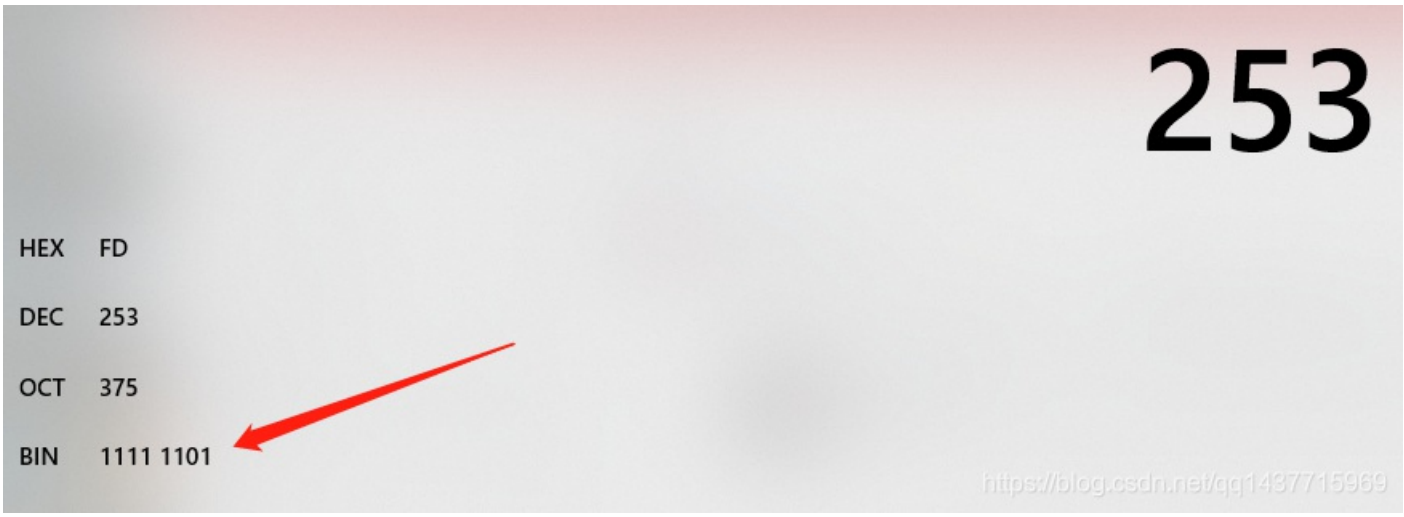
我们可以理解为图像在计算机中是以点阵的形式存在的，我们可以理解为一个二位数组或者矩阵，每个点所在的行和列是它的坐标，元素的值可以理解为当前像素的值。

2、什么原因可以实现隐写

这个原因有很多，但是最重要的还是因为人类的视觉冗余，其实是对相近的像素的敏感度比较低。所以改变部分像素的值不是很明显的话，肉眼基本察觉到不到。

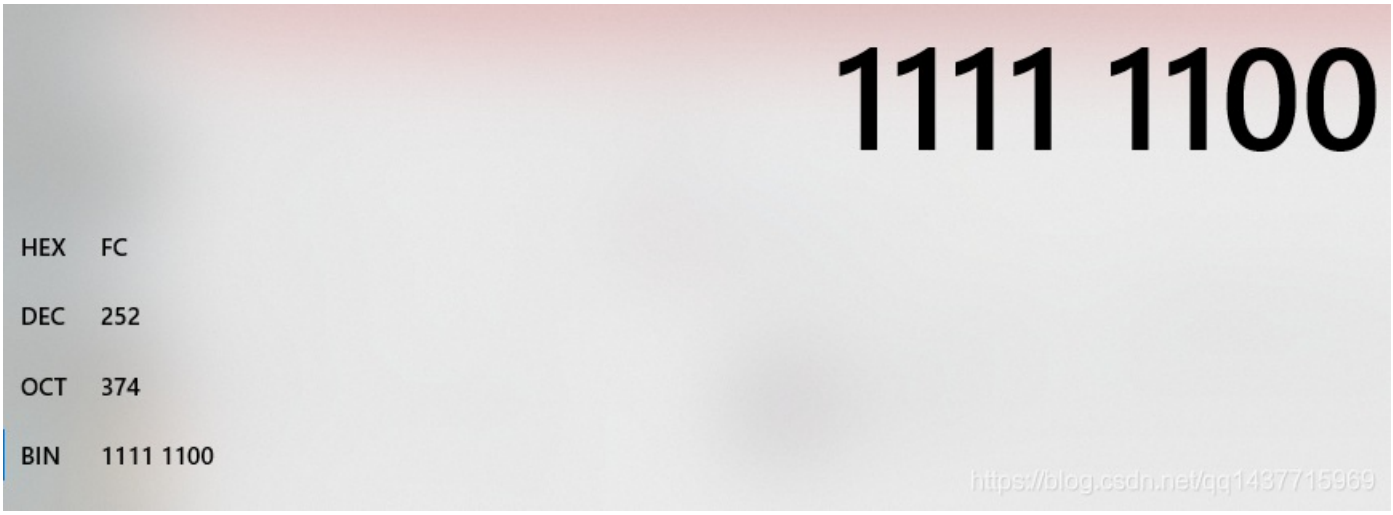
3、为什么选择最低有效位？

如果我们使用RGB方式即0—255无符号数字对某个像素点的值进行描述的时候，我们怎么修改这个数字才能实现我们写入数据的功能而且对原图的修改程度最小？我们都知道计算机是基于二进制的，也就是说我们在计算机中能见到的一切数据都是以二进制的形式存在的。例如：



比如说十进制数据253对应的二进制数据是1111 1101，即使是字符串也是可以转换为二进制类型的数据，我们只需要把要隐藏的数据先转换为二进制数据然后再将其按照某种规则差分然后按位写入图像的部分像素的二进制数据的最后一位即可。

如果图片中某点的像素值是253的话，其对应的二进制数据是1111 1101，如果我们把该数据的最后一位替换为0的话，其对应的十进制数据就是252。



因为人眼的视觉冗余，对图片中某点的像素发生上述改变时几乎是察觉不到的。因为图像隐写的目的是不让别人发现写入了数据，先不说写入的问题，如果说图像本身的变化人眼就能识别的话，何谈“隐”字?这种方式对图像的改变比较小，所以采用该方式。

4、具体实现思路

（1）获取要隐藏的数据，一般这里不管是什么，我们都可以理解为字符串，本文不涉及图像写入图像。

（2）将获取到的字符串二值化，即按照一定规则转换为二进制数据，一般是8位（不涉及中文隐写，中文占2个，因为我实验的目的虽然是实现隐写，我还想提出来呢，并不是写进去不提出来，所以我得有规则），不够的前面补0，一定要测试好对应的解码方法。

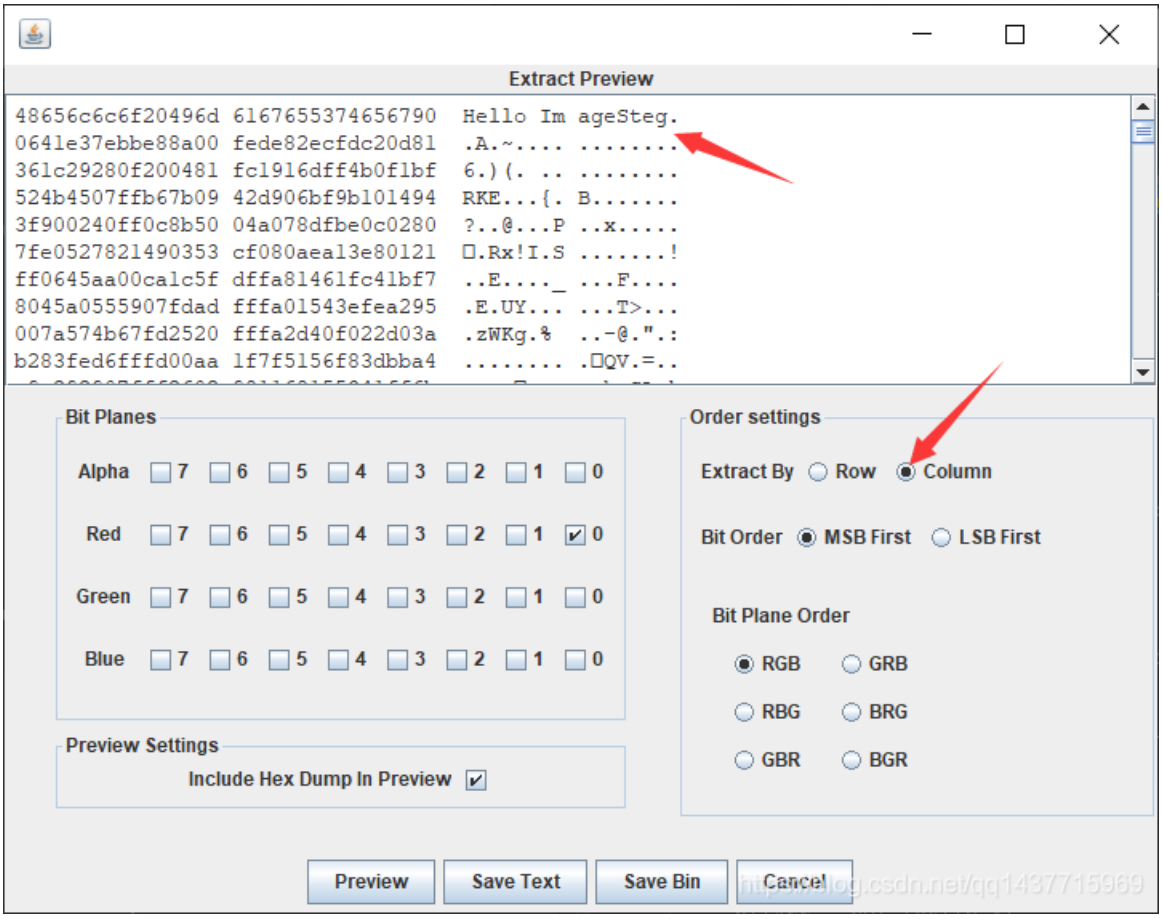
（3）准备好宿主图像，安装好python环境和PIL，我们不使用opencv实现，因为opencv可能存在着压缩，我没有实现，最后使用PIL实现的。

（4）获取图像信息（主要是高度和宽度），这里作为入门，我们不使用彩色图像，以黑白图像为例，根据二值化后的字符串的长度，对宿主图像的像素进行遍历，然后将数据依次写入对应像素的最低有效位，写入完成之后跳出循环，对目标图像进行持久化即可得到载密图像。

5、实现

```python
from PIL import  Image as im
import re

replace_reg = re.compile(r'[1|0]$')

#替换最后一位的数据，source是被替换数据，target是目标数据，就是batarget放到source最后一位
def repLstBit(source,target):
    return replace_reg.sub(target,source)
#运行结果：'123X'
print(repLstBit("111110","1"))

#字符串转换二进制，不够八位的话补齐8位
def encode(s):
    return ''.join(bin(ord(c)).replace('0b','').rjust(8,'0') for c in s)

#切割从图像中收集到的数据，就是把载密图像的对应最后一位提取出来之后需要进行切割
def cut_text(text,lenth):
    textArr = re.findall('.{'+str(lenth)+'}',text)
    tempStr = text[(len(textArr) * lenth):]
    if len(tempStr)!=0:
        textArr.append(text[(len(textArr)*lenth):])
    return textArr

#二进制转换成字符串，看上面切割方法的注释即可理解该方法存在的意义
def decode(s):
    bitArr = cut_text(s,8)
    return "".join(chr(int(i,2)) for i in bitArr)


#读取宿主图像和要写入的信息生成载密图像。
if __name__ == '__main__':
    img = im.open("D:/StegAnograpy/dove.png")
    width = img.size[0]
    height = img.size[1]
    hideInfo = "Hello ImageSteg"
    hideBitArr = encode(hideInfo)
    count = 0
    bitInfoLen = len(hideBitArr)

    print(hideBitArr)
    for i in range(width):
        for j in range(height):
            if count == bitInfoLen:
                break;
            pixel = img.getpixel((i,j));
            print(pixel[0])
            sourceBit = bin(pixel[0])[2:]
            print(sourceBit)
            rspBit = int(repLstBit(sourceBit,hideBitArr[count]),2)
            count += 1
            img.putpixel((i,j),(rspBit,rspBit,rspBit))
    img.save("D:/StegAnograpy/dove1.png")
```

6、StegSolve检测结果：



7、总结

（1）这种方式比较容易实现，但是太容易被检测到。

（2）但是本次实验的收获是直到了从0到1的细节处理。