F3隐写

原创

lzy1286427633 于 2020-05-08 17:01:38 发布

分类专栏： steg 文章标签： python 其他 算法 安全

本文链接：https://blog.csdn.net/lzy1286427633/article/details/106000907

F3隐写

文章目录

1. 引言

数字隐写技术是一种在数字媒体文件中隐藏秘密信息来打到隐秘通信的目的的技术。区别于信息密写使用负载的加密方案和密钥保证信息传输内容的机密性和完整性，隐写术通过隐藏通信行为的方式保障通信的安全性。数字图像作为使用最广泛的数字媒体，基于图像的数字隐写技术发展成熟。
JPEG有损压缩技术是一种广泛应用于网络图像传输和保存的有损图像压缩技术。作为有损压缩的一种，这一技术通过将图像转换至DCT域，在提供可分级的良好压缩率的同时降低了有损压缩对图像质量的影响。成为最流行的有损压缩格式。
F3隐写是一种针对JPEG压缩技术特性开发的数字图像隐写算法，通过将秘密信息隐藏于量化后的DCT系数的中频区域，兼顾隐写的鲁棒性和不可见性。是一种较为优秀的变换域图像隐写技术。
本次实验的目的在于实现F3隐写算法

2. 实验环境

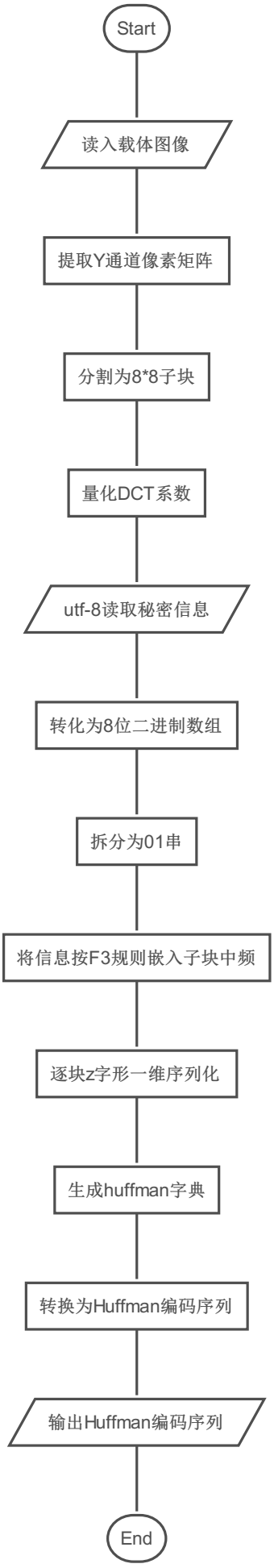
环境	版本
windows	10
python	3.8
pycharm	2019.3.3

3. 实现使用的现有外部库

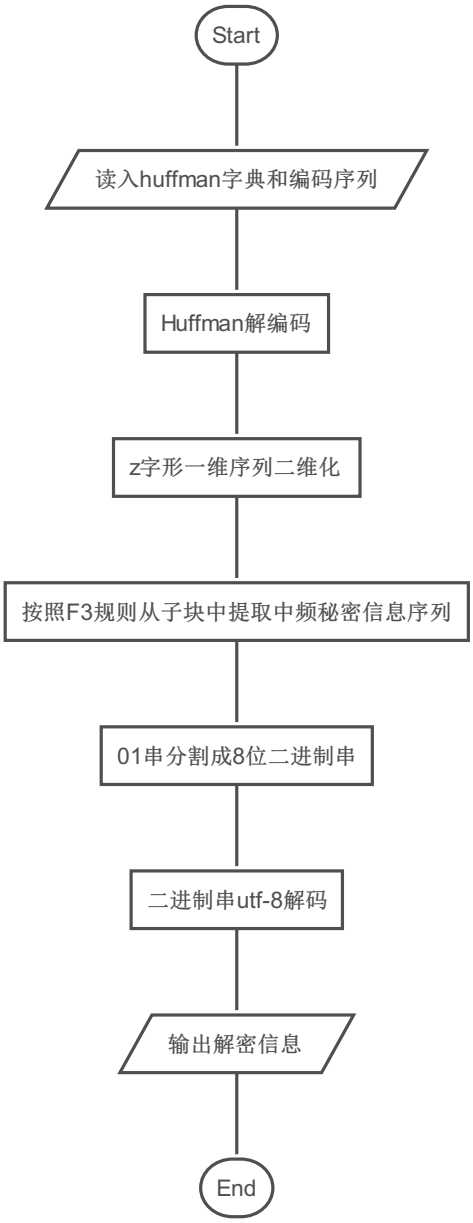
库	版本
numpy	1.18.2
pillow	7.1.2
scipy.fftpack	1.4.1
dahuffman.HuffmanCodec	0.2

4. 实现思路

具体的隐写流程如图[1]所示：


Start
读入载体图像
提取Y通道像素矩阵
分割为8*8子块
量化DCT系数
utf-8读取秘密信息
转化为8位二进制数组
拆分为01串
将信息按F3规则嵌入子块中频
逐块z字形一维序列化
生成huffman字典
转换为Huffman编码序列
输出Huffman编码序列
End

图一： F3隐写嵌入流程



图二：F3隐写提取流程

具体技术细节：

> 预处理

对载体图片的预处理： 使用pillow读入载体图片，转化为YCbCr后，提取Y通道矩阵作为隐写。
对秘密信息的预处理：

$$Y = \qquad + 0.5870G + 0.1140B$$

将秘密信息转化为由01字串表示的二进制序列。具体包括utf-8编码，数字转8位二进制，二进制转01串P。在正式的秘密信息前使用16位二进制表示嵌入的秘密信息大小。最高支持64KB。

> DCT量化

将Y通道矩阵转化为$numpy$对象，调用$numpy.vsplit()$、$numpy.hsplit()$函数均匀切割成8*8大小子块。生成子块矩阵B。B_{ij}表示位于i行j列的子块。逐块调用$ftpack.dctn(block)$进行DCT变换后,量化处理。得到各块的量化DCT系数矩阵。

>秘密信息嵌入

将序列P顺序嵌入DCT量化矩阵中， 具体嵌入规则如下：

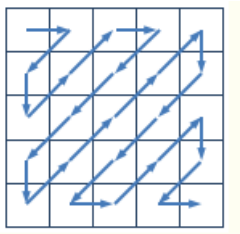
> - 每个非零的DCT系数隐藏1bit秘密信息。
> - 秘密信息为0，当前DCT系数为±1时，将当前DCT系数赋0，跳过不嵌入。读取下一个DCT系数重新嵌入。
> - 当秘密信息和当前DCT系数最低有效位相同时，不做改变；当秘密信息和当前DCT系数最低有效位不同时，将当前DCT系数的绝对值减一，符号保持相同复制给当前DCT系数。

python实现如下：

```
if not block[m][n]:
    continue
if abs(block[m][n]) > 1:
    if block[m][n] % 2 != binary[0]:
        if block[m][n] > 0:
            block[m][n] -= 1
        else:
            block[m][n] += 1
    binary.popleft()
```

> Z字形序列化

将8*8子块按Z字形排布转化为一位序列。将所有子块的一维序列连缀在一起。



图三：Z字形扫描

可以发现，每条斜线上的点的横纵坐标和是相等的。Z字形扫描就是顺着斜线线的方向读取。当横纵坐标和为奇数时，顺序为右上至左下；当横纵坐标和为偶数时，顺序为左下至右上。据此，设计以下实现：

```
def zigzag(self, mat):
    l = [[] for line in range(16)]
    for i in range(8):
        for j in range(8):
            l[i + j].append(mat[i][j])
    zigzag = []
    for i in range(15):
        zigzag += l[i] if i % 2 == 1 else reversed(l[i])
    return zigzag
```

>Huffman编码

调用 `HuffmanCodec.from_data()` 生成一维序列的Huffman编码表。调用 `huffmanDict.encode()` 根据编码表编码一维序列。

>反Z字形序列化

Huffman解编码后反Z字形序列化得到8*8子块矩阵

>提取秘密信息

逐一顺序读取DCT系数，取非零DCT系数的最低比特位连缀成串。
首先取01串前16位转化为二进制数，得到嵌入长度slen。
从01串第17位起每8位作为一个二进制数读入，直到读入slen个数。
将slen个数作为ut-8编码翻译成字符，得到的字符串即为携带的秘密信息。

```
def deSteg(self, steg_blocks):
    bin_seq = []
    for i in range(len(steg_blocks)):
        for j in range(len(steg_blocks[0])):
            block = steg_blocks[i][j]
            width, height = block.shape
            for m in range(height):
                for n in range(width):
                    if block[m][n]:
                        bin_seq.append(str(abs(block[m][n]) % 2))
    l = "".join([bin_seq[i] for i in range(16)])
    l = int(l, base=2)
    print("slen: ", l, type(l))
    bin_seq = [bin_seq[i] for i in range(16, l * 8 + 16)]
    bin_seq = np.split(np.array(bin_seq), l)
    bin_seq = np.array([chr(int("".join(item), 2)) for item in bin_seq])
    binfile = b""
    for i in bin_seq:
        binfile += i.encode("utf-8")
    return binfile
```

5.验证

实验选择的载体图片

向图片中隐写了一个文本文件，通过将提取出的文本文件和原始文件分别生成sha-256摘要。二者的摘要值如下：

```
原始文件 73897da2063f4c352a5c7a72ed52bd861034c5b9ea9674688a768b82d2ac6e7a
提取文件 73897da2063f4c352a5c7a72ed52bd861034c5b9ea9674688a768b82d2ac6e7a
before and after Steg, the secret is same
```

结果表明，二者是同一个文件，秘密信息在隐写前后没有发生丢失。

6. 总结

F3隐写算法是一种巧妙利用JPEG压缩技术在量化DCT系数后数据无损失和DCT域将图像主要特征集中体现在低频区域的特点的JPEG图像隐写算法。该算法在具有较好的鲁棒性和不可见性的同时，解决了Jsteg算法嵌入率低下的问题。
存在的问题：未能生成含有秘密信息的图片，只进行到熵编码的步骤。缺乏现有的手段由量化DCT系数或熵编码直接生成图片。而经过逆DCT化形成YCbCr图像的过程会导致隐写信息的缺失和改变，经过实验无法正确提取秘密信息。

逆DCT变换生成的含密图片，实验表明，秘密信息被损坏无法正确提取