背景

## 基于LSB的图像隐写技术

### 隐写术

隐写是指把一个文件、消息、图像或者视频隐藏到另一个文件、消息、图像或者视频的行为。与密码学不同的是，隐写术旨在隐藏消息或其他形式的信息本身的存在，不引起发送方和接收方以外的人的怀疑而完成信息的交流，而密码学则用于隐藏这些信息的内容，使得非发送方或接收方即使截获消息也无法得到所交流的信息的真实内容。隐写术的主要任务是使发生在公共信道上的秘密信息交流不被察觉，隐藏了秘密信息的图片或其他格式的载体与隐藏之前的原始数据在视觉上以及其他几个重要特征一致。

隐写术是一种秘密通信的艺术，这个术语最早在从几千年前开始，人类就着迷于密文书写，并出于多种原因和动机学习这种和研究这种技术[1]，早期密写通常是指密码学，而随着时代发展现在密写也涵括了隐写术，两者用不同的方式实现密写的目的。无论在哪个时代，隐写术广泛应用于各个领域。早期的隐写实践是使用不可见的墨水在信件中书写消息，显而易见地，军事和政治中，在不被敌方察觉的前提下向友方传递信息的能力十分关键。在数字时代，这种思想发展为在多媒体文件中隐藏其他数字消息。现代隐写术在1985年以后随着个人计算机的推广而问世，随着计算机科学技术和数学的发展以及深入研究而迅速发展并投入更广泛的应用，例如电子通信包含了传输层内的隐写码以传送如文档文件或图片文件等多媒体数据。

与密码学相似，隐写术的使用场景决定了它必须满足以下三个要求：

* 保密性：不容易被探查到隐藏消息的存在
* 可获得性：不会出现由于修改数据的载体导致秘密消息的丢失，秘密消息可以被恢复
* 完整性： 其他人无法伪造出错误信息

隐写系统可以被视为加密系统的一个特例[2]，在这个系统中我们要求密文与明文对于其他人来说难以区分。值得注意的是，隐写加密方必须首先合成一个与秘密消息无关的无害的文件。载体合成具有很大的挑战性，高效合成载体的隐写机制很少。对于这个问题，使用通过修改实现隐写的方法很好地避免了这个挑战。这种系统使用一个现有的与秘密消息无关的文件，也就是载体，作为隐写系统的输入的一部分，接下来秘密消息转换为以让人难以觉察的载体的修改，得到与载体极其相似的结果，即伪装（stego）。理论上，载体合成是最自由且强大的隐写方法，因为它可以不受限制地适用于任意的秘密消息。但在实践中，无害文件的合成是一个非常复杂且低效的过程，所以大多数众所周知的实用系统实用的都是修改载体的方法。当然除了合成和修改两种模式以外，载体选择[3]也是可行的方案，可以类比为传统密码学中使用的编码本，有多个载体或密文分别对应特定的秘密消息。然而，这种模式需要的可用载体数量太大，并不是在所有场景下都实用。所以我们在大多数情况下会选择通过修改进行隐写的模式，也就是说隐写的过程需要使用已经存在的文件作为原始的输入，而多媒体文件（如图像、音频和视频等）往往较大，包含了大规模的数据，可以找到足够的空间隐藏消息同时在不表达出可以被察觉的异常效果，是理想的载体。其中，数字图像的应用场景广泛，修改方便，且容易在互联网快速传播，成为了应用最多的载体。本文也将围绕图像隐写技术展开。

早期的隐写安全完全依赖于隐写算法，只将载体和秘密消息作为输入而不使用秘钥，被称为纯隐写系统，一旦隐写算法泄露则整个系统被破解。将秘钥引入作为输入的隐写系统则被称为秘钥隐写系统。在密码学中，加密者和解密者共享秘钥，关于算法的知识对双方区分伪装和正常消息没有帮助，这个结论即kerckhoffs原则，在隐写系统中并不总是成立[4]。符合kerckhoffs原则的系统在使用一个秘钥的实例被攻破后，使用其他秘钥的实例的安全性不受影响，与其他系统相比这样的系统具有巨大的优势，因为生成不同的秘钥恨容易，而重新设计一个算法却很困难。但事实上，基于kerckhoffs原则安全的隐写系统很难设计，同时大多数应用场景下，除了发送方和接收方外其他人对于他们的隐写系统通常一无所知，因此，不遵循kerckhoffs原则的系统也是可行的。

### LSB隐写算法

对于一个二进制整数来说，最低有效位（LSB）是最低的比特位（即第0位），决定了这个数是奇数还是偶数，这个比特位相比于其他位置的变化对于整个数值变化影响是最小的。

LSB嵌入方法是一种经典的图像隐写算法。这种方法最早被用于像素图像，在像素图像中，每个像素都是代表该点颜色强度的整数。在灰度图像中，每个点的像素值表达了该点色彩介于黑白之间的程度，而在具有三个色彩信道的RGB图像中，每个像素点由三个独立个代表红、绿、蓝三种颜色强度的值合成单元，这些像素值的取值范围通常是8bit的整数值，也就是。在颜色强度上的微小改动被察觉的可能性很小，LSB隐写算法正是利用了像素图像的这个特性，舍弃每个像素原来的最低有效位（LSB），并替换为需要隐藏的消息。接收者得到伪装完成的图像后可以通过模2操作提取新的LSB并将之还原为完整的消息。

最基础的基于LSB的隐写系统从图像的左上角开始逐位嵌入秘密消息，推进的顺序是由发送者和编程语言决定的最自然的方向。为了方便接受者确定隐藏的消息在何处结束，我们可以将消息的大小作为头部隐藏在前像素中，这里的是一个双方已经约定好的整数。通过这个简单的python程序可以演示这个隐写的过程（在python中最自然的方向是逐行嵌入）：

def convert\_message\_to\_bit(string\_text):  
 '''convert the secret message from string to bytearray'''  
 bit\_string=[]  
 for char in string\_text:  
 bit\_string.extend([int(d) for d in bin(ord(char))[2:].zfill(8)])  
 return np.array(bit\_string)  
  
def hide\_side\_information(carrier,length,pro\_text\_size):  
 '''hide some side information(i.e. the length of secret message here) in the front of the image'''  
 bit=bin(length)[2:].zfill(pro\_text\_size)  
 length\_data=np.array([int(d) for d in bit ])  
 side\_info=carrier[:pro\_text\_size]+length\_data  
 carrier[:pro\_text\_size]=side\_info  
 return carrier  
  
def binary\_array\_to\_int(arr):  
 '''convert a 0-1 array into a decimal integer'''  
 bit\_string=''.join(arr.astype(np.str))  
 return int(bit\_string,2)  
  
def hide(cover\_pic,secret\_text,pro\_text\_size):  
 '''hide secret information in a given picture(cover), return a image object'''  
 length=len(secret\_text)  
 cover=np.array(cover\_pic)  
 size=cover.shape  
 cover\_data=cover.flatten()  
 carrier\_data=cover\_data & 0b11111110 #truncated cover without LSB plane  
 l=hide\_side\_information(carrier\_data,length,pro\_text\_size)  
 secret\_bytes=convert\_message\_to\_bit(secret\_text)  
 secretbytes\_length=length\*8  
 carrier\_data[pro\_text\_size:pro\_text\_size+secretbytes\_length]=carrier\_data[pro\_text\_size:pro\_text\_size+secretbytes\_length]+secret\_bytes  
 new\_data=carrier\_data.reshape(size)  
 new\_iamge=Image.fromarray(new\_data)  
 return new\_iamge  
  
def reveal(secret\_iamge,pro\_text\_size):  
 '''extract secret message from a given picture(stego) return a secret string'''  
 im=np.array(secret\_iamge).flatten()  
 lsb\_plane=im & 0b00000001  
 length=binary\_array\_to\_int(lsb\_plane[0:pro\_text\_size])  
 secret\_bytes=lsb\_plane[pro\_text\_size:pro\_text\_size+length\*8].reshape((length,8))  
 secret\_bit=[binary\_array\_to\_int(x) for x in secret\_bytes]  
 return ''.join(chr(c) for c in secret\_bit)

这种方法使得隐藏的消息总是在同样的位置。显然，作为最经典的图像隐写算法，LSB嵌入模式许多的变体，针对这个问题，如果随机选择用于嵌入消息的像素点的位置则可以让伪装变得更加隐蔽。在伪装系统中引入伪随机数生成器（PRNG），依据其产生的序列选择嵌入消息的像素位置，可以使伪装消息的分布更加随机。这个伪随机数生成器的种子作为系统的秘钥必须在之前就由双方完成交换。

除此之外，还有一种经典的基于LSB嵌入的改进，被称作LSB匹配，体现在随机化了每个样本的篡改。不同于经典LSB嵌入直接舍弃最低有效位的操作，LSB匹配将观察该像素的最低有效位与需要嵌入的信息单元的关系，如果一致则保持最低有效位不变，否则将等概论地对该像素值，具体来说对于偶数值像素而对于奇数值像素。消息的提取同样可以使用模2运算直接完成。

基于LSB的多种隐写方法被广泛应用在空域（如像素图像）并推广到了其他整数信号发生幅度的变化难以被觉察到的场景。对于LSB隐写算法进行在多个方面进行一些微小的改进，就可以获得在隐藏效果上得到巨大的提升，因此现有的工作往往针对于LSB具体实现参数的调整而非设计LSB以外的隐写算法。

## 隐写分析

隐写分析是指对于使用隐写术隐藏的秘密消息的探查。隐写分析的主要任务是识别给定的文件中是否隐藏了可疑的秘密消息，最基础的隐写分析算法以可疑文件作为输入，而将二元分类的“是”和“否”作为输出结果。在此基础之上，有些隐写分析也加入了恢复隐藏信息的功能，但是作为隐写分析最根本的任务是探查是否存在隐藏的消息，一旦探查到隐藏消息的存在，甚至不需要恢复出具体的消息，就可以认为该隐写系统被攻破。

针对基于LSB的图像隐写分析的方法主要有测试[5]，样本对分析[6]和RS隐写分析[7]。

[1] SCHAATHUN H G. Machine learning in image steganalysis[M]. Chichester West Sussex UK: Wiley-IEEE Press, 2012.

[2] BACKES M, CACHIN C. Public-key steganography with active attacks[G]//Theory of cryptography. Springer, 2005: 210–226.

[3] FILLER T, FRIDRICH J. Complete characterization of perfectly secure stego-systems with mutually independent embedding operation[C]//Acoustics, speech and signal processing, 2009. iCASSP 2009. iEEE international conference on. IEEE, 2009: 1429–1432.

[4] FRIDRICH J, GOLJAN M, SOUKAL D. Searching for the stego-key[C]//Electronic imaging 2004. International Society for Optics; Photonics, 2004: 70–82.

[5] WESTFELD A, PFITZMANN A. Attacks on steganographic systems[C]//Information hiding. Springer, 1999: 61–76.

[6] DUMITRESCU S, WU X, MEMON N. On steganalysis of random lSB embedding in continuous-tone images[C]//Image processing. 2002. proceedings. 2002 international conference on. IEEE, 2002, 3: 641–644.

[7] FRIDRICH J, GOLJAN M, DU R. Reliable detection of lSB steganography in color and grayscale images[C]//Proceedings of the 2001 workshop on multimedia and security: New challenges. ACM, 2001: 27–30.