浙江大学

课程名称：信息安全原理

上课时间：春学期周一 7,8; 周四 1,2

小组编号：第 03 组　　**指导教师:** 蔡亮

信安原理 - 期末报告

组员：贺婷婷 唐子越 黄启涵 应承峻
张佳瑶 张雪楠 张睿嘉 童明宽 柴子炜 郭书廷
组号：03　报告：郭书廷

April 16^{th}, 2019

目录

Chapter 1

简介

隐写术，就是将信息用隐蔽的方式载入在需要传递的载体上，从而达到将信息隐藏的目的。今天的隐写术已被广泛应用于版权保护等领域。

1.1 隐写术的起源

隐写术起源于约翰尼斯·特里特米乌斯（Johannes Trithemius，1462 1516）写于 1499 年并在 1606 年出版于法兰克福的著作 Steganographia。在当时，该书被列于禁书目录之中。这本书在出版时，其中的内容被作者通过隐写术伪装成了一部关于使用鬼魂进行长距离通信的黑魔法著作，直到密钥被公布于世后，人们才知道，他们长期以来认为的魔法书中记载的竟然是一些密码学内容的隐文，这种加密方式也因此得名“隐写术”。

到 1983 年，Simmons 提出了“囚犯问题”模型，其中描述了隐写术的原理和实现过程：囚犯 Alice 和 Bob 在不同的牢房中想要谋划越狱，由于条件的限制，他们之间的任何通信都必须在看守 Wendy 的监督下，因此，为了不引起 Wendy 的怀疑，他们之间的通信不能采用加密的方式，而必须将隐秘信息隐藏于公开的无关紧要的信息之中 [1]。

1.2 古隐写术与现代隐写术

1.2.1 古隐写术

最早的隐写术通常将有用的信息与无用的伪装文本通过文字大小、字体间距、字体样式或是其他特性等方式的变化紧密结合在一起，形成所谓的隐秘文本。传统的隐秘文本加密方式较为简单，也比较容易被推测和识破，例如《圣经密码》中曾经记载着一种以“跳跃字母”的方式进行隐写的例子：

rips expained that each code is a case of adding every fourth or twelth or fiftieth to form a word

在这个序列中，当跳跃间隔为 3 时，可以得到如下的内容：

Rips ExplAineD thaT eacH codE is a Case Of adDing Every fourth or twelth or fiftieth to form a word

即 READ THE CODE。

1.2.2 现代隐写术

现代的隐写术以大数据为载体，将信息隐藏在包含大量数据的图片、音轨、视频剪辑或者文本文件中。

随着科技的发展，现代隐写术往往以大数据为载体，将信息隐藏在包含大量数据的图片、音轨、视频剪辑或者文本文件中。常见的类型有：二值图像隐写、最低比特位（LSB）替换隐写、MLSB替换隐写、随机调制隐写等等 [2]。

1.3 隐写术的发展现状

信息隐藏一直是国内外信息安全、密码学、通信等领域研究的热门话题，1992 年国际上正式提出了信息隐形性研究，1996 年 5 月国际第一届信息隐藏研讨会在英国剑桥牛顿研究所召开，至今已经举办了 9 届。在我国，信息隐藏方面的研究起步稍晚，信息隐藏研讨会自 1999 年 12 月第一次在北京举办，此后几乎每年都会召开一次。而在 2000 年后，信息隐藏在国际中正式掀起了理论研究的热潮：例如 Moulin 等人提出的基于信息论的信息隐藏理论框架、Costa dirty paper 模型、Cohen 与 Lapidoth 模型、Somekh-Baruck 模型以及并行高斯信道等模型。此后，每年都有大量的论著在国内外发表，并且产生了大量的研究成果 [3]。

Chapter 2

常用隐写技术

2.1 图像隐写 [4–8]

2.1.1 简介

隐写术是一种用于隐秘通讯的方法与技术, 它通过把秘密信息嵌入在数字图像等多媒体载体中而尽可能地不改变载体的视觉和统计特性, 以达到掩盖“正在进行秘密通讯”的目的。隐写术根据载体种类的不同可以分成图像隐写、文本隐写、音频隐写、视频隐写等。其中图像隐写占主流。一方面是因为图像是目前数字媒体中使用最为广泛的信息载体；另一方面，由于图像数据中存在较多的冗余信息，相对更适合隐藏秘密信息。针对数字图像的隐写术研究也可以有效推广到音频、视频等其他数字媒体中。

图像隐写主要利用图像中的每个字节中最不重要的比特来代替消息比特。人眼无法分辨隐藏了文件的图片和原来图片的区别，但秘密消息能够在接收端剥离出来，实现信息隐藏。研究领域覆盖数字图像处理、信息论、信号处理、统计分析、模式识别等众多领域。

2.1.2 常见算法

简单 LSB 替换 [9]

数字图像的最低有效位叫做 LSB。LSB 包含最少的图像信息，具有良好的随机性，因此 0 和 1 出现的概率相同，基本类似于随机噪声。

LSB 替换的基本原理为：如果载体图像的最低比特位与要嵌入的信息比特是相同的，则不做改变；当秘密信息比特和被嵌入在载体图像的 LSB 替换不一致时，将 LSB 替换为秘密消息比特。另一种替换方法是：如果秘密消息比特与载体图像的 LSB 不一致，若载密图像像素值为偶数则加 1，为奇数则减 1。

隐写之前，首先把秘密消息转化成可嵌入的比特流形式，在嵌入前对其进行预处理，如用密钥加密等方式。

由于图像的最低有效位的改变对图像的影响较小，所以肉眼基本看不出载密图像和载体图像

的区别, 但会在最低位面产生某些异常特性而容易被识别出来。

LSB 匹配隐写

LSB 匹配算法是对简单 LSB 替换算法一些弱点的改进。

它的基本思想是，当待嵌入的秘密消息比特与像素值的 LSB 不同时，对像素值进行随机的加 1 或减 1。将 1 改成 K 即为 ±K 隐写。当 K 为奇数时，令载体的 LSB 等于秘密消息比特；当 K 为偶数时，将秘密消息比特嵌入到载体次低比特位。

基本过程是，用 C 表示嵌入过程中的载体图像。S 表示嵌入秘密消息后的载体图像。M 表示待嵌入的秘密信息，是一个长度为 $L(M)$ 的、由 0 和 1 组成的序列比特。K 表示隐写密钥。

首先在 C 中选择 L 个载体图像像素。选定的一组像素的每个像素点的灰度值，如果 LSB 和嵌入的信息比特相同，则不会改变；否则，用隐写密钥产生一个伪随机小数，如果介于 0 和 0.5 之间则像素值加 1，否则减 1. 如果上述操作使得像素值超过范围 [0，255]，则将其截断至该范围内与载密图像最为接近的值，且截断后的 LSB 与要嵌入的秘密信息比特一致。

自适应隐写术

自适应隐写方法设计失真代价来衡量每个载体元素的适宜修改程度，在嵌入信息的同时降低总体失真。图像自适应隐写考虑载体图像的自身属性, 根据图像纹理复杂区域难于建模的特点, 有选择地将秘密信息嵌入到载体纹理复杂或者边缘丰富的区域, 提高了载密图像的抗隐写分析检测能力。

首先设计修改失真代价 (每个载体元素因修改所引起的损失) , 然后结合 STC 码 (Syndrome Trellis Codes）来完成具体的嵌入过程。

基于深度学习的隐写术 [10]

生成对抗网络 主要思想：训练生成器和判别器网络，生成器尝试生成图像的近似分布并且尽可能的将其合成真实图像，判别器则努力将真实图像与虚假图像区分开。生成器和判别器之间的对抗通过多轮迭代反馈来达到相互优化。目前基于生成对抗网络的隐写术主要集中于图像空域。

对抗样本 隐写术修改图像的目的除了传递信息还要保证修改操作不被分类器 (隐写分析) 检测到, 而对抗样本则要保证修改后的图像被错误分类。主要思想：通过主动干扰隐写分析采用的机器学习模型来提高隐写安全性。

2.2 文本隐写

2.2.1 简介

文本信息隐藏是以文本为隐藏载体的信息隐藏，它通过利用文本在格式、编码、结构、语法和语义等方面的冗余，把隐秘信息隐藏到文本之中。在互联网中，文本信息起了非常重要的作用，其

数据量也非常之大，加之文本处理的直观性更强的优点，使用文本进行信息隐藏是一个十分有效的方法。

目前，对语言隐写术的研究主要集中在如何设计合适的隐写算法，而对语言隐写术分析的研究较少。语言隐写术根据隐写时借助的自然语言的特征大体可以分为两大类，即基于语法的语言隐写术和基于语义的语言隐写术。基于语法的语言隐写术可以理解为只要求其载密文本遵循语法规则的语言隐写术，而基于语义的语言隐写术要求其载密文本既要满足语法的合法性，又要满足语义的完整性。

2.2.2 常见算法 [11–14]

行移编码

该技术是通过垂直移动文本行的位置来实现的, 通常当一行被上移或一下移时, 与其相邻的两行或其中的一行保持不动, 不动的相邻行被看作是解码过程中的参考位置它既可以用于文本文档也可以用于文件格式。根据要嵌入文件中信息的内容, 编码器将文本行上移或下移。解码器测量接收文档中相邻行的行间距来进行信息的提取。

字移编码

该技术是通过水平调整文本字符的距离来插入信息的。采用这种方式时, 相邻字之间的距离是各不相同的。在实际应用中，格式化的文档经常使用变化的单词间距来使文本更加协调，读者也更能接受字符相邻距离不同的文本，字移编码也就拥有比较好的隐藏性。

特征编码

在特征编码方法中, 观察文本文档并选择一些特征量, 再根据要嵌入的数据来修改这些特征。

利用颜色信息隐藏信息

常见的手段是运用两种相似的颜色来隐藏信息，这两种颜色在视觉上人类一般难以区别，一段信息的 01 字符串可以这样加密在文本中：若加密字符是 1，则对应的文本字符颜色较深，若加密字符是 0，则对应的文本字符颜色较浅。

不可见编码

不可见编码方法常用于非格式化的文本, 一般是在行末添加空格, 或不可见编码来加载水印, 如空格代表, 代表等方式。另外, 在标点符号和文字之间是否有空格不容易引起人注意的特点, 利用空格的排列方案进行信息的嵌入。

在实际应用中，通常是将这以上方法组合起来实现信息的隐藏。但利用文本的字间距、行间距这些信息来隐藏信息的方法稳定性比较差，当原文本重新经过文字编辑，或者重新读取并存盘后，信息就会消失。

基于语法的文本信息隐藏

还有的方法是将一些主、谓、宾按照一定的次序组合出有正常含义的句子，但这种方法的适用面也比较狭窄，特别是因为组合出的句子具有正常含义，其原文本的上下文容易出现不连贯的现象，会被怀疑。

基于语义的文本信息隐藏

在语言中，通常都存在着同音词，同义词，在人们的日常交流中，同音词，同义词的替代通常不会影响语义，但通过使用特定的同音词来同义词来表达同一个意思可以在文本中加入隐藏信息。例如, 互联网用表示, 因特网用表示。这种方法具有比较好的隐蔽性。

未来文本隐写的主要研究方向很可能是基于语义的文本信息隐藏。

2.3 音频隐写

2.3.1 简介

目前，与基于图像的隐写与隐写分析研究相比，以音频为载体的研究要少得多，主要是因为人的听觉系统比视觉系统更灵敏，将秘密信息嵌入音频媒体的难度将大大高于将其嵌入图像媒体的难度。但近年来，有越来越多的研究人员开始关注基于音频的隐写与隐写分析研究。由于信息隐藏的首要目的是让人意识不到信息的存在性，音频载体无疑是一个更为隐蔽、安全的选择。

音频隐写主要利用了掩蔽效应。掩蔽效应被定义为由于另一个声音（掩蔽音）的出现而导致一个声音的闻阈被提高的数量，通常用分贝作为单位，在音频压缩编码、语音增强等方面有很高的应用价值。掩蔽效应可根据掩蔽音和被掩蔽音是否同时存在被分为频域掩蔽和时域掩蔽，一般来说，音频载体的隐写算法主要利用了频域掩蔽效应，如 LSB 算法，但是时域掩蔽效应也有如回声隐蔽算法等的相关应用。绝对闻阈指的是对某一频率的纯音，在没有环境噪声的条件下由统计得出的可听声强度，声音信号中声强度低于绝对阈值的频率分量将不会被人耳感觉到，但在有多个声音信号的实际环境下，较弱的声音即使强度高于绝对闻阈也可能听不到，即闻阈会提高，由此产生了掩蔽效应 [15]。

2.3.2 常见算法

对以音频为载体的隐写算法按照不同的嵌入域主要分为压缩域和非压缩域算法，而非压缩域的方法又可分为时间域方法和变换域方法。

压缩域算法

音频在压缩时会利用听觉系统的一些特性减少信息的冗余度，导致能够嵌入的数据量极少，在压缩格式的音频中保证较高鲁棒性的隐藏信息方法是一个难度较高的研究方向。压缩域算法根据载体是否压缩和嵌入算法前是否解压缩文件可被分为三种算法，包括非压缩域嵌入数据后压缩、压缩域直接嵌入数据、压缩域解压缩后嵌入数据再压缩，其区别主要在于鲁棒性好坏和计算复杂度。

MP3Stego 是一款非压缩域嵌入数据后压缩的音频隐写工具，直接在音频原始数据被压缩的过程中嵌入秘密数据，原理如下。

首先介绍 MP3 格式文件压缩算法。MP3 音频编码主要由两个嵌套的循环组成, 内循环是量化编码循环, Huffman 码表对较小的量化值分配较短的码长, 如果量化编码操作的结果产生的比特数超过了可用的最大比特数限制, 增大量化步长, 直到 Huffman 编码的结果比特数足够小。外循环是噪声控制循环, 根据屏蔽阈值对量化噪声进行控制。每个子带都有一个比例因子, 如果发现在一个给定的子带中的量化噪声超过了屏蔽阈值, 那么该子带的比例因子就将被调整, 以减少量化噪声。

数字音频的频域信号在量化和编码时, 存在量化误差，MP3Stego 通过调节量化误差的大小, 将量化和编码后的长度作为信息隐藏的方法, 即长度为奇数代表信息 1 , 长度为偶数代表信息 0 , 从而将信息隐藏到最后的 MP3 比特流中。[16]

非压缩域算法

时间域算法在时间域上将秘密信息直接嵌入到数字音频信号中，计算复杂度低，易于实现，但稳定性较差，对一般信号处理如音频压缩和滤波等的抵抗能力不足。而变换域算法通过改变音频信号的频域系数来隐藏信息，虽然在一定程度上提高了算法复杂性，但其鲁棒性得到了提高。目前，国内外的趋势都是倾向于变换域算法的研究开发。

时间域算法

下面简单介绍三种常见的时间域基本算法。

最低位算法把采样点最低一位或几位的比特用秘密信息的数据比特来代替，提取时将相应最低位提取出来即可恢复嵌入的秘密信息数据流，隐写容量很大，但安全性较低，目前已有多种针对常规最低位嵌入算法的统计检测方法。

回声算法利用人类听觉系统的滞后掩蔽效应，将秘密数据作为载体数据的环境条件嵌入到音频载体中，对一些有损压缩的算法具有一定的鲁棒性。其算法可表示为

$$y(n) = s(n) + \alpha * s(n - \$elta)$$

$\alpha\delta$

其中，$y(n)$ 为加入回声后的音频信号，$s(n)$ 为原音频信号，α 为回声幅度系数，δ 为回声的延迟参数，表示回声信号滞后于原始信号的时间间隔。

扩展频谱算法的基本思想是将窄带数据扩展到整个载体信号的频率谱上，可分为跳频扩频系统和直接序列扩频系统。以跳频扩频系统为例进行说明，首先发送者和接受者需进行同步，随后载体频率谱被分成几个子带，秘密信息以伪随机方式嵌入到不同频带中。

变换域算法

变换域算法主要有相位法、小波域嵌入法等，下面做出简单介绍。

相位法 相位法是基于人耳对于音频信号的绝对相位不敏感，而对相对相位非常敏感的特点设计的，算法用不同的参考相位代表某个比特，再用参考相位替换原始音频的绝对相位，并对其音频段进行调整，以保持各段之间的相对相位保持不变，从而达到嵌入、隐藏秘密数据的目的。[18]

小波嵌入法 小波域嵌入法由芬兰 OULU 大学的 Cvejic 等人提出，主要利用了小波分析方法。他们对 16 位经度的音频信号作 5 层小波包分解，分成 32 个子带，然后在在小波系数上用 LSB 法嵌入数据，其效果与时间域上的 LSB 算法相比有了极大的提高。[19]

2.4 总结

信息隐藏方法的评价标准主要由三部分组成：嵌入信息量、透明度、鲁棒性，一个好的算法需要尽可能少地修改宿主信号样本，实现足够大的嵌入信息量和高度的鲁棒性，但是这三者往往是不可兼得的，在实际应用时需要进行一定的平衡或倾斜。

如在设计数字水印算法时，关键在于如何让载体在受到攻击后不失去其隐藏着的信息，即需要保证高度鲁棒性，此时对信息嵌入量、透明度等的要求就较低；而在使用隐写传递信息时，关键在于无法让敌人观察到此时正在发生的信息交流过程，即高透明度，相对来说对于鲁棒性等的要求就会降低。实际应用时，使用者需要根据应用场景的不同决定隐写算法所需要的特性，进行合理的选择。

表 2.4.1仅以音频隐写算法的比较进行分析以展示基本隐写方法的特点对比，以供参考。

算法	透明度	鲁棒性	隐写容量	计算复杂性
最低位 LSB 算法	依据嵌入位数而定	差	大	低
相位法	好	好	中等	中等
回声法	依据回声间隔和衰减系数而定	较好	小	高
扩展频谱法	依据嵌入强度而定	好	小	中等
时间和频率掩蔽算法	好	差	极大	高
压缩域算法	一般	一般	小	依据嵌入操作域（非压缩域则复杂，压缩域则简单）

表 2.4.1: 各种音频隐写算法的比较

Chapter 3

水印 [20–23]

3.1 简介

传统水印，是指在造纸过程中形成的，“夹”在纸中而不是在纸的表面，迎光透视时可清晰看到明暗纹理的图形、人像或文字。纸张在生产过程中用改变纸浆纤维密度，可造成传统水印。通常人民币、购物卷、粮票、证券等都采用此方式, 以防止造假。

数字水印，是在用户提供的原始数据中，利用宿主数据的冗余性，永久嵌入的，具有某些具有确定性、可鉴别性的和保密性的，且不影响宿主数据可用性的数字信号或模式。宿主数据可以是视频、音频、图像、文本、三维数字产品等。水印通常由用户提供，如表示版权信息的特殊标志、logo、用户提供的具有某些意义的序列号、文字或者是产品的其它相关信息等。除某些特殊要求外，水印信息一般是不可见的。其不可见性或透明性有相应的标准来评判。

数字水印技术发展至今，已经逐渐由传统的理论研究阶段发展到实际应用阶段，且为增加其安全性，常与密码学相结合。

3.2 来源与背景

数字水印技术的出现和发展与信息技术的飞速发展是息息相关的。上个世纪九十年代万维网的出现和繁荣使计算机网络技术用户大众化。万维网之前的网络，主要用于科学研究者之间进行信息共享，远程计算以及政府，军事部门之间的通信，网络数据主要为数据和文本，量少、无冗余，要求精确传输，可通过加密技术实现信息的安全。万维网出现之后，网络数据可以是数据、文本、声音、视频、三维场景信息等，量大，异构化，接收者对数据准确性没有严格要求，可容忍些许差错。此时，出现了网络数据安全的新挑战。数字水印技术的提出, 被认为给这些计算机网络所面临的新问题的解决带来了光明的前景。

数字水印的产生最早可追溯到 1954 年，当时，Muzak 公司的埃米利 • 希姆 • 布鲁克为带有水印的音乐作品申请了一项专利。这项专利的内容是通过间歇性地应用中心频率为 1kHz 的窄带陷波器在音乐中插入认证码。此专利在 1984 年被 Muzak 公司使用。1961 年美国专利局这样描述了该项发明：此发明使得对音乐作品进行认证成为可能，是一条防止盗版的有效途径。在 1979 年，

Szepanski 描述了一种机械探测模式，它可以用在文件上起防伪效果。1988 年，Holt 等人阐述了一种在音频信号中嵌入认证码的方法。1993 年，A.Z.Tirkel 等所撰写的“Electronic Watermark”一文中首次使用了“water mark”这一术语，标志数字水印技术作为一门正式研究学科诞生。后来二词合二为一变为“watermark”，而现在一般使用“digital watermark”来命名数字水印。

3.3 特征

3.3.1 鲁棒性

数字水印的鲁棒性又称为稳壮性，是指在经历多种无意或有意的信号处理后，数字水印仍能保持完整性或仍能被准确鉴别，即数字水印应该能够承受大量的、不同的物理和几何失真。可能的信号处理过程包括信道噪声、滤波、数/模与模/数转换、重采样、剪切、位移、尺度变化以及有损压缩编码等。在经过这些操作后，鲁棒的水印算法应仍能从水印图像中提取出嵌入的水印或证明水印的存在。

3.3.2 不可觉察性

数字水印的不可觉察性又称为隐蔽性或透明性。利用人类视觉系统或人类听觉系统的冗余特性，经过一系列隐藏处理，使目标数据没有明显的降质现象，而隐藏的数据却无法被人看见或听见。也就是说，相对于被保护的数据的使用而言，嵌入的数字水印对人的感觉器官应是不可觉察的，或者说是透明的，而且不影响被保护数据的正常使用。

3.3.3 自恢复性

所谓自恢复性是指当含有水印的图像经过一些操作或变换后，可能会使其产生较大的破坏，如果只从留下的片断数据，仍能恢复出水印信号，而且恢复过程不需要原始图像。

3.3.4 水印容量

数字水印的容量是指嵌入到原始图像中的标识信息量。嵌入的水印信息必须足以表示多媒体内容的创建者或所有者的标志信息，或购买者的序列号，从而利于解决版权纠纷，保护数字产权合法拥有者的利益。

3.3.5 小结

在数字水印技术中，水印的数据容量和鲁棒性构成了一对基本矛盾。理想的水印算法应该既能隐藏大量数据，同时可以抵抗各种信道噪声和信号变形。然而在实际的水印系统中，水印容量和鲁棒性这两个指标往往不能同时兼顾，不过这并不会影响数字水印技术的应用，因为实际应用一般只偏重其中的一个方面。如果是为了隐蔽通信，数据量显然是最重要的，由于通信方式极为隐蔽，遭遇篡改攻击的可能性很小，因而对鲁棒性要求不高。但对保证数据安全来说，情况恰恰相反，各

种保密的数据随时面临着被盗取和篡改的危险，所以鲁棒性是十分重要的，此时，隐藏数据量的要求居于次要地位。

3.4 水印的分类

3.4.1 鲁棒水印和脆弱水印

鲁棒水印 要求嵌入的水印能够经受各种常用的编辑处理，主要用于在数字作品中标识著作权信息；

脆弱水印 对信号的改动很敏感，根据脆弱水印的状态就可以判断数据是否被篡改过，主要用于完整性保护。

3.4.2 空间域水印和频率域水印

空间水印 直接在空间域中对采样点的幅度值作出改变，嵌入水印信息的称为空间域水印；

频率域水印 对变换域中的系数作出改变，嵌入水印信息的称为频率域水印。

3.4.3 非盲水印和盲水印

非盲水印 在检测过程中需要原始数据

盲水印 检测只需要密钥，不需要原始数据。

3.4.4 可见水印与不可见水印

顾名思义，分为可见的和不可见的。

3.5 主要用途

3.5.1 版权保护

即数字作品的所有者可用密钥产生一个水印，并将其嵌入原始数据，然后公开发布他的水印版本作品。当该作品被盗版或出现版权纠纷时，所有者即可利用一些方法从盗版作品或水印版作品中获取水印信号作为依据，从而保护所有者的权益。

3.5.2 加指纹

为避免未经授权的拷贝制作和发行，出品人可以将不同用户的 ID 或序列号作为不同的水印 (指纹) 嵌入作品的合法拷贝中。一旦发现未经授权的拷贝，就可以根据此拷贝所恢复出的指纹来确定它的来源。

3.5.3 标题与注释

即将作品的标题、注释等内容 (如，一幅照片的拍摄时间和地点等) 以水印形式嵌入该作品中，这种隐式注释不需要额外的带宽，且不易丢失。

3.5.4 篡改提示

当数字作品被用于法庭、医学、新闻及商业时，常需确定它们的内容是否被修改、伪造或特殊处理过。为实现该目的，通常可将原始图象分成多个独立块，再将每个块加入不同的水印。同时可通过检测每个数据块中的水印信号，来确定作品的完整性。与其他水印不同的是，这类水印必须是脆弱的，并且检测水印信号时，不需要原始数据。

3.5.5 使用控制

这种应用的一个典型的例子是 DVD 防拷贝系统，即将水印信息加入 DVD 数据中，这样 DVD 播放机即可通过检测 DVD 数据中的水印信息而判断其合法性和可拷贝性。从而保护制造商的商业利益。

3.6 图像数字水印的主要算法

3.6.1 空间域算法

典型的算法是将信息嵌入到随机选择的图像点中最不重要的像素位上，算法的鲁棒性差，水印信息很容易为滤波、图像量化、几何变形的操作破坏。

另一个常用方法是利用像素的统计特征将信息嵌入像素的亮度值中，如 Patchwork 算法，随机选择N对像素点 (a_i, b_i) ，然后将每个 ai 点的亮度值加 1 ，每个 bi 点的亮度值减 1，这样整个图像的平均亮度保持不变。检测时，计算如果这个载体确实包含了一个水印，就可以预计这个和为 2n，否则它将近似为零。

3.6.2 频率域算法

图象的频域空间中可以嵌入大量的比特而不引起可察的降质，当选择改变中频或低频分量（除去直流分量）来加入水印时，强壮性还可大大提高。频域水印技术可以利用通用的离散余弦变换，小波变换和傅立叶变换等变换方法。

3.6.3 NEC 算法

NEC 实验室的 COX 等人提出的基于扩展频谱的水印算法, 原则为：

水印信号应该嵌入源数据中对人的感觉最重要的部分。在频谱空间中，这种重要部分就是低频分量。这样，攻击者在破坏水印的过程中，不可避免地会引起图象质量的严重下降。

水印信号应该由具有高斯分布的独立同分布随机实数序列构成。这使得水印经受多拷贝联合攻击的能力大大增强。

3.6.4 变换域算法

该类算法中，大部分水印算法采用了扩展频谱通信。

3.6.5 生理模型算法

利用人的生理模型包括人类视觉系统 HVS(HumanVisualSystem) 和人类听觉系统 HAS。该模型不仅被多媒体数据压缩系统利用，同样可以供数字水印系统利用。利用视觉模型的基本思想均是利用从视觉模型导出的 JND(Just Noticeable Difference) 描述来确定在图象的各个部分所能容忍的数字水印信号的最大强度，从而能避免破坏视觉质量。也就是说，利用视觉模型来确定与图象相关的调制掩模，然后再利用其来插入水印。这一方法同时具有好的透明性和强健性。

Chapter 4

隐写分析

隐写分析的基本原理是根据隐写前后统计特性的变化进而判断分辨载体信息中是否含有秘密信息。比如对图像来说，未经隐写的原始图像一般是连续光滑而且相邻像素之间相关性较高的。但对隐写后的图像，由于秘密信息一般以比特流的形式写入图像的某一部分，因此其与载体中的其他原始信息相互独立，于是会存在图像中相邻像素之间相关性降低的情况。根据不同的目标和适用条件，也可将隐写分析分为：

盲分析 分析者可能含有秘密信息的媒体对象，但对可能适用的隐写算法和隐写内容完全不清楚；

已知消息分析 分析者知道隐藏的消息及派生类型，这可能有助于分析，但这种分析非常困难；

已知载体分析 将不含密的已知原始媒体与分析对象比较，检测其中是否存在差异；

选择隐秘载体分析 分析者在已知对方所用隐写工具和隐写内容的基础上对媒体进行检测；

选择消息分析 分析者可以使用某种隐写工具嵌入选择消息产生隐秘对象以确定其中可能设计某一隐写工具或算法的相应模式已知算法、载体和隐秘对象攻击：分析者已知隐写算法，并同时持有原件和隐秘对象。

根据隐写分析方法可以分为：

感官分析 对被测试载体做一定的预处理，然后利用人类感知和清晰分辨噪声的能力对数字载体进行分析检测；

统计分析 将原始载体的理论期望分析分布和可能的隐秘载体种提取样本分布进行比较，从而找出差别的一种检测方法；

特征分析 进行隐写分析所依赖的就是特征。首先针对某一中种或某一类信息隐写方法，对载体数据和隐秘数据等进行统计分析，找出两者某些统计量上的差异，设计相应的检测算法。

根据隐写分析所针对的隐写算法，也可主要分专用和通用两大类，前者在面对特定的隐写方法时准确性高，但具有单一性，后者则能对几种隐写方法进行检测，但准确度相对较低。隐写分析一般可以分为两个步骤：首先根据隐写过程种的缺陷或根据图像隐写前后统计特征的变化判断出是否含有秘密，其次进一步估计嵌入信息的长度、估计嵌入信息的位置，甚至估计嵌入信息的密钥，从而提取隐秘信息。

4.1 LSB 隐写的检测

隐写分析一般可以分为两个步骤：首先根据隐写过程种的缺陷或根据图像隐写前后统计特征的变化判断出是否含有秘密，其次进一步估计嵌入信息的长度、估计嵌入信息的位置，甚至估计嵌入信息的密钥，从而提取隐秘信息。

下面是一种专用隐写分析，其只针对 LSB 隐写方法。LSB 隐写方法是将像素值的最低有效位用秘密信息替换，即如果待嵌入的秘密信息比特与隐藏该比特的像素的灰度值的最低有效位相同，就不改变原始载体，否则改变灰度值的最低有效位，进行 2i -2i+1 之间的翻转。

对于自然图像，相邻像素的灰度值具有很强的相关性。像素的位平面（依次取出每个像素点的数值组成二值图位平面，比如一幅 256 灰度级的图像可以分解为 8 个位平面 28=256，其中每个像素点的灰度能够用一个 8 位二进制表示，比如当前有两个像素点，灰度值分别为 1 和 10，对应的二进制位分别为 10000000 和 01010000，则分解后组成的八个位平面分别为 10，01，00，01，00，00，00，00）越高，对灰度的贡献越大，相关性越强。而位平面越低，相关性越弱，则被隐写的可能性越大，其相当于随机的噪声，因此需要通过分析像素最低位平面的某些异常来检测。这是因为图像的最低位并不总是 0 和 1 随机均匀分布，而是在某些区域会出现与图像内容有关的结构，但 LSB 隐写会破坏这种结构，而且 LSB 的信息比特替换会引入统计上的不对称性。比如 χ^2 分析能够利用 LSB 隐写后每一对像素的灰度、颜色指数、变换系数数值分布趋于均匀的性质设计 χ^2 检验。

目前利用相邻像素相关性来用于确定载体对象中是否含有秘密信息，针对 LSB 隐写有效的检测方法有：卡方检测算法，RS 检测算法，基于最小均方差的 2LSB 检测算法，基于 WS 的 MLSB 检测算法等。

4.2 χ^2 检测

下面简单介绍对简单 LSB 替换的卡方检测算法：

由于嵌入的秘密信息可以看作 0，1 随机分布的比特流，显然值为 0 或 1 的概率均为 1/2。设图像中灰度值为 j 的像素数量为 h_j, 则如果秘密信息完全替换了载体图像像素的最低位，h_{2i} 和 h_{2i+1} 的值会比较接近，而对于过去未经信息隐写的原始图像，h_{2i} 和 h_{2i+1} 的值相差较大。对于 LSB 隐写方法来说，隐写算法虽然会改变直方图中 h_{2i} 和 h_{2i+1} 的分布，但不会改变 h_{2i} 和 h_{2i+1} 的和，这是因为像素的灰度值只可能都不变或只在 h_{2i} 和 h_{2i+1} 之间互换。所以记

$$\overline{h} = \frac{h_{2i} + h_{2i+1}}{2}$$
$$q = \frac{h_{2i} - h_{2i+1}}{2}$$

如果某个像素灰度值为 2i，则它对参数 q 的贡献为 1/2；如果值为 2i+1，则它对参数 q 的贡献为-1/2。考虑到载体图像中共有 $2\overline{h}$ 个像素的灰度值为 2i 或 2i+1，若所有像素都负载了 1 比特的秘密信息，则像素值为 2i 或 2i+1 的概率都为 1/2。由统计特性，根据正态总体下的抽样分布有：

$$\frac{h_{2i} - \overline{h_{2i}}}{\sqrt{h_{2i}}}\ N(0,1)$$

说明 h_{2i} 满足标准正态分布 $\sim N(0,1)$

考虑 χ^2 分析，设随机变量 $\xi_1, \xi_2, \ldots, \xi_n$ 相互独立，且都服从标准正态分布 N(0,1), 则有随机变量 $\eta = \xi_1^2 + \xi_2^2 + \cdots + \xi_n^2$，其服从 $\chi^2(n)$ 分布，所以有

$$r = \sum_{i=1}^{k} \frac{\left(h_{2i} - \overline{h_{2i}}\right)^2}{\overline{h_{2i}}} \sim \chi^2(k-1)$$

其中的 k 为 h_{2i} 和 h_{2i+1} 所组成的对值的数量，不计 $\overline{h_{2i}} = 0$ 的情况。r 为衡量图像是否被隐写的数值，若 r 越小，说明 h_{2i} 和 h_{2i+1} 值比较接近，被隐写带秘密信息的可能就越大。考虑该分布的密度函数，设 P 为 h_{2i} 和 h_{2i+1} 相等的概率，则有

$$P = 1 - \frac{1}{2^{\frac{k-1}{2}}\Gamma\left(\frac{k-1}{2}\right)} \int_0^{x_{k-1}^2} e^{-\frac{2}{x}} \cdot x^{\frac{k-1}{2}-1} dx$$

这意味着，P 越接近 1，则 h_{2i} 和 h_{2i+1} 越接近，带隐秘信息的可能越大，反之则图像越不可能经过隐写。对一幅顺序隐写的图像，当检测者以相同的顺序对像素组计算 P 值。P 开始接近 1，当快到秘密信息结束时，P 值开始下降，当到图像信息结束时 P 值降为 0。

由上述可见卡方检测可以判定图像中是否有信息嵌入，还能估计出秘密信息的大小。

Chapter 5

代码实践

5.1 附加式的图片隐写（图种）

```
#include <stdlib.h>
#include <stdio.h>
#include <iostream>
#include <cstring>
using namespace std;

int main()
{
        string jpg,rar;
        cout<<"Enter the name of jpg:\n";
        cin>>jpg;
        cout<<"Enter the name of rar:\n";
        cin>>rar;
//      char picname[100] = "rua.jpg";
//      char zipname[100] = "international.rar";
        char resultname[100] = "ha.jpg";

        FILE *fp1, *fp2;
        fp1 = fopen(jpg.data(), "rb");
        fp2 = fopen(rar.data(), "rb");

        FILE *w;
        w = fopen(resultname, "wb");

        int ch;

        while((ch = fgetc(fp1)) != EOF)
                fputc(ch, w);
```

```
29:
30:         while((ch = fgetc(fp2)) != EOF)
31:                 fputc(ch, w);
32:
33:         fclose(fp1);
34:         fclose(fp2);
35:         fclose(w);
36: }
```

5.2 LSB 算法实现

5.2.1 代码

```
1: def read(image):
2:     s1 = ""
3:     s2 = ""
4:     index = 0
5:     info = bytearray()
6:     for i in range(image.shape[0]):
7:         for j in range(image.shape[1]):
8:             for k in range(image.shape[2]):
9:                 if len(s1) < 32:
10:                    s1 += str(image[i][j][k]%2)
11:                else:
12:                    info_bit_len = int(s1,2)
13:                    if index <= info_bit_len:
14:                        if index % 8 == 0 and index != 0:
15:                            info.append(int(s2,2))
16:                            s2 = ""
17:                        s2 += str(image[i][j][k]%2)
18:                        index += 1
19:                    else:
20:                        return info.decode(encoding="utf8")
21:
22: def write(image, info):
23:     info_bytes = bytearray(bytes(info, encoding='utf8'))
24:     text_bit_num = len(info_bytes) * 8
25:     header = "%32s" % bin(text_bit_num)[2:]
26:     index = 0
27:
28:     info_bit = ""
29:     info_bit += header
30:     for item in info_bytes:
```

```
        for itembit in "%8s" % bin(item)[2:]:
            info_bit += itembit

    for i in range(image.shape[0]):
        for j in range(image.shape[1]):
            for k in range(image.shape[2]):
                # 写入header
                if image[i][j][k] % 2 == 0:
                    if info_bit[index] in ['0',' ']:
                        pass
                    else:
                        image[i][j][k] += 1

                else:
                    if info_bit[index] == '1':
                        pass
                    else:
                        image[i][j][k] -= 1
                index += 1
                if index == len(info_bit):
                    return
```

```
import numpy as np
import matplotlib.pyplot as plt
from scipy.misc import imread
# 加载图片
puppy = imread('puppy.bmp')
image = puppy.copy()
info = "我没有看到更广阔的天空，我没有更多的人脉，
我不能成为高晓松的校友，我不能和街上的大神们有说有笑"

print("写入信息:"+info)

write(image, info)

print("解密信息:"+read(image))

plt.subplot(1, 2, 1)
plt.imshow(puppy.astype('uint8'))
plt.title('Original image')
plt.subplot(1, 2, 2)
plt.imshow(image.astype('uint8'))
plt.title('Stegeo image')
plt.show()
print("MSE:" + str(np.sum((puppy - image)**2)))
```

5.2.2 运行结果

Listing 5.1 输出信息

```
写入信息:我没有看到更广阔的天空，我没有更多的人脉，
我不能成为高晓松的校友，我不能和街上的大神们有说有笑
解密信息:我没有看到更广阔的天空，我没有更多的人脉，
我不能成为高晓松的校友，我不能和街上的大神们有说有笑


/home/chaiziwei/anaconda3/envs/cs231n/lib/python3.6/
site-packages/ipykernel_launcher.py:5: DeprecationWarning:
 `imread` is deprecated!
`imread` is deprecated in SciPy 1.0.0, and will be removed
in 1.2.0.
Use ``imageio.imread`` instead.
```

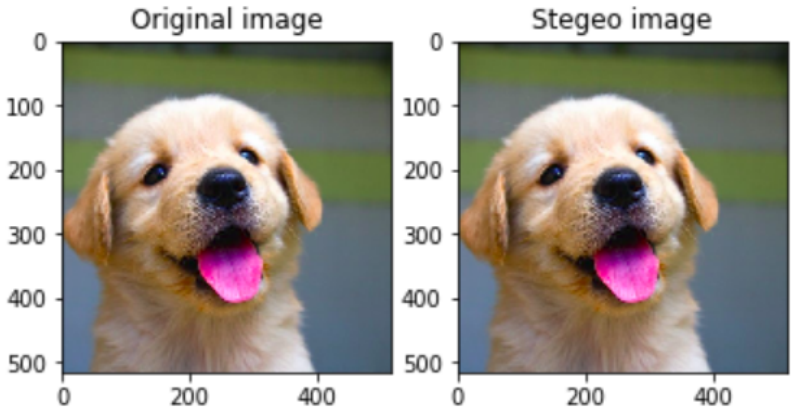


图 5.2.1: 原图像与 LSB 算法嵌入信息后的图像

5.3 LSB 检测 [24]

对 LSB 图像隐写的检测方法较多，下面介绍一种基于邻接直方图的 LSB 隐写检测的原理，实现及测试结果。

5.3.1 原理

LSB 替换隐写在图像中嵌入秘密信息时, 图像像素灰度值的最低位进行操作, 即若秘密信息位与灰度值最低位不同时, 改变灰度值的最后一位，$2m$ 变为 $2m+1$ 或 $2m+1$ 变为 $2m$, 不会是 $2m$ 变为 $2m-1$ 或 $2m+1$ 变为 $2m+2$。设一副大小为 $M\times N$ 的图像 I 在位置 (i,j) 的灰度值为

$I(i,j)$, 二维相邻像素直方图 $h(m,n)$ 的定义为

$$h(m,n) = |\{(i,j)|I(i,j) = m, I(i,j+1) = n\}|$$

因此相邻像素直方图 $h(m,n)$ 可以看成一个大小为 256×256 的矩阵。设秘密信息的嵌入率为 α, 即载体图像每个像素上平均负载 α 比特秘密信息, 而待嵌入的秘密信息是一个 0, 1 等概分布的随机比特流, 因此对于一个像素而言, 在进行最低比特位替换嵌入时, 它保持不变的概率为 $1-\alpha/2$。由此隐写前后的图像相邻像素直方图有如下关系:

$$\begin{aligned} h'(2m+1,2n+1) = &\left(1-\tfrac{\alpha}{2}\right)^2 h(2m+1,2n+1)+ \\ &\left(1-\tfrac{\alpha}{2}\right)\tfrac{\alpha}{2}h(2m+1,2n) + \tfrac{\alpha}{2}\left(1-\tfrac{\alpha}{2}\right)h(2m,2n+1) + \tfrac{\alpha^2}{4}h(2m,2n) \end{aligned} \tag{5.1}$$

$$\begin{aligned} h'(2m,2n) = &\left(1-\tfrac{\alpha}{2}\right)^2 h(2m,2n) + \left(1-\tfrac{\alpha}{2}\right)\times \\ &\tfrac{\alpha}{2}h(2m,2n+1) + \tfrac{\alpha}{2}\left(1-\tfrac{\alpha}{2}\right)h(2m+1,2n)+ \\ &\tfrac{\alpha^2}{4}h(2m+1,2n+1) \end{aligned} \tag{5.2}$$

则有

$$h'(2m+1,2n+1) - h'(2m,2n) = (1-\alpha)[h(2m+1,2n+1) - h(2m,2n)]$$

因此

$$|h'(2m+1,2n+1) - h'(2m,2n)| < |h(2m+1,2n+1) - h(2m,2n)|$$

可见隐写后 $h\,(2m+1,2n+1)$ 和 $h\,(2m,2n)$ 之间的差异会减小。同理可知, 隐写后含密图像相邻直方图中的 $h\,(2m+1,2n+1)$、$h\,(2m+1,2n)$、$h\,(2m,2n+1)$ 及 $h\,(2m,2n)$ 间的差异较之载体图像要小。

定义 F_1、F_2 如下,

$$F_1 = \sum_{m,n=0}^{127}\left(\sum_{k,l=0}^{1}|\boldsymbol{h}(2m+k,2n+l) - \frac{1}{4}\sum_{i,j=0}^{1}\boldsymbol{h}(2m+i,2n+j)|\right) \tag{5.3}$$

$$F_2 = \sum_{m,n=0}^{126}\left(\sum_{k,l=0}^{1}|\boldsymbol{h}(2m+k+1,2n+l+1) - \frac{1}{4}\sum_{i,j=0}^{1}\boldsymbol{h}(2m+i+1,2n+j+1)|\right) \tag{5.4}$$

则对于自然图像 $R = F_1/F_2 \approx 1$, 而对于载密图像 $R = F_1/F_2 \approx 1$

5.3.2 代码与实现

图片预处理

我们使用在在图像处理中常用的无压缩数据集 UCID，由于其是 RGB 图，从 RGB 文件夹中读取文件将其转化为灰度图，存储在 GREY 文件夹中。

```
1: from PIL import Image
2: import os
3: from imageio import imread, imsave
4: import numpy as np
```

```
image_num = 887
read_folder = r'RGB/'
save_folder = r'GREY/'
stegano_folder = r'STEGANO/'

# 读取UCID图片
for i in range(1, image_num):
  s = "ucid" + "{:0>5d}".format(i) + ".tif"
  img = Image.open(os.path.join(read_folder, s)).convert('L')
  savename = os.path.join(save_folder, 'grey'+s)
  img.save(savename)
```

实现计算邻接直方矩阵函数

```
import numpy as np
def calculateF(image):
  h = np.zeros((256, 256), dtype=np.uint32)
  img_flatten = image.flatten()
  for i in range(img_flatten.shape[0]-1):
    h[img_flatten[i]][img_flatten[i+1]] += 1
  F1 = 0
  F2 = 0
  for m in range(128):
    for n in range(128):
      sum1 = 0
      for k in [0, 1]:
        for l in [0, 1]:
          sum2 = 0
          for i in [0, 1]:
            for j in [0, 1]:
              sum2 += h[2*m+i][2*n+j]
          sum2 /= 4
          sum1 += np.abs(h[2*m+k][2*n+l]-sum2)

      F1 += sum1
  for m in range(127):
    for n in range(127):
      sum1 = 0
      for k in [0, 1]:
        for l in [0, 1]:
          sum2 = 0
          for i in [0, 1]:
            for j in [0, 1]:
              sum2 += h[2*m+i+1][2*n+j+1]
```

```
            sum2 /= 4
            sum1 += np.abs(h[2*m+k+1][2*n+l+1]-sum2)

        F2 += sum1
    R = F1/F2
    return R
```

Listing 5.2 运行效果

```
calculateF:  接受图片，计算邻接直方矩阵与R值

输入：  iamge: 图像

输出：  $R$
```

```
from LSB.analysis import calculateF
```

计算 R 值

计算未加密的 UCID 图像集的 R 值

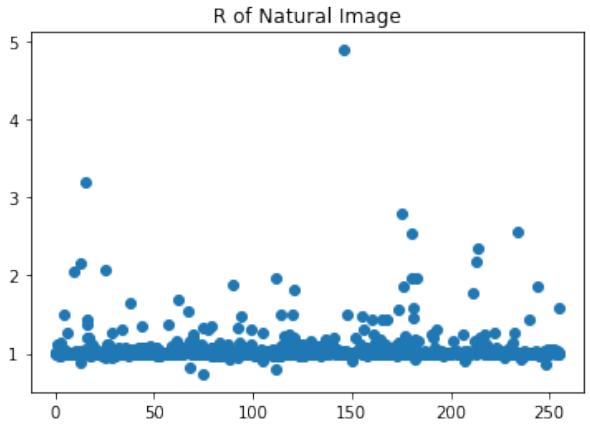
```
list_R = []
index = []

for i in range(1, image_num):
    s = "greyucid" + "{:0>5d}".format(i) + ".tif"
    image = imread(os.path.join(save_folder, s))
    list_R.append(calculateF(image))
    index.append(i)
R = np.array(list_R)
index = np.array(index, dtype = "uint8")
```

可视化

886 张自然图片的 R 值散点图如下, 可以看出其大多数在 1 附近。

```
import matplotlib
import matplotlib.pyplot as plt
plt.scatter(index, R)
plt.title('R of Natural Image')
plt.show()
```

实现加密函数




```
1: def stegano(image, info):
2:   info_bytes = bytearray(bytes(info, encoding='utf8'))
3:   text_bit_num = len(info_bytes) * 8
4:   header = "%32s" % bin(text_bit_num)[2:]
5:   index = 0
6:
7:   info_bit = ""
8:   info_bit += header
9:   for item in info_bytes:
10:    for itembit in "%8s" % bin(item)[2:]:
11:      info_bit += itembit
12:
13:  for i in range(image.shape[0]):
14:    for j in range(image.shape[1]):
15:      # 鍐欏叆header
16:      if image[i][j] % 2 == 0:
17:        if info_bit[index] in ['0',' ']:
18:          pass   # 淇濇寔涓嶅彉
19:        else:
20:          image[i][j] += 1   # +1
21:
22:      else:
23:        if info_bit[index] == '1':
24:          pass  # 淇濇寔涓嶅彉
25:        else:
26:          image[i][j] -= 1  # -1
27:      index += 1
28:      if index == len(info_bit):
29:        return (index)/(image.shape[0]*image.shape[1])
```

```
1: stegano:  接受载体灰度图片，在LSB写入信息
```

```
2:
3: 输入：  iamge: 载体图像    info: 写入信息
4:
5: 输出：  $\alpha$  嵌入率
```

```
1: from LSB.util import stegano
```

LSB 隐写

将 info_10.txt, info_30.txt, info_50.txt 中的信息写入图片, 嵌入率分别如下

```
1: image_name = "greyucid" + "{:0>5d}".format(0) + ".tif"
2: image = imread(os.path.join(save_folder, image_name))
3:
4: file_object = open('info_10.txt',encoding='UTF-8')
5: info_10 = file_object.read()
6: print("info_10.txt的嵌入率为"+str(stegano(image, info_10)))
7:
8: file_object = open('info_30.txt',encoding='UTF-8')
9: info_30 = file_object.read()
10: print("info_30.txt的嵌入率为"+str(stegano(image, info_30)))
11:
12: file_object = open('info_50.txt',encoding='UTF-8')
13: info_50 = file_object.read()
14: print("info_50.txt的嵌入率为"+str(stegano(image, info_50)))
```

Listing 5.3 运行结果

```
1:     info_10.txt的嵌入率为0.10237630208333333
2:     info_30.txt的嵌入率为0.3114420572916667
3:     info_50.txt的嵌入率为0.5142415364583334
```

在 UCID 图片集中选取 100 张图片，分别写入 info_10.txt, info_30.txt, info_50.txt 中的信息，计算三种嵌入率下的 R 值

```
1: image_num = 101
2: file_object1 = open('info_10.txt',encoding='UTF-8')
3: info_10 = file_object1.read()
4: list_R_stegano_10 = []
5: file_object2 = open('info_30.txt',encoding='UTF-8')
6: info_30 = file_object2.read()
7: list_R_stegano_30 = []
8: file_object3 = open('info_50.txt',encoding='UTF-8')
9: info_50 = file_object3.read()
10: list_R_stegano_50 = []
11:
```

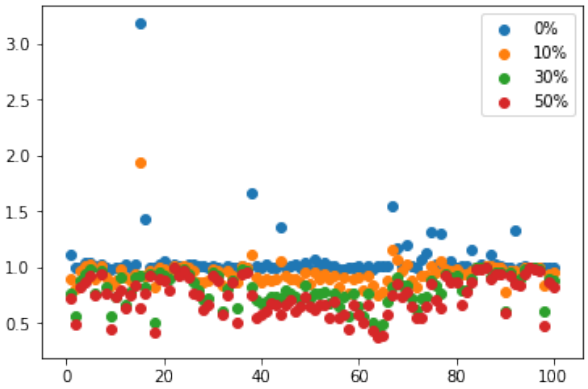
```
for i in range(1, image_num):
    image_name = "greyucid" + "{:0>5d}".format(i) + ".tif"
    image = imread(os.path.join(save_folder, image_name))
    img1, img2, img3 = image.copy(), image.copy(), image.c
opy()

    stegano(img1, info_10)
    s = "steganogreyucid_10_" + "{:0>5d}".format(i) + ".ti
f"
    imsave(os.path.join(stegano_folder, s), img1)
    list_R_stegano_10.append(calculateF(img1))

    stegano(img2, info_30)
    s = "steganogreyucid_30_" + "{:0>5d}".format(i) + ".ti
f"
    imsave(os.path.join(stegano_folder, s), img2)
    list_R_stegano_30.append(calculateF(img2))

    stegano(img3, info_50)
    s = "steganogreyucid_50_" + "{:0>5d}".format(i) + ".ti
f"
    imsave(os.path.join(stegano_folder, s), img3)
    list_R_stegano_50.append(calculateF(img3))
```

测试结果

可以观察到随着嵌入率的增加，R 值不断减小

```
plt.scatter(index[0:100], R[0:100], label = '0%')
R_stegano_10 = np.array(list_R_stegano_10)
plt.scatter(index[0:R_stegano_10.shape[0]], R_stegano_10,
label = '10%')
R_stegano_30 = np.array(list_R_stegano_30)
plt.scatter(index[0:R_stegano_30.shape[0]], R_stegano_30,
label = '30%')
R_stegano_50 = np.array(list_R_stegano_50)
plt.scatter(index[0:R_stegano_50.shape[0]], R_stegano_50,
label = '50%')
plt.legend(loc = 'upper right')
plt.show()
```

因此可设定 R 的阈值对图片进行检测，如设定阈值为 95%, 即 R 小于 95% 即认为其使用 LSB 携带了加密信息, 对嵌入率分别为 0%，10%，30%，50% 的 100 张图片进行检测，召回率与准确率如下




```
th = 0.95
R_sample = R[0:100]
R_sample[R_sample>th] = 0
mis = np.count_nonzero(R_sample)
R_stegano_10[R_stegano_10>th] = 0
print("10%嵌入率下的召回率为"+str(np.count_nonzero(R_
stegano_10)/R_stegano_10.shape[0]))
print("10%嵌入率下的准确率为"+str(np.count_nonzero(R_
stegano_10)/(np.count_nonzero(R_stegano_10)+mis)))
print()
R_stegano_30[R_stegano_30>th] = 0
print("30%嵌入率下的召回率为"+str(np.count_nonzero(R_
stegano_30)/R_stegano_30.shape[0]))
print("30%嵌入率下的准确率为"+str(np.count_nonzero(R_
stegano_30)/(np.count_nonzero(R_stegano_30)+mis)))
print()
R_stegano_50[R_stegano_50>th] = 0
print("50%嵌入率下的召回率为"+str(np.count_nonzero(R_
stegano_50)/R_stegano_30.shape[0]))
print("50%嵌入率下的准确率为"+str(np.count_nonzero(R_
stegano_50)/(np.count_nonzero(R_stegano_50)+mis)))
```

Listing 5.4 运行结果

```
    10%嵌入率下的召回率为0.51
    10%嵌入率下的准确率为1.0

    30%嵌入率下的召回率为0.81
    30%嵌入率下的准确率为1.0

    50%嵌入率下的召回率为0.92
    50%嵌入率下的准确率为1.0
```

Chapter 6

常用工具

6.1 十六进制编辑器 WinHex

WinHex 可以强制以 16 进制数字的形式读取文件。系统中不同的文件区别开来的要点是文件头。如 Jpg 文件头为 FFD8，RAR 文件的文件头是十六进制 Rar!。打开 WinHex 之后便可以查看一个 Jpg 文件是否隐藏了其他的信息。因为 jpg 文件尾 FFD9 之后的信息并不影响 jpg 图片。

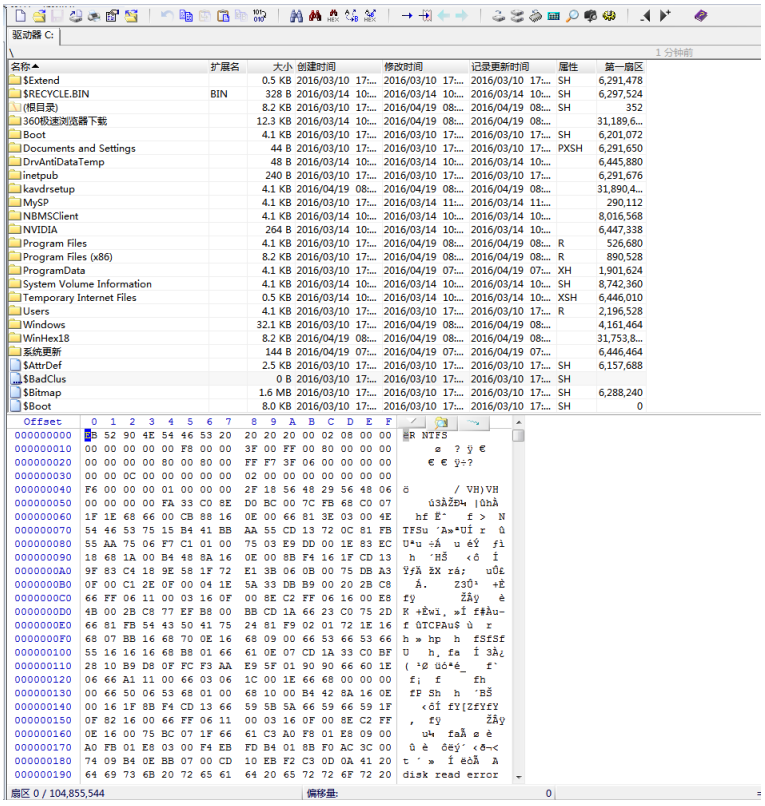


图 6.1.1: WinHex

6.2 多重文件读取器 Binwalk

一般分析某个文件，试图从文件中得到 flag 但多次尝试未果后应当查看下载的文件中是否包含其他文件。这时候一般都需要 binwalk 这个神器。binwalk 的配置环境一般是 linux。在 linux 环境下获取 root 权限后，在终端输入 binwalk。然后执行 binwalk -e 文件路径即可在原目录下分离出另一文件。

6.3 CTF 音频文件 FLAG 获取 - Mp3Stego

目前来说，音频文件使用 mp3 格式已经较为少见，因此见到 mp3 格式的文件，很有可能是将 flag 存放于 txt 后隐写于 mp3 格式文件。

6.3.1 将 txt 分离

在 cmd 命令行中，cd 到 mp3Stego 的文件夹中把需要分析的 mp3 文件拖到 Decode.exe 所在目录里选中的 Decode.exe 拖到命令行里 -X 获取隐藏的东西即可将 txt 分离到目录下

6.3.2 将 txt 写入

此过程需要在 wma 文件转换为 mp3 文件时进行；要隐藏的 TXT 文件是 aa.txt，要压缩的 WMA 文件是 bb.wma，把这两个文件放入与 Encorder.exe 和 Decoder.exe 同一个文件夹，点击 Encode TXT file into an MP3 file 即可产生一个 bb_stego.mp3 文件。

图 6.3.1: Mp3Stego

6.4 数字水印识别 Digimarc

数字水印（Digital Watermarking）技术是将一些标识信息（即数字水印）直接嵌入数字载体当中（包括多媒体、文档、软件等）或是间接表示（修改特定区域的结构），且不影响原载体的使用价值，也不容易被探知和再次修改。但可以被生产方识别和辨认。通过这些隐藏在载体中的信息，可以达到确认内容创建者、购买者、传送隐秘信息或者判断载体是否被篡改等目的。

而 digimarc 是一家专门从事数字水印的公司。主要业务有为产品加上 digimarc 条形码以及条形码的识别。

6.5 图像 LSB 分析工具 SteSlove

当两张 jpg 图片外观、大小、像素都基本相同时，可以考虑进行结合分析，即将两个文件的像素 RGB 值进行 XOR、ADD、SUB 等操作，看能否得到有用的信息。StegSolve 可以方便的进行这些操作。打开 StegSolve，选择“file”->”open”打开一张图片文件，然后选择“analyse”->”image combine”选择另一张图片，默认的 XOR 操作就可以看到隐藏的信息。点击窗口下方的箭头，可以看到不同 combine 方式下的结果。

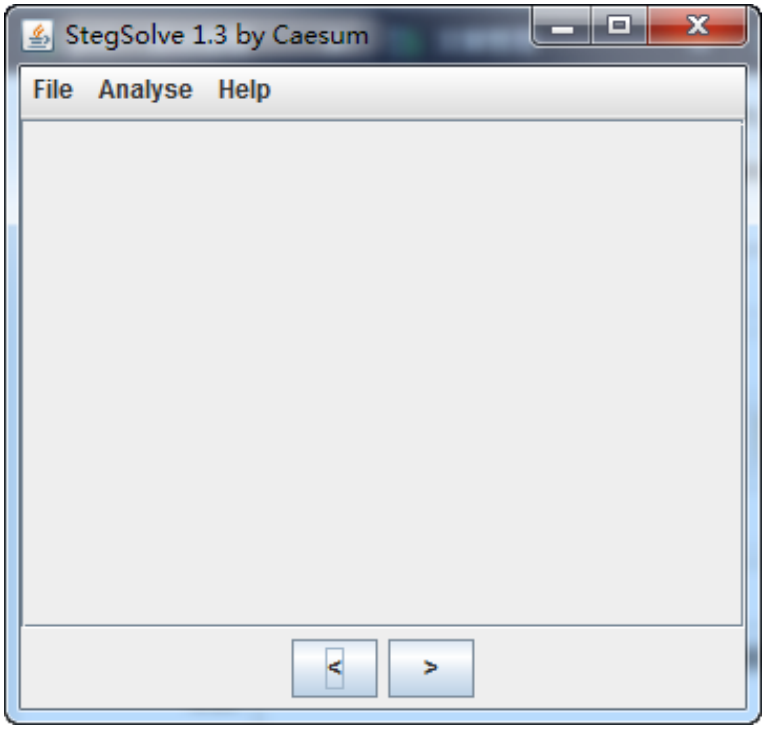


图 6.5.1: SetSolve

参考文献

[1] 维基百科 https://zh.wikipedia.org/wiki/隐写术.

[2] 张军，熊枫，张丹, Jun Z , Feng X , et al. 图像隐写分析技术综述 [J]. 计算机工程, 2013, 39(4):165-168.

[3] 伍玉良. 多载体隐写术与隐写安全分析 [D]. 中北大学.

[4] Jessica Fridrich. Steganography in digital media : Principles , Algorithms and Applications. 国防工业出版社.2014

[5] 尹浩, 林闯, 邱锋, 丁嵘. 数字水印技术综述 [J]. 计算机研究与发展,2005(07):1093-1099.

[6] 易开祥. 数字图象加密与数字水印技术研究 [D]. 浙江大学,2001.

[7] 谭慧. 数字水印技术及其应用 [J]. 信息与电脑 (理论版),2018(13):221-222+225.

[8] 蒋天. 数字水印技术及其应用. 机械工业出版社. 2011

[9] 袁琛，练倩倩.LSB 替换与匹配隐写算法研究 [J]. 数字化用户，2017（39）：242.

[10] 翟黎明, 嘉炬, 任魏翔, 徐一波, 王丽娜. 深度学习在图像隐写术与隐写分析领域中的研究进展 [J]. 信息安全学报，2018.11（6）：2-12.

[11] 王鑫辉. 基于多载体的文本隐写技术研究 [D]. 长沙理工大学,2015.

[12] 陈志立. 语言隐写术的分析与设计研究 [D]. 中国科学技术大学,2009.

[13] 高全胜. 适应即时通信的文本隐写方法研究 [D]. 青岛大学,2018.

[14] 孟威. 面向文本的自然语言隐写术保密通信系统 [J]. 淮北职业技术学院学报,2015,14(01):141-142.

[15] 谢志文. 心理声学掩蔽效应的研究 [D]. 华南理工大学,2005.

[16] 宋华, 幸丘林, 李维奇, 戴一奇.MP3Stego 信息隐藏与检测方法研究 [J]. 中山大学学报 (自然科学版),2004(S2):221-224.

[17] 汝学民. 音频隐写与分析技术研究 [D]. 浙江大学,2006.

[18] W. Bender, D. Grubl, and N. Morimoto, "Techniques for data hiding," IBM Systems Journal, vol. 35, pp. 313-336, 1996.

[19] N. Cvejic, "Algorithms for Audio Watermarking and Steganography," in Department of Electrical and Information Engineering, Information Processing Laboratory, University of Oulu, vol. PhD. Oulu, Fnland: University of Oulu, 2004.

[20] 吴海涛, 詹永照. 数字水印技术综述 [J]. 软件导刊,2015,14(08):45-49.

[21] 孙水发, 李海军, 蒋铭, 万钧力. 数字水印的应用及需求分析 [J]. 信息技术,2007(09):8-11+15.

[22] 基于椭圆曲线密码体系和小波包分解的数字水印算法.

[23] 孙水发. 数字水印技术研究 [D]. 浙江大学,2005.

[24] 邓倩岚, 林家骏, 刘婷. 基于相邻直方图的 LSB 替换隐写分析算法 [D]. 华东理工大学,2010(12).