**《网络空间安全概论》实验报告**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **姓名** | |  | | **年级** | |  |
| **学号** | |  | | **专业、班级** | |  |
| **实验名称** | **实验二 信息隐藏实验** | | | | | |
| **实验时间** |  | | **实验地点** | |  | |
| **实验成绩** |  | | **实验性质** | | **□验证性 □设计性 □综合性** | |
| 教师评价：  □算法/实验过程正确； □源程序/实验内容提交 □程序结构/实验步骤合理；  □实验结果正确； □语法、语义正确； □报告规范；  评语：  评价教师签名（电子签名）： | | | | | | |
| 一、实验目的   * 学习并掌握图像信息隐藏的基本原理和方法。 * 实现基于LSB的信息隐藏和提取算法。 * 用卡方检测对可疑图像进行LSB隐写检测。 | | | | | | |
| 二、实验项目内容   1. 编程实现基于LSB的信息隐藏和提取算法以及卡方检测算法 2. 使用LSB算法在图片中隐藏如下信息：CQUWATERMASKEXP 3. 从被隐藏数据的图片中解析出如上信息，建议使用Matlab 4. 用卡方检测算法对可疑图像进行检测 | | | | | | |
| 三、实验设计  **1.LSB算法：**  任何多媒体信息在数字化时都会产生物理随机噪声，而人的感官系统对这些随机噪声并不敏感。因此，可以通过使用秘密信息比特替换随机噪声，从而实现信息隐藏。在图像中，高位平面对图像感官质量起主要作用，去除图像最低几个位平面并不会造成画面质量的下降。利用这个原理，可用秘密信息（或称水印信息）替代载体图像低位平面以实现信息嵌入。  LSB（Least Significant Bit，最低有效位）算法选用最低位平面来嵌入信息，最低位平面对图像的视觉效果影响最轻微，因此在视觉上很难察觉。作为大数据量的信息隐藏方法，LSB在保密通信中至今仍占据相当重要的地位。其主要步骤如下：  加密：   * 读入图片 * 准备待隐藏的信息，将其转换为二进制（需要用加密算法进行加密） * 遍历图像，对像素的最低1bit置0，同时在该比特位写入1位二进制表示隐藏的信息   解密：   * 预知隐藏信息量（等同于key） * 提取出像素的最低1bit，组合成连续bit数据，转换为ASCII码对比是否与隐藏信息一致   **2.卡方检测算法：**  在信息隐藏领域，卡方检测算法常用于检测隐写术中是否存在隐藏的信息。这种方法基于对图像、音频或其他媒体的统计分析，寻找可能表明存在隐藏信息的异常模式或统计性质。卡方检测的一般步骤为：   * **选取特征：**选择用于分析的特征。例如像素值和像素直方图等。 * **建立模型：**通常使用已知的隐写术的特征和统计性质。例如，对于图像隐写术，可以建立正常图像的像素值分布模型。 * **提取样本：**从待检测的媒体中提取样本，并计算所选特征的值。这些样本可以是图像块或音频片段等。 * **计算卡方统计量：**将提取的样本与模型进行比较，衡量样本与模型之间的差异。通过统计样本的实际观测值与理论推断值之间的偏离程度，可以判断是否存在隐写信息。   在LSB算法中，如果秘密信息位与隐藏位置的像素灰度值的最低比特位相同，则不会改变原始载体。反之，则会改变其灰度值的最低位。约定：  ：一个像素被选中用于隐藏信息的概率；  ：载体图像中，值为j的像素个数；  ：隐写图像中，值为j的像素个数；  假设在秘密消息中，比特0和1是随机分布的。那么在个值为的像素中，有个像素最低比特与消息不同，像素值变为。值为的像素情况也是类似的。由此可得：    当时：    即对于隐写图像来说，值为的像素个数的观测值为，理论值为：    易得：如果图像LSB隐写，那么和一致。因此，由卡方检验原理可知，统计量    服从卡方分布。据此即可进行隐写分析。计算待检测图像统计量，的值越小，意味着和越一致，即待检测图像是隐写图像的概率越高，反之则越低。 | | | | | | |
| 四、实验过程或算法  **1.LSB算法：**  from PIL import Image  class LSB():  def encode(org, msg):# LSB编码  bmsg = ''.join(format(ord(c), '08b') for c in msg)# 将待加密信息转化为二进制串  print("Binary message: ", bmsg)  key = len(bmsg)  ofs = 0# 初始隐写位的偏移量  img = org.copy()  width, height = img.size  cap = width\*height\*3# RGB图像所能容纳的最大秘密消息比特数  if(key + ofs > cap):  raise IndexError('length out of range')  scale = 64# 隐写图像块数  width //= scale  height //= scale  idx = 0  isend = False  for x in range(width):  if(isend):  break  for y in range(height):  if(isend):  break  pixel = list(img.getpixel((x, y)))# 待写入位置的原始RGB值  for i in range(len(pixel)):  if(idx < ofs):  idx += 1  continue  pixel[i] &= 0b11111110# 消除最低比特位  pixel[i] |= int(bmsg[idx - ofs], 2)# 进行最低比特位隐写  idx += 1  if(idx == key + ofs):  isend = True  break  img.putpixel((x, y), tuple(pixel))# 覆写图像    return img, key, ofs    def decode(lsb, key, ofs):# LSB解码, 需要知道加密时的key和ofs  width, height = lsb.size  bmsg = ''  idx = 0  isend = False  scale = 64  width //= scale  height //= scale  for x in range(width):  if(isend):  break  for y in range(height):  if(isend):  break  pixel = lsb.getpixel((x,y))  for p in pixel:  if(idx < ofs):  idx += 1  continue  bmsg += str(p & 1)# 取出最低bit位  idx += 1  if(idx == key + ofs):  isend = True  break  print("Binary message: ", bmsg)  msg = ''  for i in range(0, len(bmsg), 8):  msg += chr(int(bmsg[i:i+8], 2))# 将01字符串重构为文字信息  return msg  msg = 'CS\_LAB2: CQUWATERMASKEXP'# 加密信息  img\_path = './images/'  file = 'lenna.png'  if \_\_name\_\_ == "\_\_main\_\_":    print("Message: ", msg)  org = Image.open(img\_path + file)# 读取原始图像  img, key, ofs = LSB.encode(org, msg)  img.save(img\_path + 'lsb\_' + file)# 保存覆写后的图像    print("Key: ",key)  print("Offset: ", ofs)  lsb = Image.open(img\_path + 'lsb\_' + file)  output = LSB.decode(lsb, key, ofs)# 解密  print("Message: ", output)  **2.卡方检测算法：**  from PIL import Image  from scipy.stats import chi2  import matplotlib.pyplot as plt  import numpy as np  def histogram(img\_path, file):    org = Image.open(img\_path + file)# 读取原图和隐写图  lsb = Image.open(img\_path + 'lsb\_' + file)  width, height = org.size  org\_his = [0] \* 256# 初始化灰度直方图  lsb\_his = [0] \* 256  scale = 64  width //= scale  height //= scale  print("Width: ", width)  print("Height: ", height)  for x in range(width):  for y in range(height):  r, g, b = org.getpixel((x, y))# 分别统计灰度分布情况  org\_his[r] += 1  org\_his[g] += 1  org\_his[b] += 1  r, g, b = lsb.getpixel((x, y))  lsb\_his[r] += 1  lsb\_his[g] += 1  lsb\_his[b] += 1  l = 0  r = 255  plt.figure(figsize=(10, 5))# 绘制直方图  intensity = np.arange(l, r)  plt.bar(intensity, org\_his[l:r], color = 'blue',  alpha = 0.8, width = 0.5, label = 'Original Image')  plt.bar(intensity + 0.5, lsb\_his[l:r], color='red',  alpha = 0.8, width = 0.5, label = 'LSB Image')  plt.title('RGB his Comparison')  plt.xlabel('Intensity')  plt.ylabel('Frequency')  plt.legend()  plt.show()  return org\_his, lsb\_his  def lsbchi(his):  his = np.array(his)  hi1 = his[0::2]  hi2 = his[1::2]  his3 = (hi1 + hi2) / 2  rows = np.where(his3 >= 1)[0]# 分母为零时不存在隐写  if len(rows) == 0:  return 0  k = len(rows)  r = np.sum(((hi1[rows] - his3[rows]) \*\* 2) / his3[rows])# 隐写概率  return chi2.cdf(r, k - 1)# 卡方统计量  img\_path = './images/'  file = 'lenna.png'  if \_\_name\_\_ == "\_\_main\_\_":  org, lsb = histogram(img\_path, file)  # print(org[0::2])  # print(org[1::2])  # print(lsb[0::2])  # print(lsb[1::2])  # print(sum(lsb[0::2]), sum(lsb[1::2]))  print("Original: %.2f%%"%((1 - lsbchi(org))\*100))# 原图的隐写概率  print("LSB: %.2f%%"%((1 - lsbchi(lsb))\*100))# 隐写图的隐写概率  # print(org)  # print(lsb)  # stat, p = chisquare(lsb[0::2], lsb[1::2])  # print("Chi-square statistic: ", stat)  # print("P: ", p)  # stat, p = chisquare(org[0::2], org[1::2])  # print("Chi-square statistic: ", stat)  # print("P: ", p) | | | | | | |
| 五、实验过程中遇到的问题及解决情况  **问题1：**  原图和隐写图的灰度直方图完全一致，且均不符合图片的实际分布情况。  **原因和解决办法：**  语句org\_his = lsb\_his = [0] \* 256将两个直方图对象绑定到同一个列表对象上，导致统计灰度分布时实际上将两张图的灰度值统计至同一对象中。修改为org\_his = [0] \* 256，lsb\_his = [0] \* 256即可。  **问题2：**  对隐写图进行卡方检测得到的隐写概率偏低，即卡方检测失效。  **原因和解决办法：**  实验所使用的Lenna图规模为512 \* 512 \* 3 = 786,432，而Key值仅为120，隐写率仅为0.0153%。使用8 \* 8 \* 3 = 192的子图进行隐写，并将文本量Key值增加至192，则隐写率为100%。此时计算所得的卡方统计量处于合理区间。 | | | | | | |
| 1. 实验结果及分析和（或）源程序调试过程   **1.LSB算法：**  LSB算法的计算过程如下图所示，其中加密过程的Key值为192，初始偏移量Offset为0：    可以看到LSB加解密算法工作正常，破译后的密文与明文完全一致。  写入隐写信息前后的图片如下图所示（右侧为写入秘密信息后的图片），可以看到两张图不存在可见差异。  lsb_lennalenna  **2.卡方检测算法：**  完整图像的隐写概率和灰度直方图如图所示：      可以看到，此时由于隐写率过低（0.0153%），隐写图和原始图片的隐写概率均被认为是为0%，卡方检测失效。  8 \* 8 \* 3子图（即原图的1 / 4096）的隐写概率和灰度直方图如图所示：      此时图片的隐写率为100%，隐写图和原始图片的隐写概率分别为0.54%和75.85%，存在显著差异，说明卡方检测算法工作正常，可以判断图片中存在秘密信息。 | | | | | | |