**《信息安全基础》实验报告**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **姓名** | | 黄昊 | | **年级** | | 2020级 |
| **学号** | | 20204205 | | **专业、班级** | | 20计算机科学与技术（卓越）01 |
| **实验名称** | **实验二 信息隐藏实验** | | | | | |
| **实验时间** | **2022/04/23** | | **实验地点** | | **DS3402** | |
| **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  1. 学习并掌握图像信息隐藏的基本原理和方法  2. 实现基于LSB的信息隐藏和提取算法 | | | | | | |
| 二、实验项目内容  1.使用LSB算法在图片中隐藏如下信息：CQUWATERMASKEXP  2.从被隐藏数据的图片中解析出如上信息，建议使用Matlab  3.在实际应用中，隐藏信息量通常是不可预知的，同时，攻击者也很容易从最低位像素提取到隐藏信息并对此进行密文分析。另一方面，如何确保信息来源于正确的发送者？针对这些问题，请设计完整的方案。 | | | | | | |
| 三、实验设计  由于实验项目第三条内容实际包括前两条内容，所以我们直接考虑第三条。核心思路设计如下：    如图，发送方拥有私钥e，公开公钥d，这一对密钥用于数字签名，用于认定发送方A；同时双方拥有密钥K，这一密钥用于决定密文在图片的嵌入位置。  总体思路如下：发送方使用私钥e对明文进行加密得到密文，在密钥K的作用下将密文嵌入图片的相应位置。接收方收到图片后，使用密钥K从图片中提取密文，经过公钥d的解密提取出明文。  综上所述，私钥e和公钥d实际上负责数字签名，用于确定发送方；同时密钥K的加入，加大了从图片中提取密文的难度。具体的嵌入算法将在第四节介绍。同时，隐藏信息量的确定在第四节也会详细介绍。 | | | | | | |
| 四、实验过程或算法  1、隐藏信息量的确定  为了确定隐藏信息量，我们设计了一种明文格式，该格式要求明文以字节为单位进行传输。然后，我们要求前四个字节存储明文所携带的**字符**数量。如CQUWATERMASKEXP有15个字符，故前四个字节的取值为15。在解密的时候，只要将前四个字节的结果确定下来，就知道这张图片携带了多少信息。  2、基于公钥体制RSA的数字签名  在数字签名中，RSA算法常被使用。发送方使用私钥进行加密，接收方使用公钥解密。如果解密成功，说明该密文确实由发送方发送。这一算法保证了发送方不会有其他人。  3、水印嵌入算法  考虑到原有的LSB算法中，信息通常被嵌在最低位。因此，我们对LSB算法进行了一定的改进。在密钥K的作用下，我们可以确定每一个比特在图片上的嵌入位置。具体而言，算法伪代码如下：   |  | | --- | | Algorithm chaosLSB:  输入:密钥K，嵌入信息量n，图片像素数量m  输出:嵌入位置有序列表*vn*  assert(-1<=K<=1); // 因为要使用混沌序列，要求密钥K为-1到1之间的实数。  *vn*=[ ];  *x* = *K*  for *i* = *1*…*n* do  *x = 1 – 2x2*  *loc = floor(( (x + 1) / 2) \* m)* // 根据图片像素数量进行放缩，决定信息的嵌入位置  if *loc* 在 *vn* 中:  *i = i – 1*  continue  endif  将*loc*加入*vn*的末尾  endfor |   该算法使用混沌序列决定信息的嵌入位置。如果窃听方接收到了图片，由于信息嵌入的位置是随机的，同时K的选取情况是无限种，在图片中嵌入的位置和信息量也是随机的。除非接收方将密钥K公开，否则窃听方按照嵌入顺序获取密文几乎是不可能的。  4、核心代码  import PIL  from PIL import Image  import numpy as np  import random  import math  *# ----config----*  srcPath = "Lena.bmp"  tarPath = "Lena\_hiddenMsg.bmp"  msg = "CQUWATERMASKEXP"  *# 取p = 7, q = 17, 则n = 119, phi(n) = 96, public\_k = e = 5, private\_k = d = 77*  RSA\_encrpyt\_K = 5           *# RSA算法密钥（加密签名用）*  RSA\_decrpyt\_K = 77          *# RSA算法公钥（解密验证用）*  RSA\_n = 119                 *# RSA算法的n*  LSB\_K = 0.20204205          *# LSB算法的密钥*  *# function*  def convertIdxToWH(val, h, w):      '''      用于将一维的下标转换为二维的下标，返回h\_idx和w\_idx，其中h是图像高度，w是图像宽度      '''      return val // w, val % w  *# RSA*  class RSA(object):      def \_\_init\_\_(self) -> None:          pass      def encrypt(encryptKey, s: str, n: int):          '''          每8位进行加密          (解释一下为什么用字符串：字符串的每个字符都是一个字节，不需要像Python的大整数一样进行位运算，比较方便，同时Python的Byte感觉不是很好用)          '''          ans = ""          for c in s:              val = ord(c)              val = (val \*\* encryptKey) % n              assert(0x00 <= val <= 0xff)              ans += chr(val)          print("RSA加密（签名）结束，结果为:{:s}".format(ans))          return ans      def decrypt(decryptKey, s: str, n: int, isPrint = False):          '''          每8位进行解密          '''          ans = ""          for c in s:              val = ord(c)              val = (val \*\* decryptKey) % n              assert(0 <= val < 128)              ans += chr(val)          if isPrint:              print("RSA解密结束，结果为:{:s}".format(ans))          return ans  *# BMPImageProcessor*  class BmpImageProcessor():      '''      BMP图像处理      '''      def \_\_init\_\_(self) -> None:          pass      @staticmethod      def readBmp(path: str):         *# 1D - height, 2D - width*          '''          用于读取BMP文件          '''          listImage = []          with Image.open(path) as img:              width, height = img.size              pixels = img.load()              for y in range(height):                  lst = []                  for x in range(width):                      lst.append(pixels[x, y])                  listImage.append(lst)          return np.array(listImage, dtype="uint8"), width, height      @staticmethod      def saveBmp(arr: np.array, path: str):          '''          用于保存BMP文件          '''          img = Image.fromarray(arr)          img.save(path)  class LSB(object):      '''      图像水印嵌入算法实现      '''      def \_\_init\_\_(self) -> None:          pass        def getEmbeddingArray(K: float, n: int, m: int):          '''          用于获取图片嵌入的下标          '''          ans = []          for \_ in range(n):              K = 1 - 2 \* K \*\* 2              loc = int(math.floor(((K + 1) / 2) \* float(m)))              while loc in ans:           *# 如果要写的位置有重复，则直接进行下一轮迭代*                  K = 1 - 2 \* K \*\* 2                  loc = int(math.floor(((K + 1) / 2) \* float(m)))              ans.append(loc)          return ans        @staticmethod      def encrypt(K: float, msg: str, h: int, w: int, img: np.array):          '''          function          --------          LSB算法加密            params          ------          K: 密钥\\          msg: 需要加密的信息\\          h, w: 图像的高度，宽度\\          img: 灰度图像            return          ------          img: 加密后的灰度图像          '''          n = len(msg) << 3          arr = LSB.getEmbeddingArray(K, n, h \* w)          for chIdx, ch in enumerate(msg):            *# 对每个字节进行加密*              val = ord(ch)              embedList = arr[8 \* chIdx: 8 \* (chIdx + 1)]     *# 获取当前字节的八个比特在图片中的隐写位置*              for i in range(8):                  b = (val >> (7 - i)) & 1                    *# 获取比特位，从高（指数位权值高）到低写*                  embedIdx = embedList[i]                  h\_idx, w\_idx = convertIdxToWH(embedIdx, h, w)                  img[h\_idx][w\_idx] = (img[h\_idx][w\_idx] & 0xfe) + b      *# 抹除最低位，再填上最低位*          return img        @staticmethod      def decrypt(K: float, h: int, w: int, img: np.array):          '''          function          --------          LSB算法解密            params          ------          K: 密钥\\          h, w: 图像的高度，宽度\\          img: 灰度图像            return          ------          msg: 解密后的信息          '''          arr = LSB.getEmbeddingArray(K, 32, h \* w)          length = 0  *# 获取长度信息*          for i in range(4):              embedList = arr[8 \* i: 8 \* (i + 1)]              val = 0              for j in range(8):                  h\_idx, w\_idx = convertIdxToWH(embedList[j], h, w)                  b = img[h\_idx][w\_idx] & 1                  val = (val << 1) + b  *# 这里注意要先解密才能正常解析*              decrypt\_val = ord(RSA.decrypt(RSA\_decrpyt\_K, chr(val), RSA\_n))              length = (length << 8) + decrypt\_val          print("LSB解密阶段，获取到的密文长度为:{:d}".format(length))          arr = LSB.getEmbeddingArray(K, 32 + length \* 8, h \* w)          arr = arr[32: ]          ans = ""  *# 开始获取密文*          for i in range(length):              embedList = arr[8 \* i: 8 \* (i + 1)]              val = 0  *# 对每个字节（字符）进行提取*              for j in range(8):                  h\_idx, w\_idx = convertIdxToWH(embedList[j], h, w)                  b = img[h\_idx][w\_idx] & 1                  val = (val << 1) + b              ans += chr(val)          print("LSB解密结束，得到的密文是{:s}（已去除头部长度信息）".format(ans))          return ans    if \_\_name\_\_ == "\_\_main\_\_":  *# =========加密部分=========*  *# 给信息加上头部，存储信息量*      val = len(msg)      msg = chr((val & 0xff000000) >> 24) + chr((val & 0xff0000) >> 16) + chr((val & 0xff00) >> 8) + chr(val & 0xff) + msg      print("原文的长度为{:d}，处理后的原文长度为{:d}，其中增加的4的来源是4字节的长度位，标识隐藏信息量".format(val, len(msg)))  *# 将信息用RSA算法进行数字签名*      signed\_msg = RSA.encrypt(RSA\_encrpyt\_K, msg, RSA\_n)  *# 使用LSB算法进行嵌入*      img, width, height = BmpImageProcessor.readBmp(srcPath)      *# 读取图片*      img\_encrypt = LSB.encrypt(K = LSB\_K, msg = signed\_msg, h = height, w = width, img = img)      BmpImageProcessor.saveBmp(img\_encrypt, tarPath)    *# =========解密部分=========*    *# 读取加密后的文件*      img\_encrypted\_read, width, height = BmpImageProcessor.readBmp(tarPath)  *# 将隐藏信息提取出来*      decrypt\_msg\_LSB = LSB.decrypt(K = LSB\_K, h = height, w = width, img = img\_encrypted\_read)  *# 对密文进行解密*      decrypt\_msg\_RSA = RSA.decrypt(RSA\_decrpyt\_K, decrypt\_msg\_LSB, RSA\_n, isPrint=True)        print("每个比特的隐写位置(一维)为：")      print(LSB.getEmbeddingArray(LSB\_K, len(msg) \* 8, width \* height)) | | | | | | |
| 五、实验过程中遇到的问题及解决情况  长度解析不对：    经过排查，发现是读取数据头时忘记使用RSA算法解密，导致长度解析不对。 | | | | | | |
| 六、实验结果及分析和（或）源程序调试过程    运行所给代码，运行结果如图。在代码运行的时候，打印了中间过程。可以看到，加上头部长度信息后，原文长度从15到19，增加的4个“长度”事实上就是4个Byte，其中存储了密文的长度信息。RSA加密后得到了密文，以ASCII码的形式打印，可以看出完全读不懂。经过LSB加密后，可以看到有一张图片生成：    其内容为：    可以看到，图像与原图像几乎看不出区别。之后按照设计的水印嵌入算法，将密文提取出来。去除头部信息后，将密文以ASCII码的形式打印。可以看到，这与RSA加密阶段打印的密文几乎无区别（唯一的区别是去除了头部信息0x00 00 00 0f），而前三个字节以ASCII码打印出来是不会显示的。所以LSB算法实现正确。最后使用RSA的公钥解密，可以看到密文被成功地解析出来。  最后，打印了所有比特在图像中的嵌入位。可以看到，这些位置是不可预测的。因此，本次实验实现了可以鉴别发送方的，不易解析出密文的LSB图像隐写算法。 | | | | | | |