**《信息安全基础》实验报告**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **姓名** | | **孙莹莹** | | **年级** | | **2020级** |
| **学号** | | **20204104** | | **专业、班级** | | **计算机科学与技术卓越一班** |
| **实验名称** | **实验二 信息隐藏实验** | | | | | |
| **实验时间** | **2023.4.23** | | **实验地点** | | **DS3401** | |
| **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  编程实现基于LSB的信息隐藏和提取算法 | | | | | | |
| 三、实验设计  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，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在保密通信中仍占据相当重要的地位。  LSB主要步骤：  **加密：**  1.读入图片  2.准备待隐藏的信息，将其转换为二进制  3.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：**  1.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 | | | | | | |
| 四、实验过程或算法  **1.使用LSB算法在图片中隐藏如下信息：CQUWATERMASKEXP**  对于加密，我们定义一个Encode函数，他接受一张图片的路径和需要隐藏的信息作为参数，返回的是被隐藏信息的ASCLL码的长度，即隐藏信息量（key)。  首先需要读入图片，这里调用python的image库的PIL模块辅助我们进行处理，将图片的模式设置为RGB，将信息每一位转换为ASCLL码然后拼接到一起：   1. **def** Encode(imgPath, data): 2. # 读入图片 3. img = Image.open(imgPath) 4. img = img.convert('RGB') 5. # 将待隐藏的信息转换为二进制 6. binaryData = ''.join(format(ord(c), '08b') **for** c **in** data)   然后进行信息隐藏，这里我们遍历从上到下、从左到右遍历图片的每一个像素点，然后对图片每一个像素点划分为R,G,B三个通道，每个通道首先将最低位置0（采用该通道数值和~1相与的办法），如果此时还有信息没有被隐藏到图片里(即index<len(binaryData))，那么就通过|将信息置入该通道。如果信息已经全部被隐藏到图中了，那么后面的像素点不再修改。  eg: 对于信息010:0被隐藏到到img[0][0].R,1被隐藏到到img[0][0].G,:0被隐藏到到img[0][0].B。  修改后的RGB值被保存起来，然后更新图像的像素值，将加密后的图像存储起来，同时返回被隐藏信息的ASCLL码的长度。代码如下:   1. # 遍历图像，对每个像素的最低1bit置0，并在该比特位写入1位二进制表示隐藏的信息 2. newPixelList = [] 3. index = 0 4. width = img.size[0] 5. height = img.size[1] 6. length = len(binaryData) 7. **for** x **in** range(width): 8. **for** y **in** range(height): 9. r, g, b = img.getpixel((x, y)) 10. r\_new = r 11. g\_new = g 12. b\_new = b 13. **if**(index < len(binaryData)): 14. r\_new = (r & ~1) | int(binaryData[index]) 15. index += 1 16. **if** (index < len(binaryData)): 17. g\_new = (g & ~1) | int(binaryData[index]) 18. index += 1 19. **if** (index < len(binaryData)): 20. b\_new = (b & ~1) | int(binaryData[index]) 21. index += 1 22. newPixelList.append((r\_new, g\_new, b\_new)) 24. # 更新图像的像素值 25. img.putdata(newPixelList) 27. # 保存加密后的图像 28. img.save("encrypted\_image.png") 30. **return** length   **2.实现基于LSB的信息隐藏和提取算法**  提取算法函数Decode()接受一张图片的路径和隐藏信息量（key)作为参数，返回的是被隐藏信息。  要得到隐藏的信息，只需要提取该像素点的R，G，B值，分别和1进行&操作即可。需要注意的是信息是被顺序隐藏到三个通道，所以需要判断当前提取的长度是否已经达到了被隐藏信息的长度，对于后面的像素点可以直接跳过。代码如下：   1. **def** Decode(imgPath,length): 2. # 读入加密的图像 3. img = Image.open(imgPath) 4. # 提取隐藏数据 5. binaryData = '' 6. pixel\_list = list(img.getdata()) 7. i = 0; 8. **for** pixel **in** pixel\_list: 9. r, g, b = pixel 10. **if**(i < length): 11. binaryData += str(r & 1) 12. i += 1 13. **else**: 14. **break**; 15. **if** (i < length): 16. binaryData += str(g & 1) 17. i += 1 18. **else**: 19. **break**; 20. **if** (i < length): 21. binaryData += str(b & 1) 22. i += 1 23. **else**: 24. **break**;   提取到隐藏的信息后，只需要将他们八位八位地从ASCLL码还原成字符然后返回即可：   1. # 将二进制数据转换为字符串 2. data = '' 3. **for** i **in** range(0, len(binaryData), 8): 4. data += chr(int(binaryData[i:i+8], 2)) 6. **return** data   最后调用上述两个函数实现加密解密：   1. message = "CQUWATERMASKEXP" 2. length = Encode("test.bmp", message) 3. extractedMessage = Decode("encodedImage.png",length) 4. **print**(extractedMessage)   **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。**  一套完整的信息隐藏方案有以下几个步骤：  1.加密：将原始消息使用加密技术转换为不可读的格式，只有持有正确密钥的人才能解密和读取。可以使用对称加密算法（如AES）或非对称加密算法（如RSA）来加密消息。  2.数字签名：发送者使用自己的私钥对消息进行签名，接收者使用发送者的公钥验证签名的有效性，以确保消息来源正确。可以使用数字证书颁发机构（CA）来验证公钥的真实性。  3.隐藏：基于频率分析的隐写算法，将加密后的消息嵌入图像、音频或视频中。  因此本实验中我们设计的方案首先采用EdDSA数字签名，然后将签名和信息长度加到信息前面，之后整体用ASE加密算法对信息进行加密，然后，最后将这些信息用LSB算法隐藏到图片中，实现信息加密。  方案中具体实现步骤如下：   * **使用EdDSA进行数字签名**   EdDSA是一种基于椭圆曲线密码学的数字签名算法。它使用了一种称为Edwards曲线的特殊曲线，并结合哈希函数和随机数生成器来实现数字签名。要验证一个EdDSA签名，需要使用相同的公钥、消息和签名，以及与签名创建时使用的随机数相同的随机数。具体步骤如下：  1.将原始消息使用哈希函数处理，得到哈希值H。  2.将签名拆分为两个部分：R和s。其中R是一个椭圆曲线上的点，s是一个整数。  3.对R进行一些必要的验证，例如检查它是否满足特定的条件（例如，它必须在Edwards曲线上），并检查它是否与公钥匹配。  4.计算哈希值的哈希值，用于生成一个固定长度的字节数组。  5.计算w = s^-1 mod L，其中s^-1表示s的逆元mod L，L是Edwards曲线的阶。  6.计算u = H \* w mod L和v = R \* w mod L。  7.计算点P = u \* B + v \* A，其中B是Edwards曲线的基点，A是公钥对应的点。  8.如果P等于R，则验证通过。  如果所有步骤都通过，则可以确定签名是由公钥持有者创建的，并且消息没有被篡改。  用python实现生成签名和验证的代码如下：   1. **import** ed25519 3. # 生成公钥和私钥 4. sk, vk = ed25519.create\_keypair() 6. # 消息 7. message = b"By LUY!" 9. # 签名 10. signature = sk.sign(message) 12. # 验证签名 13. **try**: 14. vk.verify(signature, message) 15. **print**("Signature is valid") 16. **except**: 17. **print**("Signature is invalid")   首先使用create\_keypair()函数生成了一个公钥和私钥对。然后，定义一个消息message，并使用私钥s对其进行签名。最后，我们使用公钥v来验证签名，如果签名有效则打印出Signature is valid。   * **AES加密算法：**   AES加密算法流程如下：    这里主要调用python库实现AES加密算法。首先是加密函数，需要注意的是AES算法是基于对二进制数据块的操作来实现的，而文本字符串包含的字符不能直接传递给算法进行处理。因此在进行加密之前需要将文本转换为字节类型。加密步骤首先是对明文进行填充，转换为16字节的倍数，然后进行加密，最后返回密文。代码实现如下：   1. # 将明文转换为16字节的倍数 2. **def** pad\_text(text): 3. **while** len(text) % 16 != 0: 4. text += b' ' 5. **return** text 7. # 加密函数 8. **def** aes\_encrypt(text, key): 9. # 初始化加密器 10. cipher = AES.new(key.encode(), AES.MODE\_ECB) 11. # 对明文进行填充 12. padded\_text = pad\_text(text) 13. # 加密明文 14. encrypted\_text = cipher.encrypt(padded\_text) 15. # 返回Base64编码后的密文 16. **return** base64.b64encode(encrypted\_text)   解密算法则是调用函数解密以后，去除掉填充信息即可，代码如下：   1. # 解密函数 2. **def** aes\_decrypt(ciphertext, key): 3. # 初始化解密器 4. cipher = AES.new(key.encode(), AES.MODE\_ECB) 5. # 解密密文 6. decrypted\_text = cipher.decrypt(base64.b64decode(ciphertext)) 7. # 去除填充信息并返回明文 8. **return** decrypted\_text.rstrip(b' ')   关于这部分代码测试可以看最后实验结果部分   * **LSB部分前文已经详细叙述，不再赘述** * **将上述三部分联合起来，进行信息加密**   首先是根据消息生成数字签名，把他的长度b记录下来（这样做是为了解密后可以将签名内容和消息内容分开处理），然后将数字签名添加到消息的前面，将他们总长度a记录下来，然后再把a添加到原本的消息前面，组成发送出去的信息，这些信息首先通过AES进行加密，密文然后传送给LSB模块进行信息隐藏，生成图片。加密步骤完成。  这里为了方便假设消息的前3位存放的是除了长度以外信息（签名+消息）的总长，在实际应用中信息长度可能很大，可以设置前8位存放长度信息  解密的时候则是首先用LSB解密算法从图片中提取信息（这里假设收消息的一方不知道信息有多长，但是知道数字签名的长度b和存放长度信息的位长为3,因此从图片中读信息的时候是读取了所有像素点RGB最低位的信息），然后通过AES解密算法解密密文，先从前三位读出消息的总长a，然后提取b位的数字签名进行验证，判断信息来源是否正确，如果正确，则读取剩下的a-b位信息，即为发送的信息。  调用过程如下：   1. # 测试代码 2. text = b'SYY is Lucky Serena' 3. strText = str(text, encoding='utf-8') 4. key = 'Sixteen byte key' 5. print(('The original information is: ',strText) 6. # 签名 7. # 生成公钥和私钥 8. sk, vk = ed25519.create\_keypair() 9. signature = sk.sign(text) 10. strSignature = str(signature, encoding='utf-8') 11. b = len(strSignature) #数字签名长度 12. a = b + len(strText)  #总长度 13. sendmsg = str(a) + strSignature + strText #发送出去的消息 14. sendmsg = bytes(sendmsg,  encoding='utf8') 15. #AES加密 16. encryptedText = aes\_encrypt(sendmsg, key) 17. readEncryted = str(encryptedText, encoding='utf-8') 18. **print**(Use AES to encrypt:', readEncryted) 19. #LSB加密 20. length = Encode("Lena.bmp", readEncryted) 21. **print**("length= ",length) 22. #LSB解密 23. extractedMessage = Decode("encrypted\_image.png",length) 24. **print**('Extraccted LSB Message: ', extractedMessage) 25. #AES解密 26. byteextractedMessage = bytes(extractedMessage, encoding='utf8') 27. decryptedText = aes\_decrypt(byteextractedMessage, key) 28. #读取长度和数字签名 29. readDecry = str(decryptedText, encoding='utf-8') 30. s = "" 31. **for** i **in** range(0,3): 32. s += readDecry[i] 33. sign\_sent = "" 34. **for** i **in** range(3,3+b+1): 35. sign\_sent += readDecry[i] 36. sign\_set = bytes(sign\_sent,encoding='utf8') 37. # 验证签名 38. **try**: 39. vk.verify(sign\_sent, strText) 40. **print**("Signature is valid") 41. **except**: 42. **print**("Signature is invalid") 43. **for** i **in** range(3+b+1,4+int(s)): #输出明文 44. **print**(readDecry[i]) | | | | | | |
| 五、实验过程中遇到的问题及解决情况  1.如果在读入图像后直接调用r,g,b = img.getpixel((x, y))来获取x,y点的RGB数值,会出现错误：    原因是读入图像后采用的模式是不一定的，如果需要提取RGB通道的数值，需要先将图像设置为RGB模式，故需要使用convert函数，修改代码如下：    2.一开始解码时直接采用提取length个像素点的RGB值进行解码，发现解码出来的信息多出来了一部分：  原因是忘记了信息被编入了三个通道，所以每提取R,G,B一个通道后都要对长度加一，判断是否已经提取完毕。修改后代码成功运行。 | | | | | | |
| 六、实验结果及分析和（或）源程序调试过程  **1.使用LSB算法在图片中隐藏如下信息：CQUWATERMASKEXP**  采用Lena图像，加密后的图像如图：    可以看到图像基本没有什么改变，信息能够较好地被隐藏。  **2.实现基于LSB的信息隐藏和提取算法**  提取信息的输出如下，发现可以正确提取到隐藏的信息：    **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。**  对于这个问题，我们设计的方案是采用EdDSA数字签名,然后将签名加信息长度和签名加到信息签名，对整体用ASE加密算法进行加密，然后，将这些信息用LSB算法隐藏到图片中，实现信息加密。解密则是先从图片中提取信息，解密信息，然后验证签名，最后提取信息。   1. ASE算法测试结果：     可以看到能够成功实现加密解密。   1. 整体方案实现结果：   隐藏信息后的图片：可以看到图片并没有大的变化，不容易暴露信息。    程序输出：可以看到成功通过AES进行加密，同时签名验证成功，最后提取出来了发送的消息。    （3）思考与总结：  为了解决攻击者可以通过分析最低位像素来发现这些隐藏的信息的这个问题，首先可以考虑使用其他的像素位来隐藏信息，这是就需要双方之间传递密钥，说明隐藏信息的是哪一个像素位，但是如果像素位选择不当可能会造成图片有较大的改变（可以考虑一些本来看起来就杂乱无章的抽象派图片）。  其次是可以使用更加高级的隐藏算法，比如使用基于人工智能的技术，例如深度学习模型，它们可以根据像素数据自动提取和嵌入信息，以避免被最低位像素攻击所检测到。此外，还可以使用特定的加密算法对嵌入的信息进行加密，以保护其机密性，并且只有授权用户才能解密。还有基于频率分析的隐写算法，这种算法通过分析媒体中的频率分布来隐藏信息。例如，在图像中，可以根据像素的亮度和颜色分布来嵌入信息。这种算法对于像素分析攻击也比较鲁棒。  如果要确定信息来源于正确的发送者，可以使用数字签名技术，数字签名可以验证消息的完整性、真实性和来源。发送者向消息添加数字签名，接收者可以使用发送者公钥来验证签名，以确定消息是否被篡改过且确实来自该发送者。  基于上述分析，一套完整的信息隐藏方案有以下几个步骤：  1.加密：将原始消息使用加密技术转换为不可读的格式，只有持有正确密钥的人才能解密和读取。可以使用对称加密算法（如AES）或非对称加密算法（如RSA）来加密消息。  2.数字签名：发送者使用自己的私钥对消息进行签名，接收者使用发送者的公钥验证签名的有效性，以确保消息来源正确。可以使用数字证书颁发机构（CA）来验证公钥的真实性。  3.隐藏：基于频率分析的隐写算法，将加密后的消息嵌入图像、音频或视频中。  4.提取：将隐藏的消息从上述载体中提取出来，并使用相应的密钥和验证技术对其进行解密和验证签名的有效性。  5.鲁棒性：考虑到防止攻击者破解或篡改消息，可以添加错误检测和纠正码，使用抗剪切和抗旋转的隐写术技术，或使用多重验证技术。 | | | | | | |