**《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** | **实验一 加解密算法的实现** | | | | | |
| **实验时间** | **2023/4/9** | | **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  掌握频度分析法原理和Feistel加解密原理 | | | | | | |
| 二、实验项目内容   1. 使用频度分析法解密文本，并写出替换表 2. 编程实现Feistel加密/解密 | | | | | | |
| 三、实验设计  考虑到有两个实验内容，本次实验使用Python语言执行，分两个文件。考虑到频度分析法中，由于频率相近的字母也有很多，有多种可能性。因此，我们在通过已有的频度粗略给出破译结果后，根据已有文档给出的二元和三元词的频度，进行交互式的微调。  Feistel网络则采用类实现。其中密钥长度为128， 分组大小选择64，循环次数为16，子密钥生成算法采用马特赛特旋转演算法，F函数为  F(x,y)= (0x996177f3 ^ x) \* (y & 0x3312ff78)，使用的是非线性函数。 | | | | | | |
| 四、实验过程或算法  1、频度分析法：  代码的大体思路为：使用freq\_statistic统计密文中字母的出现频率，然后使用getMapping函数，结合实际的字母出现频率去匹配明文和密文之间的映射关系。然后在decrypt函数去破译密文。然后在interactive\_dec rypter函数中进行交互式的调整，根据材料所给信息和自身猜测微调映射关系，等到微调合适后再给出最终结果。微调的逻辑如下：    如图，虚线表示原有的映射关系，红线表示调整后的映射关系，如果在明文中，认为D的位置应该是Y，那么我们同时也会将Y的位置换成D。  代码实现如下：  encrypted\_txt = "UZ QSO VUOHXMOPV GPOZPEVSG ZWSZ OPFPESX UDBMETSX AIZ VUEPHZ HMDZSHZO WSFP APPD TSVP QUZW YMXUZUHSX EPYEPOPDZSZUFPO MB ZWP FUPZ HMDJ UD TMOHMQ"  freq\_list = ["E", \               "T", "A", "O", "I", "N", "S", "H", "R", \               "D", "L", \               "C", "U", "M", "W", "F", "G", "Y", "P", "B", \               "V", "K", "J", "X", "Q", "Z"]  def freq\_statistic(txt):      lst = [0 for \_ in range(26)]      tot = 0      for ch in txt:          if ch == ' ':              continue          else:              tot += 1              idx = ord(ch) - ord('A')              lst[idx] += 1      for idx, val in enumerate(lst):          lst[idx] = (chr(idx + ord('A')), val / tot)      return lst  def getMapping(freq\_list, statistic\_list):      statistic\_list.sort(key = lambda x: x[1], reverse=True)      decrypt\_dic, encrypt\_dic = {}, {}      for i in range(26):          decrypt\_dic[statistic\_list[i][0]] = freq\_list[i]          encrypt\_dic[freq\_list[i]] = statistic\_list[i][0]      decrypt\_dic[" "] = " "      encrypt\_dic[" "] = " "      return decrypt\_dic, encrypt\_dic    def decrypt(mapping, txt):      ans = ""      for ch in txt:          ans+= mapping[ch]      return ans    def mappingAdjusting(decrypt\_mapping, encrypt\_mapping, v1, v2):      k1 = "A"      k2 = "A"      for k, v in decrypt\_mapping.items():          if (v == v1):              k1 = k          if (v == v2):              k2 = k      decrypt\_Mapping[k1] = v2      decrypt\_Mapping[k2] = v1      encrypt\_Mapping[v1] = k2      encrypt\_Mapping[v2] = k1      return decrypt\_mapping, encrypt\_mapping  def encoderPrint(decrypt\_mapping):      for k, v in decrypt\_mapping.items():          if (k == " "):              continue          print("{:s} -> {:s}".format(k, v))  def interactive\_decrypter(encrypted\_txt, decrypt\_Mapping, encrypt\_Mapping):      count = 1      while True:          decrypt\_txt = decrypt(decrypt\_Mapping, encrypted\_txt)          print("第 {:d} 次的解密结果为: {:s}\n\t请输入您认为应该变换的字母(输入格式：I P)，若认为是最终结果，请输入-1：".format(count, decrypt\_txt))          a = input()          if a.strip(" ") == "-1":              print("破译结束。")              print("密码本为：")              encoderPrint(encrypt\_Mapping)              print("破译结果为：")              print(decrypt\_txt)              break          else:              v1, v2 = a.split()          count += 1          decrypt\_Mapping, encrypt\_Mapping = mappingAdjusting(decrypt\_Mapping, encrypt\_Mapping, v1, v2)  lst = freq\_statistic(encrypted\_txt)  decrypt\_Mapping, encrypt\_Mapping = getMapping(freq\_list, lst)  decrypt\_txt = decrypt(decrypt\_Mapping, encrypted\_txt)  interactive\_decrypter(encrypted\_txt, decrypt\_Mapping, encrypt\_Mapping)  2、Feistel网络：  代码大体思路为：  首先初始化确定分组大小，初始种子（用于生成随机种子），F函数，循环次数。之后使用encrypt()函数进行加密，decrypt()函数进行解密。这两个函数的作用是对输入的比特或文本进行分组，然后对每组数据调用\_\_setEncrypt()函数或\_\_setDecrypt()函数进行加密或解密。\_\_Feistel\_bl ock()函数的作用在于使用单个Feistel块进行单次正向运算，而\_\_Inv\_Fei stel\_block()函数用于在解密过程中进行单次反向运算。  代码实现如下：  import random  class Feistel(object):      def \_\_init\_\_(self,  seed : int = 9961, \                          encryptTime : int = 16, \                          charSetSize : int = 4, \                          function = lambda x, y: ((x + 1) \*\* y % ((x + 1) \* y)) + x + y, \                          debugging : bool = False \                  ) -> None:          self.seed = seed          self.encryptTime = encryptTime          random.seed(self.seed)          self.subseed = self.\_\_generate\_sub\_seed()          self.function = function        *# Feistel网络中的非线性函数*          self.charSetSize = charSetSize  *# 几个字符为一组*          self.debug = debugging          assert(charSetSize >= 4)        *# 要求至少四个字符为一组（一个字符两个字节有点少，希望至少是八个字节）*        @property      def \_\_MOD(self):          return (self.charSetSize \* 8 // 2)        def \_\_generate\_sub\_seed(self):          return [random.randint(0, 0xffff\_ffff\_ffff\_ffff\_ffff\_ffff\_ffff\_ffff) for \_ in range(self.encryptTime)]        def \_\_Feistel\_block(self, L0, R0, K1):          L1 = R0          R1 = L0 ^ (self.function(R0, K1) % self.\_\_MOD)  *# 保证输出空间在[0,0xffffffff]之间*          return L1, R1        def \_\_Inv\_Feistel\_block(self, L1, R1, K1):          R0 = L1          L0 = R1 ^ (self.function(L1, K1) % self.\_\_MOD)          return L0, R0        def \_\_setEncrypt(self, L, R):       *# 分组后的加密*          for i in range(self.encryptTime):              L, R = self.\_\_Feistel\_block(L, R, self.subseed[i])          return L, R        def \_\_setDecrypt(self, L, R):       *# 分组后的解密*          for i in range(self.encryptTime):              L, R = self.\_\_Inv\_Feistel\_block(L, R, self.subseed[self.encryptTime - 1 - i])          return L, R        def \_\_setString2bit(self, s):       *# 分组后的字符串转换成比特数*          ans = 0          for ch in s:              ans = (ans << 8) + ord(ch)          return ans        def \_\_setBit2string(self, n):       *# 分组后的比特转换成字符串*          lst = []          for \_ in range(self.charSetSize):              ch = chr(n & 0xff)              lst.append(ch)              n = n >> 8          lst = lst[::-1]          return  "".join(lst)        def encrypt(self, s):          rem = len(s) % self.charSetSize          if rem != 0:              s = s + (self.charSetSize - rem) \* " "               *# padding*          lst = []          for i in range(int(len(s)) // self.charSetSize):        *# 计算字母所对应的ascii码并分组*              l = i \* self.charSetSize              substr = s[l: l + self.charSetSize]              if self.debug == True:                  print("子串为: {:s}, 转换为ascii码后对应的十六进制数字: 0x{:x}".format(substr, self.\_\_setString2bit(substr)))              lst.append(self.\_\_setString2bit(substr))          ans = []          bitLen = 8 \* self.charSetSize                           *# 对每组进行加密*          for bits in lst:              modulo = 2 \*\* (bitLen // 2)              R = bits % modulo              L = (bits - R) >> (bitLen // 2)              Ln, Rn = self.\_\_setEncrypt(L, R)              val = (Ln << (bitLen // 2)) + Rn              ans.append(val)              print("明文数字: 0x{:x}, 密文数字: 0x{:x}".format(bits, val))          ret = 0          for bits in ans:              ret = (ret << bitLen) + bits          return ret                              *# 最后返回一个极大的数字，看成二进制即可*        def decrypt(self, bits: int, n: int):       *# n为字符个数，bits为比特*          ans = []          if n % self.charSetSize != 0:           *# 填充*              n = n + (self.charSetSize - n % self.charSetSize)          for i in range(n // self.charSetSize):  *# 运用位运算从整数提取出分组后的比特信息*              val = bits & (2 \*\* (self.charSetSize \* 8) - 1)              ans.append(val)              bits = bits >> (self.charSetSize \* 8)          ans = ans[ : :-1]                       *# 由于ans最后面的bits存的是最开始的字串，故需要倒转*          s = ""          bitLen = 8 \* self.charSetSize           *# 计算比特长度*          for idx, val in enumerate(ans):              modulo = 2 \*\* (bitLen // 2)              R = val % modulo              L = (val - R) >> (bitLen // 2)      *# 计算L, R*              L0, R0 = self.\_\_setDecrypt(L, R)              decryptVal = (L0 << (bitLen // 2)) + R0              s += self.\_\_setBit2string(decryptVal)              print("密文: 0x{:x}；明文: 0x{:x}；子串: {:s}".format(val, decryptVal, self.\_\_setBit2string(decryptVal)))          return s.strip(" ")                     *# 去除空格*    raw\_txt = "CQUINFORMATIONSECURITYEXP" *# cqu information security exp*  def f(x, y):      return ((0x996177f3 ^ x) \* (y & 0x3312ff78))  model = Feistel(seed = 2, encryptTime = 16, charSetSize = 8, function = f, debugging = True)  val = model.encrypt(raw\_txt)  s = model.decrypt(val, len(raw\_txt))  print("最终解密结果：",s) | | | | | | |
| 五、实验过程中遇到的问题及解决情况  问题主要在Feistel网络中，因为涉及到很多位运算。主要问题是没有注意F函数的值域，导致经过F函数的运算，每组的取值超出了应有的取值（如：分组本身是64个bit表示的，最后变成65甚至66位表示，导致最终的解密过程的分组操作不对，导致最后的乱码）。      如上图，分组为7\*8 = 56bits，由于没有取余操作，在解密过程中，密文分组不对，导致错误。下面是修改方案：    正确结果如下： | | | | | | |
| 六、实验结果及分析和（或）源程序调试过程  （一）、频度分析法  使用单个字母的频度统计后，得到的密文如下：    注意到YEITERDAY极有可能是YESTERDAY，我们选择让I与S对换。结果如下：    注意到开头的字母长度为2和3，往常见词进行考虑，MAS很容易想到WAS，根据这个想法，OT应该是IT，因此，我们选择M与W对换，O与I对换，结果如下：    可以看到词句大概成型。注意到WITU单词，应该是WITH，U与H对换，结果如下：    现在很多单词基本成型，现在注意到单词REPRESEUTATICES，现在让U和N对换，C与V对换。结果如下：    注意到PULITIOAL，应该为POLITICAL之误，O与U对换，U与C对换，结果如下：    注意到FEEN，不难猜出是BEEN，选择FB对换。结果如下：    注意到BFT，不难猜出是常见介词BUT。F与U对换。    注意到INGORMAL，猜测可能是INFORMAL或IMNORMAL，考虑到N换成M后CONTACTS会变化，因此优先换F和G。    可以看到已经是个通顺的英文句子了。现在退出交互式程序。最终结果如下。    其中映射关系为密文映射到明文。稍微检查可以发现是正确的。  （二）、Feistel网络  选择分组为8 \* 8 = 64bits， 密钥长度为128， 循环次数为16，子密钥生成算法为马特赛特旋转演算法，对于文本的填充方案为填充空格，最终结果如下： | | | | | | |