**《信息安全概论》实验报告**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **姓名** | | 谭星 | | **年级** | | 2019级 |
| **学号** | | 20191584 | | **专业、班级** | | 计算机科学与技术卓越01班 |
| **实验名称** | 实验一 加解密算法的实现 | | | | | |
| **实验时间** | **2022.3.22** | | **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RSA加解密原理 | | | | | | |
| 二、实验项目内容  1. 使用频度分析法解密以下文本，并给出替换表：  UZ QSO VUOHXMOPV GPOZPEVSG ZWSZ OPFPESX UDBMETSX AIZ VUEPHZ HMDZSHZO WSFP APPD TSVP QUZW YMXUZUHSX EPYEPOPDZSZUFPO MB ZWP FUPZ HMDJ UD TMOHMQ  2. 编程实现RSA加密解密以下文本：  CQUINFORMATIONSECURITYEXP1 | | | | | | |
| 三、实验设计  1. 频度分析原理­­­  对于任何一种书面语言而言，不同的字母或字母组合出现的频率各不相同。如果以这种语言书写足够长的文本，都呈现出大致相同的特征字母分布规律，如下图所示：  表格  描述已自动生成  在上表中，不少字母出现的概率近乎相等。为了应用方便，常将英文字母表按字母出现的概率大小分类，分类情况见下表：  --------------------------  极高频 E  次高频 T A O I N S H R  中等频 D L  低频 C U M W F G Y P B  甚低频 V K J X Q Z  --------------------------  语言的单字母统计特性没有反映出英文双字母和多字母的特征，在双字母中统计出概率最大的30对字母按概率大小排列为：  th he in er an re ed on es st en at to nt ha nd ou ea ng as or ti is et it ar te se hi of  类似的，我们还可以考察英文课文中三字母出现的频率。按Beker在1982年统计的结果(样本总数100 360)得到概率最大的20组三字母按概率大小排列为： the    ing    and    her    ere    ent    tha    nth    was    eth    for    dth    hat    she    ionhis    sth    ers    ver  特别，the出现的频率几乎为ing的3倍。  2. RSA  1977年，三位数学家Rivest、Shamir 和 Adleman 设计了一种算法，可以实现非对称加密。这种算法用他们三个人的名字命名，叫做RSA算法。从那时直到现在，RSA算法一直是最广为使用的"非对称加密算法"。毫不夸张地说，只要有计算机网络的地方，就有RSA算法。  这种算法非常可靠，密钥越长，它就越难破解。根据已经披露的文献，目前被破解的最长RSA密钥是768个二进制位。也就是说，长度超过768位的密钥，还无法破解（至少没人公开宣布）。因此可以认为，1024位的RSA密钥基本安全，2048位的密钥极其安全。RSA流程如下：  第一步，随机选择两个不相等的质数p和q。  第二步，计算p和q的乘积n。  第三步，计算n的欧拉函数φ(n)。  第四步，选择一个整数e，需满足1< e <φ(n)，且e与φ(n) 互质。  第五步，计算e和φ(n)的模反元素d。  第六步，将n和e封装成公钥，n和d封装成私钥。  **加密**  m^e ≡ c (mod n) ，信息m加密成C  **解密**  c^d ≡ m (mod n) | | | | | | |
| 四、实验过程或算法  1. 频度分析解密  先绘制标准字母频率统计条形图和给定密文的字母频率统计条形图，结果如下：  C:\Users\PH\AppData\Roaming\Tencent\Users\1447540812\QQ\WinTemp\RichOle\N[Q1~)Z5@1_F({91YGMIA3B.png  观察可知，密文中的字母P和Z出现频率较高，极有可能对应于明文中的字母E和T，故暂时将P译为E，Z译为T。之后的字母统计频率两图较相似，可初步作以下判断：(S, U, O, M, H, D, E) 可能对应于次高频字母集合 (A, O, I, N, S, H, R)；后面的以此类推。故第一次尝试直接将各字母按照上图的频率排序依次对应，输出结果为：    结果中出现了TUE和TUAT两种字母组合，可知T和E的对应极可能正确。据此还可联想到单词THE和THAT，故尝试将密文中的W对应明文中的H（原对应于U），输出结果为：    此次修改后使THE和THAT正确，观察句子开头为OT，仅两个字母，且T已经修改正确，因此着手修改O。满足可放在句首、在双字母统计概率较高、第二个字母为T这三个条件的单词只能是IT，故尝试将密文中的U对应明文中的I（原对应于O），输出结果为：    目前明文中已对应的字母有T, H, E, I，针对新出现的PITH，因为P和W同属于低频字母，所以猜测单词为WITH，故尝试将密文中的Q对应明文中的W（原对应于P），输出结果为：    句首为IT，且第二个单词目前为WAN，W和A均已对应，所以不难猜测出正确单词为WAS，故尝试将密文中的O对应明文中的S（原对应于N），输出结果为：    句中的YESTEULAY和单词YESTERDAY相近，故尝试将密文中的E对应明文中的R（原对应于U），将密文中的V对应明文中的D（原对应于L）输出结果为：    将第三个单词DISNLOSED修改为DISCLOSED，故尝试将密文中的H对应明文中的C（原对应于N），输出结果为：    将HANE修改为HAVE，故尝试将密文中的F对应明文中的V（原对应于N），输出结果为：    U和N同属于次高频，将COUTACTS修改为CONTACTS，故尝试将密文中的D对应明文中的N（原对应于U），输出结果为：    观察单词MEEN，与HAVE相连的一般为BEEN，故尝试将密文中的A对应明文中的B（原对应于M），输出结果为：    长单词REMRESENTATIVES与REPRESENTATIVES相近，故尝试将密文中的Y对应明文中的P（原对应于M），输出结果为：    句末的FOSCOW前为IN，所以FOSCOW极可能是个地名，猜测为MOSCOW，故尝试将密文中的T对应明文中的M（原对应于F），输出结果为：    目前句子大体意思已经清晰，句中的BGT前后连接两个形容词，猜测为连词BUT，故尝试将密文中的I对应明文中的U（原对应于G），输出结果为：    至此，整个密文已解密完成。  2． RSA加解密  依赖库gmpy2中针对大数的操作可较快完成RSA算法的计算需求。根据输入的密钥长度来随机生成两个对应长度的大素数，借助is\_prime()、gcd()、invert()、powmod()等函数计算相应的公钥和私钥，最后对转换为数字的字符文本进行加密解密。  其中，在随机生成指定bit位数的大素数时，使用randrange() 函数指定数字的上下限。因为下限2 \*\* (num\_bits - 1) + 1 必定为奇数，且素数除2以外也必定为奇数，故设置步长为2，避免了一些合数的产生。 | | | | | | |
| 五、实验过程中遇到的问题及解决情况  1. 在频度分析法解密时，猜测的单词错误，导致解密过程无法继续。在此情况下我选择重新分析，多此尝试后成功解密。  2. 在RSA计算密钥时，涉及到对大数的素性测试、求最大公约数、求模逆元等操作，各种高等算法所涉及的数学知识理解起来比较费劲，最后查阅资料时发现了gmpy2库，可用于大数之间的数学运算。  3. 在RSA加解密时，起初我不知道如何将字符文本转换为数字进行加密。第一次尝试将字符串中各字符对应的ASCII码连接在一起形成新数字，再进行操作。但在解密时得到的数字串可能有多种ASCII码的拆分方式，即形成多种字符串，不便于操作。因此最终将文本先转换为字节序列，加密时先将字节序列转换为整数再运算，解密时将计算的整数结果先转换为字节序列，再转换为字符串，这样就可以恢复明文。 | | | | | | |
| 六、实验结果及分析和（或）源程序调试过程  1. 频度分析法解密  最终解密后的明文及对应的替换表均如下图所示：    2. RSA加解密  指定密钥长度为1024bit，所得中间参数及加解密结果如下，从两次的结果来看，字符文本转换为数字的结果和解密后的结果是不变的，但密钥和密文是随机的。      3. 源代码   1. **import** matplotlib.pyplot as plt 2. **import** numpy as np 3. # 绘制标准字母统计频率条形图 4. # 字母及其统计频率 5. letter = ['E', 'T', 'A', 'O', 'N', 'I', 'S', 'R', 'H', 'L', 'D', 'C', 'U', 'P', 'F', 'M', 'W', 'Y', 'B', 'G', 'V', 'K', 6. 'Q', 'X', 'J', 'Z'] 7. f\_standard = [0.123, 0.096, 0.081, 0.079, 0.072, 0.072, 0.066, 0.06, 0.051, 0.04, 0.037, 0.032, 0.031, 0.023, 0.023, 0.023, 0.02, 8. 0.019, 0.016, 0.016, 0.009, 0.005, 0.002, 0.002, 0.001, 0.001] 9. z = np.linspace(0, 26, 26) 10. plt.xlabel('letter') 11. plt.ylabel('frequency') 12. plt.bar(x=z, height=f\_standard, width=0.8) 13. plt.xticks(z, letter) 14. plt.title('Standard Letter Frequency') 16. # 绘制给定密文的字母统计频率图 17. str = 'UZ QSO VUOHXMOPV GPOZPEVSG ZWSZ OPFPESX UDBMETSX AIZ VUEPHZ HMDZSHZO WSFP APPD TSVP QUZW YMXUZUHSX EPYEPOPDZSZUFPO MB ZWP FUPZ HMDJ UD TMOHMQ' 18. alphabet = ['A', 'B', 'C', 'D', 'E', 'F', 'G', 'H', 'I', 'J', 'K', 'L', 'M', 'N', 'O', 'P', 'Q', 'R', 'S', 'T', 'U', 19. 'V', 'W', 'X', 'Y', 'Z'] 20. length = len("".join(str.split())) 21. f\_ciphertext = [] # 密文字母频率 22. **for** i **in** range(26): 23. f\_ciphertext.append(str.count(alphabet[i]) / length) 24. temp = dict(zip(alphabet, f\_ciphertext)) 25. alphabet, f\_ciphertext = zip(\*sorted(temp.items(), key=**lambda** x: x[1], reverse=True)) 26. plt.figure() 27. plt.xlabel('letter') 28. plt.ylabel('frequency') 29. plt.bar(x=z, height=f\_ciphertext, width=0.8) 30. plt.xticks(z, alphabet) 31. plt.title('Ciphertext Letter Frequency') 32. alphabet = list(alphabet) 34. # 调整替换表的映射关系 35. **def** revise\_table(a, b): 36. idx1 = alphabet.index(a) 37. idx2 = alphabet.index(b) 38. alphabet[idx1] = b 39. alphabet[idx2] = a 41. revise\_table('W','E') # W和E交换 42. revise\_table('M','U') # M和U交换 43. revise\_table('Q','B') # Q和B交换 44. revise\_table('H','O') # H和O交换 45. revise\_table('E','D') # E和S交换 46. revise\_table('V','X') # V和X交换 47. revise\_table('H','F') # H和F交换 48. revise\_table('F','J') # F和J交换 49. revise\_table('D','J') # D和J交换 50. revise\_table('A','Y') # A和Y交换 51. revise\_table('Y','B') # Y和B交换 52. revise\_table('T','B') # T和B交换 53. revise\_table('I','J') # I和J交换 55. plaintext = "" 56. replacetable = dict(zip(alphabet, letter)) 57. # 按照替换表进行解密 58. **for** i **in** range(len(str)): 59. **if** str[i] == ' ': 60. plaintext = plaintext + ' ' 61. **continue** 62. plaintext = plaintext + replacetable[str[i]] 64. cnt = 0 65. # 输出替换表 66. **print**("替换表对应关系如下（密文，明文）：") 67. **for** i **in** sorted(replacetable): 68. cnt = cnt + 1 69. **if** cnt < 13: 70. end\_char = ' ' 71. **else**: 72. end\_char = '\n' 73. cnt = 0 74. **print**((i, replacetable[i]), end = end\_char) 75. # 输出明文 76. **print**("对应的明文为：") 77. **print**(plaintext) 78. plt.show() 79. **import** random 80. **import** gmpy2 82. # 随机生成两个不相等的大素数p和q 83. **def** get\_p\_q(num\_bits): 84. **while** (1): 85. p = random.randrange(2 \*\* (num\_bits - 1) + 1, 2 \*\* num\_bits, 2) # 随机生成指定bit位数的数字 86. **if** gmpy2.is\_prime(p): 87. **break** 88. **while** (1): 89. q = random.randrange(2 \*\* (num\_bits - 1) + 1, 2 \*\* num\_bits, 2) # 随机生成指定bit位数的数字 90. **if** gmpy2.is\_prime(q) & (p != q): 91. **break** 92. **print**('p : %i'%p) 93. **print**('q : %i'%q) 94. **return** p, q 96. # 计算n 97. **def** get\_n(p, q): 98. n = p \* q 99. **print**('n : %i' %n) 100. **return** n 102. # 计算n的欧拉函数值 103. **def** get\_euler(p, q): 104. **return** (p - 1) \* (q - 1) 106. # 计算与M互质的e 107. **def** get\_e(M): 108. **while** (1): 109. e = random.randint(1, M) 110. **if** gmpy2.gcd(e, M) == 1: 111. **print**('e : %i' %e) 112. **return** e 114. # 计算e和M的模反元素d 115. **def** get\_d(e, M): 116. d = gmpy2.invert(e, M) 117. **print**('d : %i' %d) 118. **return** d 120. **def** encrypt(text, e, n): 121. # 将字符串转换为字节序列后，再转为整数进行加密 122. num = int.from\_bytes(bytes(text, encoding='utf-8'), 'big') 123. **print**('num : %i' %num) 124. num\_encrypt = gmpy2.powmod(num, e, n) 125. **return** num\_encrypt 127. **def** decrypt(ciphertext, d, n): 128. num\_decrypt = gmpy2.powmod(ciphertext, d, n) 129. # 将解密后的数字明文先转换为字节序列，再转换为字符串 130. **return** str(int(num\_decrypt).to\_bytes((num\_decrypt.bit\_length() + 7) // 8, 'big'), encoding='utf-8') 132. **def** RSA(text, num\_bits): 133. p, q = get\_p\_q(num\_bits) 134. n = get\_n(p, q) 135. M = get\_euler(p, q) 136. e = get\_e(M) 137. d = get\_d(e, M) 138. ciphertext = encrypt(text, e, n) 139. **print**('密文（16进制）：%#x' % ciphertext) 140. plaintext = decrypt(ciphertext, d, n) 141. **print**('明文：%s' %plaintext) 143. **if** \_\_name\_\_ == "\_\_main\_\_": 144. num\_bits = input('请输入密钥长度：') 145. text = 'CQUINFORMATIONSECURITYEXP1' 146. RSA(text, int(num\_bits)//2) | | | | | | |