**《信息安全概论》实验报告**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **姓名** | | *李燕琴* | **年级** | | | *2019级* |
| **学号** | | *20195633* | **专业、班级** | | | *计算机科学与技术（卓越）02班* |
| **实验名称** | **加解密算法的实现** | | | | | |
| **实验时间** | **2022-03-22** | | | **实验地点** | **寝室** | |
| **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文本，并写出替换表  2. 编程实现RSA加密/解密 | | | | | | |
| 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算法**  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：  Step1, 随机选择两个不相等的质数 和 。  Step2，计算 和 的乘积 。  Step3，计算 的欧拉函数 。  Step4，选择一个整数 ，需满足 ，且 与 互质。  Step5，计算 和 的模反元素 。  Step6，将 和 封装成公钥， 和 封装成私钥。  **加密**   ，信息m加密成C  **解密**  ，密文c解密为m  **原理参考**  [RSA算法原理 · 看云 (kancloud.cn)](https://www.kancloud.cn/kancloud/rsa_algorithm/48493) | | | | | | |
| 四、实验过程或算法  **1、频度分析解密**   1. **import** string 2. **from** collections **import** Counter 4. ''''' 参考词频分析 ''' 5. # 单个字母 6. single = ['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'] 7. # 两个字母 8. double = ['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'] 9. # 三个字母 10. triple = ['the', 'ing', 'and', 'her', 'ere', 'ent', 'tha', 'nth', 'was', 'eth', 'for', 'dth', 'hat', 'she', 'ion', 'his', 'sth', 'ers', 'ver'] 11. # 单个字母组成的词 12. s\_word = ['a'] 13. # 两个字母组成的词 14. d\_word = ['he', 'in', 'an', 'on', 'at', 'to', 'as', 'or', 'is', 'it', 'hi', 'of'] 15. # 三个字母组成的词 16. t\_word = ['the', 'her', 'was', 'for', 'hat', 'she', 'his'] 17. # 字母表 18. alpha = list(string.ascii\_lowercase) 20. ''''' 加密字符串 ''' 21. s = "UZ QSO VUOHXMOPV GPOZPEVSG ZWSZ OPFPESX UDBMETSX AIZ VUEPHZ HMDZSHZO WSFP APPD TSVP QUZW YMXUZUHSX EPYEPOPDZSZUFPO MB ZWP FUPZ HMDJ UD TMOHMQ" 22. # 分词 23. words = s.split(" ") 24. # 获取词组 25. single\_word = [] 26. double\_word = [] 27. triple\_word = [] 28. other\_word = [] 29. **for** w **in** words: 30. **if** len(w)==1: 31. single\_word.append(w) 32. **elif** len(w)==2: 33. double\_word.append(w) 34. **elif** len(w)==3: 35. triple\_word.append(w) 36. **else**: 37. other\_word.append(w) 38. # 获取字母词频 39. cnt\_single = Counter() 40. cnt\_double = Counter() 41. cnt\_triple = Counter() 42. **for** w **in** words: 43. **for** i **in** w: 44. cnt\_single[i]+=1 45. l = 0 46. **for** r **in** range(2,len(w)): 47. cnt\_double[w[l:r]]+=1 48. l+=1 49. l = 0 50. **for** r **in** range(3,len(w)): 51. cnt\_triple[w[l:r]]+=1 52. l+=1 54. ''''' 词频分析和词组分析 ''' 55. **print**("="\*10+"词频上："+"="\*10) 56. **print**(single) 57. **print**(cnt\_single,'\n') 59. **print**(double) 60. **print**(cnt\_double,'\n') 62. **print**(triple) 63. **print**(cnt\_triple,'\n') 65. **print**("="\*10+"词组上："+"="\*10) 66. **print**(s\_word) 67. **print**(single\_word,'\n') 69. **print**(d\_word) 70. **print**(double\_word,'\n') 72. **print**(t\_word) 73. **print**(triple\_word,'\n') 75. **print**(other\_word,'\n')  78. ''''' 分析解密 ''' 79. # 映射关系 80. map = { 81. 'U':'i', 82. 'Z':'t', 83. 'D':'n', 85. 'Q':'w', 86. 'S':'a', 87. 'O':'s', 89. 'W':'h', 90. 'P':'e', 92. 'M':'o', 93. 'B':'f', 95. 'F':'v', 96. 'A':'b', 98. 'I':'u', 99. 'E':'r', 100. 'X':'l', 101. 'T':'m', 102. 'V':'d', 103. 'H':'c', 104. 'Y':'p', 105. 'G':'y', 106. 'J':'g' 107. } 108. # 结果输出 109. res = "" 110. **for** i **in** s: 111. **if** map.get(i)==None: 112. res += i 113. **else**: 114. res += map[i] 115. **print**(s) 116. **print**(res)   **2、RSA算法**   1. # -\*- coding: utf-8 -\*- 2. **import** time 3. **from** logging **import** error 5. # 欧几里得算法：求两个数字的最大公约数 6. **def** gcd(a, b): 7. **return** gcd(b,a%b) **if** b **else** a 9. # 扩展欧几里的算法: 计算 ax + by = 1中的x与y的整数解（a与b互质） 10. **def** ext\_gcd(a, b): 11. **if** b == 0: 12. **return** 1,0 13. **else**: 14. x1, y1 = ext\_gcd(b, a % b) 15. **return** y1, x1 - a // b \* y1 17. # 十进制转换36进制 18. **def** ten\_2\_r(d,r=36): 19. res = '' 20. **while** d!=0: 21. num = d%r 22. **if** 0<=num<=9: 23. res += str(num) 24. **elif** 10<=num<36: 25. res += chr(num-10+ord('A')) 26. **else**: 27. error("不符合36进制规则") 28. d = d//r 29. **return** res[::-1] 31. # 36进制转换十进制 32. **def** r\_2\_ten(s,r=36): 33. res = 0 34. **for** i **in** s: 35. **if** i.isdigit(): 36. num = int(i) 37. **elif** i.isupper(): 38. num = ord(i)-ord('A')+10 39. **elif** i.islower(): 40. num = ord(i)-ord('a')+10 41. **else**: 42. error("不符合36进制规则") 43. res = r \* res + num 44. **return** res 46. # 快速幂 47. **def** quick\_pow(base, exponet, n): 48. ans = 1 49. **while** exponet: 50. **if** exponet&1: 51. ans = ans \* base % n 52. base = base \* base % n 53. exponet = exponet >> 1 54. **return** ans 56. # 生成公钥私钥 57. **def** gen\_rsa\_key(p, q): 58. # step1 计算p\*q，其中p，q为超大质数 59. n = p \* q 61. # step2 计算与n互质的整数个数 欧拉函数 62. ''''' 63. eluer(p\*q) = eluer(p)\*eluer(q) 64. p为质数时, eluer(p) = p-1 65. ''' 66. eluer = (p - 1) \* (q - 1) 68. # step3 选取e, 一般选取65537 69. e = 65537 71. # step4 计算模反元素d，扩展欧几里得算法 72. x, y = ext\_gcd(e, eluer) 73. # 计算出的x不能是负数，如果是负数，说明p、q、e选取失败，不过可以把x加上eluar，使x为正数，才能计算。 74. **if** x < 0: 75. x = x + eluer 77. # step5 返回公钥，私钥 78. **return** (n, e), (n, x) 80. # 加密，m加密为c 81. **def** encrypt(m, pubkey): 82. n = pubkey[0] 83. e = pubkey[1] 84. # 需要被加密的信息转化成数字，m视为字母和数字的组合，即36进制 85. # 注意：长度小于秘钥n的长度，如果信息长度大于n的长度，那么分段进行加密，分段解密即可 86. m = r\_2\_ten(m,36) 87. c = quick\_pow(m,e,n) 88. **return** c 90. # 解密，c解密为m 91. **def** decrypt(c, selfkey): 92. n = selfkey[0] 93. d = selfkey[1] 94. m = quick\_pow(c, d, n) 95. m = ten\_2\_r(m,36) 96. **return** m 98. **if** \_\_name\_\_ == "\_\_main\_\_": 99. start\_time = time.time() 100. # 公钥私钥中用到的两个大质数p,q，都是1024位 101. p = 106697219132480173106064317148705638676529121742557567770857687729397446898790451577487723991083173010242416863238099716044775658681981821407922722052778958942891831033512463262741053961681512908218003840408526915629689432111480588966800949428079015682624591636010678691927285321708935076221951173426894836169 102. q = 144819424465842307806353672547344125290716753535239658417883828941232509622838692761917211806963011168822281666033695157426515864265527046213326145174398018859056439431422867957079149967592078894410082695714160599647180947207504108618794637872261572262805565517756922288320779308895819726074229154002310375209 103. # 生成公钥私钥 104. pubkey, selfkey = gen\_rsa\_key(p, q) 105. m = 'CQUINFORMATIONSECURITYEXP1' 106. **print**("原文: %s" % m) 107. # 加密 108. c = encrypt(m, pubkey) 109. **print**("密文: %s" % c) 110. # 解密 111. d = decrypt(c, selfkey) 112. **print**("明文: %s" % d) 113. **print**('用时: %.6fs'%(time.time()-start\_time))   算法处理如下：   * 原文处理：通过将原文视为36进制，转化为十进制进行幂运算；解密后，再将大数字转为36进制显示即可 * 欧拉函数：质数性质 * 模反元素：扩展欧几里得算法 * 大数幂运算：快速幂算法   **3、参考代码**  [Python 25行代码实现的RSA算法详解\_python\_脚本之家 (jb51.net)](https://www.jb51.net/article/138018.htm) | | | | | | |
| 五、实验过程中遇到的问题及解决情况  1. 频度分析解密上  最开始从词频方面，基本不好下手；之后想到了将一些短字符的词语猜测出来，如长度小于3的词语。再根据已经猜出来的字符，进一步猜测长度为4的词语。这个时候，一般只剩下几个字符难以确定了，于是就采取这次字符遍历26个字母，如果能组成熟悉的单词，则替换字符。由此就可以判断出所有的字符了。  2、rsa加密解密算法上  ①原文字符串的处理：  因为之后会涉及幂运算，所以需要将字符串转为数字写。本实验采用的方法，因为给定的原文比较短，将其视为字母和数字的组合，通过36进制转10进制，达到转为数字的目的。  但是上述处理方法，存在一个弊端，即36进制表示的数字非常大，转为10进制难以表示较长的字符串。在群里询问各方道友后，得知另一种方法，即将每个字符用其asscii码表示，然后加空格或特殊字符或固定字长（有点像计算机网络的数字链路层编码方式了，hhh）等方法，以转为数字且能区分每个字符的位置。但是，由于本人有点懒，故没有实现。  ②大数幂运算  这里的幂运算非常大，如果做完幂之后再取模，难以提供足够大的内存空间，且计算时间也很长（乘法很耗时），比较可行的算法是快速幂（很巧的是，前几周刚刚学了这个算法）。极大减少乘法的次数，且保证需要存储的结果不会很大，以满足时间上和空间上的限制。 | | | | | | |
| 六、实验结果及分析和（或）源程序调试过程  1、频度分析  根据五中提到的方法，这里的词频和词组上，统计如下：    结果映射表如下：  map = {      'U':'i',      'Z':'t',      'D':'n',        'Q':'w',      'S':'a',      'O':'s',      'W':'h',      'P':'e',        'M':'o',      'B':'f',      'F':'v',      'A':'b',      'I':'u',      'E':'r',      'X':'l',      'T':'m',      'V':'d',      'H':'c',      'Y':'p',      'G':'y',      'J':'g'  }  最终结果如下：    2、rsa算法  运行时长只需0.03s | | | | | | |