|  |
| --- |
| **Github（或者Coding）账号：phitooo** |
| **个人博客关于密码学实验的链接：** |
| **实验题目（中文）：实验一** |
| **实验摘要（中文）：**   1. 当流密码的一次性密钥被使用多次时就不再具有语义安全性，可以结合多个密文破解出明文。 2. 破解类维吉尼亚密码给出一段密文和加密程序，告知明文只包含大小写字母、标点符号和空格，没有数字和其他字符。 3. XOR猜测密钥长度后，将密文分成相应长度块，计算相邻块间的汉明距离，具有最小归一化汉明距离的块长度很可能是真实密钥长度。用字母频率分析的方法爆破当前位置的密钥。 |
| **题目描述（清楚描述题目中文，写出自己的理解，请勿复制原题目）**  任务一  **Many Time Pad**  让我们来看看流密码密钥被多次使用时会出现什么问题。下面是 11 个十六进制编码的密码文本，它们是用同一个流密码密钥对 11 个明文进行加密的结果。您的目标是解密最后一个密码文本，并提交其中的密文作为解决方案。  提示：将密文进行 XOR，并考虑空格与 [a-zA-Z] 字符进行 XOR 时会发生什么。  任务二    **任务三**    **任务四**  监控系统网络服务器的一个漏洞泄露了 管理员账户密码的 SHA1 哈希值。 密码的哈希值为67ae1a64661ac8b4494666f58c4822408dd0a3e4  此外，登录终端的键盘上有输入密码的明显痕迹，因为登录成功后软件的导航只能通过箭头键完成。  密码是什么？ |
| **过程（包括背景，原理：必要的公式，图表；步骤，如有必要画出流程图，给出主要实现步骤代码）**  **任务一**  空格space的ASCII码为0010 0000，大写字母A～Z的ASCII码为01000001～01011010，小写字母a～z的ASCII码为 01100001～01111010。因此可以用space和字母做XOR操作，对字母进行大小写切换； 而两个字母做XOR操作，结果将不在字母范围内。  又由于将两个密文做XOR操作相当于将两个密文对应的明文做XOR操作，如果结果中某个位置出现字母，则说明这两个明文的其中一个在该位置可能为空格。故对11个密文两两做XOR操作，然后通过结果判断不同明文中可能存在空格的位置，然后将对应位置上的密文和space做XOR操作，就可得到对应位置的密钥信息，当获取到足够多的密钥信息后，即可对目标密文进行解密。   1. NUM\_CIPHER = len(ciphertexts) 2. THRESHOLD\_VALUE = 7  # 认为该位置是空格的阈值 3. ​ 4. ​ 5. def strxor(a, b): 6. if len(a) > len(b): 7. return "".join([chr(ord(x) ^ ord(y)) for (x, y) in zip(a[:len(b)], b)]) 8. else: 9. return "".join([chr(ord(x) ^ ord(y)) for (x, y) in zip(a, b[:len(a)])]) 10. ​ 11. ​ 12. def letter\_position(s): 13. position = [] 14. for idx in range(len(s)): 15. if ('A' <= s[idx] <= 'Z') or ('a' <= s[idx] <= 'z') or s[idx] == chr(0): 16. position.append(idx) 17. return position 18. ​ 19. ​ 20. def find\_space(cipher): 21. space\_position = {} 22. space\_possible = {} 23. *# 两两异或* 24. for cipher\_idx\_1 in range(NUM\_CIPHER): 25. space\_xor = [] 26. c = ''.join( 27. [chr(int(d, 16)) for d in [cipher[cipher\_idx\_1][i:i + 2] for i in range(0, len(cipher[cipher\_idx\_1]), 2)]]) 28. for cipher\_idx\_2 in range(NUM\_CIPHER): 29. *# 16进制字符串转ascii码* 30. e = ''.join([chr(int(d, 16)) for d in 31. [cipher[cipher\_idx\_2][i:i + 2] for i in range(0, len(cipher[cipher\_idx\_2]), 2)]]) 32. plain\_xor = strxor(c, e) 33. if cipher\_idx\_2 != cipher\_idx\_1: 34. *# 记录可能的空格位置* 35. space\_xor.append(letter\_position(plain\_xor)) 36. space\_possible[cipher\_idx\_1] = space\_xor 37. ​ 38. for cipher\_idx\_1 in range(NUM\_CIPHER): 39. spa = [] 40. for position in range(400): 41. count = 0 42. for cipher\_idx\_2 in range(NUM\_CIPHER - 1): 43. if position in space\_possible[cipher\_idx\_1][cipher\_idx\_2]: 44. count += 1 45. if count > THRESHOLD\_VALUE: 46. spa.append(position) 47. space\_position[cipher\_idx\_1] = spa 48. return space\_position 49. ​ 50. ​ 51. *# 破解出密钥key* 52. def calculate\_key(cipher): 53. key = [0] \* 200 54. space = find\_space(cipher) 55. for cipher\_idx\_1 in range(NUM\_CIPHER): 56. for position in range(len(space[cipher\_idx\_1])): 57. idx = space[cipher\_idx\_1][position] \* 2 58. a = cipher[cipher\_idx\_1][idx] + cipher[cipher\_idx\_1][idx + 1] 59. key[space[cipher\_idx\_1][position]] = int(a, 16) ^ ord(' ') 60. key\_str = "" 61. for k in key: 62. key\_str += chr(k) 63. return key\_str 64. ​ 65. ​ 66. result = "" 67. key = calculate\_key(ciphertexts) 68. key\_hex = ''.join([hex(ord(c)).replace('0x', '') for c in key]) 69. ​ 70. f = ''.join([chr(int(d, 16)) for d in [ciphertexts[10][i:i + 2] for i in range(0, len(ciphertexts[10]), 2)]]) 71. for letter in strxor(f, key): 72. if ' ' <= letter <= '~ ': 73. result += letter 74. print(result)   结果  key\_hex = "66396e89c9dbd8cb9874352acd6395102eafce78aa7fed28a006bc98d29c5b69b0339a19f8aa401a9c6d708f80c066c763fef0123148cdd8e82d05ba98777335daefcecd59c433a6b268b60bf4ef03c9a61100bb09a3161edc704a3"  **任务二**  本质上是密钥重复使用问题，假设密钥长度为klen，则相隔n\*klen的密文使用同一密钥进行加密。所以首先猜测密钥长度klen，再将密文分割成klen组，对每一组遍历密钥的所有可能取值，并与密文异或，找到并记录异或结果全为明文字符的可能密钥值。   1. *# 找出可以与subarr中的值异或成可显示字符的密钥值* 2. *# subarr是在Vigenere密码中用同一密钥加密的密文* 3. def findkey(sub\_arr): 4. required\_chars = [chr(32), chr(33), chr(34), chr(44), chr(45), chr(46), chr(58), chr(63), chr(95)]  *# 标点和空格* 5. for x in range(65, 91): 6. required\_chars.append(chr(x))  *# 大写字母* 7. for x in range(97, 123): 8. required\_chars.append(chr(x))  *# 小写字母* 9. *# print(required\_chars)* 10. all\_keys = [] 11. res\_keys = [] 12. for x in range(0x00, 0xff + 1):  *# 枚举当前字节处所有可能的密钥* 13. all\_keys.append(x) 14. res\_keys.append(x) 15. for i in all\_keys: 16. for s in sub\_arr: 17. if chr(s ^ i) not in required\_chars: 18. res\_keys.remove(i) 19. break 20. return res\_keys 21. ​ 22. ​ 23. ciphertext = ​ 35. arr = []  *# 将16进制密文划分成8位2进制数* 36. for x in range(0, len(ciphertext), 2): 37. arr.append(int(ciphertext[x:2 + x], 16)) 38. ​ 39. for key\_length in range(1, 14):  *# 密钥长度1~13* 40. for class\_number in range(0, key\_length):  *# 密文划分成key\_length类，每一类和同一密钥值异或* 41. sub\_arr = arr[class\_number::key\_length]  *# 分割出同一组的密文* 42. res\_keys = findkey(sub\_arr) 43. print('key\_length= ', key\_length, 'class\_number= ', class\_number, 'keys= ', res\_keys) 44. if len(res\_keys) > 1:  *# 如果有多个可能的密钥值，查看具体明文情况进行选择* 45. for x in res\_keys: 46. print(x) 47. for s in sub\_arr: 48. print(chr(s ^ x), end='') 49. print("\n")     可以看到只有在密钥长度为7时，找到了可能密钥值，其中第一组有两个可能密钥值，分析异或结果判断密钥是31的概率更高，得到密钥为[186, 31, 145, 178, 83, 205, 62]。  直接用密钥解密，   1. keys = [186, 31, 145, 178, 83, 205, 62] 2. plaintext = "" 3. for i in range(0, len(arr)): 4. plaintext = plaintext + chr(arr[i] ^ keys[i % len(keys)])  *# 解密* 5. print(plaintext)   结果为：  Cryptography is the practice and study of techniques for, among other things, secure communication in the presence of attackers. Cryptography has been used for hundreds, if not thousands, of years, but traditional cryptosystems were designed and evaluated in a fairly ad hoc manner. For example, the Vigenere encryption scheme was thought to be secure for decades after it was invented, but we now know, and this exercise demonstrates, that it can be broken very easily.  **任务三**  这一题可以认为是上一题的加强。猜测密钥长度后，将密文分成相应长度块，计算相邻块间的汉明距离，具有最小归一化汉明距离的块长度很可能是真实密钥长度。用字母频率分析的方法爆破当前位置的密钥。   1. import string 2. import re 3. from operator import itemgetter, attrgetter 4. import base64 5. ​ 6. ​ 7. def English\_Scoring(t): 8. letter\_frequency = { 9. 'a': .08167, 'b': .01492, 'c': .02782, 'd': .04253, 10. 'e': .12702, 'f': .02228, 'g': .02015, 'h': .06094, 11. 'i': .06094, 'j': .00153, 'k': .00772, 'l': .04025, 12. 'm': .02406, 'n': .06749, 'o': .07507, 'p': .01929, 13. 'q': .00095, 'r': .05987, 's': .06327, 't': .09056, 14. 'u': .02758, 'v': .00978, 'w': .02360, 'x': .00150, 15. 'y': .01974, 'z': .00074, ' ': .15000 16. } 17. return sum([letter\_frequency.get(chr(i), 0) for i in t.lower()])  *# 总字母频率* 18. ​ 19. ​ 20. def Single\_XOR(s, single\_character): 21. t = b'' 22. *# s = bytes.fromhex(s)* 23. *# t = the XOR'd result* 24. for i in s: 25. t = t + bytes([i ^ single\_character]) 26. return t 27. ​ 28. ​ 29. def ciphertext\_XOR(s): 30. \_data = [] 31. *# s = bytes.fromhex(s)* 32. *# key = ord (single\_character)* 33. *# ciphertext = b''* 34. for single\_character in range(256): 35. ciphertext = Single\_XOR(s, single\_character) 36. score = English\_Scoring(ciphertext) 37. data = { 38. 'Single character': single\_character, 39. 'ciphertext': ciphertext, 40. 'score': score 41. } 42. \_data.append(data) 43. score = sorted(\_data, key=lambda score: score['score'], reverse=True)[0]  *# 字母频率最大的* 44. return score 45. ​ 46. ​ 47. def Repeating\_key\_XOR(\_cipher, \_key):  *# 破解出明文* 48. message = b'' 49. length = len(\_key) 50. for i in range(0, len(\_cipher)): 51. message = message + bytes([\_cipher[i] ^ \_key[i % length]]) 52. return message 53. ​ 54. ​ 55. *# 计算汉明距离* 56. def hamming\_distance(a, b): 57. distance = 0 58. for i, j in zip(a, b): 59. byte = i ^ j 60. distance = distance + sum(k == '1' for k in bin(byte))  *# 转成二进制* 61. return distance 62. ​ 63. ​ 64. *# 猜测密钥长度* 65. def Get\_the\_keysize(ciphertext): 66. data = [] 67. for keysize in range(2, 41): 68. block = [ciphertext[i:i + keysize] for i in range(0, len(ciphertext), keysize)]  *# 划分成等长的块* 69. distances = [] 70. for i in range(0, len(block) - 1, 1):  *# 逐个计算两个相邻块之间的汉明距离* 71. block1 = block[i] 72. block2 = block[i + 1] 73. distance = hamming\_distance(block1, block2) 74. distances.append(distance / keysize) 75. \_distance = sum(distances) / len(distances) 76. \_data = {  *# 记录长度和归一化距离* 77. 'keysize': keysize, 78. 'distance': \_distance 79. } 80. data.append(\_data) 81. \_keysize = sorted(data, key=lambda distance: distance['distance'])[0]  *# 具有最小归一化汉明距离的KEYSIZE，很可能是真实密钥长度* 82. return \_keysize 83. ​ 84. ​ 85. def Break\_repeating\_key\_XOR(ciphertext): 86. \_keysize = Get\_the\_keysize(ciphertext)  *# 猜测得到的密钥长度* 87. keysize = \_keysize['keysize'] 88. print(keysize) 89. key = b'' 90. message = b'' 91. block = [ciphertext[i:i + keysize] for i in range(0, len(ciphertext), keysize)] 92. for i in range(0, keysize): 93. new\_block = [] 94. t = b'' 95. for j in range(0, len(block) - 1): 96. s = block[j] 97. t = t + bytes([s[i]])  *# 相同位置的组合在一起，被同一密钥加密* 98. socre = ciphertext\_XOR(t) 99. key = key + bytes([socre['Single character']])  *# repeating-key* 100. *# cipher = cipher + socre['ciphertext']* 101. for k in range(0, len(block)): 102. message = message + Repeating\_key\_XOR(block[k], key) 103. return message, key 104. ​ 105. ​ 106. if \_\_name\_\_ == '\_\_main\_\_': 107. with open('6.txt') as of: 108. ciphertext = of.read() 109. ciphertext = base64.b64decode(ciphertext)  *# base64解密，解密后为bytes* 110. message, key = Break\_repeating\_key\_XOR(ciphertext) 111. print("message:", bytes.decode(message), "\nkey:", bytes.decode(key))   结果：    **任务四**  给出用户输入密码所使用的按键，密码的长度和密码SHA1之后的值。  简单的爆破问题，假设每个键只按了一次，再分析出所有可能的按键组合。   1. import hashlib 2. import itertools 3. import datetime 4. ​ 5. hash1 = "67ae1a64661ac8b4494666f58c4822408dd0a3e4" 6. str1 = "QqWw%58(=0Ii\*+nN"  *# 字符集合* 7. str2 = [['Q', 'q'], ['W', 'w'], ['%', '5'], ['8', '('], ['=', '0'], ['I', 'i'], ['\*', '+'], ['n', 'N']] 8. ​ 9. ​ 10. def sha\_encrypt(str): 11. hash = hashlib.sha1() 12. hash.update(str.encode('utf-8')) 13. encrypts = hash.hexdigest() 14. return encrypts 15. ​ 16. ​ 17. st3 = "0" \* 8  *# 假设每个键只按了一次，len(key)=8* 18. str4 = "" 19. str3 = list(st3) 20. starttime = datetime.datetime.now() 21. for a in range(0, 2):  *# 共256种组合* 22. str3[0] = str2[0][a] 23. for b in range(0, 2): 24. str3[1] = str2[1][b] 25. for c in range(0, 2): 26. str3[2] = str2[2][c] 27. for d in range(0, 2): 28. str3[3] = str2[3][d] 29. for e in range(0, 2): 30. str3[4] = str2[4][e] 31. for f in range(0, 2): 32. str3[5] = str2[5][f] 33. for g in range(0, 2): 34. str3[6] = str2[6][g] 35. for h in range(0, 2): 36. str3[7] = str2[7][h] 37. newS = "".join(str3) 38. for i in itertools.permutations(newS, 8): 39. str4 = sha\_encrypt("".join(i)) 40. if str4 == hash1: 41. print("password:", ("".join(i))) 42. endtime = datetime.datetime.now() 43. print("time:", ((endtime - starttime).seconds), "s") 44. ​   运行结果：  password: (Q=win\*5 |
| **总结（完成心得与其它，主要自己碰到的问题和解决问题的方法）**  **任务一一开始看到破解出的文本中有轻微错误，通过调整认为该位置是空格的阈值和两两异或密文的个数，多次进行尝试。最后分析出第8组的明文为“ We can see the point where the chip is unhappy if a wrong bit is sent and consumes”。**  **这个实验也告诉我们，对于流密码，我们绝对不要二次使用密钥，不然是绝对不安全的。** |
| **参考文献（包括参考的书籍，论文，URL等，很重要）**  <https://download.csdn.net/download/dxxmsl/87699394?spm=1001.2014.3001.5503>  [**https://blog.csdn.net/m0\_51605778/article/details/127317714?utm\_medium=distribute.pc\_relevant.none-task-blog-2~default~baidujs\_baidulandingword~default-5-127317714-blog-122388477.235^v38^pc\_relevant\_default\_base&spm=1001.2101.3001.4242.4&utm\_relevant\_index=8**](https://blog.csdn.net/m0_51605778/article/details/127317714?utm_medium=distribute.pc_relevant.none-task-blog-2~default~baidujs_baidulandingword~default-5-127317714-blog-122388477.235%5ev38%5epc_relevant_default_base&spm=1001.2101.3001.4242.4&utm_relevant_index=8)  **https://blog.csdn.net/xiewenbonb/article/details/122392089?spm=1001.2101.3001.6650.14&utm\_medium=distribute.pc\_relevant.none-task-blog-2%7Edefault%7EBlogCommendFromBaidu%7ERate-14-122392089-blog-127317714.235%5Ev38%5Epc\_relevant\_default\_base&depth\_1-utm\_source=distribute.pc\_relevant.none-task-blog-2%7Edefault%7EBlogCommendFromBaidu%7ERate-14-122392089-blog-127317714.235%5Ev38%5Epc\_relevant\_default\_base&utm\_relevant\_index=15** |