|  |
| --- |
| **Github账号：Ystar0511** |
| **个人博客关于密码学实验的链接：**  **https://github.com/Ystar0511/-/tree/main/%E5%AF%86%E7%A0%81%E5%AD%A6%E5%AE%9E%E9%AA%8C** |
| **实验题目：**   1. **Coursera Dan Boneh 第一周编程作业**   **多次使用同一密钥的密码破译**   1. **编程作业1 可选题目**   **编写一个程序，用于"破解"由类维吉尼亚密码生成的密文，该密码使用逐字节异或操作代替模26加法。**   1. [**http://www.cryptopals.com/sets/1**](http://www.cryptopals.com/sets/1)   **(1) 十六进制转Base64**  **(2) 固定异或**  **(3) 单字节异或密码**  **(4) 检测单字符异或**  **(5) 实现重复密钥异或**  **(6) 破解重复密钥异或**   1. **MTC3 破解SHA1哈希密码**   [**https://www.mysterytwisterc3.org/en/challenges/level-2/cracking-sha1-hashed-passwords**](https://www.mysterytwisterc3.org/en/challenges/level-2/cracking-sha1-hashed-passwords) |
| **实验摘要：**  **1.Coursera Dan Boneh 第一周编程作业**   * **题目**：多次使用同一密钥的密码破译 * **要求**：分析并破解使用相同密钥多次加密的密文。   **2.编程作业1 可选题目**   * **题目**：类维吉尼亚密码破译 * **要求**：编写程序破解使用逐字节异或（而非模26加法）的类维吉尼亚密码生成的密文。   **3.Cryptopals 挑战集第1组**   * **包含任务**：   1. **十六进制转Base64**：实现十六进制字符串到Base64编码的转换。   2. **固定异或**：对两个等长缓冲区执行逐字节异或操作。   3. **单字节异或密码**：破解被单字节异或加密的密文。   4. **检测单字符异或**：从多个密文中识别出使用单字符异或加密的那一个。   5. **实现重复密钥异或**：使用重复密钥对明文进行异或加密。   6. **破解重复密钥异或**：分析和破解使用重复密钥异或加密的密文。   **4.MTC3 挑战：破解SHA1哈希密码**   * **题目**：破解SHA1哈希密码 * **要求**：通过密码分析或暴力破解方法，恢复给定SHA1哈希值对应的原始密码。 * **链接**：[MTC3挑战页面](https://www.mysterytwisterc3.org/en/challenges/level-2/cracking-sha1-hashed-passwords) |
| **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）**   1. **多次使用同一密钥的破译**   **本题目要求分析多个使用相同流密码密钥加密的密文。在流密码体系中，密钥重复使用会导致密文间产生特定的数学关系，通过分析这些关系并利用明文的统计特征（如英文字母频率），可以逐步推导出加密密钥和原始明文内容。这是理解密钥管理重要性的典型案例。**   1. **类维吉尼亚密码破解**   **要求实现一个能破解维吉尼亚密码变种的程序。该变种使用字节异或取代传统的模26加法运算。破解过程需要先推测密钥长度，然后将密文按密钥长度分组，对每组使用单字节异或破解技术，最后结合字符频率分析恢复完整密钥。**   1. **Cryptopals密码挑战集**   **这一系列挑战涵盖密码学基础技能：数据编码：十六进制与Base64格式互转异或运算：实现等长缓冲区的按位异或操作单字节破解：通过穷举和评分机制破解简单异或加密加密检测：从多个密文中识别单字符异或加密的样本循环密钥：实现并破解重复密钥异或加密系统**   1. **SHA1哈希密码破解**   **在此挑战中，需要破解一组SHA1哈希值对应的原始密码。虽然SHA1是单向哈希函数，但通过暴力破解、字典攻击或彩虹表等技术，可以对弱密码进行有效还原。该任务揭示了强密码策略在安全体系中的关键作用** |
| **过程**  **1、代码**  msg\_1 = bytes.fromhex(  '315c4eeaa8b5f8aaf9174145bf43e1784b8fa00dc71d885a804e5ee9fa40b16349c146fb778cdf2d3aff021dfff5b403b510d0d0455468aeb98622b137dae857553ccd8883a7bc37520e06e515d22c954eba5025b8cc57ee59418ce7dc6bc41556bdb36bbca3e8774301fbcaa3b83b220809560987815f65286764703de0f3d524400a19b159610b11ef3e')  msg\_2 = bytes.fromhex(  '234c02ecbbfbafa3ed18510abd11fa724fcda2018a1a8342cf064bbde548b12b07df44ba7191d9606ef4081ffde5ad46a5069d9f7f543bedb9c861bf29c7e205132eda9382b0bc2c5c4b45f919cf3a9f1cb74151f6d551f4480c82b2cb24cc5b028aa76eb7b4ab24171ab3cdadb8356f')  msg\_3 = bytes.fromhex(  '32510ba9a7b2bba9b8005d43a304b5714cc0bb0c8a34884dd91304b8ad40b62b07df44ba6e9d8a2368e51d04e0e7b207b70b9b8261112bacb6c866a232dfe257527dc29398f5f3251a0d47e503c66e935de81230b59b7afb5f41afa8d661cb')  msg\_4 = bytes.fromhex(  '32510ba9aab2a8a4fd06414fb517b5605cc0aa0dc91a8908c2064ba8ad5ea06a029056f47a8ad3306ef5021eafe1ac01a81197847a5c68a1b78769a37bc8f4575432c198ccb4ef63590256e305cd3a9544ee4160ead45aef520489e7da7d835402bca670bda8eb775200b8dabbba246b130f040d8ec6447e2c767f3d30ed81ea2e4c1404e1315a1010e7229be6636aaa')  msg\_5 = bytes.fromhex(  '3f561ba9adb4b6ebec54424ba317b564418fac0dd35f8c08d31a1fe9e24fe56808c213f17c81d9607cee021dafe1e001b21ade877a5e68bea88d61b93ac5ee0d562e8e9582f5ef375f0a4ae20ed86e935de81230b59b73fb4302cd95d770c65b40aaa065f2a5e33a5a0bb5dcaba43722130f042f8ec85b7c2070')  msg\_6 = bytes.fromhex(  '32510bfbacfbb9befd54415da243e1695ecabd58c519cd4bd2061bbde24eb76a19d84aba34d8de287be84d07e7e9a30ee714979c7e1123a8bd9822a33ecaf512472e8e8f8db3f9635c1949e640c621854eba0d79eccf52ff111284b4cc61d11902aebc66f2b2e436434eacc0aba938220b084800c2ca4e693522643573b2c4ce35050b0cf774201f0fe52ac9f26d71b6cf61a711cc229f77ace7aa88a2f19983122b11be87a59c355d25f8e4')  msg\_7 = bytes.fromhex(  '32510bfbacfbb9befd54415da243e1695ecabd58c519cd4bd90f1fa6ea5ba47b01c909ba7696cf606ef40c04afe1ac0aa8148dd066592ded9f8774b529c7ea125d298e8883f5e9305f4b44f915cb2bd05af51373fd9b4af511039fa2d96f83414aaaf261bda2e97b170fb5cce2a53e675c154c0d9681596934777e2275b381ce2e40582afe67650b13e72287ff2270abcf73bb028932836fbdecfecee0a3b894473c1bbeb6b4913a536ce4f9b13f1efff71ea313c8661dd9a4ce')  msg\_8 = bytes.fromhex(  '315c4eeaa8b5f8bffd11155ea506b56041c6a00c8a08854dd21a4bbde54ce56801d943ba708b8a3574f40c00fff9e00fa1439fd0654327a3bfc860b92f89ee04132ecb9298f5fd2d5e4b45e40ecc3b9d59e9417df7c95bba410e9aa2ca24c5474da2f276baa3ac325918b2daada43d6712150441c2e04f6565517f317da9d3')  msg\_9 = bytes.fromhex(  '271946f9bbb2aeadec111841a81abc300ecaa01bd8069d5cc91005e9fe4aad6e04d513e96d99de2569bc5e50eeeca709b50a8a987f4264edb6896fb537d0a716132ddc938fb0f836480e06ed0fcd6e9759f40462f9cf57f4564186a2c1778f1543efa270bda5e933421cbe88a4a52222190f471e9bd15f652b653b7071aec59a2705081ffe72651d08f822c9ed6d76e48b63ab15d0208573a7eef027')  msg\_10 = bytes.fromhex(  '466d06ece998b7a2fb1d464fed2ced7641ddaa3cc31c9941cf110abbf409ed39598005b3399ccfafb61d0315fca0a314be138a9f32503bedac8067f03adbf3575c3b8edc9ba7f537530541ab0f9f3cd04ff50d66f1d559ba520e89a2cb2a83')  msg\_target = bytes.fromhex(  '32510ba9babebbbefd001547a810e67149caee11d945cd7fc81a05e9f85aac650e9052ba6a8cd8257bf14d13e6f0a803b54fde9e77472dbff89d71b57bddef121336cb85ccb8f3315f4b52e301d16e9f52f904')  msgs = [msg\_1, msg\_2, msg\_3, msg\_4, msg\_5, msg\_6, msg\_7, msg\_8, msg\_9, msg\_10]  def bytesxor(a, b):  return bytes([x ^ y for (x, y) in zip(a, b)])  def is\_printable\_ascii(b):  return 32 <= b <= 126  def is\_letter(b):  return (ord('a') <= b <= ord('z')) or (ord('A') <= b <= ord('Z'))  def attack\_many\_time\_pad(ciphertexts):  max\_len = max(len(c) for c in ciphertexts)  key = [None] \* max\_len  for i, c1 in enumerate(ciphertexts):  for j, c2 in enumerate(ciphertexts):  if i == j:  continue  xored = bytesxor(c1, c2)  for k in range(min(len(c1), len(c2))):  if is\_letter(xored[k]):  if is\_printable\_ascii(c1[k] ^ ord(' ')):  key\_candidate = c1[k] ^ ord(' ')  if key[k] is None:  key[k] = key\_candidate  valid = True  for c in ciphertexts:  if k < len(c):  decrypted = c[k] ^ key\_candidate  if not is\_printable\_ascii(decrypted) and decrypted != ord('\n'):  valid = False  break  if valid:  key[k] = key\_candidate  for k in range(max\_len):  if key[k] is None:  candidates = {}  for key\_byte in range(256):  valid = True  for c in ciphertexts:  if k < len(c):  decrypted = c[k] ^ key\_byte  if not is\_printable\_ascii(decrypted) and decrypted != ord('\n'):  valid = False  break  if valid:  letter\_count = 0  for c in ciphertexts:  if k < len(c):  decrypted = c[k] ^ key\_byte  if is\_letter(decrypted) or decrypted == ord(' '):  letter\_count += 1  candidates[key\_byte] = letter\_count  if candidates:  key[k] = max(candidates, key=candidates.get)  return key  key = attack\_many\_time\_pad(msgs)  key\_bytes = bytes([k if k is not None else 0 for k in key])  print("目标密文解密结果:")  target\_decrypted = bytesxor(key\_bytes, msg\_target)  print(target\_decrypted)  print("\n所有密文解密结果:")  for i, msg in enumerate(msgs, 1):  decrypted = bytesxor(msg, key\_bytes)  print(f"msg\_{i}: {decrypted}")  **2、代码：**  import string from collections import Counter  def find\_key\_length(ciphertext, max\_length=20):    coincidences = []  for length in range(1, max\_length + 1):  avg\_coincidence = 0  for i in range(length):  segment = ciphertext[i::length]  freq = Counter(segment)  total = len(segment)  if total == 0:  continue  coincidence = sum(count \* (count - 1) for count in freq.values()) / (total \* (total - 1))  avg\_coincidence += coincidence  avg\_coincidence /= length  coincidences.append((length, avg\_coincidence))   coincidences.sort(key=lambda x: x[1], reverse=True)  return coincidences[0][0]  def find\_key\_byte(ciphertext, key\_length, index, english\_freq):   segment = ciphertext[index::key\_length]  if not segment:  return 0   freq = Counter(segment)  total = len(segment)  max\_score = -1  best\_byte = 0  for key\_byte in range(256):  score = 0  for ct\_byte, count in freq.items():  pt\_byte = ct\_byte ^ key\_byte   if chr(pt\_byte) in string.ascii\_letters or chr(pt\_byte) == ' ':  prob = english\_freq.get(chr(pt\_byte).lower(), 0)  score += count \* prob  if score > max\_score:  max\_score = score  best\_byte = key\_byte  return best\_byte  def decrypt(ciphertext, key):   plaintext = bytearray()  key\_length = len(key)  for i, ct\_byte in enumerate(ciphertext):  key\_byte = key[i % key\_length]  plaintext.append(ct\_byte ^ key\_byte)  return plaintext  def encrypt(plaintext, key):   ciphertext = bytearray()  key\_length = len(key)  for i, pt\_byte in enumerate(plaintext):  key\_byte = key[i % key\_length]  ciphertext.append(pt\_byte ^ key\_byte)  return ciphertext  def main():   plaintext = b'This is a sample plaintext for Vigenere like XOR cipher.'  key = b'secret'    ciphertext = encrypt(plaintext, key)  print(f"密文: {ciphertext}")    guessed\_key\_length = find\_key\_length(ciphertext)  print(f"猜测的密钥长度: {guessed\_key\_length}")    english\_freq = {  'a': 0.08167, 'b': 0.01492, 'c': 0.02782, 'd': 0.04253,  'e': 0.12702, 'f': 0.02228, 'g': 0.02015, 'h': 0.06094,  'i': 0.06966, 'j': 0.00153, 'k': 0.00772, 'l': 0.04025,  'm': 0.02406, 'n': 0.06749, 'o': 0.07507, 'p': 0.01929,  'q': 0.00095, 'r': 0.05987, 's': 0.06327, 't': 0.09056,  'u': 0.02758, 'v': 0.00978, 'w': 0.02360, 'x': 0.00150,  'y': 0.01974, 'z': 0.00074  }  guessed\_key = bytearray()  for i in range(guessed\_key\_length):  key\_byte = find\_key\_byte(ciphertext, guessed\_key\_length, i, english\_freq)  guessed\_key.append(key\_byte)  print(f"猜测的密钥: {guessed\_key}")  print(f"实际的密钥: {key}")    decrypted = decrypt(ciphertext, guessed\_key)  print(f"解密结果: {decrypted.decode('ascii', errors='ignore')}")  print(f"实际的明文: {plaintext.decode('ascii')}")  if \_\_name\_\_ == "\_\_main\_\_":  main()  **输出结果：**  **3、代码：**  import base64 from collections import Counter  def hamming\_distance(s1, s2):  *"""计算两个字节串的汉明距离"""* if len(s1) != len(s2):  raise ValueError("两个字节串长度必须相等")  distance = 0  for b1, b2 in zip(s1, s2):   xor = b1 ^ b2  distance += bin(xor).count('1')  return distance  def find\_key\_length(ciphertext, max\_length=40):  *"""找到最可能的密钥长度"""* distances = []  for keysize in range(2, max\_length + 1):   blocks = [ciphertext[i:i+keysize] for i in range(0, 4\*keysize, keysize) if i+keysize <= len(ciphertext)]  if len(blocks) < 2:  continue  total\_distance = 0  pair\_count = 0  for i in range(len(blocks)):  for j in range(i+1, len(blocks)):  total\_distance += hamming\_distance(blocks[i], blocks[j])  pair\_count += 1  if pair\_count == 0:  continue  avg\_distance = total\_distance / pair\_count  normalized\_distance = avg\_distance / keysize  distances.append((keysize, normalized\_distance))   distances.sort(key=lambda x: x[1])  return [d[0] for d in distances[:3]]  def score\_english(text):  *"""根据英文字符频率计算文本的得分，得分越高越可能是英文"""* # 英文字符（含空格）的频率表  freq = {  ' ': 13.0, 'e': 12.702, 't': 9.056, 'a': 8.167, 'o': 7.507,  'i': 6.966, 'n': 6.749, 's': 6.327, 'h': 6.094, 'r': 5.987,  'd': 4.253, 'l': 4.025, 'c': 2.782, 'u': 2.758, 'm': 2.406,  'w': 2.360, 'f': 2.228, 'g': 2.015, 'y': 1.974, 'p': 1.929,  'b': 1.492, 'v': 0.978, 'k': 0.772, 'j': 0.153, 'x': 0.150,  'q': 0.095, 'z': 0.074  }  text = text.lower()  count = Counter(text)  total = sum(count.values())  if total == 0:  return 0  score = 0  for char, cnt in count.items():  score += freq.get(char, 0) \* (cnt / total)  return score  def single\_byte\_xor(ciphertext):  *"""破解单字节异或，返回最优密钥和明文"""* best\_score = 0  best\_key = 0  best\_plaintext = b''  for key in range(256):  plaintext = bytes([b ^ key for b in ciphertext])  current\_score = score\_english(plaintext.decode('ascii', errors='ignore'))  if current\_score > best\_score:  best\_score = current\_score  best\_key = key  best\_plaintext = plaintext  return best\_key, best\_plaintext  def break\_repeating\_key\_xor(ciphertext):  *"""破解重复密钥异或"""* possible\_key\_lengths = find\_key\_length(ciphertext)  best\_key = b''  best\_plaintext = b''  best\_score = 0  for keysize in possible\_key\_lengths:   blocks = [ciphertext[i::keysize] for i in range(keysize)]   key = []  for block in blocks:  key\_byte, \_ = single\_byte\_xor(block)  key.append(key\_byte)  key = bytes(key)   plaintext = bytes([ciphertext[i] ^ key[i % keysize] for i in range(len(ciphertext))])   score = score\_english(plaintext.decode('ascii', errors='ignore'))   if score > best\_score:  best\_score = score  best\_key = key  best\_plaintext = plaintext  return best\_key, best\_plaintext  def main():  # 读取base64编码的密文文件  with open('ciphertext.txt', 'r') as f:  base64\_cipher = f.read()  ciphertext = base64.b64decode(base64\_cipher)   key, plaintext = break\_repeating\_key\_xor(ciphertext)  print("密钥:", key.decode('ascii'))  print("明文:", plaintext.decode('ascii', errors='ignore'))  if \_\_name\_\_ == "\_\_main\_\_":  main()  文本  描述已自动生成**输出结果：**  **文本  描述已自动生成文本  描述已自动生成4、代码**  # -\*- coding: cp936 -\*- import time import multiprocessing import itertools import hashlib  def Func(C):  return [c for c in itertools.permutations(C, 8) if hashlib.sha1(  ''.join(c).encode()).hexdigest() == '67ae1a64661ac8b4494666f58c4822408dd0a3e4']  def Check(s):  return s[:3].count('0')>0 and s[3:].count('0')>0  if \_\_name\_\_ == "\_\_main\_\_":  stime = time.perf\_counter()  keyChars = [('Q', 'q'), ('W', 'w'), ('%', '5'), ('(', '8'),('=', '0'), ('I', 'i'), ('\*', '+'), ('N', 'n')]  Choose = filter(Check,[str(bin(i))[2:].zfill(8) for i in range(256)])   pool = multiprocessing.Pool()  for res in [pool.apply\_async(  Func, ([keyChars[j][int(i[j])] for j in range(len(i))], )) for i in Choose]:   if res.get() != []:  print('The key is:', ''.join(res.get()[0]))   print('Timer:', round(time.perf\_counter() - stime, 2), 's')  **输出结果：**  **文本  描述已自动生成** |
| **总结**  **实验一整体上是一个从古典密码到现代哈希函数的“密码分析入门之旅”。它从最基础的编码和异或操作开始，逐步深入到破解流密码（多次密钥使用、维吉尼亚密码）的核心技术——即利用算法使用不当（密钥重用）或明文统计特性来瓦解系统的安全性。最后，通过破解哈希密码的挑战，揭示了即使使用安全的哈希函数，弱密码本身依然是系统中最薄弱的一环。** |
| **参考文献**  **无** |