|  |
| --- |
| **Github账号：yetanshenshui** |
| **个人博客关于密码学实验的链接：**https://github.com/yetanshenshui/XDU\_Cryptography |
| **实验题目：**   1. **多次使用同一密钥的流密码**   **2. "破解"类似维吉尼亚密码的方法生成的密文**  ****3. 一组密码学挑战****  **十六进制转 Base64，固定异或，单字节异或密码，检测单字符异或，实现重复密钥异或，破解重复密钥异或，ECB 模式下的 AES，检测 ECB 模式下的 AES**   1. **MTC3 破解 SHA1 哈希密码** |
| **实验摘要：**  **本系列实验旨在通过系统性的密码学攻防实践，全面培养密码分析能力与编程实现技能。实验包含三大核心模块：**  **1. 流密码分析：针对多次使用相同密钥的异或加密系统，通过密文间异或运算和英文文本统计特征，破解最后一组密文中的秘密信息。**  **2. Cryptopals基础挑战：完成8个循序渐进的密码学任务，涵盖Base64编码、异或操作、单字节密钥破解、重复密钥异或加解密、AES-ECB模式应用与识别等基础密码学技术。**  **3. SHA1哈希破解：根据泄露的SHA1哈希值，结合键盘指纹痕迹和德语键盘布局的侧信道信息，还原管理员账户密码。**  **实验内容环环相扣，从古典密码分析到现代哈希破解，从纯密码学攻击到侧信道辅助分析，完整展现了密码系统的实际漏洞与攻防场景。通过"由建到破"的实践路径，深入理解密码算法的实现机制、安全边界和典型弱点。** |
| **题目描述**  **问题 1**  **多次使用同一密钥的流密码**  **下面是十一个十六进制编码的密文，它们都是使用同一个流密码密钥加密十一个明文的结果。目标是解密最后一个密文，并提交其中的秘密消息作为答案。**  **提示：将这些密文进行异或操作，并考虑当空格字符与 [a-zA-Z] 范围内的字符进行异或时会发生什么。**  密文1  315c4eeaa8b5f8aaf9174145bf43e1784b8fa00dc71d885a804e5ee9fa40b16349c146fb778cdf2d3aff021dfff5b403b510d0d0455468aeb98622b137dae857553ccd8883a7bc37520e06e515d22c954eba5025b8cc57ee59418ce7dc6bc41556bdb36bbca3e8774301fbcaa3b83b220809560987815f65286764703de0f3d524400a19b159610b11ef3e  密文2  234c02ecbbfbafa3ed18510abd11fa724fcda2018a1a8342cf064bbde548b12b07df44ba7191d9606ef4081ffde5ad46a5069d9f7f543bedb9c861bf29c7e205132eda9382b0bc2c5c4b45f919cf3a9f1cb74151f6d551f4480c82b2cb24cc5b028aa76eb7b4ab24171ab3cdadb8356f  密文3  32510ba9a7b2bba9b8005d43a304b5714cc0bb0c8a34884dd91304b8ad40b62b07df44ba6e9d8a2368e51d04e0e7b207b70b9b8261112bacb6c866a232dfe257527dc29398f5f3251a0d47e503c66e935de81230b59b7afb5f41afa8d661cb  密文4  32510ba9aab2a8a4fd06414fb517b5605cc0aa0dc91a8908c2064ba8ad5ea06a029056f47a8ad3306ef5021eafe1ac01a81197847a5c68a1b78769a37bc8f4575432c198ccb4ef63590256e305cd3a9544ee4160ead45aef520489e7da7d835402bca670bda8eb775200b8dabbba246b130f040d8ec6447e2c767f3d30ed81ea2e4c1404e1315a1010e7229be6636aaa  密文 5  3f561ba9adb4b6ebec54424ba317b564418fac0dd35f8c08d31a1fe9e24fe56808c213f17c81d9607cee021dafe1e001b21ade877a5e68bea88d61b93ac5ee0d562e8e9582f5ef375f0a4ae20ed86e935de81230b59b73fb4302cd95d770c65b40aaa065f2a5e33a5a0bb5dcaba43722130f042f8ec85b7c2070  密文6  32510bfbacfbb9befd54415da243e1695ecabd58c519cd4bd2061bbde24eb76a19d84aba34d8de287be84d07e7e9a30ee714979c7e1123a8bd9822a33ecaf512472e8e8f8db3f9635c1949e640c621854eba0d79eccf52ff111284b4cc61d11902aebc66f2b2e436434eacc0aba938220b084800c2ca4e693522643573b2c4ce35050b0cf774201f0fe52ac9f26d71b6cf61a711cc229f77ace7aa88a2f19983122b11be87a59c355d25f8e4  密文7  32510bfbacfbb9befd54415da243e1695ecabd58c519cd4bd90f1fa6ea5ba47b01c909ba7696cf606ef40c04afe1ac0aa8148dd066592ded9f8774b529c7ea125d298e8883f5e9305f4b44f915cb2bd05af51373fd9b4af511039fa2d96f83414aaaf261bda2e97b170fb5cce2a53e675c154c0d9681596934777e2275b381ce2e40582afe67650b13e72287ff2270abcf73bb028932836fbdecfecee0a3b894473c1bbeb6b4913a536ce4f9b13f1efff71ea313c8661dd9a4ce  密文8  315c4eeaa8b5f8bffd11155ea506b56041c6a00c8a08854dd21a4bbde54ce56801d943ba708b8a3574f40c00fff9e00fa1439fd0654327a3bfc860b92f89ee04132ecb9298f5fd2d5e4b45e40ecc3b9d59e9417df7c95bba410e9aa2ca24c5474da2f276baa3ac325918b2daada43d6712150441c2e04f6565517f317da9d3  密文9  271946f9bbb2aeadec111841a81abc300ecaa01bd8069d5cc91005e9fe4aad6e04d513e96d99de2569bc5e50eeeca709b50a8a987f4264edb6896fb537d0a716132ddc938fb0f836480e06ed0fcd6e9759f40462f9cf57f4564186a2c1778f1543efa270bda5e933421cbe88a4a52222190f471e9bd15f652b653b7071aec59a2705081ffe72651d08f822c9ed6d76e48b63ab15d0208573a7eef027  密文10  466d06ece998b7a2fb1d464fed2ced7641ddaa3cc31c9941cf110abbf409ed39598005b3399ccfafb61d0315fca0a314be138a9f32503bedac8067f03adbf3575c3b8edc9ba7f537530541ab0f9f3cd04ff50d66f1d559ba520e89a2cb2a83  目标密文（解密这个）：  32510ba9babebbbefd001547a810e67149caee11d945cd7fc81a05e9f85aac650e9052ba6a8cd8257bf14d13e6f0a803b54fde9e77472dbff89d71b57bddef121336cb85ccb8f3315f4b52e301d16e9f52f904  为完整起见，这里是用于生成这些密文的 Python 脚本。  import sys  MSGS = ( ---  11 secret messages  --- )  def strxor(a, b):     # xor two strings of different lengths      if len(a) > len(b):         return "".join([chr(ord(x) ^ ord(y)) for (x, y) in zip(a[:len(b)], b)])      else:         return "".join([chr(ord(x) ^ ord(y)) for (x, y) in zip(a, b[:len(a)])])  def random(size=16):      return open("/dev/urandom").read(size)  def encrypt(key, msg):      c = strxor(key, msg)      print      print c.encode('hex')      return c  def main():      key = random(1024)      ciphertexts = [encrypt(key, msg) for msg in MSGS]  **问题2**  **编写一个程序，用于"破解"由类似维吉尼亚密码的方法生成的密文，其中使用了按位异或（XOR）运算来代替模 26 的加法。**  **具体来说，以下密文：是通过使用以下 C 程序加密英文文本生成的：**  #include <stdio.h>  #define KEY\_LENGTH 2 // Can be anything from 1 to 13  main(){  unsigned char ch;  FILE \*fpIn, \*fpOut;  int i;  unsigned char key[KEY\_LENGTH] = {0x00, 0x00};  /\* of course, I did not use the all-0s key to encrypt \*/  fpIn = fopen("ptext.txt", "r");  fpOut = fopen("ctext.txt", "w");  i=0;  while (fscanf(fpIn, "%c", &ch) != EOF) {  /\* avoid encrypting newline characters \*/  /\* In a "real-world" implementation of the Vigenere cipher,  every ASCII character in the plaintext would be encrypted.  However, I want to avoid encrypting newlines here because  it makes recovering the plaintext slightly more difficult... \*/  /\* ...and my goal is not to create "production-quality" code =) \*/  if (ch!='\n') {  fprintf(fpOut, "%02X", ch ^ key[i % KEY\_LENGTH]); // ^ is logical XOR  i++;  }  }    fclose(fpIn);  fclose(fpOut);  return;  }  **（当然，在加密时，我使用了随机的密钥长度，并且密钥的每个字节都是随机选择的。）明文中包含大小写字母、标点符号和空格，但不包含数字。**  ****要求恢复原始明文****  ****问题3****  ****一组密码学挑战****   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** 7. **ECB 模式下的 AES** 8. **检测 ECB 模式下的 AES**   **问题4**  **MTC3 破解 SHA1 哈希密码**  **某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。该密码的哈希值为： 67ae1a64661ac8b4494666f58c4822408dd0a3e4**  **此外，登录终端键盘上留下了所输入密码的明显痕迹，因为成功登录后，软件内的导航仅通过方向键完成。**  **密码是什么？**    **备注：请注意是德语键盘布局！** |
| **实验过程**  **问题 1：多次使用同一密钥的流密码**  **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):  if len(a) > len(b):  return bytes([x ^ y for (x, y) in zip(a[:len(b)], b)])  else:  return bytes([x ^ y for (x, y) in zip(a, b[:len(a)])])  key = [0] \* len(msg\_7)  def isalpha(b):  return (ord('a') <= b <= ord('z')) or (ord('A') <= b <= ord('Z'))  for i, msg\_i in enumerate(msgs):  may\_not\_space = [0] \* len(msg\_i)  for j, msg\_j in enumerate(msgs):  if i != j:  xored = bytesxor(msg\_i, msg\_j)  for k, xb in enumerate(xored):  if (not isalpha(xb)) and xb != 0:  may\_not\_space[k] += 1  for j, may\_not in enumerate(may\_not\_space):  if may\_not <= 2:  key\_byte = msg\_i[j] ^ ord(' ')  if key[j] == 0:  key[j] = key\_byte  continue  if key[j] != key\_byte:  reliable = True  for m in msgs:  if j >= len(m):  continue  byte = m[j] ^ key\_byte  if not isalpha(byte) and byte != ord(' '):  reliable = False  break  if reliable:  key[j] = key\_byte  print(key)  print(bytesxor(bytes(key), msg\_target))  for msg in msgs:  print(bytesxor(msg, bytes(key)))  **运行结果：**    **可以看到第一行的结果，有一部分的乱码，在人为修正后得出明文（最终结果）为：**  The secret message is: when using a stream cipher, never use the key more than once  **问题2："破解"类似维吉尼亚密码的方法生成的密文**  **代码：**  import string  ciphertext将十六进制字母转为十进制数字，便于计算  def hex\_to\_ascii(hex\_text):  ascii\_list = []  for i in range(0, len(hex\_text), 2):  ascii\_list.append(int(hex\_text[i:i + 2], 16))  return ascii\_list  #枚举所有key值，根据明文是否合法，确定key值  def find\_possible\_keys(byte\_group):  valid\_chars = string.ascii\_letters + ',' + '.' + ' '  potential\_keys = []  confirmed\_keys = []  for i in range(0x00, 0xFF):  potential\_keys.append(i)  confirmed\_keys.append(i)  for key in potential\_keys:  for byte in byte\_group:  if chr(key ^ byte) not in valid\_chars:  confirmed\_keys.remove(key)  break  return confirmed\_keys  #枚举得key长度和key值  cipher\_bytes = hex\_to\_ascii(ciphertext)  actual\_key\_length = 0  vigenere\_like\_keys = []  for length in range(1, 14):  temp\_keys = []  for index in range(0, length):  byte\_group = cipher\_bytes[index::length]  keys = find\_possible\_keys(byte\_group)  if not keys:  break  else:  temp\_keys.insert(index, keys)  if temp\_keys:  actual\_key\_length = length  vigenere\_like\_keys = temp\_keys  print(length)  print(f"key:{temp\_keys}")  #得到明文  decrypted\_text = ''  for i in range(0, len(cipher\_bytes)):  decrypted\_text = decrypted\_text + chr(cipher\_bytes[i] ^ vigenere\_like\_keys[i % actual\_key\_length][0])  print(decrypted\_text)  **运行结果：**    **得到最终结果为：**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.  ****问题3：一组密码学挑战****  **十六进制转 Base64，固定异或，单字节异或密码，检测单字符异或**  **代码：**  import base64  # 1.十六进制转 Base64  def hex\_to\_base64(hex\_string):  # 将十六进制字符串转换为字节  raw\_bytes = bytes.fromhex(hex\_string)  # 将字节编码为base64  base64\_encoded = base64.b64encode(raw\_bytes)  # 返回base64字符串（解码为普通字符串）  return base64\_encoded.decode('ascii')  hex\_input = "49276d206b696c6c696e6720796f757220627261696e206c696b65206120706f69736f6e6f7573206d757368726f6f6d"  result = hex\_to\_base64(hex\_input)  print(f"输出: {result}")  # 2.固定异或  def fixed\_xor(hex1, hex2):  # 将十六进制字符串转换为字节  bytes1 = bytes.fromhex(hex1)  bytes2 = bytes.fromhex(hex2)  # 检查长度是否相等  if len(bytes1) != len(bytes2):  raise ValueError("输入字符串长度必须相等")  # 执行XOR操作  result\_bytes = bytes(a ^ b for a, b in zip(bytes1, bytes2))  # 将结果转换回十六进制字符串  return result\_bytes.hex()  # 测试  hex1 = "1c0111001f010100061a024b53535009181c"  hex2 = "686974207468652062756c6c277320657965"  # 执行XOR操作  result = fixed\_xor(hex1, hex2)  print(f"输入1: {hex1}")  print(f"输入2: {hex2}")  print(f"结果: {result}")  # 3.单字节异或密码  def single\_byte\_xor(ciphertext\_hex, key):  ciphertext\_bytes = bytes.fromhex(ciphertext\_hex)  return bytes([b ^ key for b in ciphertext\_bytes])  def english\_score(text\_bytes):  # 常见英文字母频率（近似值）  freq = {  b' ': 15, b'e': 13, b't': 12, b'a': 8, b'o': 8, b'i': 7, b'n': 7,  b's': 6, b'h': 6, b'r': 6, b'd': 4, b'l': 4, b'c': 3, b'u': 3,  b'm': 2, b'w': 2, b'f': 2, b'g': 2, b'y': 2, b'p': 2, b'b': 1,  b'v': 1, b'k': 1, b'j': 1, b'x': 0, b'q': 0, b'z': 0  }  score = 0  for byte in text\_bytes:  # 转换为小写进行评分  char = bytes([byte]).lower()  if char in freq:  score += freq[char]  elif 32 <= byte <= 126: # 可打印ASCII字符  score += 0.5  else: # 非可打印字符，严重扣分  score -= 10  return score  def break\_single\_byte\_xor(ciphertext\_hex):  best\_score = float('-inf')  best\_key = None  best\_plaintext = None  # 尝试所有可能的单字节密钥 (0-255)  for key in range(256):  try:  plaintext = single\_byte\_xor(ciphertext\_hex, key)  score = english\_score(plaintext)  if score > best\_score:  best\_score = score  best\_key = key  best\_plaintext = plaintext  except:  continue  return best\_key, best\_plaintext, best\_score  # 测试  ciphertext = "1b37373331363f78151b7f2b783431333d78397828372d363c78373e783a393b3736"  key, plaintext, score = break\_single\_byte\_xor(ciphertext)  print(f"最佳密钥: {key} (ASCII: '{chr(key) if 32 <= key <= 126 else 'non-printable'}')")  print(f"解密文本: {plaintext.decode('ascii', errors='replace')}")  print(f"评分: {score}")  # 4.检测单字符异或  def detect\_single\_char\_xor(hex\_strings):  def score\_text(text):  # 常见英文字母频率  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, ' ': 0.13000  }  score = 0  for char in text.lower():  if char in freq:  score += freq[char]  return score  def single\_char\_xor\_decrypt(hex\_string, key):  """使用单字符密钥解密XOR加密的十六进制字符串"""  bytes\_data = bytes.fromhex(hex\_string)  result = bytes([b ^ key for b in bytes\_data])  return result  best\_score = -1  best\_result = None  best\_string = None  best\_key = None  for hex\_string in hex\_strings:  # 尝试所有可能的单字节密钥  for key in range(256):  try:  decrypted = single\_char\_xor\_decrypt(hex\_string, key)  text = decrypted.decode('ascii', errors='ignore')  # 计算得分  current\_score = score\_text(text)  if current\_score > best\_score:  best\_score = current\_score  best\_result = text  best\_string = hex\_string  best\_key = key  except:  continue  return best\_string, best\_result, best\_key, best\_score  # 读取文件内容  with open('file-4.txt', 'r') as f:  content = f.read().strip()  # 将内容分割成60个字符的字符串  hex\_strings = [line.strip() for line in content.split('\n')]  # 检测单字符XOR  encrypted\_string, decrypted\_text, key, score = detect\_single\_char\_xor(hex\_strings)  print(f"使用的密钥: {key} (字符: '{chr(key) if 32 <= key <= 126 else 'non-printable'}')")  运行结果：    **实现重复密钥异或，破解重复密钥异或**  **代码：**  import base64  from itertools import combinations  # 5.实现重复密钥异或  def repeating\_key\_xor(plaintext, key):  # 将文本和密钥转换为字节  plaintext\_bytes = plaintext.encode()  key\_bytes = key.encode()  # 执行重复密钥XOR  encrypted\_bytes = bytes([  plaintext\_bytes[i] ^ key\_bytes[i % len(key\_bytes)]  for i in range(len(plaintext\_bytes))  ])  # 返回十六进制字符串  return encrypted\_bytes.hex()  # 测试数据  plaintext = """Burning 'em, if you ain't quick and nimble  I go crazy when I hear a cymbal"""  key = "ICE"  # 加密  result = repeating\_key\_xor(plaintext, key)  print(f"\n加密结果: {result}")  # 6.破解重复密钥异或  # 计算汉明距离  def hamming\_distance(s1, s2):  return sum(bin(b1 ^ b2).count('1') for b1, b2 in zip(s1, s2))  # 计算平均标准化汉明距离  def avg\_normalized\_hamming(ciphertext, keysize, num\_blocks=4):  blocks = [ciphertext[i \* keysize:(i + 1) \* keysize] for i in range(num\_blocks)]  total\_distance = 0  count = 0  for a, b in combinations(blocks, 2):  if len(a) == len(b):  total\_distance += hamming\_distance(a, b) / len(a)  count += 1  return total\_distance / count if count > 0 else float('inf')  # 频率分析得分  def frequency\_score(text):  freq = {}  for char in text:  freq[char] = freq.get(char, 0) + 1  common\_chars = b'etaoin shrdlu'  score = sum(freq.get(char, 0) for char in common\_chars)  return score  # 单字节XOR破解  def single\_byte\_xor(ciphertext):  best\_score = 0  best\_key = 0  best\_plaintext = b''  for key in range(256):  plaintext = bytes([b ^ key for b in ciphertext])  score = frequency\_score(plaintext)  if score > best\_score:  best\_score = score  best\_key = key  best\_plaintext = plaintext  return best\_key, best\_plaintext  # 主解密函数  def break\_repeating\_key\_xor(ciphertext):  # 步骤1: 找到最可能的密钥长度  best\_keysize = 2  best\_score = float('inf')  for keysize in range(2, 41):  score = avg\_normalized\_hamming(ciphertext, keysize)  if score < best\_score:  best\_score = score  best\_keysize = keysize  print(f"最可能的密钥长度: {best\_keysize}")  # 步骤2: 分块并转置  blocks = [ciphertext[i:i + best\_keysize] for i in range(0, len(ciphertext), best\_keysize)]  transposed = []  for i in range(best\_keysize):  block = bytes([block[i] for block in blocks if len(block) > i])  transposed.append(block)  # 步骤3: 对每个转置块进行单字节XOR破解  key = []  for block in transposed:  key\_byte, \_ = single\_byte\_xor(block)  key.append(key\_byte)  key = bytes(key)  print(f"找到的密钥: {key}")  # 步骤4: 使用密钥解密  plaintext = bytes([ciphertext[i] ^ key[i % len(key)] for i in range(len(ciphertext))])  return plaintext, key  # 主程序  if \_\_name\_\_ == "\_\_main\_\_":  # 读取并解码base64  with open('file-6.txt', 'r') as f:  encoded\_data = f.read().strip()  ciphertext = base64.b64decode(encoded\_data)  # 解密  plaintext, key = break\_repeating\_key\_xor(ciphertext)  print("\n解密后的明文:")  print(plaintext.decode('utf-8', errors='ignore'))  print(f"\n使用的密钥: {key.decode('utf-8', errors='ignore')}")  **运行结果：**    **ECB 模式下的 AES，检测 ECB 模式下的 AES**  **代码：**  from Crypto.Cipher import AES  import base64  # 7.ECB 模式下的 AES  def decrypt\_aes\_ecb(ciphertext, key):  cipher = AES.new(key, AES.MODE\_ECB)  plaintext = cipher.decrypt(ciphertext)  return plaintext  def pkcs7\_unpad(data):  padding\_length = data[-1]  return data[:-padding\_length]  # 主程序  if \_\_name\_\_ == "\_\_main\_\_":  # 读取并解码base64  with open('file-7.txt', 'r') as f:  encoded\_data = f.read().strip()  # 解码base64得到密文  ciphertext = base64.b64decode(encoded\_data)  # 密钥  key = b"YELLOW SUBMARINE"  # 解密  decrypted\_data = decrypt\_aes\_ecb(ciphertext, key)  # 移除填充并解码为文本  try:  # 尝试移除PKCS7填充  plaintext = pkcs7\_unpad(decrypted\_data)  result = plaintext.decode('utf-8')  except:  # 如果移除填充失败，直接解码  result = decrypted\_data.decode('utf-8', errors='ignore')  print("解密后的内容:")  print(result)  # 8.检测 ECB 模式下的 AES  def detect\_ecb\_simple(hex\_strings):  """简单检测ECB模式"""  for i, hex\_str in enumerate(hex\_strings, 1):  ciphertext = bytes.fromhex(hex\_str)  block\_size = 16  blocks = [ciphertext[j:j + block\_size] for j in range(0, len(ciphertext), block\_size)]  if len(blocks) != len(set(blocks)):  return i, hex\_str  return None, None  # 主程序  with open('file-8.txt', 'r') as f:  hex\_strings = [line.strip() for line in f if line.strip()]  line\_num, ecb\_ciphertext = detect\_ecb\_simple(hex\_strings)  if ecb\_ciphertext:  print(f"检测到ECB模式加密的密文在行号: {line\_num}")  print(f"密文: {ecb\_ciphertext}")  else:  print("未检测到ECB模式加密的密文")  **运行结果：**    **问题4：MTC3 破解 SHA1 哈希密码**  **代码：**  import hashlib  import itertools  import time  SHA1\_HASH\_TARGET = "67ae1a64661ac8b4494666f58c4822408dd0a3e4"  CHAR\_SETS = [['Q', 'q'], ['W', 'w'], ['5', '%'], ['8', '('], ['=', '0'], ['I', 'i'], ['\*', '+'], ['n', 'N']]  def sha1\_encrypt(input\_string):  sha = hashlib.sha1(input\_string.encode())  hashed\_value = sha.hexdigest()  return hashed\_value  # 暴力破解  start\_time = time.time()  initial\_string = "0" \* 8  current\_password = list(initial\_string)  for i in range(2):  current\_password[0] = CHAR\_SETS[0][i]  for j in range(2):  current\_password[1] = CHAR\_SETS[1][j]  for k in range(2):  current\_password[2] = CHAR\_SETS[2][k]  for l in range(2):  current\_password[3] = CHAR\_SETS[3][l]  for m in range(2):  current\_password[4] = CHAR\_SETS[4][m]  for n in range(2):  current\_password[5] = CHAR\_SETS[5][n]  for o in range(2):  current\_password[6] = CHAR\_SETS[6][o]  for p in range(2):  current\_password[7] = CHAR\_SETS[7][p]  permutation = "".join(current\_password)  for perm in itertools.permutations(permutation, 8):  candidate\_password = "".join(perm)  hashed\_candidate = sha1\_encrypt(candidate\_password)  if hashed\_candidate == SHA1\_HASH\_TARGET:  print("password:", candidate\_password)  end\_time = time.time()  print(f"time:{end\_time - start\_time}s")  exit(0)  **运行结果：**    **最终结果：**(Q=win\*5 |
| **总结**  **本系列密码学实验构成了一个从基础到实战的完整学习路径。通过三个紧密相连的模块，系统性地展现了密码学攻防的核心要点。首先是流密码分析实验，揭示了密钥重用带来的致命安全隐患，通过异或运算和统计特性分析，展示了如何从多个密文中恢复明文，体现了密码分析中"由建到破"的思维转换。**  **在此基础上，Cryptopals挑战集提供了循序渐进的技能训练，从基础的编码转换到复杂的重复密钥破解，再到AES-ECB模式的应用与识别，构建了完整的密码学知识体系。这些基础技能最终在SHA1哈希破解实验中得到了综合运用，该实验不仅要求掌握哈希算法的特性，更需要结合键盘指纹、键盘布局等侧信道信息，展现了真实场景中多维度信息整合的分析能力。**  **整体而言，这些实验生动诠释了密码学中"安全在于实践"的真谛。从算法实现到漏洞利用，从纯理论分析到多源信息整合，实验者能够亲身体会密码系统的脆弱环节和防御要点，建立起对密码学攻防的立体认知，为深入网络安全领域奠定了坚实的实践基础。** |
| **参考文献**  **《现代密码学》（第四版）（杨波编著）**  http://www.cryptopals.com  https://www.mysterytwisterc3.org  https://www.coursera.org/learn/crypto  https://github.com/acetyl-lwx/Cryptography\_assignment\_XDU |