|  |
| --- |
| **Github（或者Coding）账号：AmieeLove** |
| **个人博客关于密码学实验的链接：**  [**AmieeLove/cryptography: experiment1 (github.com)**](https://github.com/AmieeLove/cryptography) |
| **实验题目：**   1. **多次一密(many time pad)**   **让我们看看当一个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文，它们是用流密码对11个明文进行加密的结果，它们都使用相同的流密码密钥。您的目标是解密最后一个密文，并在解决方案中提交秘密消息。**  **提示：将密文XOR在一起，并考虑当空格与[a- z A-Z]中的字符XOR时会发生什么。**  **ciphertext #1:**  **315c4eeaa8b5f8aaf9174145bf43e1784b8fa00dc71d885a804e5ee9fa40b16349c146fb778cdf2d3aff021dfff5b403b510d0d0455468aeb98622b137dae857553ccd8883a7bc37520e06e515d22c954eba5025b8cc57ee59418ce7dc6bc41556bdb36bbca3e8774301fbcaa3b83b220809560987815f65286764703de0f3d524400a19b159610b11ef3e**  **ciphertext #2:**  **234c02ecbbfbafa3ed18510abd11fa724fcda2018a1a8342cf064bbde548b12b07df44ba7191d9606ef4081ffde5ad46a5069d9f7f543bedb9c861bf29c7e205132eda9382b0bc2c5c4b45f919cf3a9f1cb74151f6d551f4480c82b2cb24cc5b028aa76eb7b4ab24171ab3cdadb8356f**  **ciphertext #3:**  **32510ba9a7b2bba9b8005d43a304b5714cc0bb0c8a34884dd91304b8ad40b62b07df44ba6e9d8a2368e51d04e0e7b207b70b9b8261112bacb6c866a232dfe257527dc29398f5f3251a0d47e503c66e935de81230b59b7afb5f41afa8d661cb**  **ciphertext #4:**  **32510ba9aab2a8a4fd06414fb517b5605cc0aa0dc91a8908c2064ba8ad5ea06a029056f47a8ad3306ef5021eafe1ac01a81197847a5c68a1b78769a37bc8f4575432c198ccb4ef63590256e305cd3a9544ee4160ead45aef520489e7da7d835402bca670bda8eb775200b8dabbba246b130f040d8ec6447e2c767f3d30ed81ea2e4c1404e1315a1010e7229be6636aaa**  **ciphertext #5:**  **3f561ba9adb4b6ebec54424ba317b564418fac0dd35f8c08d31a1fe9e24fe56808c213f17c81d9607cee021dafe1e001b21ade877a5e68bea88d61b93ac5ee0d562e8e9582f5ef375f0a4ae20ed86e935de81230b59b73fb4302cd95d770c65b40aaa065f2a5e33a5a0bb5dcaba43722130f042f8ec85b7c2070**  **ciphertext #6:**  **32510bfbacfbb9befd54415da243e1695ecabd58c519cd4bd2061bbde24eb76a19d84aba34d8de287be84d07e7e9a30ee714979c7e1123a8bd9822a33ecaf512472e8e8f8db3f9635c1949e640c621854eba0d79eccf52ff111284b4cc61d11902aebc66f2b2e436434eacc0aba938220b084800c2ca4e693522643573b2c4ce35050b0cf774201f0fe52ac9f26d71b6cf61a711cc229f77ace7aa88a2f19983122b11be87a59c355d25f8e4**  **ciphertext #7:**  **32510bfbacfbb9befd54415da243e1695ecabd58c519cd4bd90f1fa6ea5ba47b01c909ba7696cf606ef40c04afe1ac0aa8148dd066592ded9f8774b529c7ea125d298e8883f5e9305f4b44f915cb2bd05af51373fd9b4af511039fa2d96f83414aaaf261bda2e97b170fb5cce2a53e675c154c0d9681596934777e2275b381ce2e40582afe67650b13e72287ff2270abcf73bb028932836fbdecfecee0a3b894473c1bbeb6b4913a536ce4f9b13f1efff71ea313c8661dd9a4ce**  **ciphertext #8: 315c4eeaa8b5f8bffd11155ea506b56041c6a00c8a08854dd21a4bbde54ce56801d943ba708b8a3574f40c00fff9e00fa1439fd0654327a3bfc860b92f89ee04132ecb9298f5fd2d5e4b45e40ecc3b9d59e9417df7c95bba410e9aa2ca24c5474da2f276baa3ac325918b2daada43d6712150441c2e04f6565517f317da9d3**  **ciphertext #9:**  **271946f9bbb2aeadec111841a81abc300ecaa01bd8069d5cc91005e9fe4aad6e04d513e96d99de2569bc5e50eeeca709b50a8a987f4264edb6896fb537d0a716132ddc938fb0f836480e06ed0fcd6e9759f40462f9cf57f4564186a2c1778f1543efa270bda5e933421cbe88a4a52222190f471e9bd15f652b653b7071aec59a2705081ffe72651d08f822c9ed6d76e48b63ab15d0208573a7eef027**  **ciphertext #10:**  **466d06ece998b7a2fb1d464fed2ced7641ddaa3cc31c9941cf110abbf409ed39598005b3399ccfafb61d0315fca0a314be138a9f32503bedac8067f03adbf3575c3b8edc9ba7f537530541ab0f9f3cd04ff50d66f1d559ba520e89a2cb2a83**  **target ciphertext (decrypt this one):**  **32510ba9babebbbefd001547a810e67149caee11d945cd7fc81a05e9f85aac650e9052ba6a8cd8257bf14d13e6f0a803b54fde9e77472dbff89d71b57bddef121336cb85ccb8f3315f4b52e301d16e9f52f904**  **为了完整起见，下面是用于生成密文的python脚本。**  **（看不懂也没关系）**  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] |
| **题目描述**  **流密码中一次一密是具有完美的加密性的，但是流密码的密钥使用多次就不安全了。本题就是破解多次一密的流密码。**  **题目给出了10个密文，并且全都是用同一个密钥异或。我们要做的是把目标密文解密。**  **根据提示发现空格与字母异或可以将字母大小写转换，而字母与字母异或则是不可见字符。破解思路就是将10条密文两两异或，出现字母的地方说明明文的这个位置是空格，得到明文后与密文异或就能拿到密钥，这就是这个题的解题思路。** |
| **过程**  **Step1定义异或函数、初始化阈值、key**  def xor\_bytes(seq1: bytes, seq2: bytes) -> bytes:  return bytes(b1 ^ b2 for b1, b2 in zip(seq1, seq2))  key = [0] \* 200  # 我们后续需要调整这个空格数的阈值,得到最接近的答案。  # 当阈值过高,则会导致本来确实为空格的字节被忽略,导致密钥不完整  # 当阈值过低,则会导致本来不为空格的字节误认为是空格,导致密钥错误  # 所有寻找一个能够正确识别最大多数空格的阈值  space\_threshold = 6  ciphertexts\_length = len(ciphertexts) |
| **Step2密文两两异或得到key**  **例如对第一条密文来说，设一个candidate\_space列表来记录可能的空格位置，长度是第一条密文的长度，然后依次与其他的密文异或，找出异或结果中出现字母或字节为0的位置，并把该处的candidate\_space值++。然后对camdidate\_space遍历，寻找其中的值超过阈值的位置，将密文此处的值与空格异或得到key。**  for i in range(ciphertexts\_length):      ciphertext1 = bytes.fromhex(ciphertexts[i])      ciphertext1\_length = len(ciphertext1)      candidate\_space = [0] \* ciphertext1\_length        for j in range(ciphertexts\_length):          if j == i:              continue          ciphertext2 = bytes.fromhex(ciphertexts[j])          xor\_text = xor\_bytes(ciphertext1, ciphertext2)            len\_xor\_text = len(xor\_text)          for k in range(len\_xor\_text):              if xor\_text[k: k + 1].isalpha() or xor\_text[k: k + 1] == chr(0):                  candidate\_space[k] += 1        for s in range(ciphertext1\_length):          if candidate\_space[s] >= space\_threshold:              key[s] = ciphertext1[s] ^ ord(' ')    print("根据猜测的密钥求解的明文：", xor\_bytes(key, bytes.fromhex(ciphertexts[-1])).decode()) |
| **Step3修正明文**  **Thm secuet message is: Whtn usi|g wsstream cipher, never use the key more than once**  **由于是猜测的,密钥可能并不完全正确,我们可以自主修正一下：**  **The secret message is: When using a stream cipher, never use the key more than once** |
| **总结**  **我觉得整体不算太难，思路比较清晰，这个题的突破点就是在于题目的提示，空格与字母的异或还是字母，而字母与字母的异或是不可见字符。** |
| **参考文献**  **参考博客**  [**DanBoneh's Cryptography Week 1 | ZephyrRyan**](https://zephyrryan.github.io/posts/DanBoneh's-Cryptography-Week1/) |

|  |
| --- |
| **实验题目：**   1. **PA1 option**   **编写一个程序，允许您“破解”使用类似vigenere的密码生成的密文，其中使用逐字节异或而不是加法模26。**  **密文：这段密文是由下列c代码产生的。**  #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，我使用随机密钥长度并随机选择密钥的每个字节。明文包含大写字母、小写字母、标点符号和空格，但不包含数字。** |
| **题目描述**  **首先我们要猜出密钥长度。之后我们根据密钥长度将密文分组，对密钥一位一位的猜测，在0~255之间，最后解出明文。** |
| **过程**  **Step1猜测key长度**  **根据c代码我们可以看出密钥最大长度是13，那么我们就依次使密钥长度为1，2，3…13，算出各自的密钥的适配指数，选取最大的指数相应的长度作为密钥长度**  def compare\_bytes(a: bytes, b: bytes):      fitting\_index = 0      for le1, le2 in zip(a, b):          if le1 ==le2:              fitting\_index += 1      return fitting\_index / min(len(a), len(b))  def guess\_key\_length(cipher\_bytes: bytes):      fitting\_index\_dict = {}      for key\_len in range(1, max\_key\_length+1):          ciphertext\_chunks = []          for i in range(0, ciphertext\_length, key\_len):              ciphertext\_chunks.append(cipher\_bytes[i: i+key\_len])          chunks\_length = len(ciphertext\_chunks)          fitting\_index = 0          for i in range(chunks\_length):              fitting\_sum = 0              for j in range(i+1, chunks\_length):                  fitting\_sum += compare\_bytes(ciphertext\_chunks[i], ciphertext\_chunks[j])              fitting\_index += (fitting\_sum / (chunks\_length - i))          fitting\_index\_dict[key\_len] = fitting\_index      return fitting\_index\_dict    **适配指数最高的是1然后就是7，由于当key长度为1时0~255之间都无正确的密钥，于是我们选取7作为密钥长度。**  **Step2猜测key**  **因为根据c代码可以看出明文只包含大小写字母、标点、空格，我们根据异或出的明文是否在大小写字母、标点、空格之间判断key的位猜测是否正确。**  def get\_key(key\_len: int, cipher\_text\_byte\_chunks):      key = []      for i in range(key\_len):          length = len(cipher\_text\_byte\_chunks[i])          for k in range(256):              for l in range(length):                  if not judge(cipher\_text\_byte\_chunks[i][l] ^ k):                      break                  if l == length - 1:                      key.append(k)        return key  **Step3得出明文**  **根据密文长度，扩展密钥，然后再依次异或**  def repeating\_xor(text: bytes, key) -> bytes:      quotient, remainder = divmod(len(text), len(key))      key\_extend = bytes(key \* quotient + key[:remainder])      return bytes([x ^ y for x, y in zip(text, key\_extend)])  **plaintext：** **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.**  **Key：** **[186, 31, 145, 178, 83, 205, 62]** |
| **总结**  **这个题目相比较上一个题要难一些，难就难在如何去猜测密钥长度** |
| **参考文献**  **参考博客**  [**Vigenere-like cipher | ZephyrRyan**](https://zephyrryan.github.io/posts/Vigenere-like-cipher/) |
|  |
| **实验题目：**   1. http://www.cryptopals.com/sets/1   **(1)十六进制转换成base64( Convert hex to base64)**  **str:**  **49276 d206b696c6c696e6720796f757220627261696e206c696b65206120706f69736f6e6f7573206d757368726f6f6d**  **Should produce:**  **SSdtIGtpbGxpbmcgeW91ciBicmFpbiBsaWtlIGEgcG9pc29ub3VzIG11c2hyb29t**  **所以勇往直前，让梦想成真吧。您将需要在剩下的练习中使用此代码。**  **(2) 异或(Fixed XOR)**  **编写一个函数，取两个长度相等的缓冲区并生成它们的异或组合。**  **如果你的函数工作正常，那么当你给它输入字符串时：**  **1c0111001f010100061a024b53535009181c**  **…十六进制解码后，当异或对：**  **686974207468652062756 c6c277320657965**  **…应该生产:**  **746865206 b696420646f6e277420706c6179**  **(3) 单个字节异或密码(Single-byte XOR cipher)**  **十六进制编码字符串：**  **1b37373331363f78151b7f2b783431333d78397828372d363c78373e783a393b3736**  **…对单个字符进行异或处理。找到密钥，解密信息。**  **你可以用手做。但是不要写代码来帮你做。**  **如何?设计一些方法来“评分”一段英文明文。字符频率是一个很好的指标。评估每个输出并选择得分最高的输出。**  **(4)检测单字符异或(Detect single-character XOR)**  **该文件中的60个字符串中有一个已通过单字符异或加密。**  **找到它。**  **（第3条中的代码应该会有所帮助。）**  **(5) 重复键异或(Implement repeating-key XOR)**  **这是一部重要的英语作品的开头一节：**  **Burning 'em, if you ain't quick and nimble**  **I go crazy when I hear a cymbal**  **在“ICE”键下，使用重复键异或。**  **在重复键异或中，您将依次应用键的每个字节；**  **明文的第一个字节将对I、下一个C、下一个E进行异或，然后对第四个字节再次进行异或，以此类推。它应该是：0b3637272a2b2e63622c2e69692a23693a2a3c6324202d623d63343c2a2622632432427272727272727272a20430a652e652a3124333a653e2b2027630c692b20283165286326302e27282f使用重复密钥XOR功能加密一堆东西。**  **加密你的邮件，加密您的密码文件，您的.sig文件。感受一下。我保证，我们不会浪费你的时间。**  **(6)破解重复键异或 (Break repeating-key XOR)**  **这里有一份文件。**  **它是在用重复密钥异或加密后被加密的。**  **解密它。**  **方法如下:**  **1.设KEYSIZE为键的猜测长度；**  **尝试从2到（比如）40**  **2.编写一个函数来计算两个字符串之间的编辑距离/汉明距离。**  **汉明距离就是不同比特的数量。**  **This is a test**  **和**  **Wokka wokka！！！**  **汉明距离是37。**  **在继续之前，请确保代码一致。**  **3.对于每个KEYSIZE，取第一个KEYSIZE值的字节，第二个KEYSIZE值的字节，并找出它们之间的编辑距离。通过除以KEYSIZE将结果归一化。**  **4.具有最小规范化编辑距离的KEYSIZE可能是键。您可以继续使用最小的2-3个KEYSIZE值。或者取4个KEYSIZE块而不是2个，然后取距离的平均值。**  **5.现在您可能知道KEYSIZE：将密文分解成KEYSIZE长度的块。**  **6.现在把这些块调换一下：做一个是每个块的第一个字节的块，和一个是第二个字节的块**  **7.把每个块当作单个字符的异或来求解。您已经有了这样做的代码。**  **8.对于每个块，产生最好看的直方图的单字节异或键是该块的重复键异或键字节。把它们放在一起，你就有钥匙了。**  **这段代码在后面会变得非常有用。在统计上打破重复密钥异或（“Vigenere”）显然是一种学术练习，一种“密码101”的事情。但更多的人“知道”如何打破它，而不是真正打破它，类似的技术打破了更重要的东西。** |
| **题目描述**   1. 这个没什么好说的，直接调用base64 2. 字符串转字节串，异或，列表中的值转16进制，.join()连接列表中的值。 3. 根据提示字符频率我们想到遍历0~255解出密文，遍历明文并计算这段明文中的字符频率是否与实际相符。 4. 这道 challenge 是以 challenge3 为基础的，不同的是，尽管 challenge4 是一个密文，但他藏在多个文本中，因此，我们只需要利用 challenge3 中的代码，先选出每一串字符最有可能的明文，再从其中选出拟合程度最高的明文，即为正确答案。 5. 重复密钥 XOR 加密其实是维吉尼亚密码的一种变体，也比较简单 6. 这个题目应该是最难的，首先我们应该了解什么是汉明距离，就是比特位不同的数量，我们可以计算异或后1的数量来得出汉明距离。然后根据提示一步一步做。 |
| **过程**  **(1)(2)比较简单，就不详细说了**  **(3)总体思路：遍历密钥解密得到各个明文文本，使用字符频率作为指标对明文进行评分，评估各个明文文本并选择评分最高的明文文本，其对应的密钥就是需要找到的密钥。**  **以下是26个字母出现的频率**  CHARACTER\_FREQ = {      'a': 0.0651738, 'b': 0.0124248, 'c': 0.0217339, 'd': 0.0349835, 'e': 0.1041442, 'f': 0.0197881, 'g': 0.0158610,      'h': 0.0492888, 'i': 0.0558094, 'j': 0.0009033, 'k': 0.0050529, 'l': 0.0331490, 'm': 0.0202124, 'n': 0.0564513,      'o': 0.0596302, 'p': 0.0137645, 'q': 0.0008606, 'r': 0.0497563, 's': 0.0515760, 't': 0.0729357, 'u': 0.0225134,      'v': 0.0082903, 'w': 0.0171272, 'x': 0.0013692, 'y': 0.0145984, 'z': 0.0007836, ' ': 0.1918182  }  **Step1遍历备选密钥返回得分最高的字符作为key**  def Traversal\_singlechar(hex\_string):      candidate=[]      for candidate\_key in range(256):          plaintext=single\_xor(candidate\_key,hex\_string)          score=get\_score(plaintext)          result={          'score':score,          'plaintext':plaintext,          'key':candidate\_key          }          candidate.append(result)      return sorted(candidate,key=lambda c:c['score'])[-1]  **Step2异或得出明文**  **Plaintext: Cooking MC's like a pound of bacon**  **Key: X**  **(4)题目描述里说得很清晰了，这里不再赘述，主要还是依赖第3题**  **(5)这里只给出重复异或函数**  def repeating\_key\_xor(plaintext, key):      ciphertext = b''      i = 0      for byte in plaintext:          ciphertext += bytes([byte ^ key[i]])          # Cycle i to point to the next byte of the key          i = i + 1 if i < len(key) - 1 else 0      return ciphertext   1. **Step1计算汉明距离**   **异或后计算1的个数**  def hamming\_distance(a,b) :      distance = 0      for i ,j in zip(a,b) :          byte = i^j          distance = distance + sum(k == '1' for k in bin(byte) )      return distance  **Step2猜测密钥长度**  **提示说密钥长度为2~40**  **然后对每一个KEYSIZE把密文分成block\_size为KEYSIZE大小的块，计算汉明距离并求和，找出最小汉明距离的KEYSIZE。这里求出KEYSIE为29.**  for keysize in range(2,41) :          block = [ciphertext[i:i+keysize] for i in range(0,len(ciphertext),keysize)]          distances = []          for i in range(0,len(block),2) :              try:                  block1 = block[i]                  block2 = block[i+1]                  distance = hamming\_distance(block1,block2)                  distances.append(distance / keysize)              except :                  break          \_distance = sum(distances) / len(distances)          \_data = {              'keysize' : keysize,              'distance': \_distance          }          data.append(\_data)      \_keysize = sorted(data, key = lambda distance:distance['distance'])[0]  **Step3将密文按照KEYSIZE分组，并进行单字节异或找到key，得到明文**  **单字节异或在之前已经说过，不再赘述。** |
| **总结**  **第二小题第一次忘记了转成16进制会有前缀”0x”，后来取[2：]**  **Set1总体来说不难，只是最后一个破解重复键异或花费了很长时间。** |
| **参考文献**  **参考博客**  [**ZephyrRyan**](https://zephyrryan.github.io/)  [**Cryptopals\_set1\_challenge6\_Break repeating-key XOR\_ciphey repeating-key xor-CSDN博客**](https://blog.csdn.net/pangyuanzi/article/details/114763139) |
|  |
| **实验题目：**   1. **MTC3 Cracking SHA1-Hashed Passwords**   **题目太长了而且是pdf，所以题目链接附在这里了**  [**Cracking SHA1-hashed passwords (mysterytwister.org)**](https://mysterytwister.org/media/challenges/pdf/mtc3-kitrub-07-sha1crack-en.pdf) |
| **题目描述**  根据键盘上的印记我们推断↑↓←→和8 6 2 4为控制键不存在密码里，所以密码中可能的字符有Q q W w I i \* + N n( 8 % 5 = 0  依次组合，寻找sha1值相等的组合。 |
| **过程**  **过程比较简单，就是排列组合爆破出sha1值相等的密码，具体过程可以看代码，不难。** |
| **总结**  **这是sha1使用的一个真实例子，服务器不会直接存储用户的password而是存储其哈希值，根据用户的输入，再次计算其哈希值是否一致，来实现身份验证。** |
| **参考文献**  **参考博客**  [**[MTC3]Cracking SHA1-Hashed Passwords - Elpsywk - 博客园 (cnblogs.com)**](https://www.cnblogs.com/elpsycongroo/p/7669786.html) |
|  |