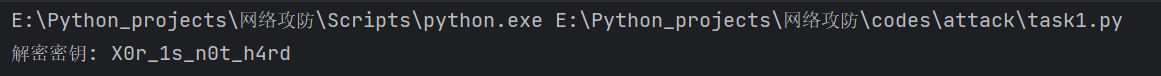
# 1.key简单变换

该解密程序将密文的每一位与密文下标的数值进行异或操作

secret = b"\x58\x31\x70\x5C\x35\x76\x59\x69\x38\x7D\x55\x63\x38\x7F\x6A"  
  
def decrypt(secret):  
 szKey = bytearray(20)  
 for i in range(len(secret)):  
 szKey[i] = secret[i] ^ i return szKey.split(b'\x00')[0].decode()  
  
print("解密密钥:", decrypt(secret))

运行结果为



获取明文密钥

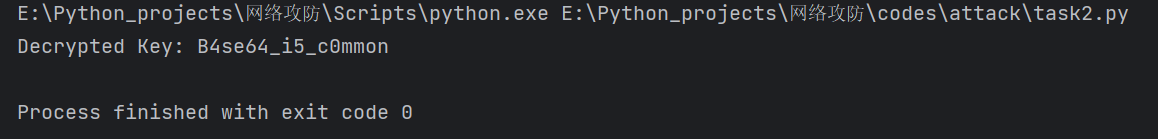
X0r\_1s\_n0t\_h4rd

# 2.key编码转换

将密文base64解密即可获取解密后的密钥

import base64  
  
secret = "QjRzZTY0X2k1X2MwbW1vbg=="  
  
def decrypt\_secret(secret):  
 decoded\_secret = base64.b64decode(secret)  
 return decoded\_secret.decode()  
  
print("Decrypted Key:", decrypt\_secret(secret))

运行结果为



获取明文密钥

B4se64\_i5\_c0mmon

# 3.key密钥加密

首先使用提供的密钥进行多次交换操作来“混洗”排列数组。每个元素都与另一个通过密钥和当前状态计算得出的索引的元素交换。密钥长度可能不等于 256，因此使用 key[i % len(key)] 确保即使密钥较短也能重复使用。

然后因为RC4 是对称的，这意味着加密和解密使用相同的操作。

函数接受已初始化的 S 数组和要处理的数据（可以是明文或密文）。

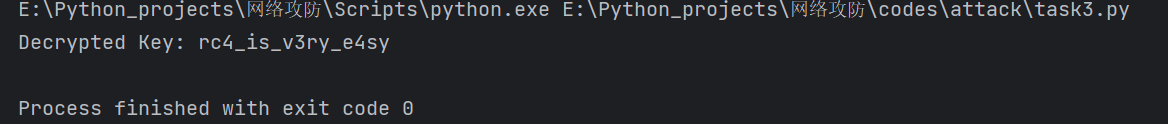
使用两个指针 i 和 j 在 S 数组中移动，并在处理每个数据字节时进行交换和异或操作，生成最终的输出。

最后首先将密钥字符串转换为它的 ASCII 数值列表，然后用 rc4\_init 函数初始化 S 数组。

然后，用 rc4\_crypt 函数解密密文。

def rc4\_init(key):  
 S = list(range(256))  
 j = 0 for i in range(256):  
 j = (j + S[i] + key[i % len(key)]) % 256 S[i], S[j] = S[j], S[i]  
 return S  
  
def rc4\_crypt(S, data):  
 i = 0 j = 0 out = []  
 for char in data:  
 i = (i + 1) % 256 j = (j + S[i]) % 256 S[i], S[j] = S[j], S[i]  
 out.append(chr(ord(char) ^ S[(S[i] + S[j]) % 256]))  
 return ''.join(out)  
  
secret = "\x6E\xDC\xE5\xE7\xB7\x89\x3C\xC5\x81\xE7\xD0\xB5\xCB\xAA\x91\xDE" rc4key = "I\_am\_not\_the\_flag"  
  
def decrypt\_secret(secret, rc4key):  
 S = rc4\_init([ord(char) for char in rc4key])  
 decrypted = rc4\_crypt(S, secret)  
 return decrypted  
  
print("Decrypted Key:", decrypt\_secret(secret, rc4key))

运行结果为



获取明文密钥

rc4\_is\_v3ry\_e4sy

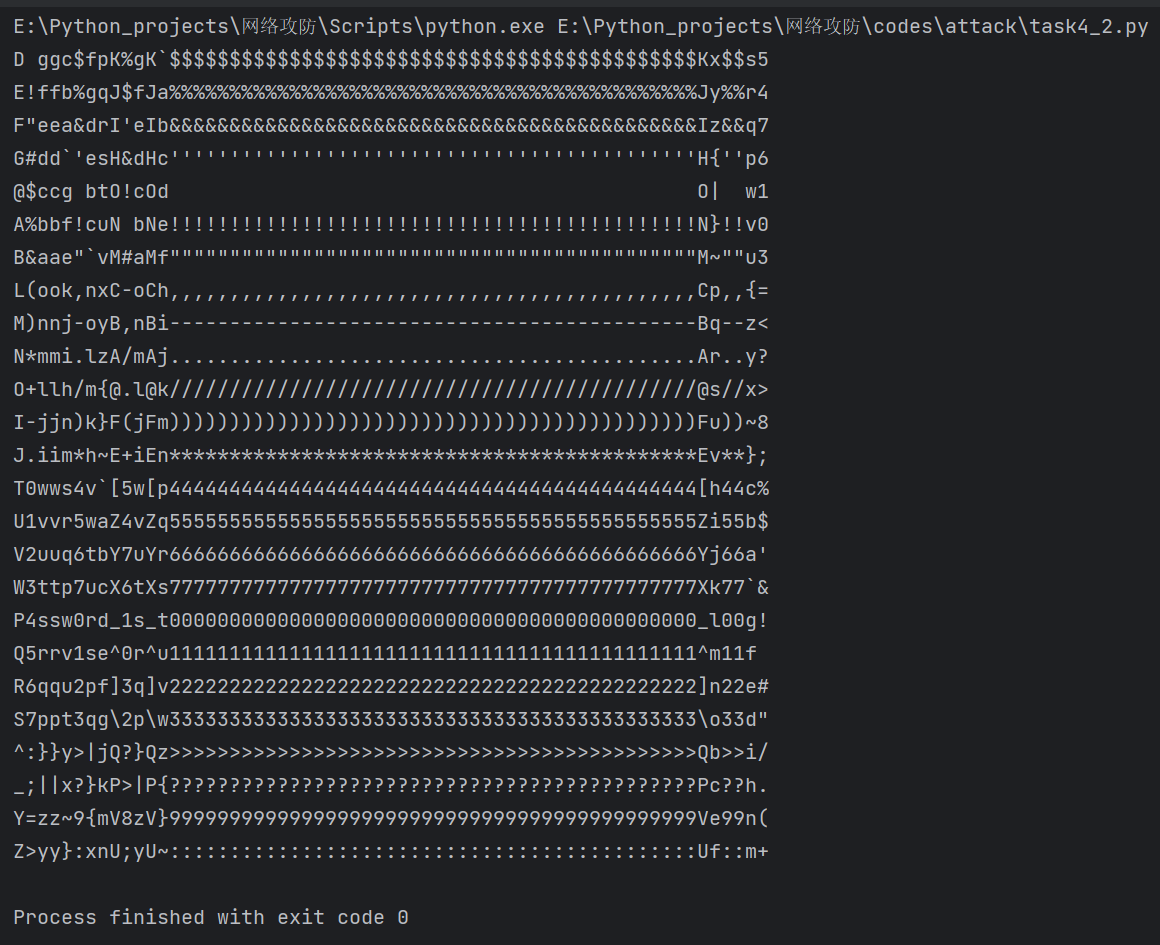
# 4.密钥空间过短

首先分析加密程序，提示用户输入一个密钥（szKey），该密钥应具有确切的长度（63个字符）。然后进行MD5 散列计算：如果密钥长度正确，程序将使用 MD5 算法对密钥进行散列计算，得到一个散列值（szHash）。之后进行散列比较：然后将计算得到的散列值与预定义的 secret 散列值进行比较。之后如果散列值匹配，程序将使用密钥的第二个字符 (szKey[1]) 对一个加密的字符串 (enflag) 进行异或操作来解密它。解密成功后，程序输出格式化的字符串 ctf{}，包含解密的内容。

因为在异或操作时只使用了szKey[1],因此在这里我们可以对szKey[1]的值进行循环爆破

c = [0x64,0x00,0x47,0x47,0x43,0x04,0x46,0x50,0x6B,0x05,0x47,0x6B,0x40,0x04,0x04,0x04,0x04,0x04,0x04,0x04,0x04,0x04,0x04,0x04,0x04,0x04,0x04,0x04,0x04,0x04,0x04,0x04,0x04,0x04,0x04,0x04,0x04,0x04,0x04,0x04,0x04,0x04,0x04,0x04,0x04,0x04,0x04,0x04,0x04,0x04,0x04,0x04,0x04,0x04,0x04,0x04,0x04,0x6B,0x58,0x04,0x04,0x53,0x15]  
  
  
for i in range(0,255):  
 s = "" for c\_s in c:  
 out = c\_s^i if out > 126 or out < 32:  
 break s += chr(out)  
 if out > 126 or out < 32:  
 continue print(s)

爆破结果为



观察结果发现有P4ssw0rd\_1s\_t00000000000000000000000000000000000000000000\_l00g!字符串看起来像一个有效的密钥，因为其格式符合通常的密钥字符串风格。

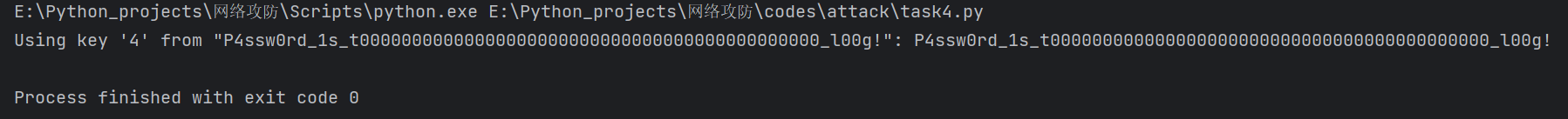
利用怀疑的密钥字符串编写解密程序。

通过对每个字节执行异或操作来解密数据。这里将密钥字符转换为其 ASCII 数值，然后与 enflag 中的每个字节进行异或操作。解密的结果是通过列表推导生成的，每个字节解密后转换为字符，然后所有字符合并成一个字符串。

然后提取猜测密钥的第二个字符 (key[1]) 作为解密密钥，编写程序如下

enflag = bytearray([0x64,0x00,0x47,0x47,0x43,0x04,0x46,0x50,0x6B,0x05,0x47,0x6B,0x40,0x04,0x04,0x04,0x04,0x04,0x04,0x04,0x04,0x04,0x04,0x04,0x04,0x04,0x04,0x04,0x04,0x04,0x04,0x04,  
0x04,0x04,0x04,0x04,0x04,0x04,0x04,0x04,0x04,0x04,0x04,0x04,0x04,0x04,0x04,0x04,  
0x04,0x04,0x04,0x04,0x04,0x04,0x04,0x6B,0x58,0x04,0x04,0x53,0x15])  
  
# Possible keys from brute force output  
possible\_keys = [  
 "P4ssw0rd\_1s\_t00000000000000000000000000000000000000000000\_l00g!",  
]  
  
def decrypt(enflag, key\_char):  
 return ''.join(chr(b ^ ord(key\_char)) for b in enflag)  
  
# Use second character from each key to decrypt  
for key in possible\_keys:  
 decrypted\_flag = decrypt(enflag, key[1])  
 print(f"Using key '{key[1]}' from \"{key}\": {decrypted\_flag}")

运行结果为



成功获取明文密钥

P4ssw0rd\_1s\_t000000000000000000000000000000000000000000\_l00g!