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)

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:

315c4eeaa8b5f8aaf9174145bf43e1784b8fa00dc71d885a804e5ee9fa40b16349c146fb778cdf2d3aff021dfff5b403b510d0d0455468aeb98622b137dae857553ccd8883a7bc37520e06e515d22c954eba5025b8cc57ee59418ce7dc6bc41556bdb36bbca3e8774301fbcaa3b83b2208095600987815f65286764703de0f3d524400a19b159610b11ef3e

ciphertext #2:

234c02ecbbfbafa3ed18510abd11fa724fcda2018a1a8342cf064bbde548b12b07df44ba7191d9606ef4081ffde5ad46a5069d9f7f543bedb9c861bf29c7e205132eda9382b0bc2c5c4b45f919cf3a9f1cb74151f6d551f4480c82b2cb24cc5b028aa76eb7b4ab24171ab3cdadb8356f

ciphertext #3:

32510ba9a7b2bba9b8005d43a304b5714cc0bb0c8a34884dd91304b8ad40b62b07df44ba6e9d8a2368e51d04e0e7b207b70b9b8261112bacb6c866a232dfe257527dc29398f5f3251a0d47e503c66e935de81230b59b7afb5f41afa8d661cb

ciphertext #4:

32510ba9aab2a8a4fd06414fb517b5605cc0aa0dc91a8908c2064ba8ad5ea06a029056f47a8ad3306ef5021eafe1ac01a81197847a5c68a1b78769a37bc8f4575432c198ccb4ef63590256e305cd3a9544ee4160ead45aef520489e7da7d835402bca670bda8eb775200b8dabbba246b130f040d8ec6447e2c767f3d30ed81ea2e4c1404e1315a1010e7229be6636aaa

ciphertext #5:

3f561ba9adb4b6ebec54424ba317b564418fac0dd35f8c08d31a1fe9e2
4fe56808c213f17c81d9607cee021dafe1e001b21ade877a5e68bea88d
61b93ac5ee0d562e8e9582f5ef375f0a4ae20ed86e935de81230b59b73
fb4302cd95d770c65b40aaa065f2a5e33a5a0bb5dcaba43722130f042f
8ec85b7c2070

ciphertext #6:

32510bfbacfbb9befd54415da243e1695ecabd58c519cd4bd2061bbde
24eb76a19d84aba34d8de287be84d07e7e9a30ee714979c7e1123a8bd9
822a33ecaf512472e8e8f8db3f9635c1949e640c621854eba0d79eccf52
ff111284b4cc61d11902aebc66f2b2e436434eacc0aba938220b084800
c2ca4e693522643573b2c4ce35050b0cf774201f0fe52ac9f26d71b6cf6
1a711cc229f77ace7aa88a2f19983122b11be87a59c355d25f8e4

ciphertext #7:

32510bfbacfbb9befd54415da243e1695ecabd58c519cd4bd90f1fa6e
a5ba47b01c909ba7696cf606ef40c04afe1ac0aa8148dd066592ded9f87
74b529c7ea125d298e8883f5e9305f4b44f915cb2bd05af51373fd9b4a
f511039fa2d96f83414aaaf261bda2e97b170fb5cce2a53e675c154c0d9
681596934777e2275b381ce2e40582afe67650b13e72287ff2270abcf7
3bb028932836fbdecfecee0a3b894473c1bbeb6b4913a536ce4f9b13f1
efff71ea313c8661dd9a4ce

ciphertext #8:

315c4eeaa8b5f8bffd11155ea506b56041c6a00c8a08854dd21a4bbde5
4ce56801d943ba708b8a3574f40c00fff9e00fa1439fd0654327a3bfc8
60b92f89ee04132ecb9298f5fd2d5e4b45e40ecc3b9d59e9417df7c95b
ba410e9aa2ca24c5474da2f276baa3ac325918b2daada43d6712150441
c2e04f6565517f317da9d3

ciphertext #9:

271946f9bbb2aeadec111841a81abc300ecaa01bd8069d5cc91005e9f
e4aad6e04d513e96d99de2569bc5e50eeeca709b50a8a987f4264edb68
96fb537d0a716132ddc938fb0f836480e06ed0fcd6e9759f40462f9cf5
7f4564186a2c1778f1543efa270bda5e933421cbe88a4a52222190f471
e9bd15f652b653b7071aec59a2705081ffe72651d08f822c9ed6d76e48
b63ab15d0208573a7eef027

ciphertext #10:

466d06ece998b7a2fb1d464fed2ced7641ddaa3cc31c9941cf110abbf
409ed39598005b3399ccfafb61d0315fca0a314be138a9f32503bedac8

067f03adbf3575c3b8edc9ba7f537530541ab0f9f3cd04ff50d66f1d559ba520e89a2cb2a83

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

实验题目：

2. 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:

F96DE8C227A259C87EE1DA2AED57C93FE5DA36ED4EC87EF2C63AAE5B9A7EFFD673BE4ACF7BE8923CAB1ECE7AF2DA3DA44FCF7AE29235A24C963FF0DF3CA3599A70E5DA36BF1ECE77F8DC34BE129A6CF4D126B

F5B9A7CFEDF3EB850D37CF0C63AA2509A76FF9227A55B9A6FE3D720A850D97AB1DD35ED5FCE6BF0D138A84CC931B1F121B44ECE70F6C032BD56C33FF9D320ED5CDF7AFF9226BE5BDE3FF7DD21ED56CF71F5C036A94D963FF8D473A351CE3FE5DA3CB84DDB71F5C17FED51DC3FE8D732BF4D963FF3C727ED4AC87EF5DB27A451D47EFD9230BF47CA6BFEC12ABE4ADF72E29224A84CDF3FF5D720A459D47AF59232A35A9A7AE7D33FB85FCE7AF5923AA31EDB3FF7D33ABF52C33FF0D673A551D93FFCD33DA35BC831B1F43CBF1EDF67F0DF23A15B963FE5DA36ED68D378F4DC36BF5B9A7AFFD121B44ECE76FEDC73BE5DD27AFCD773BA5FC93FE5DA3CB859D26BB1C63CED5CDF3FE2D730B84CDF3FF7DD21ED5ADF7CF0D636BE1EDB79E5D721ED57CE3FE6D320ED57D469F4DC27A85A963FF3C727ED49DF3FFFDD24ED55D470E69E73AC50DE3FE5DA3ABE1EDF67F4C030A44DDF3FF5D73EA250C96BE3D327A84D963FE5DA32B91ED36BB1D132A31ED87AB1D021A255DF71B1C436BF479A7AF0C13AA14794

这段密文是由下列 c 代码产生的。

```c
#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
```

```
1 : 5.4054
2 : 2.6859
3 : 1.5482
4 : 1.1983
5 : 0.9524
6 : 0.7179
7 : 4.7134
8 : 0.5238
9 : 0.4511
10 : 0.4307
11 : 0.4277
12 : 0.2911
13 : 0.248
```

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

实验题目：

3. http://www.cryptopals.com/sets/1

(1)十六进制转换成 base64(Convert hex to base64)

str:

49276

d206b696c6c696e6720796f757220627261696e206c696b65206120706

f69736f6e6f7573206d757368726f6f6d

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：

1b37373331363f78151b7f2b783431333d78397828372d363c78373e78

3a393b3736

…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：

0b3637272a2b2e63622c2e69692a23693a2a3c6324202d623d63343c2a26226324342427272727272727272a20430a652e652a3124333a653e2b2027630c692b20283165286326302e27282f 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
```

(7)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
Cryptopals_set1_challenge6_Break repeating-key XOR_ciphey repeating-key xor-CSDN 博客

实验题目：
4. 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)

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)