作者：上海交通大学蜚语安全（G.O.S.S.I.P）研究小组实习生张一苇

链接：https://cryptopals.com/

参考：https://github.com/ickerwx/cryptopals

Set 2：Block crypto

Set 2的练习与分组密码学有关，在这一组里你会看到大多数加密网络软件中实现的方法，在Set 1的基础上完成练习，是对真实世界中加密算法的安全攻击的简单入门。

目录：

Challenge 9: Implement PKCS#7 padding

题目要求：

分组密码将固定大小（通常为8或16字节）的文本块转化为密文，大多数时候我们需要加密不规则大小的信息，但是通常不想改变消息的内容。这时，通常的做法是通过填充来生成一段块大小整数倍的文本，最流行的填充策略是PKCS#7。

PKCS#7这种方法是在最后一个块中添加一定长度的字节，填充长度为使文本长度能够达到BLOCKSIZE整数倍的增加字节数，填充内容为填充长度字节重复。

示例：

已知string="YELLOW SUBMARINE"，长度为16字节，将其填充到20字节

paddingstring="YELLOW SUBMARINE\x04\x04\x04\x04"，需要填充的长度为4字节，因此在字符串最后添加填充“\x04\x04\x04\x04”

实现方法：

函数输入参数为待填充的字符串string，和需要的填充后的字符串长度length。

填充方法PKCS7padding：首先计算出字符串的长度psize，然后得到填充部分数组[psize]*psize，将其转化为字符串添加到原字符串后面。

去掉填充removePKCS7padding：取字符串最后一位转化为数字即为填充的长度padnum，返回0到len(string) - padnum长度的字符串，即为原始字符串。

```
def PKCS7padding(string, length):
    psize = length - len(string)
    pad = [psize] * psize
    return string + bytelist_to_str(pad)

def removePKCS7padding(string):
    padnum = ord(string[len(string) - 1])
    return string[ : len(string) - padnum]

string="YELLOW SUBMARINE"
paddingstring=PKCS7padding(string,20)
print paddingstring
print removePKCS7padding(paddingstring)
```

输出结果：

Challenge 10: Implement CBC mode

题目要求：在AES ECB的基础上，实现AES CBC。

CBC模式一种分组密码模式，允许我们加密不规则的消息，但是同样只能加密完整的块。在CBC模式中，下一次块加密之前，每一个密文块加到下一个明文块上。第一个明文块需要与一个初始化向量IV相加。

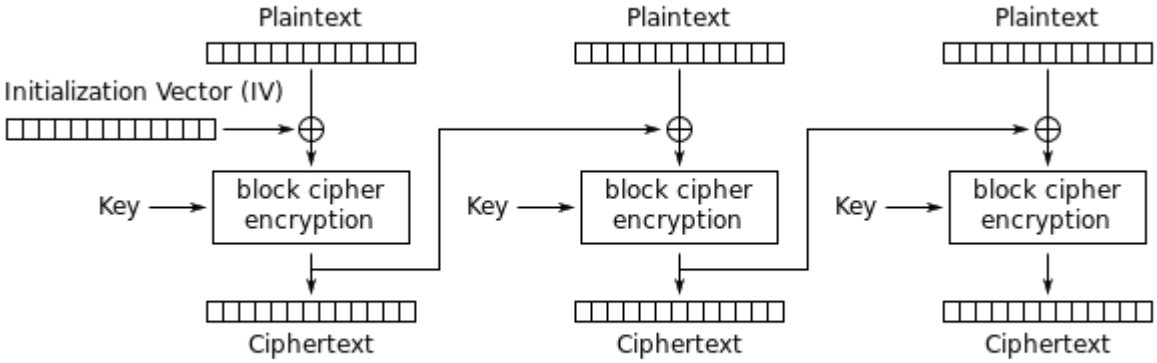
已知待解密的文本文件，密钥key，IV，来得到解密后的文本。

key="YELLOW SUBMARINE"

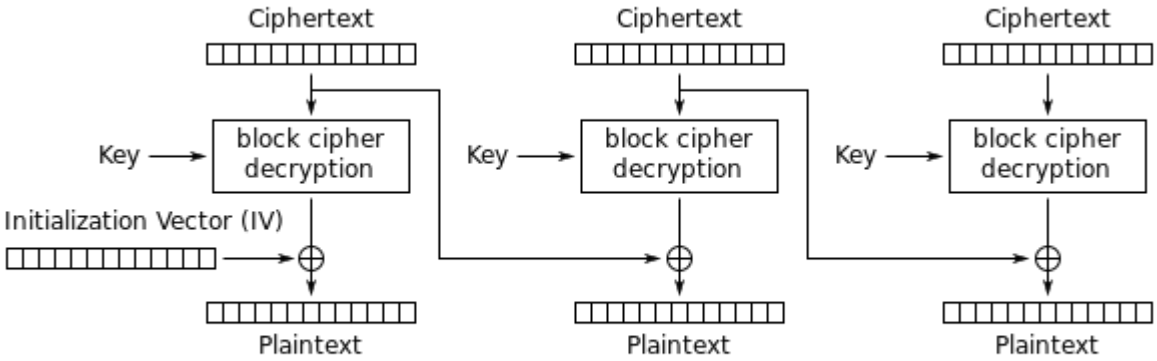
IV=全0

(ps：IV全0，以及固定IV、key，是一种密码学误用，十分不安全，建议不要使用；通常需要随机生成key和IV。)

实现方法：AES CBC加解密流程如下，其中block cipher encryption指代AES ECB加密，block cipher decryption指代AES ECB解密。（图源wiki）



Cipher Block Chaining (CBC) mode encryption



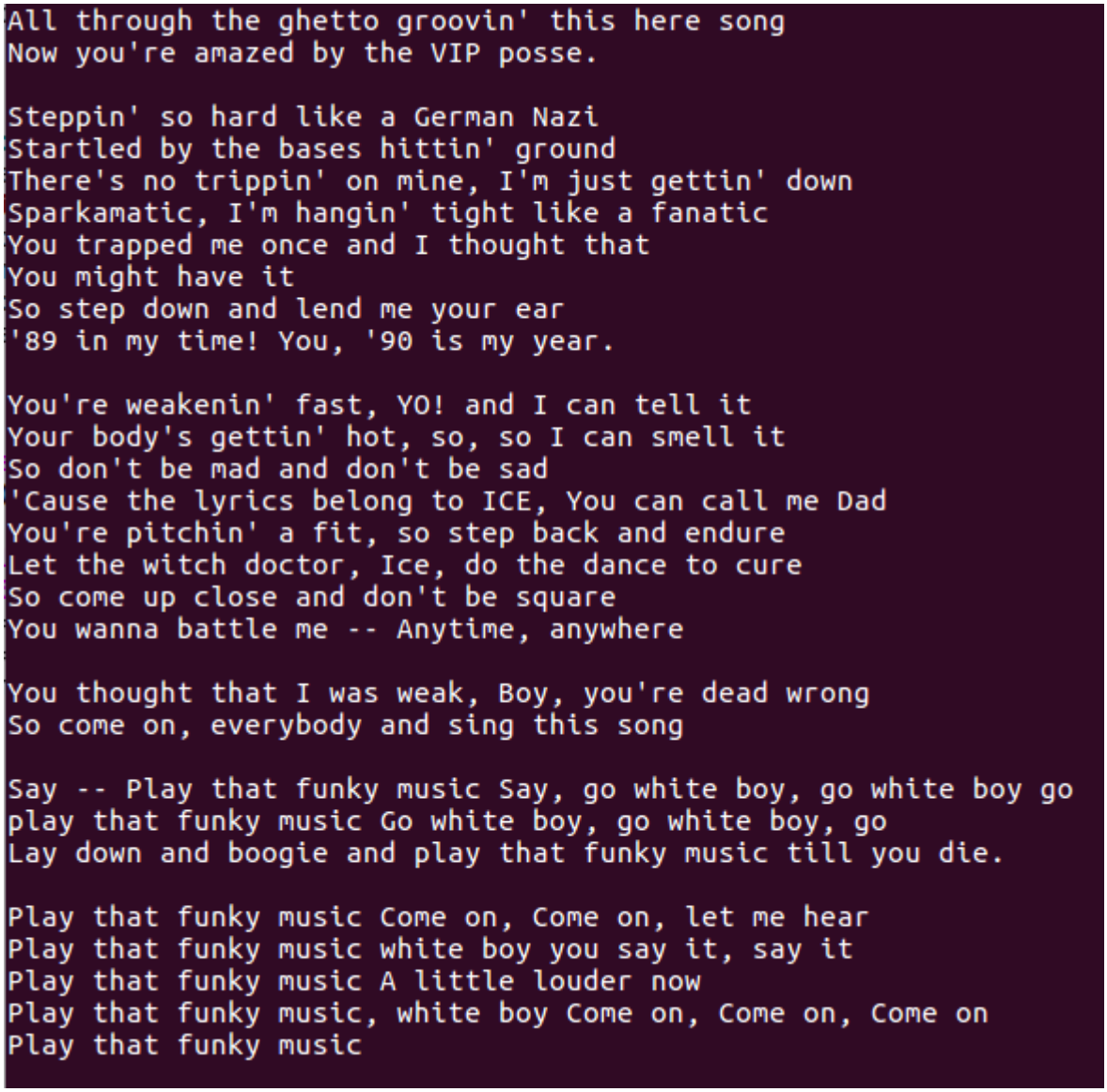
Cipher Block Chaining (CBC) mode decryption

```python
def aes_cbc_encrypt(key, iv, string):
    pnum = (len(string) / 16 + 1) * 16
    paddingString = PKCS7padding(string, pnum)
    ciphertext = ""
    for i in range(pnum / 16):
        block = str_to_bytelist(paddingString[i * 16 : i * 16 + 16])
        xorBlock = bytelist_to_str(xor(block, iv))
        encryptBlock = aes_ecb_encrypt(key, xorBlock)
        ciphertext += encryptBlock
        iv = str_to_bytelist(encryptBlock)
    return ciphertext

def aes_cbc_decrypt(key, iv, string):
    plaintext = ""
    for i in range(len(string) / 16):
        block = aes_ecb_decrypt(key, string[i * 16 : i * 16 + 16])
        xorBlock = xor(str_to_bytelist(block), iv)
        decryptBlock = bytelist_to_str(xorBlock)
        plaintext += decryptBlock
        iv = str_to_bytelist(string[i * 16 : i * 16 + 16])
    return plaintext

```

```
f = open('challenge10.txt', 'r')
s = f.read()
ciphertext = base64.b64decode(s)
key = "YELLOW SUBMARINE"
iv = [0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0]
plaintext = aes_cbc_decrypt(key, iv, ciphertext)
plaintext = removePKCS7padding(plaintext) #去掉明文末尾的填充
print plaintext
```

输出结果:

```
kydahe@kydahe-virtual-machine:~/文档/crypto$ python test.py
I'm back and I'm ringin' the bell
A rockin' on the mike while the fly girls yell
In ecstasy in the back of me
Well that's my DJ Deshay cuttin' all them Z's
Hittin' hard and the girlies goin' crazy
Vanilla's on the mike, man I'm not lazy.

I'm lettin' my drug kick in
It controls my mouth and I begin
To just let it flow, let my concepts go
My posse's to the side yellin', Go Vanilla Go!

Smooth 'cause that's the way I will be
And if you don't give a damn, then
Why you starin' at me
So get off 'cause I control the stage
There's no dissin' allowed
I'm in my own phase
The girlies sa y they love me and that is ok
And I can dance better than any kid n' play

Stage 2 -- Yea the one ya' wanna listen to
It's off my head so let the beat play through
So I can funk it up and make it sound good
1-2-3 Yo -- Knock on some wood
For good luck, I like my rhymes atrocious
Supercalafragilisticexpialidocious
I'm an effect and that you can bet
I can take a fly girl and make her wet.

I'm like Samson -- Samson to Delilah
There's no denyin', You can try to hang
But you'll keep tryin' to get my style
Over and over, practice makes perfect
But not if you're a loafer.

You'll get nowhere, no place, no time, no girls
Soon -- Oh my God, homebody, you probably eat
Spaghetti with a spoon! Come on and say it!

VIP. Vanilla Ice yep, yep, I'm comin' hard like a rhino
Intoxicating so you stagger like a wino
So punks stop trying and girl stop cryin'
Vanilla Ice is sellin' and you people are buyin'
'Cause why the freaks are jockin' like Crazy Glue
Movin' and groovin' trying to sing along
```



Challenge 11: An ECB/CBC detection oracle

题目要求：

实现三个函数

函数1——产生随机的AES密钥，16字节。

函数2——使用随机产生的未知密钥进行加密；加密前，在明文前面添加5-10字节随机值，在明文后面添加5-10字节随机值；随机选取ECB/CBC加密模式，概率各为1/2，如果是CBC还需要产生随机IV。

函数3——检测每次执行的块加密的模式（ECB or CBC）。

实现方法：

1. 随机数的生成，可以使用random.randint(0，255)方法来生成0到255之间的随机数字。
2. 加密流程：先在字符串前加5-10随机值，在字符串后面添加5-10随机值；然后对整个字符串进行PKCS#7填充；产生一个随机数，决定加密模式如果为1，则为EBC加密，否则为CBC加密；最后通过ECB的特性检测ECB模式，方法详情参考challenge 8。

```python
def generate_randomdata(length):
    data = []
    for i in range(length):
        data.append(random.randint(0, 255))
    return data

def random_encrypt(key,iv,string):
    flag = random.randint(1, 2)
    randomFront=bytelist_to_str(generate_randomdata(random.randint(5, 10)))
    randomBack = bytelist_to_str(generate_randomdata(random.randint(5, 10)))
    padRandomString = PKCS7padding(randomFront + string + randomBack)
    print flag
    if flag == 1:#ECB MODE
        ciphertext = aes_ecb_encrypt(key, padRandomString)
    else:#CBC MODE
        ciphertext = aes_cbc_encrypt(key, iv, padRandomString)
    return ciphertext

def detect_mode(ciphertext):
    blocks = [b for b in chunks(ciphertext,16)]
    detects = []
    for b in blocks:
        if blocks.count(b) > 1:
            detects.append(b)
    if detects:
        print("ECB MODE detected!")

key=bytelist_to_str(generate_randomdata(16))
iv=generate_randomdata(16)
string="I have a dream that one day every valley shall be exalted, and every hill and mountain shall be made low, the rough places will be made plain, and the crooked places will be made straight; and the glory of the Lord shall be revealed and all flesh shall see it together." * 10
ciphertext = random_encrypt(key, iv, string)
detect_mode(ciphertext)
```

输出结果：

```
kydahe@kydahe-virtual-machine:~/文档/crypto$ python test.py
2
kydahe@kydahe-virtual-machine:~/文档/crypto$ python test.py
1
ECB MODE detected!
```

Challenge 12: Byte-at-a-time ECB decryption (Simple)

题目要求：

1. 在加密前，在明文后面添加一段文本（添加之前需要对该文本进行Base64解码）。文本如下

```
Um9sbGluJyBpbiBteSA1LjAKV2l0aCBteSByYWctdG9wIGRvd24gc28gbXkg
aGFpciBjYW4gYmxvdwpUaGUgZ2lybGllcyBvbiBzdGFuZGJ5IHdhdmluZyBq
dXN0IHRvIHNheSBoaQpEaWQgeW91IHN0b3A/IE5vLCBJIGp1c3QgZHJvdmUg
YnkK
```

2. 使用一个恒定未知的密钥，通过ECB模式加密。加密函数格式为：

AES-128-ECB(your-string || unknown-string, random-key)

现在，已知加密函数的接口encrypt(string)，需要解密"unknown-string"。

实现方法：即解密方法。

（1）通过不断对"A"、"AA"、"AAA".....加密，通过密文块的长度以及填充规则，来获得加密块的大小。

（2）测试加密模式是否为ECB。

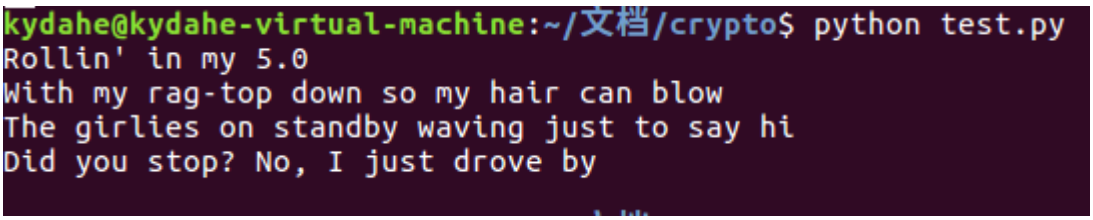
（3）将your-string设置为"AAAAAAA"，进行加密，提取第一个密文块，为"AAAAAAAB1"的加密结果，B1为unknown-string的第一个字符；然后尝试对"AAAAAAAX"加密，X为任意字符，同样提取第一个密文块，与刚才密文块对比，找到与其相同的密文块，即可以找到unknown-string的第一个字符。

（4）以此类推，即可求得unknown-string。

```python
unknownString = base64.b64decode("Um9sbGluJyBpbiBteSA1LjAKV2l0aCBteSByYWctdG9wIGRvd24gc28gbXkgaGFpciBjYW4gYmxvdwpUaGUgZ2lybGllcyBvbiBzdGFuZGJ5IHdhdmluZyBqdXN0IHRvIHNheSBoaQpEaWQgeW91IHN0b3A/IE5vLCBJIGp1c3QgZHJvdmUgYnkK")
key = bytelist_to_str(generate_randomdata(16))
def encrypt(string):
    plaintext = bytelist_to_str(string) + unknownString
    ciphertext = aes_ecb_encrypt(key, PKCS7padding(plaintext))
    return str_to_bytelist(ciphertext)

def detect_blocksize():
    ulen = len(encrypt([]))
    p1 = p2 = ''
    l1 = l2 = ulen
    while l1 == l2:
        p1 += 'A'
        l2 = len(encrypt(str_to_bytelist(p1)))
    l1 = l2
    while l1 == l2:
        p2 += 'A'
        l2 = len(encrypt(str_to_bytelist(p1 + p2)))
    #返回: unknown-string长度，填充长度，加密块大小
    return (ulen - (len(p1) - 1), len(p1) - 1, len(p2))

def generate_ciphertexts(buffer):#生成不同字符加密后的字典
    buffers = {}
```

```
    for b in range(256):
        buffers[b] = encrypt(buffer + [b])
    return buffers.items()

def guess_bytes(ulen, blocksize):
    buffer = [0x41] * (blocksize-1)
    count = 0
    recoveredBytes = []
    for i in range(ulen):
        if len(recoveredBytes) > 0 and len(recoveredBytes) % blocksize == 0:
            count += 1
            buffer.extend([0x41] * blocksize)
        ciphertexts = generate_ciphertexts(buffer[len(recoveredBytes):] + recoveredBytes)
        ciphertext = [c for c in chunks(encrypt(buffer[len(recoveredBytes):]), blocksize)][count]
        for b, cipher in ciphertexts:
            if [c for c in chunks(cipher, blocksize)][count] == ciphertext:
                recoveredBytes.append(b)
    return recoveredBytes

l, _, blocksize=detect_blocksize()
unknown_str = bytelist_to_str(guess_bytes(l, blocksize))
print removePKCS7padding(unknown_str)
```

输出结果：



Challenge 13: ECB cut-and-paste

题目要求：

有一个配置函数profile_for，类似于结构化cookie，格式如下

```
函数profile_for:
profile_for("foo@bar.com")产生以下
{
  email: 'foo@bar.com',
  uid: 10,
  role: 'user'
}
编码为 email=foo@bar.com&uid=10&role=user
```

profile_for函数中不允许&和=出现，这样也就不允许普通用户设置邮箱地址为"...(email)&role=admin"。

有两个函数用于加解密：

A. 产生一个随机AES密钥，加密编码后的用户配置文件；

B. 解密用户配置文件，并对其进行语法分析。

现在attacker获得了加密后的文件，使用profile_for()和密文本身，生成一个解密后role=admin的配置文件。

实现方法：

profile_for：通过字符串的replace方法来破化恶意信息。

parse_string：首先将使用split(';')对字符串进行分割，对分割后的每一块进行分析。以'='为基准，前面部分为关键字，后面部分为值，添加到account字典中。

attack()攻击方法：

1. 已知输入字符串前面需要添加"email="，长度为6，因此输入10个'A'，以使得后面有用的信息在第二个块中，并从块的起始位置开始。所以，可以输入"AAAAAAAAAAadmin\x0b\x0b\x0b\x0b\x0b\x0b\x0b\x0b\x0b\x0b\x0b"，"\x0b"可以当作填充部分。
2. 对其填充加密，取第二个密文块，即为"admin\x0b\x0b\x0b\x0b\x0b\x0b\x0b\x0b\x0b\x0b\x0b"的密文adminBlock。
3. 输入字符串"admin1@me.com"为管理员邮箱，经过profile_for函数配置，长度为32，再进行填充加密，密文长度应为48，填充为'\x0b'。
4. 取加密后的字符串的前32位，与adminBlock连接，即为"email=admin1@me.com&uid=10&role=admin\x0b\x0b\x0b\x0b\x0b\x0b\x0b\x0b\x0b\x0b\x0b"的密文。
5. 进行解密，就可以得到role=admin。

```
key = bytelist_to_str(generate_randomdata(16))
def profile_for(string):
    string.replace('&','_')
    string.replace('=','_')
    return 'email=' + string + '&uid=10&role=user'

def parse_string(string):
```

```
    account = {}
    parts = string.split('&')
    for kv in parts:
        k,v = kv.split('=')
        account[k] = v
    return account

def attack():
    admintext = profile_for('A'*10 + 'admin' + '\x0b'*0xb)
    adminCiphertext = aes_ecb_encrypt(key, PKCS7padding(admintext))
    adminBlock = adminCiphertext[16:32]
    #明文为'email=AAAAAAAAAAadmin\x0b\x0b\x0b\x0b\x0b\x0b\x0b\x0b\x0b\x0b\x0b'
    #在第二个块中包括admin加密后的结果

    plaintext = profile_for("admin1@me.com")#长度为36
    ciphertext = aes_ecb_encrypt(key, PKCS7padding(plaintext))
    adminCipher = cipher[:-16] + adminBlock
    #为"email=admin1@me.com&uid=10&role=admin\x0b\x0b\x0b\x0b\x0b\x0b\x0b\x0b\x0b\x0b\x0b"加密后的密文，填充刚好为\x0b
    adminplain = aes_ecb_decrypt(key, admincipher)#role=admin
    print plaintext
    print adminplain

attack()
```

输出结果：

```
kydahe@kydahe-virtual-machine:~/文档/crypto$ python test.py
email=admin1@me.com&uid=10&role=user
email=admin1@me.com&uid=10&role=admin
```

Challenge 14: Byte-at-a-time ECB decryption (Harder)

题目要求：在Challenge 12的基础上，还需要在明文前添加一段随机长度的随机字符串，则加密函数变为：

AES-128-ECB(random-prefix || attacker-controlled || target-bytes, random-key)

已知加密函数接口encrypt(string)，解密target-bytes。

实现方法：

与Challenge 12基本相同，要考虑random-prefix。

1. 求target-bytes的长度和加密块的大小。

（1）通过不断对"A"、"AA"、"AAA".....加密，通过密文块的长度以及填充规则，可以得到random-prefix+target-bytes的总长度以及加密块的大小。

（2）加密两个块大小的'A'列表，然后不断的添加'A'，当出现两个相同的块的时候，可以得到单独random-prefix的填充长度。

（3）加密三个块大小的'A'，找到其中两个相同块起始位置的offset，则第一个相同块前面部分为random-prefix及其填充，从而可以得到random-prefix的长度；从总长度中减掉random-prefix长度，即为target-bytes的长度。

2. 猜测得到random-prefix

（3）将your-string设置为"A" * random-prefix填充长度 + "AAAAAAA"，进行加密，提取第一个密文块，为"AAAAAAAB1"的加密结果，B1为unknown-string的第一个字符；然后尝试对"AAAAAAAX"加密，X为任意字符，同样提取第一个密文块，与刚才密文块对比，找到与其相同的密文块，即可以找到target-bytes的第一个字符。

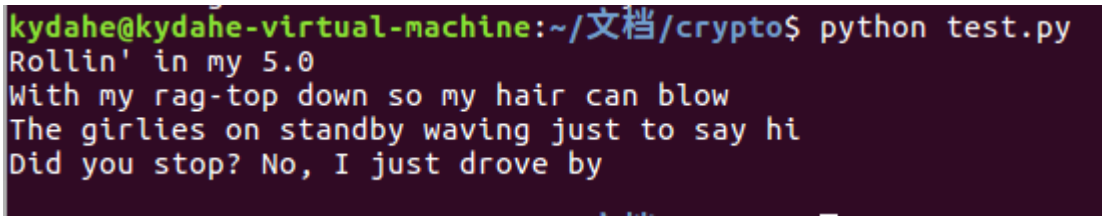
（4）以此类推，即可求得target-bytes。

```
unknownString = base64.b64decode("Um9sbGluJyBpbiBteSA1LjAKV2l0aCBteSByYWctdG9wIGRvd24gc28gbXkgaGFpciBjYW4gYmxvdwpUaGUgZ2lybGllcyBvbiBzdGFuZGJ5IHdhdmluZyBqdXN0IHRvIHNheSBoaQpEaWQgeW91IHN0b3A/IE5vLCBJIGp1c3QgZHJvdmUgYnkK")
key = bytelist_to_str(generate_randomdata(16))
randomPrefix = bytelist_to_str(generate_randomdata(random.randint(2, 32)))
def encrypt(string):
    plaintext = randomPrefix + bytelist_to_str(string) + unknownString
    ciphertext = aes_ecb_encrypt(key, PKCS7padding(plaintext))
    return str_to_bytelist(ciphertext)

def detect_blocksize():
    ruplen = len(encrypt([]))
    p1 = p2 = ''
    l1 = l2 = ruplen
    while l1 == l2:
        p1 += 'A'
        l2 = len(encrypt(str_to_bytelist(p1)))
    l1 = l2
    while l1 == l2:
        p2 += 'A'
        l2 = len(encrypt(str_to_bytelist(p1 + p2)))
    paddinglen = len(p1) - 1 #对于random-prefix+target-bytes的填充长度
    blocksize = len(p2)
    rulen = ruplen - paddinglen #random-prefix+target-bytes的长度

    #先加密两个块大小的buf，然后其中不断添加元素，直至在密文中可以得到两个相同的块，则对于单独random-prefix字符串的填充长度为len(buf)-blocksize*2
    buf = [0x41] * blocksize * 2
    flag = False
```

```
    while not flag and len(buf) < blocksize * 3:
        ciphertext = encrypt(buf)
        ctBlock = [c for c in chunks(ciphertext, blocksize)]
        for i in range(len(ctBlock) - 1):
            if ctBlock[i] == ctBlock[i + 1]:
                flag = True
        if not flag:
            buf += [0x41]
    rpadlen = len(buf) - blocksize * 2 #只有random-prefix的填充长度

    #加密三个块大小的buf，求出两个相同块的位置offset，则前面为random-prefix及填充
    offset = 0
    buf = [0x41]*blocksize*3
    ciphertext = encrypt(buf)
    cBlock = [c for c in chunks(ciphertext, blocksize)]
    for i in range(len(cBlock) - 1):
        if cBlock[i] == cBlock[i + 1]:
            offset = i
            break
    randomprefixlen = offset * blocksize - rpadlen #random-prefix长度
    #返回：random-prefix长度，target-bytes长度，加密块大小
    return (randomprefixlen, ruplen - paddinglen - randomprefixlen,
blocksize)

def generate_ciphertexts(buffer):
    buffers = {}
    for b in range(256):
        buffers[b] = encrypt(buffer + [b])
    return buffers.items()

def guess_bytes(randomprefixlen, ulen, blocksize):
    buffer = [0x41] * (blocksize - randomprefixlen % blocksize + blocksize
- 1)
    count = randomprefixlen / blocksize + 1
    recoveredBytes = []
    for i in range(ulen):
        if len(recoveredBytes) > 0 and len(recoveredBytes) % blocksize ==
0:
            count += 1
            buffer.extend([0x41] * blocksize)
        ciphertexts = generate_ciphertexts(buffer[len(recoveredBytes):] +
recoveredBytes)
        ciphertext = [c for c in
chunks(encrypt(buffer[len(recoveredBytes):]), blocksize)][count]
        for b, cipher in ciphertexts:
            if [c for c in chunks(cipher, blocksize)][count] == ciphertext:
                recoveredBytes.append(b)
    return recoveredBytes
```

```
70
71  rlen, ulen, blocksize = detect_blocksize()
72  unknown_str = bytelist_to_str(guess_bytes(rlen, ulen, blocksize))
73  print removePKCS7padding(unknown_str)
```

输出结果：



Challenge 15: PKCS#7 padding validation

题目要求：检查一段文本，是否为有效的PKCS#7填充，如果是，则去掉填充。

示例：

string1="ICE ICE BABY\x04\x04\x04\x04" 为有效填充，结果为"ICE ICE BABY"。

string2="ICE ICE BABY\x05\x05\x05\x05" 不是有效填充。

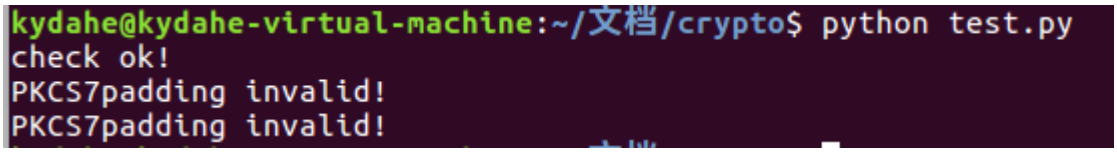
string3="ICE ICE BABY\x01\x02\x03\x04" 不是有效填充。

实现方法：

1. 提取字符串最后一位c，转化为10进制数，即为填充长度paddingCount。
2. 比较字符串的倒数paddingCount的位置上的字符是否等于c，如果相等，则为有效填充。

```
1  def checkPKCS7padding(string):
2      l = len(string)
3      c = string[l-1]
4      paddingCount = ord(c)
5      for i in range(paddingCount):
6          if string[l-1-i] != c:
7              print "PKCS7padding invalid!"
8              return False
9      print "check ok!"
10     return True
11
12 string1="ICE ICE BABY\x04\x04\x04\x04"
13 checkPKCS7padding(string1)
14 string2="ICE ICE BABY\x05\x05\x05\x05"
15 checkPKCS7padding(string2)
16 string3="ICE ICE BABY\x01\x02\x03\x04"
17 checkPKCS7padding(string3)
```

输出结果：



Challenge 16: CBC bitflipping attacks

题目要求：

首先生成一个随机AES密钥，然后实现两个功能。

功能1：对于userdata，在前面添加 "comment1=cooking%20MCs;userdata="，在字符串后面添加 ";comment2=%20like%20a%20pound%20of%20bacon"，然后对该文本进行填充和加密，返回加密后的结果。对于userdata的内容，不允许存在";"和"="。

功能2：解密字符串，语法分析查找是否存在"admin=true;"，返回True或False。

如果函数1实现正确，那么不会出现函数2中查找的字符串。

现在，需要做的是修改密文，基于CBC模式的比特翻转攻击，来使其可以找到。

实现方法：

profile_for(string)：输入用户数据字符串string，如果其中包含";"或"="，则替换为"="。将字符串前面和后面添加指定的字符串。

parsestring(string)：首先将使用split(';')对字符串进行分割，对分割后的每一块进行分析。以'='为基准，前面部分为关键字，后面部分为值，添加到account字典中。

attack()攻击方法：

1. 在userdata中如果输入";admin=true"，经过配置函数之后，会更改为"_admin_true"。
2. 我们可以计算出两个更改后的'_'位置为32和38；由于CBC模式是将前一个密文块异或到后一个明文块上进行加密的，因此我们可以通过更改相应密文的前一个密文块，来实现将其解密为';admin=true'。

```
key = bytelist_to_str(generate_randomdata(16))
iv = generate_randomdata(16)
def profile_for(string):
    string = string.replace(';', '_')
    string = string.replace('=', '_')
    return "comment1=cooking%20MCs;userdata=" + string +
";comment2=%20like%20a%20pound%20of%20bacon"

def parse_string(string):
    account = {}
    part = string.split(';')
    for kv in part:
        try:
```

```
            k,v = kv.split('=',1)
        except:
            k = kv
            v = ''
        account[k] = v
    return account

def encrypt(string):
    plaintext = profile_for(string)
    return aes_cbc_encrypt(key, iv, PKCS7padding(plaintext))

def decrypt(ciphertext):
    plaintext = aes_cbc_decrypt(key, iv, ciphertext)
    return parse_string(plaintext)

def attack():
    adminPlain = ";admin=true"

 #comment1=cooking%20MCs;userdata=_admin_true;comment2=%20like%20a%20pound%
20of%20bacon
    ciphertext = encrypt(adminPlain)
    adminCipher = str_to_bytelist(ciphertext)
    #第一个'_'下标为32，CBC通过更改前一个块的对应内容即下标16，来更改'_'
    #第二个'_'下标为38，前一个块对应下标为22
    adminCipher[16] = adminCipher[16] ^ ord('_') ^ ord(';')
    adminCipher[22] = adminCipher[22] ^ ord('_') ^ ord('=')
    plaintext = decrypt(bytelist_to_str(adminCipher))
    print decrypt(ciphertext)
    print plaintext

attack()
```

输出结果：

```
kydahe@kydahe-virtual-machine:~/文档/crypto$ python test.py
{'userdata': '_admin_true', 'comment1': 'cooking%20MCs', 'comment2': '%20like%20a%20poun
d%20of%20bacon\x0b\x0b\x0b\x0b\x0b\x0b\x0b\x0b\x0b\x0b\x0b\x10\x10\x10\x10\x10\x10\x10\x
10\x10\x10\x10\x10\x10\x10\x10\x10\x10'}
{'admin': 'true', 'comment1': 'cookingv\xb2(\x04 \xb7-\xdf\x94\xb9\xb4\xae\x8a\xccJ~', '
comment2': '%20like%20a%20pound%20of%20bacon\x0b\x0b\x0b\x0b\x0b\x0b\x0b\x0b\x0b\x0b\x0b
\x10\x10\x10\x10\x10\x10\x10\x10\x10\x10\x10\x10\x10\x10\x10\x10'}
```