the cryptopals crypto challenges

链接：https://cryptopals.com/

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

the cryptopals crypto challenges 中包含8组练习，共48个挑战。我们可以通过完成挑战，来学习密码学的相关知识。

这些挑战源于现实世界系统和加密结构的弱点，通过挑战中提供的足够的信息，我们可以了解基础密码学概念，以及常见的密码误用和密码算法的弱点。当完成这些挑战后，我们不仅可以了解密码系统的构建方式，还可以了解它们是如何受到攻击的。

Set 0: Start

以下内容运行环境为：VMware Workstation ubuntu-16.04.4虚拟机，Python 2.7.12。

这一部分不包括在挑战中，是为了后续解题方便自行添加的。

将一些基本的编码转换方法、基本的XOR异或运算、ECB加密、以及分块方法写到一个python程序中，以便后续解题时方便调用。

Set 0 包括：十六进制、字节数组、字符串两两之间的相互转化，字节数组的异或XOR运算，AES-ECB加密算法，分块方法。

如果对这些算法概念熟悉，可以跳过这部分，后面的某些挑战需要调用这些方法。当然可能还会有其他的方法或手段来实现这些方法，这里我们只列出了一种方法。

related method:

- 十六进制字符串转化为字节数组

  先将十六进制字符串解码，对每个字符取ASCII值，然后添加到列表中

  ```
  def hex_to_bytelist(hexString):
      return [ord(c) for c in hexString.decode("hex")]
  ```

- 字节数组转化为十六进制字符串

  先将数组中的每一个元素转化为十六进制hex()，如果x<15，需要在前面补'0'，将这些十六进制字符串连接起来

  ```
  def bytelist_to_hex(byteList):
      return "".join(hex(x)[2:] if x > 15 else '0' + hex(x)[2:] for x in byteList)
  ```

- 字符串转化为十六进制字符串

  直接将字符串编码为十六进制

  ```
  def str_to_hex(string):
      return string.encode("hex")
  ```

- 十六进制字符串转化为字符串

  直接将十六进制字符串解码为字符串

```
def hex_to_str(hexString):
    return hexString.decode("hex")
```

- 字节数组转化为字符串

  利用前面的方法，可以先将字节数组转化为十六进制，然后在调用十六进制转化为字符串的方法

  （ps：还有其他的方法，这里只说明一种）

```
def bytelist_to_str(byteList):
    return hex_to_str(bytelist_to_hex(byteList))
```

- 字符串转化为字节数组

  利用前面的方法，先将字符串转化为十六进制，然后再将十六进制字符串转化为字节数组

```
def str_to_bytelist(string):
    return hex_to_bytelist(str_to_hex(string))
```

- XOR

```
#字节数组异或
def xor(b1, b2):
    res = []
    for i in range(len(b1)):
        res.append(b1[i] ^ b2[i])
    return res
```

- AES ECB

  直接使用from Crypto.Cipher import AES调用模块，构造AES ECB生成器，调用其encrypt和decrypt方法进行加解密。

```
def aes_ecb_encrypt(key, plaintext):
    crypto = AES.new(key, AES.MODE_ECB)
    ciphertext = crypto.encrypt(plaintext)
    return ciphertext

def aes_ecb_decrypt(key, ciphertext):
    crypto = AES.new(key, AES.MODE_ECB)
    plaintext = crypto.decrypt(ciphertext)
    return plaintext
```

- 分块

  将字符串string以n字节长度的大小进行切分，切分后的每块合为一个列表。

```
def chunks(string, n):
    for i in xrange(0, len(string), n):
        yield s[i:i+n]
```

Set 1: Basics

第一部分是基础知识相关的，比较容易，包括一些密码学编码、常用的基础加密方式以及简单的破解方式等。

目录：

Challenge 1: Convert hex to base64 Challenge 2: Fixed XOR Challenge 3: Single-byte XOR cipher Challenge 4: Detect single-character XOR Challenge 5: Implement repeating-key XOR Challenge 6: Break repeating-key XOR Challenge 7: AES in ECB mode Challenge 8: Detect AES in ECB mode

Challenge 1 : Convert hex to base64

题目要求：实现一个函数，将十六进制字符串转化为base64编码字符串。

示例：

输入十六进制字符串string = "49276d206b696c6c696e6720796f757220627261696e206c696b65206120706f69736f6e6f7573206d757368726f6f6d"

返回base64编码result = "SSdtIGtpbGxpbmcgeW91ciBicmFpbiBsaWtlIGEgcG9pc29ub3VzIG11c2hyb29t"

实现方法：

1. 首先将十六进制字符串解码，decode方法是将字符串解码为unicode编码。
2. 然后再将字符串编码为base64

```
def hex_to_base64(hexString):
    return base64.b64encode(hexString.decode('hex'))

string="49276d206b696c6c696e6720796f757220627261696e206c696b65206120706f69736f6e6f7573206d757368726f6f6d"
print hex_to_base64(string)
```

输出结果：

```
kydahe@kydahe-virtual-machine:~/文档/crypto$ python test.py
SSdtIGtpbGxpbmcgeW91ciBicmFpbiBsaWtlIGEgcG9pc29ub3VzIG11c2hyb29t
```

Challenge 2: Fixed XOR

题目要求：实现一个函数，输入为两个等长的字符串，输出为这两个字符串的异或结果。功能为实现两个等长的字符串异或。

示例：输入s1，s2，输出为res

s1 = "1c0111001f010100061a024b53535009181c"

s2 = "686974207468652062756c6c277320657965"

res = "746865206b696420646f6e277420706c6179"

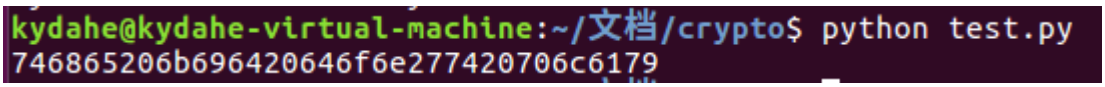
实现方法：

Set 0 中的xor方法是实现两个字节数组的异或。

1. 若要实现两个等长的字符串的异或，可以先使用Set 0 中的str_to_bytelist方法将字符串转化为字节数组
2. 然后调用xor方法进行异或操作
3. 最后在调用bytelist_to_hex方法将字节数组转化为十六进制字符串。

```
def fixed_xor(s1, s2):
    return bytelist_to_hex(xor(hex_to_bytelist(s1), hex_to_bytelist(s2)))

s1 = "1c0111001f010100061a024b53535009181c"
s2 = "686974207468652062756c6c277320657965"
print fixed_xor(s1, s2)
```

输出结果：



Challenge 3: Single-byte XOR cipher

题目要求：给定一个十六进制字符串hexString，它是某个字符串与一个单字符key异或进行加密之后得到的结果。现在已知hexString，我们需要找到key，来解密字符串。

hexString="1b37373331363f78151b7f2b783431333d78397828372d363c78373e783a393b3736"

实现方法：

1. 在这个挑战中，我们可以通过遍历0-255的字符key，对hexString进行逐字符异或解密；
2. 通过解密后的英文文本中的字符频率，来对测试key解密结果进行评估；
3. 如果为正常的英文文本，那么它的字符频率应该尽可能的大，选择其中得分最高的key，即有很大概率为正确的key。

其中，CHARACTER_FREQ为字符频率表（ps: 使用当前wiki的字母频率解密不正确，猜测可能是字符频率发生了变化）

get_score方法：输入参数为待评估的字符串，输出为该字符的频率得分。通过找到字符串中的每一个字符在字符频率表中对应的频率，相加之和，即为该字符串的频率得分。

singlebyte_xor方法：该方法实现单字符异或加密。输入为单字符密钥key和待加密的字符串，输出为加密后的字符串。通过将待加密的字符串中的每一个字符与单字符密钥key进行异或，从而进行加密，返回加密结果（ps：加密解密相同）。

traversal_singlebyte方法：输入为解密的字符串hexString，输出为解密结果。在该方法中，遍历0-255的字符key，调用singlebyte_xor方法对输入字符串进行解密，调用get_score方法对解密后的字符串进行评估。使用列表的方式存储每个密钥key及其对应的信息，列表中的元素是字典类型，包括key、解密结果plaintext以及字符频率得分score。使用sorted方法对列表进行排序，取得字符频率得分最高的解密结果返回。

```
CHARACTER_FREQ = { #字符频率表
    '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}

def get_score(string): #计算字符串的频率得分
    score = 0
    for ch in string:
        ch = ch.lower()
        if ch in CHARACTER_FREQ:
            score += CHARACTER_FREQ[ch]
    return score

def singlebyte_xor(key, string): #单字符异或加密
    res = ""
    for i in string:
        ch = chr(key ^ ord(i))
        res += ch
    return res

def traversal_singlebyte(string): #遍历单个字符解密，评估
    candidate = []
    for key in range(256):
        plaintext = singlebyte_xor(key, string)
        score = get_score(plaintext)
        res={'key': key, 'plaintext': plaintext, 'score': score}
        candidate.append(res)
    #取得分最高的返回
    return sorted(candidate, key = lambda c:c['score'])[-1]

hexString="1b37373331363f78151b7f2b783431333d78397828372d363c78373e783a393b3736"
print traversal_singlebyte(hexString.decode('hex'))
```

输出结果：

```
kydahe@kydahe-virtual-machine:~/文档/crypto$ python test.py
{'plaintext': "Cooking MC's like a pound of bacon", 'score': 2.2641049, 'key': 88}
```

Challenge 4: Detect single-character XOR

题目要求：在这个挑战中，给出一个文件challenge4.txt，文件中的一个字符串是由单字符异或方法加密的，再进行十六进制编码，找到并解密这个字符串。

实现方法：

1. 参考challenge 3 实现，对文件中的每个字符串调用traversal_singlebyte方法，进行遍历和评估
2. 对于文件中的每个字符串都会返回一个得分最高的对象，选取这其中分数最高的一个。

```
f = open('challenge4.txt', 'r')
hexStringPool = f.readlines() #读取字符串
candidate = [] #存放对于每个字符串来说单字符异或加密得分最高的结果
for line in hexStringPool:
    line = line.strip() #移除字符串头尾的空格或换行符
    string = line.decode('hex') #16进制解码
    #对每个字符串调用traversal_singlebyte方法，将得分最高的存放在candidate列表中
    candidate.append(traversal_singlebyte(string))

#返回列表中得分最高的一项的解密结果，即为所要寻找的字符串解密结果。
print sorted(candidate, key = lambda c:c['score'])[-1]['plaintext']
```

输出结果：

```
kydahe@kydahe-virtual-machine:~/文档/crypto$ python test.py
Now that the party is jumping
```

Challenge 5: Implement repeating-key XOR

题目要求：实现一个函数，给定待加密的字符串和密钥，该函数通过使用重复密钥异或（repeating-key XOR）的方法，加密一段英文文本，返回加密后的结果。

示例：

待加密的字符串 string = "Burning 'em, if you ain't quick and nimble\nI go crazy when I hear a cymbal"

密钥 key = "ICE"

重复密钥异或加密结果 res = "0b3637272a2b2e63622c2e69692a23693a2a3c6324202d623d63343c2a26226324272765272a282b2f20430a652e2c652a3124333a653e2b2027630c692b20283165286326302e27282f"

实现方法：

1. 我们可以将待加密的字符串按keysize的大小进行分块
2. 将每块分别与密钥key进行逐字符异或加密
3. 最后将每块的加密结果合并到一起。

异或方法与challenge 2 类似，先将每块字符串和密钥转化为字节数组，调用xor方法进行异或操作，然后将结果转化为十六进制字符串，最后将每一个块的加密结果合并在一起。

```
def repeatingkey_xor(string, key):
    res="" #字符串连接，存放每一次加密后的结果
    #xrange方法使得每次处理len(key)长度的字符串
    for i in xrange(0, len(string), len(key)):
        res += bytelist_to_hex(xor(str_to_bytelist(string[i:i+len(key)]), str_to_bytelist(key)))
    return res

string = "Burning 'em, if you ain't quick and nimble\nI go crazy when I hear a cymbal"
key = "ICE"
print repeatingkey_xor(string, key)
```

输出结果：

```
kydahe@kydahe-virtual-machine:~/文档/crypto$ python test.py
0b3637272a2b2e63622c2e69692a23693a2a3c6324202d623d63343c2a26226324272765272a
282b2f20430a652e2c652a3124333a653e2b2027630c692b20283165286326302e27282f
```

Challenge 6: Break repeating-key XOR

题目要求：在该挑战中提供了一个文件challenge6.txt，文件中是使用重复密钥异或方法加密后，再经过base64编码后得到的文本，我们需要找到密钥，对其进行解密。

实现方法：

（1）猜测密钥长度KEYSIZE，尝试2到40

（2）汉明距离hamming_distance，即为两个字符串异或之后，二进制格式1的个数

（3）对于每个KEYSIZE，求得每个解密块两两之间的汉明距离，对KEYSIZE取平均值（大概选取四个KEYSIZE块就够了）

（4）选取2-3个最小汉明距离的KEYSIZE（如果选取了正确的密钥长度，密文块与密文块两两之间的汉明距离等于对应的明文块与明文块两两之间的汉明距离，两两块之间的汉明距离的值应该趋于小）

（5）获得KEYSIZE之后，对每一块的对应字节组合后的字符串与单字符异或，从而可以破解密钥。（例如，每一块的第一字节组合后的字符串，如果分组正确，那么他们所对应的密钥字节是相同的，这是可以使用破解单字符异或的方法，来逐字符的破解密钥）

（6）对不同KEYSIZE解密后的明文进行评估，选取得分最高的一组。

```
def hamming_distance(s1, s2):
        dis = 0
        for i in range(min(len(s1), len(s2))):
            b = bin(ord(s1[i]) ^ ord(s2[i]))
            dis += b.count('1')
        return dis

def guess_keysize(string):
    keys = []
    for keysize in range(2, 40):
        blocks = []
        count = 0
        dis = 0
        for i in range(0, len(string), keysize):
            count += 1
            blocks.append(string[i:i+keysize])
            if count == 4:
                break
        #选取四个块，两两组合求汉明距离
        pairs = itertools.combinations(blocks, 2)
        for (x, y) in pairs:
            dis += hamming_distance(x, y)
        ndis = dis / keysize
        key = {'keysize': keysize, 'distance': ndis}
        keys.append(key)
    return sorted(keys, key=lambda c:c['distance'])[0:3]

def guess_key(keysize, string):
```

```
    key = ''
    for i in range(keysize):
        now_str = ''
        #获取每个块相同位置的字符
        for index, ch in enumerate(string):
            if index % keysize == i:
                now_str += ch
        key += chr(traversal_singlebyte(now_str)['key'])
    return key

def break_repeatingkey_xor(string):
    keysizes = guess_keysize(string)
    candidate = []
    plains = []
    for keysize in keysizes:
        key = guess_key(keysize['keysize'], string)
        #二元组：重复密钥异或解密明文，对应密钥key
        plains.append((hex_to_str(repeatingkey_xor(string, key)), key))
    return sorted(plains, key=lambda c:get_score(c[0]))[-1]

f = open('challenge6.txt', 'r')
s = f.read()
string=base64.b64decode(s)
res=break_repeatingkey_xor(string)
print 'plaintext: \n'+res[0]
print 'key: \n'+res[1]
```

输出结果：

```
kydahe@kydahe-virtual-machine:~/文档/crypto$ python test.py
plaintext:
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
All through the ghetto groovin' this here song
Now you're amazed by the VIP posse.

Steppin' so hard like a German Nazi
Startled by the bases hittin' ground
There's no trippin' on mine, I'm just gettin' down
Sparkamatic, I'm hangin' tight like a fanatic
You trapped me once and I thought that
You might have it
So step down and lend me your ear
'89 in my time! You, '90 is my year.

You're weakenin' fast, YO! and I can tell it
Your body's gettin' hot, so, so I can smell it
So don't be mad and don't be sad
'Cause the lyrics belong to ICE, You can call me Dad
You're pitchin' a fit, so step back and endure
Let the witch doctor, Ice, do the dance to cure
So come up close and don't be square
You wanna battle me -- Anytime, anywhere

You thought that I was weak, Boy, you're dead wrong
So come on, everybody and sing this song

Say -- Play that funky music Say, go white boy, go white boy go
play that funky music Go white boy, go white boy, go
Lay down and boogie and play that funky music till you die.

Play that funky music Come on, Come on, let me hear
Play that funky music white boy you say it, say it
Play that funky music A little louder now
Play that funky music, white boy Come on, Come on, Come on
Play that funky music

key:
Terminator X: Bring the noise
```

Challenge 7: AES in ECB mode

题目要求：该挑战提供了一个文件challenge7.txt，文件中是经过AES-128 ECB模式加密后的base64编码的文本，已知密钥key，对加密后的文本进行解密。

key = "YELLOW SUBMARINE"

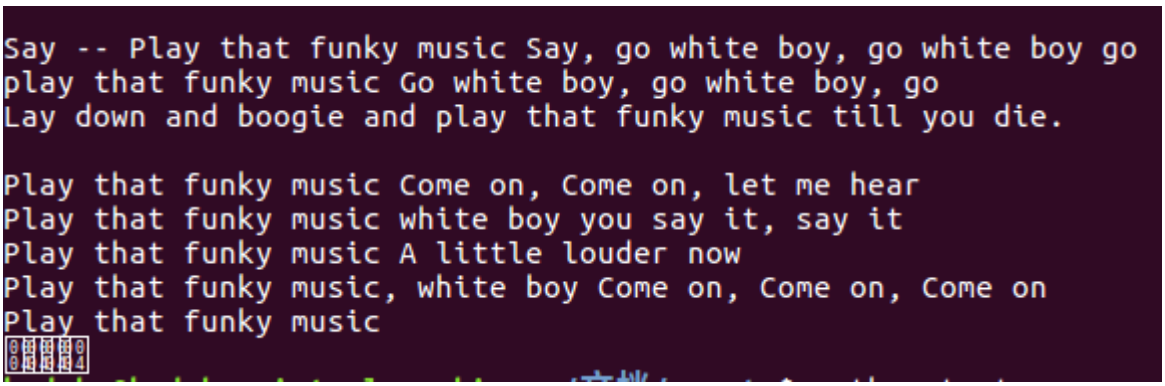
实现方法：

直接使用from Crypto.Cipher import AES调用模块，构造AES ECB生成器，调用其encrypt和decrypt方法进行加解密。注意到解密后的明文中末尾部分包含填充，因此我们可以使用removePKCS7padding方法去掉填充（方法的具体实现见challenge 9）

```
def aes_ecb_encrypt(key, plaintext):
    crypto = AES.new(key, AES.MODE_ECB)
    ciphertext = crypto.encrypt(plaintext)
    return ciphertext

def aes_ecb_decrypt(key, ciphertext):
    crypto = AES.new(key, AES.MODE_ECB)
    plaintext = crypto.decrypt(ciphertext)
    return plaintext

f = open('challenge7.txt', 'r')
s = f.read()
ciphertext = base64.b64deocde(s)
key = "YELLOW SUBMARINE"
plaintext = aes_ecb_decrypt(key, ciphertext) #这一步解密后的明文包含填充部分
plaintext = removePKCS7padding(plaintext) #去掉明文末尾的填充
print plaintext
```

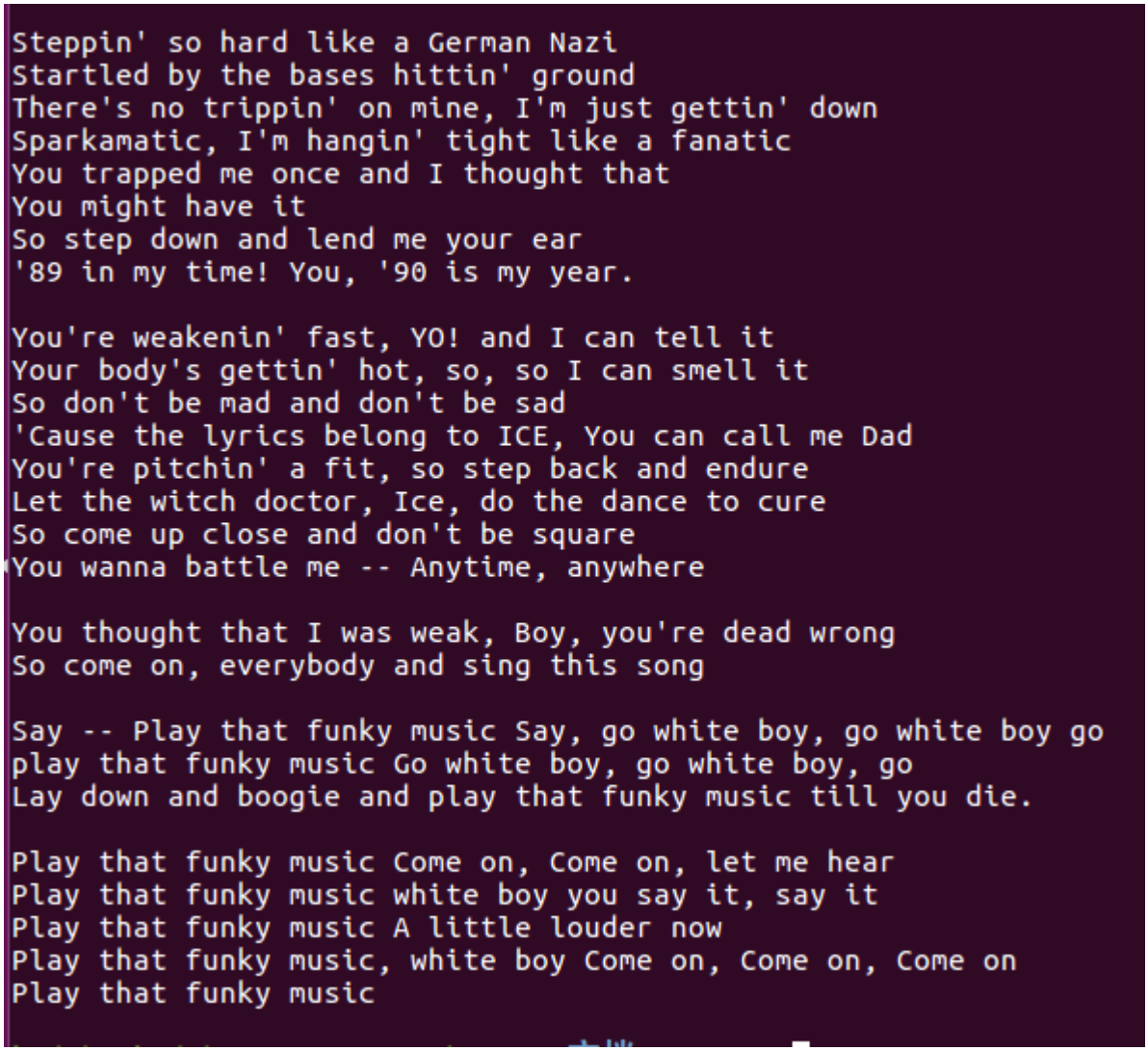
输出结果：

（1）未去掉填充的末尾结果



（2）去掉填充后的完整结果

```
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
All through the ghetto groovin' this here song
Now you're amazed by the VIP posse.
```



Challenge 8: Detect AES in ECB mode

题目要求：该挑战提供一个文件，文件内容是十六进制编码的若干加密字符串，其中一个字符串是通过AES ECB模式加密的，key未知。我们需要通过密文找到这个AES ECB加密的字符串。

实现方法：

由于ECB模式加密是静态和确定的，相同的16字节明文块总是生成相同16字节密文块。所以我们可以通过查找加密字符串中是否有相同的密文块，来判断是否为ECB模式。如果存在，则为ECB模式。

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

f = open('challenge8.txt', 'r')
i = 1
```

```
for line in f:
    print i, ': ', detect_ecb(line)
    i = i + 1
```

输出结果：只有第133行的字符串被检测出ECB模式