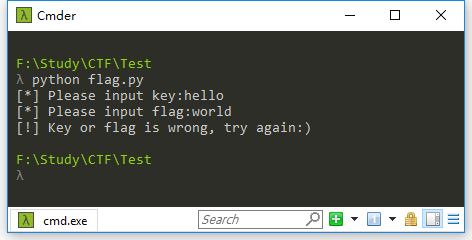
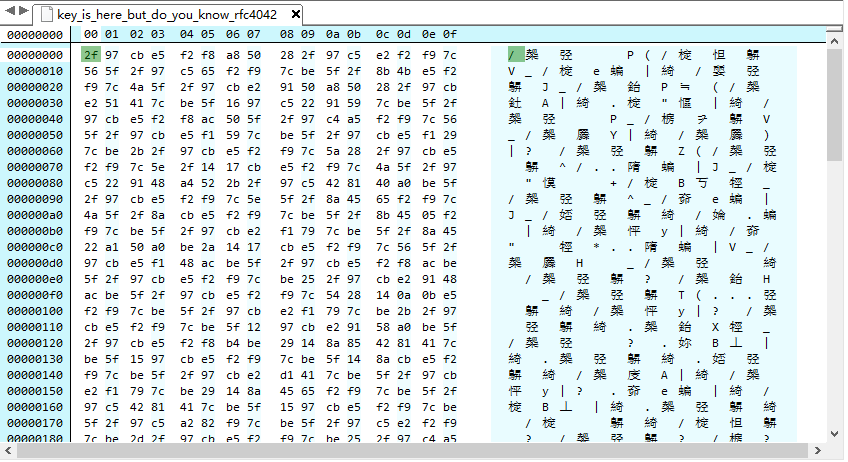
1. 获取key

下载得到题目包you\_need\_python.7z，解压得到flag.py和key\_is\_here\_but\_do\_you\_know\_rfc4042和两个文件，尝试运行flag.py：



要求输入key，由另一个文件名key\_is\_here\_but\_do\_you\_know\_rfc4042，可知key需要从key\_is\_here\_but\_do\_you\_know\_rfc4042中获得，查看key\_is\_here\_but\_do\_you\_know\_rfc4042文件的内容：

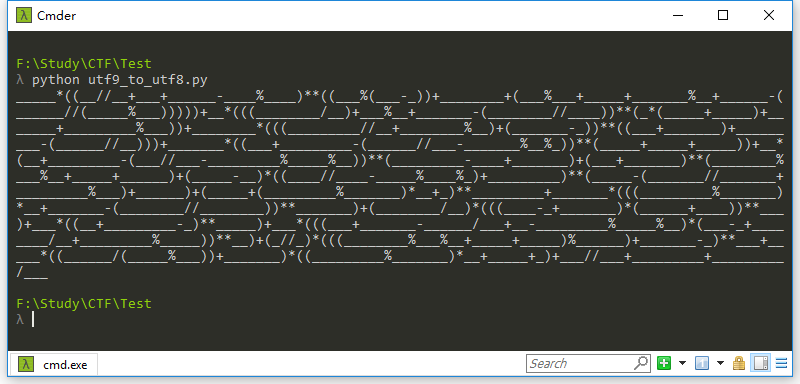


文件内容是一些乱码，但是文件名提示了rfc4042，于是网上查询相关资料，知道rfc4042中定义了utf9和utf18两种Unicode转换编码格式，在了解转换原理之后，根据题意推测是将uft9编码转化了utf8编码，通过资料查询得知python已有utf9模块（如果没有查询到也可以自己动手编写转换代码），使用pip安装utf9模块：

1. pip search utf9
2. pip install utf9

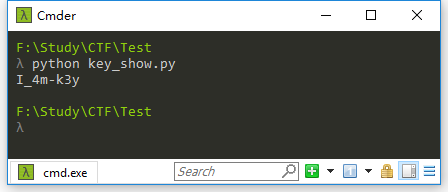
编写代码将文件内容的uft9编码转化utf8编码：

1. #coding utf-8
2. #utf9\_to\_utf8.py
3. **import** utf9
5. utf9\_file = open('key\_is\_here\_but\_do\_you\_know\_rfc4042','rb')
6. utf9\_data = utf9\_file.read()
7. decoded\_data = utf9.utf9decode(utf9\_data)
8. **print** decoded\_data
9. decoded\_file = open('decoded','w')
10. decoded\_file.write(decoded\_data)
11. decoded\_file.close()



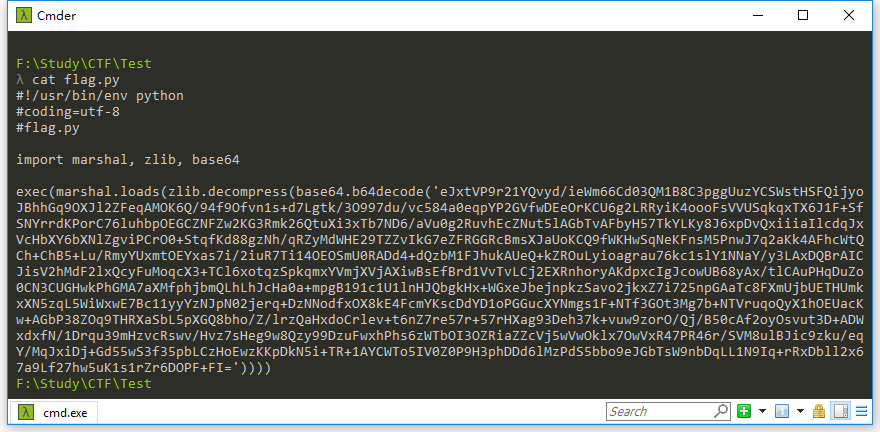
得到一堆符号串，但是经常仔细观察，除了“\_”符号外，其他符号都是Python中的算数运算符，“(”，“)”括号表示优先级，然后开脑洞“\_”为数字“1”，“\_\_”为数字“2”，依次类推“\_\_\_\_\_\_\_\_\_”为数字“9”，在熟悉了utf9模块的使用后尝试编写转换代码，代码执行后得到数字：5287002131074331513，尝试转换为16进制然后转换为ASCII字符：

1. #coding=utf-8
2. #key\_show.py
3. **import** binascii
5. \_ = 1
6. \_\_ = 2
7. \_\_\_ = 3
8. \_\_\_\_ = 4
9. \_\_\_\_\_ = 5
10. \_\_\_\_\_\_ = 6
11. \_\_\_\_\_\_\_ = 7
12. \_\_\_\_\_\_\_\_ = 8
13. \_\_\_\_\_\_\_\_\_ = 9
15. **print** binascii.a2b\_hex(hex(eval("\_\_\_\_\_\*((\_\_//\_\_+\_\_\_+\_\_\_\_\_\_-\_\_\_\_%\_\_\_\_)\*\*((\_\_\_%(\_\_\_-\_))+\_\_\_\_\_\_\_\_+(\_\_\_%\_\_\_+\_\_\_\_\_+\_\_\_\_\_\_\_%\_\_+\_\_\_\_\_\_-(\_\_\_\_\_\_//(\_\_\_\_\_%\_\_\_)))))+\_\_\*(((\_\_\_\_\_\_\_\_/\_\_)+\_\_\_%\_\_+\_\_\_\_\_\_\_-(\_\_\_\_\_\_\_\_//\_\_\_\_))\*\*(\_\*(\_\_\_\_\_+\_\_\_\_\_)+\_\_\_\_\_\_\_+\_\_\_\_\_\_\_\_\_%\_\_\_))+\_\_\_\_\_\_\_\_\*(((\_\_\_\_\_\_\_\_\_//\_\_+\_\_\_\_\_\_\_\_%\_\_)+(\_\_\_\_\_\_\_-\_))\*\*((\_\_\_+\_\_\_\_\_\_\_)+\_\_\_\_\_\_\_\_\_-(\_\_\_\_\_\_//\_\_)))+\_\_\_\_\_\_\_\*((\_\_\_+\_\_\_\_\_\_\_\_\_-(\_\_\_\_\_\_//\_\_\_-\_\_\_\_\_\_\_%\_\_%\_))\*\*(\_\_\_\_\_+\_\_\_\_\_+\_\_\_\_\_))+\_\_\*(\_\_+\_\_\_\_\_\_\_\_\_-(\_\_\_//\_\_\_-\_\_\_\_\_\_\_\_\_%\_\_\_\_\_%\_\_))\*\*(\_\_\_\_\_\_\_\_\_-\_\_\_\_+\_\_\_\_\_\_\_)+(\_\_\_+\_\_\_\_\_\_\_)\*\*(\_\_\_\_\_\_\_\_%\_\_\_%\_\_+\_\_\_\_\_+\_\_\_\_\_\_)+(\_\_\_\_\_-\_\_)\*((\_\_\_\_//\_\_\_\_-\_\_\_\_\_%\_\_\_\_%\_)+\_\_\_\_\_\_\_\_\_)\*\*(\_\_\_\_\_-(\_\_\_\_\_\_\_//\_\_\_\_\_\_\_+\_\_\_\_\_\_\_\_\_%\_\_\_)+\_\_\_\_\_\_)+(\_\_\_\_\_+(\_\_\_\_\_\_\_\_\_%\_\_\_\_\_\_\_)\*\_\_+\_)\*\*\_\_\_\_\_\_\_\_\_+\_\_\_\_\_\_\_\*(((\_\_\_\_\_\_\_\_\_%\_\_\_\_\_\_\_)\*\_\_+\_\_\_\_\_\_\_-(\_\_\_\_\_\_\_\_//\_\_\_\_\_\_\_\_))\*\*\_\_\_\_\_\_\_)+(\_\_\_\_\_\_\_\_/\_\_)\*(((\_\_\_\_-\_+\_\_\_\_\_\_\_)\*(\_\_\_\_\_\_+\_\_\_\_))\*\*\_\_\_)+\_\_\_\*((\_\_+\_\_\_\_\_\_\_\_\_-\_)\*\*\_\_\_\_\_)+\_\_\_\*(((\_\_\_+\_\_\_\_\_\_\_-\_\_\_\_\_\_/\_\_\_+\_\_-\_\_\_\_\_\_\_\_\_%\_\_\_\_\_%\_\_)\*(\_\_\_-\_+\_\_\_\_\_\_\_\_/\_\_+\_\_\_\_\_\_\_\_\_%\_\_\_\_\_))\*\*\_\_)+(\_//\_)\*(((\_\_\_\_\_\_\_\_%\_\_\_%\_\_+\_\_\_\_\_+\_\_\_\_\_)%\_\_\_\_\_\_)+\_\_\_\_\_\_\_-\_)\*\*\_\_\_+\_\_\_\_\_\*((\_\_\_\_\_\_/(\_\_\_\_\_%\_\_\_))+\_\_\_\_\_\_\_)\*((\_\_\_\_\_\_\_\_\_%\_\_\_\_\_\_\_)\*\_\_+\_\_\_\_\_+\_)+\_\_\_//\_\_\_+\_\_\_\_\_\_\_\_\_+\_\_\_\_\_\_\_\_\_/\_\_\_"))[2:][:-1])



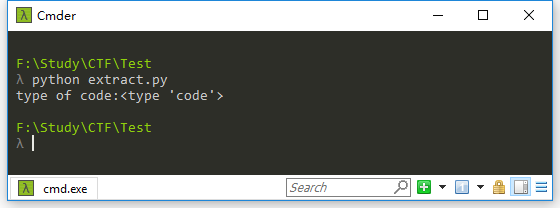
得到key：I\_4m-k3y

1. 分析flag.py



查询有关marshal, zlib, base64模块和exec函数的资料，反向推测源代码或字节码先是使用marshal模块序列化，之后使用zlib压缩，最后使用base64编码，而exec语句可以用来执行储存在字符串或者文件中的python语句或python字节码。所以尝试提取exec语句执行的内容：

1. **import** marshal, zlib, base64
3. code = marshal.loads(zlib.decompress(base64.b64decode('')))
4. **print** "type of code:%s" %type(code)



得知code类型为Python代码对象（PyCodeObject），为了便于题目讲解，以下简单阐述个人对Python解释器运行原理，pyc文件格式以及Python代码对象的理解。

当你使用命令python demo.py（这里的python其实是python解释器中的CPython）时 ，会启动 Python解释器，Python解释器首先会检查当前目录是否存在相应的demo.pyc或demo.pyo文件（pyo文件是经过Python解释器编译优化，然后将内存中的字节码对象序列化并加上pyo文件头信息的可储存的二进制文件），如果都存在优先检查运行demo.pyo，如果不存在相应的demo.pyc或demo.pyo那么Python解释器就会编译demo.py，因为Python解释器检查运行pyc文件和pyo文件两者过程类似，下面以pyc文件为例，如果存在相应的pyc文件Python解释器则验证demo.pyc文件头的前四个字节幻数（magic number）的值，以确保当前文件是pyc文件格式且当前Python解释器版本能支持，如果验证不通过则给出提示退出，如果验证通过继续检查demo.pyc文件头的后四个字节py源文件修改时间（mtime，modify time），如果demo.pyc文件头的mtime和现在demo.py源文件的修改时间一样，那么Python解释器就会把demo.pyc（不包括文件头的8个字节）加载到内存并将demo.pyc文件反序列化为字节码对象（PyCodeObject）交给Python虚拟机（Python虚拟机只是Python解释器中实现的一部分，为了便于理解人们把这部分抽象命名为Python虚拟机）处理执行，运行完毕Python解释器退出。如果demo.pyc文件头的mtime早于现在py源文件的修改时间，说明之前的demo.py源文件经过了修改需要重新编译。

Python解释器所做的工作就不深入了，可以简单理解为Python解释器将demo.py加载到内存编译一个对应的字节码对象，然后交由Python虚拟机程序处理执行，Python 虚拟机会从编译得到的PyCodeObject对象中依次读入每一条字节码指令，并在当前的上下文环境中执行这条字节码指令。而执行完毕之后Python编译器会根据情况生成相应的pyc文件，生成过程为Python解释器将内存的Python代码对象反序列化并加上相应的py文件头信息一起写入到硬盘，所以pyc文件只是Python代码对象（PyCodeObject）在硬盘上的表现形式，生成pyo文件过程也类似，只是多了Python解释器优化的过程。

3、提取得到pyc文件

所以我们要做的便是编写代码将code这个Python代码对象加上相应pyc文件头信息提取出来写入磁盘生成pyc文件，生成pyc文件目的是便于我们反编译pyc文件得到相应的py源码。当然你也可以通过Python自带的dis模块慢慢分析pyc文件中的字节码指令。

1. #!/usr/bin/env python
2. #coding = utf-8
3. #PyCodeObject\_to\_pyc.py
4. **import** py\_compile, imp, os, marshal, zlib, base64
6. code = marshal.loads(zlib.decompress(base64.b64decode('')))
8. **def** PyCodeObject\_to\_pyc(py\_code\_obj, pyc\_file):
9. with open(pyc\_file, 'wb') as pyc:
10. pyc\_magic = imp.get\_magic()
11. pyc.write(pyc\_magic)
12. mtime = long(os.fstat(pyc.fileno()).st\_mtime)
13. py\_compile.wr\_long(pyc, mtime)
14. marshal.dump(py\_code\_obj, pyc)
15. pyc.flush()
16. pyc.close()
18. **def** main():
19. PyCodeObject\_to\_pyc(code, 'extract.pyc')
21. **if** \_\_name\_\_ == '\_\_main\_\_':
22. main()

4、反编译pyc文件得到py源文件

将生成的pyc通过Python工具如uncompyle2等反编译得到py源码，这里直接通过在线的pyc反编译[网站](http://tool.lu/pyc/)得到py源码：

1. #!/usr/bin/env python
2. # encoding: utf-8
4. **import** hashlib
6. **def** sha1(string):
7. **return** hashlib.sha1(string).hexdigest()
9. **def** calc(strSHA1):
10. r = 0
11. **for** i **in** strSHA1:
12. r += int('0x%s' % i, 16)
13. **return** r
15. **def** encrypt(plain, key):
16. keySHA1 = sha1(key)
17. intSHA1 = calc(keySHA1)
18. r = []
19. **for** i **in** range(len(plain)):
20. r.append(ord(plain[i]) + int('0x%s' % keySHA1[i % 40], 16) - intSHA1)
21. intSHA1 = calc(sha1(plain[:i + 1])[:20] + sha1(str(intSHA1))[:20])
23. **return** ''.join(map((**lambda** x: str(x)), r))
25. **if** \_\_name\_\_ == '\_\_main\_\_':
26. key = raw\_input('[\*] Please input key:')
27. plain = raw\_input('[\*] Please input flag:')
28. encryptText = encrypt(plain, key)
29. cipherText = '-185-147-211-221-164-217-188-169-205-174-211-225-191-234-148-199-198-253-175-157-222-135-240-229-201-154-178-187-244-183-212-222-164'
30. **if** encryptText == cipherText:
31. **print** '[>] Congratulations! Flag is: %s' % plain
32. exit()
33. **else**:
34. **print** '[!] Key or flag is wrong, try again:)'
35. exit()

5、分析py源文件中加密算法

sha1函数使用sha1算法计算返回40位16进制散列值，calc函数计算40位16进制散列值中每位的整型值并相加，最后返回整型的总和值。

encrypt函数为核心加密算法函数：

1. **def** encrypt(plain, key):
2. keySHA1 = sha1(key)
3. intSHA1 = calc(keySHA1)
4. r = []
5. **for** i **in** range(len(plain)):
6. r.append(ord(plain[i]) + int('0x%s' % keySHA1[i % 40], 16) - intSHA1)
7. intSHA1 = calc(sha1(plain[:i + 1])[:20] + sha1(str(intSHA1))[:20])
9. **return** ''.join(map((**lambda** x: str(x)), r))

由第4行的for可以知道到明文长度和密文长度相同，核心加密语句为第6，7行，算法使用ord函数取得明文每个字符的ASCII整型值，int函数内容为明文每个字符位置模40访问由调用sha1函数返回的40位16进制keySHA1字符串中的16进制数并转化为10进制数与由调用calc函数返回的整型值相减，然后将ord函数和int计算所得值作为密文添加到r列表，第7行更新intSHA1值，第9行转换为“-185-147-211...”格式并返回。

这里我们知道了密文cipherText，密钥key，加密算法encrypt，从而能逆推出解密算法，只要把密文值减去int函数中的值并对结果使用chr函数取得明文plain。

6、编写解密代码

1. #coding:utf-8
2. #decrypt.py
4. **import** hashlib
6. **def** sha1(string):
7. **return** hashlib.sha1(string).hexdigest()
9. **def** calc(strSHA1):
10. r = 0
11. **for** i **in** strSHA1:
12. r += int("0x%s" % i, 16)
13. **return** r
15. **def** decrypt(strCipher,strKey):
16. listCipher = map(**lambda** x: int(x),strCipher.replace('-',' -')[1:].split(' '))
17. strKeySHA1 = sha1(strKey)
18. intSHA1 = calc(strKeySHA1)
19. strPlain = ''
20. **for** i **in** range(len(listCipher)):
21. strPlain += chr(listCipher[i] + intSHA1 - int("0x%s" % strKeySHA1[i%40],16))
22. intSHA1 = calc(sha1(strPlain[:(i + 1)])[:20] + sha1(str(intSHA1))[:20])
23. **return** strPlain
25. **if** \_\_name\_\_ == '\_\_main\_\_':
26. strCipher= '-185-147-211-221-164-217-188-169-205-174-211-225-191-234-148-199-198-253-175-157-222-135-240-229-201-154-178-187-244-183-212-222-164'
27. strKey = 'I\_4m-k3y'
28. strPlain = decrypt(strCipher, strKey)
29. **print** strPlain

最终获得flag：

