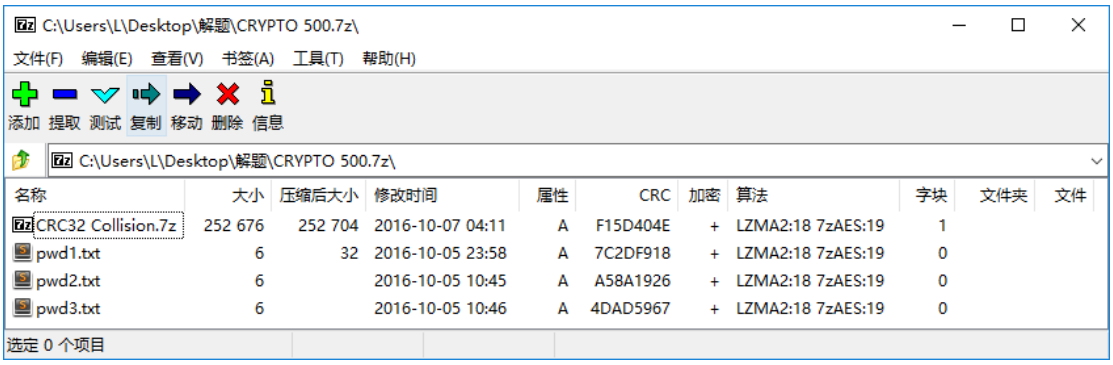
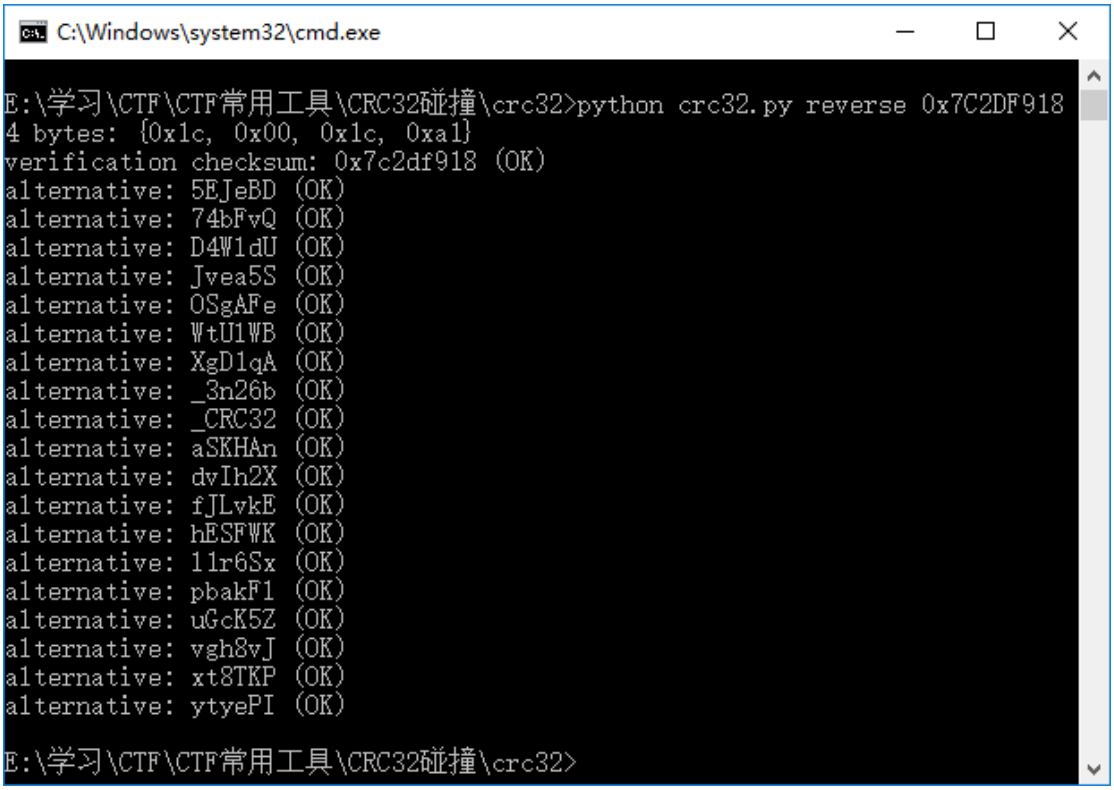
1、第一层：CRC32 碰撞

得到题目压缩包：



尝试解压，却发现需要密码，根据压缩包里的文件和文件名，猜测是题意是想让我们通过对 pwd1，pwd2 和 pwd3 文本文件进行 CRC32 碰撞获得文本内容，从而得到组合密码。

通过网上查找资料，了解 CRC32 碰撞原理，了解原理之后可以在 github 上找到可利用的 Python 脚本。



分别对 3 个 CRC32 值进行碰撞，找到最可能的组合：_CRC32_i5_n0t_s4f3，输入密码解压进入下一层。

如果是善于动手编程能力较好的小伙伴也可以自己编写一个脚本来跑，下面给出自己的 demo：

```python
# -*- coding: utf-8 -*-
# crc32Collision.py
import threading
import binascii
import time

def breakpassword():
    start=time.clock()
    crc_num=set([0x7C2DF918,0xA58A1926,0x4DAD5967])
    x = range(32,128)
    for i in x:
        for j in x:
            for k in x:
                for l in x:
                    for m in x:
                        for n in x:
                            mutex.acquire()
                            string=chr(i)+chr(j)+chr(k)+chr(l)+chr(m)+chr(n)

                            if binascii.crc32(string) in crc_num:
                                print "crc32 of %s is-> %s" %(string,hex(binascii.crc32(string)))
                                f=open("string.txt",'a')
                                f.write(string)
                                f.close()
                            mutex.release()
    end=time.clock()
    print "Used time: %f s" % (end - start)

def main(thread_num):
    print "breaking,please wait!"
    global mutex
    mutex=threading.Lock()
    threads=[]
    for x in xrange(0,thread_num):
        threads.append(threading.Thread(target=breakpassword))
        for t in threads:
            t.start()

        for t in threads:
            t.join()

if __name__ == '__main__':
    main(10)
```

2、第二层：维吉尼亚基于字典攻击

解压之后得到下图三个文件：

ciphertext.txt	2016/10/6 23:21	TXT 文件	1 KB
Find password.7z	2016/10/7 4:10	7Z 文件	3 KB
keys.txt	2016/10/7 2:25	TXT 文件	411 KB
tips.txt	2016/10/7 3:37	TXT 文件	1 KB

打开 tips.txt 读懂题意，要求我们在 keys.txt 中找到密钥解密 ciphertext.txt，解密密文之后便可以得到 Find password.7z 的解压密码。通过网上大量的资料查阅，了解维吉尼亚密码加密解密原理和算法，搜索相应的解密工具可以在 http://inventwithpython.com/hacking/diff/找到可以模板，需要读懂相应模块并修改使用：

```
# -*- coding: utf-8 -*-
#vigenereDictionaryHacker.py
import detectEnglish

LETTERS = 'ABCDEFGHIJKLMNOPQRSTUVWXYZ'

def translateMessage(key, message, mode):
    translated = [] # 存储加密/解密消息字符串
    keyIndex = 0
    key = key.upper()
    for symbol in message: # 遍历每个消息里的字符的消息
        num = LETTERS.find(symbol.upper())
        if num != -1: # -1 意味着转换为大写在 LETTERS 找不到
            if mode == 'encrypt':
                num += LETTERS.find(key[keyIndex]) # 加密时相加
            elif mode == 'decrypt':
                num -= LETTERS.find(key[keyIndex]) # 解密时相减
            num %= len(LETTERS) # 处理潜在的循环
            # 添加转换后加密/解密字符
            if symbol.isupper():
                translated.append(LETTERS[num])
            elif symbol.islower():
                translated.append(LETTERS[num].lower())
            keyIndex += 1 # 继续下一个用密钥字符来解密
            if keyIndex == len(key):
                keyIndex = 0
        else:
            # 字符不在 LETTERS 里直接添加
            translated.append(symbol)
    return ''.join(translated)
```

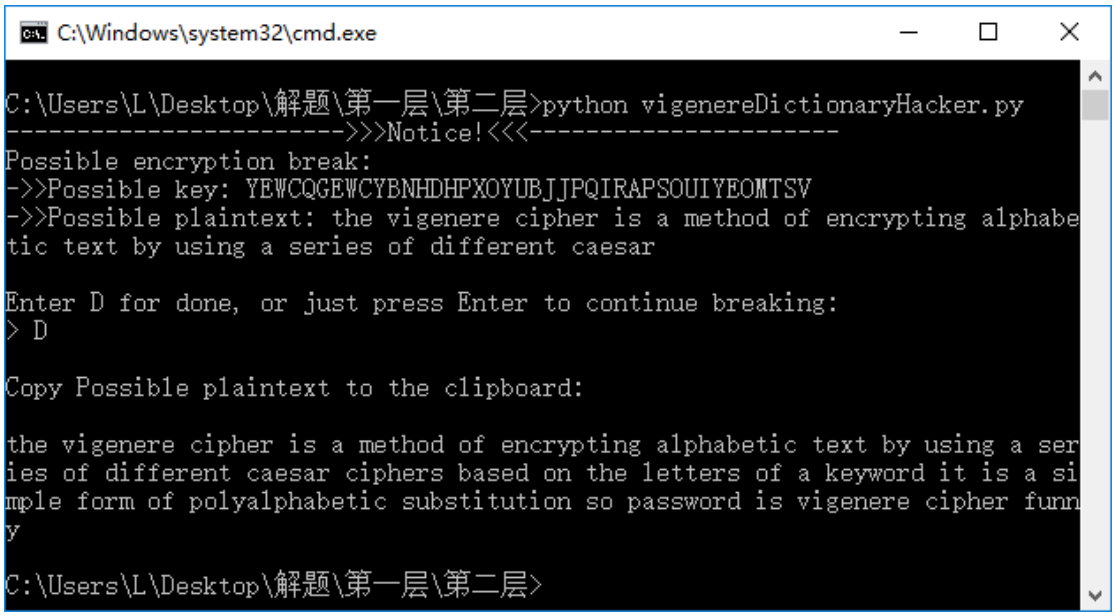
```
31.
32. def decryptMessage(key, message):
33.     return translateMessage(key, message, 'decrypt')
34.
35. def hackVigenere(ciphertext):
36.     fo = open('keys.txt')
37.     words = fo.readlines()
38.     fo.close()
39.     for word in words:
40.         word = word.strip()
41.         decryptedText = decryptMessage(word, ciphertext)
42.         if detectEnglish.isEnglish(decryptedText, wordPercentage=40):
43.             print('------------------------>>>Notice!<<<----------------------')
44.             print('Possible encryption break:')
45.             print('->>Possible key: ' + str(word))
46.             print('->>Possible plaintext: ' + decryptedText[:100])
47.             print('Enter D for done, or just press Enter to continue breaking:')
48.             response = raw_input('> ')
49.             if response.upper().startswith('D'):
50.                 return decryptedText
51.
52. def main():
53.     ciphertext = """rla xymijgpf ppsoto wq u nncwel ff tfqlgnxwzz sgnlwduzmy vcyg ib bhfbe u tnaxua ff satzmpibf vszqen eyvlatq cnzhk dk hfy mnciuzj ou s yygusfp bl dq e okcvpa hmsz vi wdimyfqqjqubzc hmpmbgxifbgi qs lciyaktb jf cln tkspy drywuz wucfm"""
54.     hackedMessage = hackVigenere(ciphertext)
55.     if hackedMessage != None:
56.         print('\nCopy Possible plaintext to the clipboard:\n')
57.         print(hackedMessage)
58.     else:
59.         print('Failed to hack encryption.')
60.
61. if __name__ == '__main__':
62.     main()
```

```
1. # -*- coding: utf-8 -*-
2. # detectEnglish.py
3. # 英文单词探测模块
4. # 模块引用:
```

```
#   import detectEnglish
#   detectEnglish.isEnglish(someString) # 返回真或假
# 模块需要一个包含常见英文单词的"words.txt"，下载地址：http://invpy.com/dictionary.txt

UPPERLETTERS = 'ABCDEFGHIJKLMNOPQRSTUVWXYZ'
LETTERS_AND_SPACE = UPPERLETTERS + UPPERLETTERS.lower() + ' \t\n'

def loadDictionary():
    dictionaryFile = open('words.txt')
    englishWords = {}
    for word in dictionaryFile.read().split('\n'):
        englishWords[word] = None
    dictionaryFile.close()
    return englishWords

ENGLISH_WORDS = loadDictionary()

def getEnglishCount(message):
    message = message.upper()
    message = removeNonLetters(message)
    possibleWords = message.split()
    # print possibleWords
    if possibleWords == []:
        return 0.0 # 没有单词返回 0.0

    matches = 0
    for word in possibleWords:
        if word in ENGLISH_WORDS:
            matches += 1
    return float(matches) / len(possibleWords)

def removeNonLetters(message):
    lettersOnly = []
    for symbol in message:
        if symbol in LETTERS_AND_SPACE:
            lettersOnly.append(symbol)
    return ''.join(lettersOnly)

def isEnglish(message, wordPercentage=20, letterPercentage=85):
    # 默认设置转换后的 message 中单词的 20%能在 words.txt 中的单词列表找到
    # 默认设置转换后的 message 中 85%是字母或空格
    # (not punctuation or numbers).
    wordsMatch = getEnglishCount(message) * 100 >= wordPercentage
```

```
    numLetters = len(removeNonLetters(message))
    messageLettersPercentage = float(numLetters) / len(message) * 100
    lettersMatch = messageLettersPercentage >= letterPercentage
    return wordsMatch and lettersMatch
```

解密结果：



所以 Find password.7z 的解压密码就是：vigenere cipher funny

3、第三层：SHA1

解压 Find password.7z 得到如下文件：

Easy SHA1.7z	2016/10/7 4:10	7Z 文件	2 KB
U need unzip password.txt	2016/10/7 3:26	TXT 文件	1 KB

打开 U need unzip password.txt 读懂题意，题目要求我们通过编写脚本爆破 SHA1 的明文密文对，得到明文作为解压密码进入下一关。

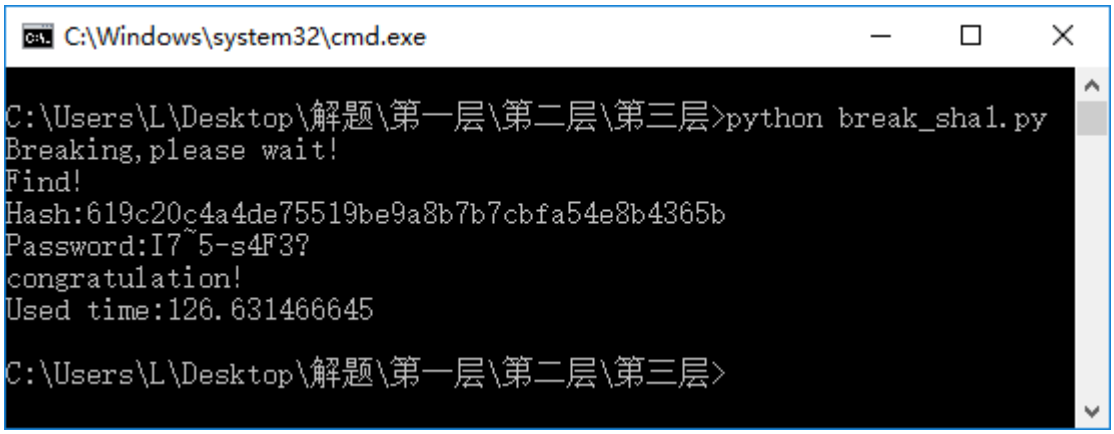
了解了 SHA1 密码之后，通过查阅资料发现我们可以使用 Python 的 hashlib 模块来进行爆破，下面给出 demo：

```
#break_sha1.py
import hashlib
import time

def match(h,pwd):
    hl=list(h)
```

```
    if hl[0]=='6':
        if hl[1]=='1':
            if hl[2]=='9':
                if hl[3]=='c':
                    if hl[4]=='2':
                        if hl[5]=='0':
                            if hl[6]=='c':
                                if hl[8]=='a':
                                    if hl[16]=='9':
                                        if hl[24]=='b':
                                            if hl[32]=='e':
                                                print "Find!"
                                                print "Hash:%s" %h
                                                print "Password:%s" %pwd
                                                matched=1
                                                return matched

    else:
        matched=0
        return matched

def generate():
    x=range(32,128)
    for i in x:
        for j in x:
            for k in x:
                for l in x:
                    pwd=chr(i)+'7'+chr(j)+'5-'+chr(k)+'4'+chr(l)+'3?'
                    sha1_hash=hashlib.sha1()
                    sha1_hash.update(pwd)
                    h=sha1_hash.hexdigest()
                    matched=match(h,pwd)
                    if matched:
                        print "congratulation!"
                        return 0
                    else:
                        pass

def main():
    start=time.clock()
    print "Breaking,please wait!"
    generate()
    end=time.clock()
    print "Used time:%s" %(end-start)
```

```python

if __name__ == '__main__':
    main()
```

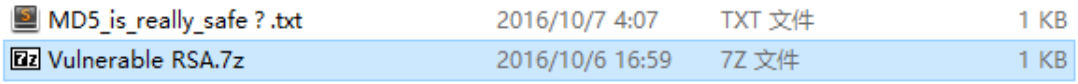
结果：



所以得到 Easy SHA1.7z 的解压密码：I7~5-s4F3?

4、第四层：MD5 不再安全

解压 Easy SHA1.7z 之后得到如下文件：



MD5_is_really_safe ? .txt	2016/10/7 4:07	TXT 文件	1 KB
Vulnerable RSA.7z	2016/10/6 16:59	7Z 文件	1 KB

打开 MD5_is_really_safe？.txt 文本，题目要让我们找到两个不同程序但是他们的 MD5 值却相同，这题考察我们对 MD5 安全性的感知度。通过搜索引擎可以找到大量关于王小云教授对 MD5 破解相关资料，百度关键词：MD5 碰撞 MD5 校验真的安全吗？ MD5 真的已靠不住？等等。

小编发现，其实早在2011年，就有几位密码学家使用“构造前缀碰撞法”（chosen-prefix collisions）制作出了md5一样但运行结果不一样的exe可执行文件。

HelloWorld-colliding.exe

GoodbyeWorld-colliding.exe

下面这两个程序会在屏幕上打印出不同的字符，但是它们的 MD5 都是一样的。

这几位密码学家所使用的计算机是一台 Sony PS3，且仅用了不到两天。

王小云的破解使md5堡垒的轰然倒塌，使越来越多基于md5的保护变得不可靠、不安全。随着技术的发展，md5终将被更加安全的算法所替代。

附快速MD5碰撞生成器fastcoll下载地址

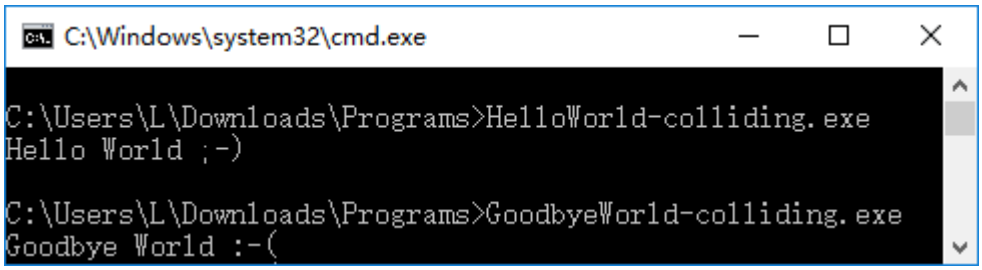
fastcoll_v1.0.0.5.exe.zip

源代码：

fastcoll_v1.0.0.5_source.zip

本文由 安全客 原创发布，如需转载请注明来源及本文地址。
本文地址：http://bobao.360.cn/news/detail/768.html

下载 HelloWorld-colliding.exe GoodbyeWorld-colliding.exe 运行结果：



Hello World ;-)已在提示里，那么另一个程序的输出就是：Goodbye World :-(

根据提示输入做为 Vulnerable RSA.7z 的解压密码，解压成功进入下一关。

5、第五层：Vulnerable RSA

解压 Vulnerable RSA.7z 后得到如下文件：

flag.enc	2016/10/6 16:43	Wireshark captu...	1 KB
rsa_public_key.pem	2016/10/6 16:01	PEM 文件	1 KB

在了解 RSA 的原理和常见的攻击方法后，我们用 OpeSSL 来导入公钥查看模数 n，指数 e。

```
1. openssl rsa -inform PEM -in rsa_public_key.pem -noout -modulus -text -pubin
```

```
root@kali: ~
File  Edit  View  Search  Terminal  Help
root@kali:~# openssl rsa -inform PEM -in rsa_public_key.pem -noout -modulus -text -pubin
Public-Key: (1026 bit)
Modulus:
    02:8f:ff:9d:d3:e6:fe:97:81:64:9e:b7:fe:5e:93:
    03:cf:69:63:47:c4:11:0b:c4:ba:39:69:f0:b1:16:
    69:84:0c:51:d8:1a:68:42:b6:df:2b:09:0f:21:cd:
    76:d4:37:1a:8c:0e:47:04:8c:96:5e:ca:5b:46:91:
    3a:fb:b8:da:05:20:72:a0:56:6d:70:39:c6:18:ab:
    a9:06:57:59:b0:59:e2:9e:48:5d:c5:06:1a:16:ac:
    63:12:94:38:d9:35:4e:65:df:57:47:54:6b:85:db:
    3d:69:98:19:c4:b7:73:2d:f9:27:c7:08:4a:5d:52:
    d6:e6:d6:aa:c1:44:62:34:25
Exponent:
    01:f8:fb:a4:10:05:2d:f7:ed:a3:46:2f:1a:ac:d6:
    9e:40:76:04:33:ca:33:57:67:cd:73:05:a3:d0:90:
    80:5a:5f:d4:05:dd:6e:ea:70:e9:8f:0c:a1:e1:cf:
    25:47:48:67:1b:f0:c9:80:06:c2:0e:ee:1d:62:79:
    04:35:09:fe:7a:98:23:8b:43:91:60:a5:61:2d:a7:
    1e:90:45:14:e8:12:80:61:7e:30:7c:3c:d3:31:3f:
    a4:c6:fc:a3:31:59:d0:44:1f:bb:18:d8:3c:af:4b:
    d4:6f:6b:92:97:a8:0a:14:2d:d6:9b:f1:a3:57:cc:
    b5:e4:c2:00:b6:d9:0f:15:a3
Modulus=28FFF9DD3E6FE9781649EB7FE5E9303CF696347C4110BC4BA3969F0B11669840C51D81A6842B6DF2B
090F21CD76D4371A8C0E47048C965ECA5B46913AFBB8DA052072A0566D7039C618ABA9065759B059E29E485DC
5061A16AC63129438D9354E65DF5747546B85DB3D699819C4B7732DF927C7084A5D52D6E6D6AAC144623425
root@kali:~#
```

可以看到指数（Exponent）很大，在 RSA 中我们知道 ed ≡ 1 (mod ϕ(n))，如果 n 确定，e 非常大，就会导致 d 很小，就会出现维纳攻击（Wiener's attack），攻击原理是使用连分式（Continued fraction）去求得 d。

了解原理后我们可以在 github 找到基于维纳攻击的工具 rsa-wiener-attack，然后将其中的 RSAwienerHacker.py 改写一下：

```
#RSAwienerAttack.py
import ContinuedFractions, Arithmetic

def hack_RSA(e,n):
    '''''
    Finds d knowing (e,n)
    applying the Wiener continued fraction attack
    '''
    frac = ContinuedFractions.rational_to_contfrac(e, n)
    convergents = ContinuedFractions.convergents_from_contfrac(frac)

    for (k,d) in convergents:
        #check if d is actually the key
        if k!=0 and (e*d-1)%k == 0:
            phi = (e*d-1)//k
            s = n - phi + 1
            # check if the equation x^2 - s*x + n = 0
            # has integer roots
            discr = s*s - 4*n
```

```
            if(discr>=0):
                t = Arithmetic.is_perfect_square(discr)
                if t!=-1 and (s+t)%2==0:
                    print("\nHacked!")
                    return d

def main():
    e=3546111024413075720565721818279258991983453502287537309310893932754639165444566268942454150961078344657784095323731871253185546147225993017915289162128393681210660355410088082615345005860236527677122716257852042809646880046803283001248496804771053025193773700925781078271168213918262109723203776149675478276197
    n=460657813884289609896372056585544172485318117026246263899744329237492701820627219556007788200590119136173895989001382151536006853823326382892363143604314518686388786002989248800814861248595075326277099645338694977097459168530898776007293695728101976069423971696524237755227187061418202849911479124793990722597
    print "e="
    print e
    print "n="
    print n
    d=hack_RSA(e,n)
    print "d="
    print d

if __name__ == '__main__':
    main()
```

结果：

```
root@kali: ~/rsa-wiener-attack
File  Edit  View  Search  Terminal  Help
root@kali:~/rsa-wiener-attack# python RSAwienerAttack.py
e=
3546111024413075720565721818279258991983453502287537309310893932754639165444566268942454150961078344657784095323731871253185546147225993017915289162128393681210660355410088082615345005860236527677122716257852042809646880046803283001248496804771053025193773700925781078271168213918262109723203776149675478276197
n=
4606578138842896098963720565855441724853181170262462638997443292374927018206272195560077882005901191361738959890013821515360068538233263828923631436043145186863887860029892488008148612485950753262770996453386949770974591685308987760072936957281019760694239716965242377552271870614182028499114791247939907225 97

Hacked!
d=
8264667972294275017293339772371783322168822149471976834221082393409363691895
root@kali:~/rsa-wiener-attack#
```

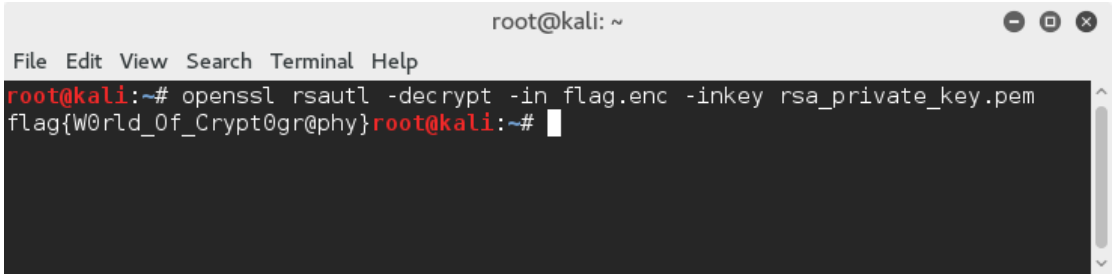
我们得到了私钥 d，且知道了 e，那我们就可以使用 rsatool 来生产私钥文件：

```
1. rsatool.py -e 3546111024413075720565721818279258991983453502287537309310893932754639165444566268942454150961078344657784095323731871253185546147225993017915289162128393681210660355410088082615345005860236527677122716257852042809646880046803283001248496804771053025193773700925781078271168213918262109723203776149675478276 19 -n 4606578138842896098963720565855441724853181170262462638997443292374927018206272195560077882005901191361738959890013821515360068538233263828923631436043145186863887860029892488008148612485950753262770996453386949770974591685308987760072936957281019760694239716965242377552271870614182028499114791247939907225 97 -d 8264667972294275017293339772371783322168822149471976834221082393409363691895 -o rsa_private_key.pem -f PEM
```

```
root@kali: ~
File  Edit  View  Search  Terminal  Help
root@kali:~# rsatool.py -e 3546111024413075720565721818279258991983453502287537309310893932754639165444566268942454150961078344657784095323731871253185546147225993017915289162128393681210660355410088082615345005860236527677122716257852042809646880046803283001248496804771053025193773700925781078271168213918262109723203776149675478276 19 -n 4606578138842896098963720565855441724853181170262462638997443292374927018206272195560077882005901191361738959890013821515360068538233263828923631436043145186863887860029892488008148612485950753262770996453386949770974591685308987760072936957281019760694239716965242377552271870614182028499114791247939907225 97 -d 8264667972294275017293339772371783322168822149471976834221082393409363691895 -o rsa_private_key.pem -f PEM
Using (n, d) to initialise RSA instance

n =
28fff9dd3e6fe9781649eb7fe5e9303cf696347c4110bc4ba3969f0b11669840c51d81a6842b6df2
b090f21cd76d4371a8c0e47048c965eca5b46913afbb8da052072a0566d7039c618aba9065759b05
9e29e485dc5061a16ac63129438d9354e65df5747546b85db3d699819c4b7732df927c7084a5d52d
6e6d6aac144623425

e =
1f8fba410052df7eda3462f1aacd69e40760433ca335767cd7305a3d090805a5fd405dd6eea70e98
f0ca1e1cf254748671bf0c98006c20eee1d6279043509fe7a98238b439160a5612da71e904514e81
280617e307c3cd3313fa4c6fca33159d0441fbb18d83caf4bd46f6b9297a80a142dd69bf1a357ccb
5e4c200b6d90f15a3

d =
1245a2e4c321ada55905c249b7e09640f88a41cabd63c932b44e010d3788c977

p =
225ffc6f7c6d8959696663e4cdb90d4a250a99a2122bf6b3d9a4431a1a8d2e6cd7848af4e396b97e
c1109ea9cf30cf67af2215cc2cbd3a754ff0aa6407df9b49b

q =
13156850d0559551936f425e83ec24bf0905591f13e4a07857f8849f36b4f904431b50aa4509baf1
369583e32fff2b25af10e39de868cea7706efee8c36fb663f

Saving PEM as rsa_private_key.pem
root@kali:~#
```

得到 rsa_private_key.pem，于是我们利用 OpenSSL 对 flag.enc 解密：

```
1. openssl rsautl -decrypt -in flag.enc -inkey rsa_private_key.pem
```



最终得到 flag：flag{W0rld_Of_Crypt0gr@phy}