菜鸡刷CTF（四）

原创

大白羊想学习 于 2019-08-02 18:55:13 发布 5397 收藏 2

分类专栏： CTF 文章标签： CTF

本文链接：https://blog.csdn.net/weixin_44728238/article/details/97371023

高手进阶（4）

题目12：wtc_rsa_bbq

难度系数：

题目来源： tinyctf-2014

题目描述：暂无

解题：

可以看到cry200的前几个字符为504B0304，这是zip文件的文件头，于是我们将其保存为zip文件。zip文件中包括key.pem文件和cipher.bin文件。

我们使用openssl可以获得公钥的信息，其中N比较大，我们可以尝试用费马定理去破解p和q

```
import gmpy2
def isqrt(n):
  x = gmpy2.mpz(n)
  y = gmpy2.mpz((x + n // x) // 2)
  while y < x:
    x = y
    y = (x + n // x) // 2
  return x
n = gmpy2.mpz(679579929445417090796373314958352880510177361729462192689791243102196769018146341419940569102496651824937299
738537456135576544432049079391002898300198797221924437483724643855477018147774288639336155228900597734744173869318844255692475545835684556709775073336109109846752772077385437851448215328141096660261438308826704924125013674189002595250203294762596028965332899775208773320006393676223592135995088535678385027358187339301997688102383636136
137538127006868529224901937059267136080402498877432166613191344576652312438714566169382753700465260156
143199707252590250373280921870375431796105736552074603804910696325743691317781099016524628381496580097091743568617678702048038069242893159548715952220738591853532842685873312671737282216587819901194012521678998260475885832473622985727
362298572788397512447184590473718761927222549389515152849248566281272354480116085470203137413924993107429630882329043259530840382880179158766903387472716587067982240678129096467201138461960380680792953534366991098348930562736587887507018304132204828793333486018470656056870548645765390879310100168147762901154890481163444481122296433539288905754638768700952492512406731460604412447424967642115057341729310735982961515248311488333222612609961196477124084039771246638999119951171033639602489932159455170406551791127198312487436473494597495007637073871513066738387201949468771251327190683519610856620025566675621926314874396030840219278916815000865282846642529711462753894749115541586140580337162009750195223178235180176734250985597156373563381550790868104361886324053097508335099051385614181800664174756482090785602210645518723893919206065164862091614302474277173767210274990238910117421644634219154493542479444330391986088773150664240652037397933338319430212833196600978295172801790024303762943223958292905027627071463633404313069465721168381112461333905393158190230869565483634526574253400121555735123517319351960115057100697984253076563406503138996670196215058454245883679008273985640483229330920592583738721183501876697489989501614020773672392914623155639709219835737212685155663671557075851092437571437899014501489942395016846236551636379971057692229594943250641132787380383941390997030426400545923793467820029807517860639514911381350718072026344114427138700477301899713331704181120773900158527297665949189552554882800273593488113047743490525046122703696950080616239685749707583471176065779395554462380674306857878865027422013727988520108938859999419465483361034025171872484927656994901513948065911490283205309663657333204840476504296444200426929020926139116081523101524090361675265336198228514115895029703639032)
e = 65537
i = gmpy2.mpz(isqrt(n))
p= gmpy2.mpz(0)
q=gmpy2.mpz(0)
while True:
  #if (n-(i*(n/i))==0):
  if(n%i==0):
    p = gmpy2.mpz(i)
    q = gmpy2.mpz(n/i)
    break
  i += 1
  print(i)
  print('\n')
print(p)
print(q)
```

接下来我们可以用获得的p和q来破译加密文件了，值得注意的是，.bin文件是而二进制文件
终于向python3低头了，换成了2，所以下面的代码都是适用于2咯

```
n = 679579929445417090796373314958352880510177361729462192689791243102196769018146341419940569102496651824937299
738537456135576544432049079391002898300198797221924437483724643855477018147774288639336155228900597734744173869318844255692475545835684556709775073336109109846752772077385437851448215328141096660261438308826704924125013674189002595250203294762596028965332899775208773320006393676223592135995088535678385027358187339301997688102383636136
137538127006868529224901937059267136080402498877432166613191344576652312438714566169382753700465260156143199707252590250373280921870375431796105736552074603804910696325743691317781099016524628381496580097091743568617678702048038069242893159548715952220738591853532842685873312671737282216587819901194012521678998260475885832473622985727
```

```
883975124471845904737187619272225493895151528492485662812723544801160854702031374139249931074296308823290432595308403828801791587669033874727165870679822406781290964672011384619603806807929535343669910983489305627365878875070183041322048287933334860184706560568705486457653908793101001681477629011548904811634444811222964335392889057546387687009524925124067314606044124474249676421150573417293107359829615152483114883332226126099611964771240840397712466389991199511710336396024899321594551704065517911271983124874364734945974950076370738715130667383887201949468771251327190683519610856620025566675621926314874396030840219278916815000865282846642529711462753894749115541586140580337162009750195223178235180176734250985597156373563381550790868104361886324053097508335099051385614181800664174756482090785602210645518723893919206065164862091614302474277173767210274990238910117421644634219154493542479444330391986088773150664240652037397933338319430212833196600978295172801790024303762943223958292905027627071463633404313069465721168381112461333905393158190230869565483634526574253400121555735123517319351960115057100697984253076563406503138996670196215058454245883679008273985640483229330920592583738721183501876697489989501614020773672392914623155639709219835737212685155663671557075851092437571437899014501489942395016846236551636379971057692229594943250641132787380383941390997030426400545923793467820029807517860639514911381350718072026344114427138700477301899713331704181120773900158527297665949189552554882800273593488113047743490525046122703696950080616239685749707583471176065779395554462380674306857878865027422013727988520108938859999419465483361034025171872484927656994901513948065911490283205309663657333204840476504296444200426929020926139116081523101524090361675265336198228514115895029703639031
p = 260687538913047604611581784183499992008451760653785074016920718323190075912750802620741024634382067897986936407072164858654195035172056629230527593989978466296098220579031027154724041642557344915423242566240332284307941704981938020407661739104882149483433969072174052675580644164713319546908058746795753762649896919296312912702039493797237183118384499796757914157936629672274357161577311007769338358918777509136488505959298049498018461823273893799669038335872602861372239911549680531979970541528181711754520442958622706059603000655429088872469604344301647170492716926975027102329909258120989355802343558462047316328752521306824960655230501606894696454547299868668551619819367530969111390441903346427083466873717778205853984905381790237396069409514016317067322105019049986366714245693681256834514176535381210683575566036929609986694319036120560201862975426434985372574748540556000702736560930190396248279747095930579691977036607335841985416541503515963817848780705827093841092321324825882821075072622553561916744473353548759606182352622008019859330726239330083782855413787656845382479896229326659871568516059343215385393836820040891163826148343904385963328442668250413125432533185139042633099387310968853063208000057718438859818276275817262048678368751842523947169311954597690408896814857060351
q = 260687538913047604611581784183499992008451760653785074016920718323190075912750802620741024634382067897986936407072164858654195035172056629230527593989978466296098220579031027154724041642557344915423242566240332284307941704981938020407661739104882149483433969072174052675580644164713319546908058746795753762649896919296312912702039493797237183118384499796757914157936629672274357161577311007769338358918777509136488505959298049498018461823273893799669038335872602861372239911549680531979970541528181711754520442958622706059603000655429088872469604344301647170492716926975027102329909258120989355802343558462047316328752521306824960655230501606894696454547299868668551619819367530969111390441903346427083466873717778205853984905381790237396069409514016317067322105019049986366714245693681256834514176535381210683575566036929609986694319036120560201862975426434985372574748540556000702736560930190396248279747095930579691977036607335841985416541503515963817848780705827093841092321324825882821075072622553561916744473353548759606182352622008019859330726239330083782855413787656845382479896229326659871568516059343215385393836820040891163826148343904385963328442668250413125432533185139042633099387310968853063208000057718438859818276275817262048678368751842523947169311954597690408896814857060353
e = 65537

def egcd(a, b):
  if a == 0:
    return (b, 0, 1)
  else:
    g, y, x = egcd(b % a, a)
    return (g, x - (b // a) * y, y)

def modinv(a, m):
  g, x, y = egcd(a, m)
  if g != 1:
    raise Exception('modular inverse does not exist')
  else:
    return x % m

cipher = open('cipher.bin', 'rb').read().encode('hex')
cipher = int(cipher, 16)
```

```
fi = (p-1)*(q-1)
d = modinv(e, fi)

flag = pow(cipher, d, n)
print ('%x' % flag).decode('hex')
```

结果为
Congratulations! Here is a treat for you:
flag{how_d0_you_7urn_this_0n?}

题目13：cr4-poor-rsa
难度系数：
题目来源： alexctf-2017
题目描述：暂无

解题：首先我们看到文件中有key.pub文件，我们可以使用openssl来解析它，指令为：
openssl rsa -pubin -in key.pub -text -noout
结果为：

```
RSA Public-Key: (399 bit)
Modulus:
    52:a9:9e:24:9e:e7:cf:3c:0c:bf:96:3a:00:96:61:
    77:2b:c9:cd:f6:e1:e3:fb:fc:6e:44:a0:7a:5e:0f:
    89:44:57:a9:f8:1c:3a:e1:32:ac:56:83:d3:5b:28:
    ba:5c:32:42:43
Exponent: 65537 (0x10001)
```

```
from Crypto.PublicKey import RSA
import gmpy2, base64

pub = open("key.pub", "r").read()
pub = RSA.importKey(pub)

n = long(pub.n)
print "n"
print n
e = long(pub.e)
print "e"
print e

#w/ n, get p and q from factordb.com
p = 863653476616376575308866344984576466644942572246900013156919
print "p"
print p
q = 96544530432699819479828222884248473243845717059599952342690l
print "q"
print q

d = long(gmpy2.invert(e,(p-1)*(q-1)))
print "d"
print d

key = RSA.construct((n,e,d))

secret = base64.b64decode("Ni45iH4UnXSttNuf0Oy80+G5J7tm8sBJuDNN7qfTIdEKJow4siF2cpSbP/qIWDjSi+w=")
print key.decrypt(secret)
```

结果为：ALEXCTF{SMALL_PRIMES_ARE_BAD}

题目14：Decode_The_File
难度系数： 8.0
题目来源： RCTF-2015
题目描述：暂无

解题：首先打开文件后很显然是一个经过base64编码后的文件，我们首先想到用base64解码
解码后我们可以看到一段代码，这段代码中给出了一个代码原来的网址，通过对比可以看到二者还是有一定的差别，在这些不同的地方中就可以掺杂一些信息

> ###
>
> **Documentation**
>
> ###
>
> **Author: Todd Whiteman**
>
> **Date: 7th May, 2003**
>
> **Verion: 1.1**
>
> **Homepage: http://home.pacific.net.au/~twhitema/des.html**
>
> **Modifications to 3des CBC code by Matt Johnston 2004**
>
> **This algorithm is a pure python implementation of the DES algorithm.**
>
> **It is in pure python to avoid portability issues, since most DES implementations are programmed in C (for performance reasons).**
>
> **Triple DES class is also implemented, utilising the DES base. Triple DES is either DES-EDE3 with a 24 byte key, or DES-EDE2 with a 16 byte key.**
>
> **See the README.txt that should come with this python module for the implementation methods used. “””"A pure python implementation of the DES and TRIPLE DES encryption algorithms pyDes.des(key, [mode], [IV])**
>
> pyDes.triple_des(key, [mode], [IV]) key -> String containing the
> encryption key. 8 bytes for DES, 16 or 24 bytes
> for Triple DES mode -> Optional argument for encryption type, can be either
> pyDes.ECB (Electronic Code Book) or pyDes.CBC (Cypher Block Chaining) IV -> Optional argument, must be supplied if using CBC mode. Must be 8 bytes Example: from pyDes import * data = “Please
> encrypt my string” k = des(“DESCRYPT”, " ", CBC, “\0\0\0\0\0\0\0\0”) d
> = k.encrypt(data) print "Encypted string: " + d print “Decypted string: " + k.decrypt(d) See the module source (pyDes.py) for more examples of use. You can slo run the pyDes.py file without and
> arguments to see a simple test. Note: This code was not written for
> high-end systems needing a fast
> implementation, but rather a handy portable solution with small usage. “””
>
> **Modes of cryptin / cyphering ECB = 0 CBC = 1**

Modes of crypting / cyphering ECB = 0 CBC = 1

##

DES

class des:
“”“DES encryption/decrytpion class
Supports ECB (Electronic Code Book) and CBC (Cypher Block Chaining) modes.
pyDes.des(key,[mode], [IV])
key -> The encryption key string, must be exactly 8 bytes
mode -> Optional argument for encryption type, can be either pyDes.ECB
(Electronic Code Book), pyDes.CBC (Cypher Block Chaining)
IV -> Optional string argument, must be supplied if using CBC mode.
Must be 8 bytes in length.
“””
# Permutation and translation tables for DES
__pc1 = [56, 48, 40, 32, 24, 16, 8,
0, 57, 49, 41, 33, 25, 17,
9, 1, 58, 50, 42, 34, 26,
18, 10, 2, 59, 51, 43, 35,
62, 54, 46, 38, 30, 22, 14,
6, 61, 53, 45, 37, 29, 21,
13, 5, 60, 52, 44, 36, 28,
20, 12, 4, 27, 19, 11, 3
]
# number left rotations of pc1
__left_rotations = [
1, 1, 2, 2, 2, 2, 2, 2, 1, 2, 2, 2, 2, 2, 2, 1
]
# permuted choice key (table 2)
__pc2 = [
13, 16, 10, 23, 0, 4,
2, 27, 14, 5, 20, 9,
22, 18, 11, 3, 25, 7,
15, 6, 26, 19, 12, 1,
40, 51, 30, 36, 46, 54,
29, 39, 50, 44, 32, 47,
43, 48, 38, 55, 33, 52,
45, 41, 49, 35, 28, 31
]
# initial permutation IP
__ip = [57, 49, 41, 33, 25, 17, 9, 1,
59, 51, 43, 35, 27, 19, 11, 3,
61, 53, 45, 37, 29, 21, 13, 5,
63, 55, 47, 39, 31, 23, 15, 7,
56, 48, 40, 32, 24, 16, 8, 0,
58, 50, 42, 34, 26, 18, 10, 2,
60, 52, 44, 36, 28, 20, 12, 4,
62, 54, 46, 38, 30, 22, 14, 6
]
# Expansion table for turning 32 bit blocks into 48 bits
__expansion_table = [
31, 0, 1, 2, 3, 4,
3, 4, 5, 6, 7, 8,
7, 8, 9, 10, 11, 12,
11, 12, 13, 14, 15, 16,
15, 16, 17, 18, 19, 20,
19, 20, 21, 22, 23, 24,
23, 24, 25, 26, 27, 28,

```
27, 28, 29, 30, 31, 0
]
# The (in)famous S-boxes
__sbox = [
# S1
[14, 4, 13, 1, 2, 15, 11, 8, 3, 10, 6, 12, 5, 9, 0, 7,
0, 15, 7, 4, 14, 2, 13, 1, 10, 6, 12, 11, 9, 5, 3, 8,
4, 1, 14, 8, 13, 6, 2, 11, 15, 12, 9, 7, 3, 10, 5, 0,
15, 12, 8, 2, 4, 9, 1, 7, 5, 11, 3, 14, 10, 0, 6, 13],
# S2
[15, 1, 8, 14, 6, 11, 3, 4, 9, 7, 2, 13, 12, 0, 5, 10,
3, 13, 4, 7, 15, 2, 8, 14, 12, 0, 1, 10, 6, 9, 11, 5,
0, 14, 7, 11, 10, 4, 13, 1, 5, 8, 12, 6, 9, 3, 2, 15,
13, 8, 10, 1, 3, 15, 4, 2, 11, 6, 7, 12, 0, 5, 14, 9],
# S3
[10, 0, 9, 14, 6, 3, 15, 5, 1, 13, 12, 7, 11, 4, 2, 8,
13, 7, 0, 9, 3, 4, 6, 10, 2, 8, 5, 14, 12, 11, 15, 1,
13, 6, 4, 9, 8, 15, 3, 0, 11, 1, 2, 12, 5, 10, 14, 7,
1, 10, 13, 0, 6, 9, 8, 7, 4, 15, 14, 3, 11, 5, 2, 12],
# S4
[7, 13, 14, 3, 0, 6, 9, 10, 1, 2, 8, 5, 11, 12, 4, 15,
13, 8, 11, 5, 6, 15, 0, 3, 4, 7, 2, 12, 1, 10, 14, 9,
10, 6, 9, 0, 12, 11, 7, 13, 15, 1, 3, 14, 5, 2, 8, 4,
3, 15, 0, 6, 10, 1, 13, 8, 9, 4, 5, 11, 12, 7, 2, 14],
# S5
[2, 12, 4, 1, 7, 10, 11, 6, 8, 5, 3, 15, 13, 0, 14, 9,
14, 11, 2, 12, 4, 7, 13, 1, 5, 0, 15, 10, 3, 9, 8, 6,
4, 2, 1, 11, 10, 13, 7, 8, 15, 9, 12, 5, 6, 3, 0, 14,
11, 8, 12, 7, 1, 14, 2, 13, 6, 15, 0, 9, 10, 4, 5, 3],
# S6
[12, 1, 10, 15, 9, 2, 6, 8, 0, 13, 3, 4, 14, 7, 5, 11,
10, 15, 4, 2, 7, 12, 9, 5, 6, 1, 13, 14, 0, 11, 3, 8,
9, 14, 15, 5, 2, 8, 12, 3, 7, 0, 4, 10, 1, 13, 11, 6,
4, 3, 2, 12, 9, 5, 15, 10, 11, 14, 1, 7, 6, 0, 8, 13],
# S7
[4, 11, 2, 14, 15, 0, 8, 13, 3, 12, 9, 7, 5, 10, 6, 1,
13, 0, 11, 7, 4, 9, 1, 10, 14, 3, 5, 12, 2, 15, 8, 6,
1, 4, 11, 13, 12, 3, 7, 14, 10, 15, 6, 8, 0, 5, 9, 2,
6, 11, 13, 8, 1, 4, 10, 7, 9, 5, 0, 15, 14, 2, 3, 12],
# S8
[13, 2, 8, 4, 6, 15, 11, 1, 10, 9, 3, 14, 5, 0, 12, 7,
1, 15, 13, 8, 10, 3, 7, 4, 12, 5, 6, 11, 0, 14, 9, 2,
7, 11, 4, 1, 9, 12, 14, 2, 0, 6, 10, 13, 15, 3, 5, 8,
2, 1, 14, 7, 4, 10, 8, 13, 15, 12, 9, 0, 3, 5, 6, 11],
]
# 32-bit permutation function P used on the output of the S-boxes
__p = [
15, 6, 19, 20, 28, 11,
27, 16, 0, 14, 22, 25,
4, 17, 30, 9, 1, 7,
23,13, 31, 26, 2, 8,
18, 12, 29, 5, 21, 10,
3, 24
]
# final permutation IP^-1
__fp = [
39, 7, 47, 15, 55, 23, 63, 31,
38, 6, 46, 14, 54, 22, 62, 30,
37, 5, 45, 13, 53, 21, 61, 29,
36, 4, 44, 12, 52, 20, 60, 28,
```

```
35, 3, 43, 11, 51, 19, 59, 27,
34, 2, 42, 10, 50, 18, 58, 26,
33, 1, 41, 9, 49, 17, 57, 25,
32, 0, 40, 8, 48, 16, 56, 24
]
# Type of crypting being done
ENCRYPT = 0x00
DECRYPT = 0x01
# Initialisation
def init(self, key, mode=ECB, IV=None):
if len(key) != 8:
raise ValueError("Invalid DES key size. Key must be exactly 8 bytes long.")
self.block_size = 8
self.key_size = 8
self.__padding = ''
# Set the passed in variables
self.setMode(mode)
if IV:
self.setIV(IV)
self.L = []
self.R = []
self.Kn = [ [0] * 48 ] * 16 # 16 48-bit keys (K1 - K16)
self.final = []
self.setKey(key)
def getKey(self):
"""getKey() -> string"""
return self.__key
def setKey(self, key):
"""Will set the crypting key for this object. Must be 8 bytes."""
self.__key = key
self.__create_sub_keys()
def getMode(self):
"""getMode() -> pyDes.ECB or pyDes.CBC"""
return self.__mode
def setMode(self, mode):
"""Sets the type of crypting mode, pyDes.ECB or pyDes.CBC"""
self.__mode = mode
def getIV(self):
"""getIV() -> string"""
return self.__iv
def setIV(self, IV):
"""Will set the Initial Value, used in conjunction with CBC mode"""
if not IV or len(IV) != self.block_size:
raise ValueError("Invalid Initial Value (IV), must be a multiple of " + str(self.block_size) + " bytes")
self.__iv = IV
def getPadding(self):
"""getPadding() -> string of length 1. Padding character."""
return self.__padding
def __String_to_BitList(self, data):
"""Turn the string data, into a list of bits (1, 0)'s"""
l = len(data) * 8
result = [0] * l
pos = 0
for c in data:
i = 7
ch = ord©
while i >= 0:
if ch & (1 << i) != 0:
result[pos] = 1
```

```
else:
result[pos] = 0
pos += 1
i -= 1
return result
def __BitList_to_String(self, data):
“”“Turn the list of bits -> data, into a string”””
result = ‘’
pos = 0
c = 0
while pos < len(data):
c += data[pos] << (7 - (pos % 8))
if (pos % 8) == 7:
result += chr©
c = 0
pos += 1
return result
def __permutate(self, table, block):
“”“Permutate this block with the specified table”””
return map(lambda x: block[x], table)
# Transform the secret key, so that it is ready for data processing
# Create the 16 subkeys, K[1] - K[16]
def __create_sub_keys(self):
“”“Create the 16 subkeys K[1] to K[16] from the given key”””
key = self.__permutate(des.__pc1, self.__String_to_BitList(self.getKey()))
i = 0
# Split into Left and Right sections
self.L = key[:28]
self.R = key[28:]
while i < 16:
j = 0
# Perform circular left shifts
while j < des.__left_rotations[i]:
self.L.append(self.L[0])
del self.L[0]
self.R.append(self.R[0])
del self.R[0]
j += 1
# Create one of the 16 subkeys through pc2 permutation
self.Kn[i] = self.__permutate(des.__pc2, self.L + self.R)
i += 1
# Main part of the encryption algorithm, the number cruncher ?
def __des_crypt(self, block, crypt_type):
“”“Crypt the block of data through DES bit-manipulation”””
block = self.__permutate(des.__ip, block)
self.L = block[:32]
self.R = block[32:]
# Encryption starts from Kn[1] through to Kn[16]
if crypt_type == des.ENCRYPT:
iteration = 0
iteration_adjustment = 1
# Decryption starts from Kn[16] down to Kn[1]
else:
iteration = 15
iteration_adjustment = -1
i = 0
while i < 16:
# Make a copy of R[i-1], this will later become L[i]
tempR = self.R[:]
# Permutate R[i - 1] to start creating R[i]
```

```
self.R = self.__permutate(des.__expansion_table, self.R)
# Exclusive or R[i - 1] with K[i], create B[1] to B[8] whilst here
self.R = map(lambda x, y: x ^ y, self.R, self.Kn[iteration])
B = [self.R[:6], self.R[6:12], self.R[12:18], self.R[18:24], self.R[24:30], self.R[30:36], self.R[36:42],
self.R[42:]]
# Optimization: Replaced below commented code with above
#j = 0
#B = []
#while j < len(self.R):
# self.R[j] = self.R[j] ^ self.Kn[iteration][j]
# j += 1
# if j % 6 == 0:
# B.append(self.R[j-6:j])
# Permutate B[1] to B[8] using the S-Boxes
j = 0
Bn = [0] * 32
pos = 0
while j < 8:
# Work out the offsets
m = (B[j][0] << 1) + B[j][5]
n = (B[j][1] << 3) + (B[j][2] << 2) + (B[j][3] << 1) + B[j][4]
# Find the permutation value
v = des.__sbox[j][(m << 4) + n]
# Turn value into bits, add it to result: Bn
Bn[pos] = (v & 8) >> 3
Bn[pos + 1] = (v & 4) >> 2
Bn[pos + 2] = (v & 2) >> 1
Bn[pos + 3] = v & 1
pos += 4
j += 1
# Permutate the concatination of B[1] to B[8] (Bn)
self.R = self.__permutate(des.__p, Bn)
# Xor with L[i - 1]
self.R = map(lambda x, y: x ^ y, self.R, self.L)
# Optimization: This now replaces the below commented code
#j = 0
#while j < len(self.R):
# self.R[j] = self.R[j] ^ self.L[j]
# j += 1
# L[i] becomes R[i - 1]
self.L = tempR
i += 1
iteration += iteration_adjustment
# Final permutation of R[16]L[16]
self.final = self.__permutate(des.__fp, self.R + self.L)
return self.final
# Data to be encrypted/decrypted
def crypt(self, data, crypt_type):
“”“Crypt the data in blocks, running it through des_crypt()”””
# Error check the data
if not data:
return ‘’
if len(data) % self.block_size != 0:
if crypt_type == des.DECRYPT: # Decryption must work on 8 byte blocks
raise ValueError(“Invalid data length, data must be a multiple of " + str(self.block_size) + " bytes\n.”)
if not self.getPadding():
raise ValueError(“Invalid data length, data must be a multiple of " + str(self.block_size) + " bytes\n. Try
setting the optional padding character”)
else:
```

```
data += (self.block_size - (len(data) % self.block_size)) * self.getPadding()
# print “Len of data: %f” % (len(data) / self.block_size)
if self.getMode() == CBC:
if self.getIV():
iv = self.__String_to_BitList(self.getIV())
else:
raise ValueError(“For CBC mode, you must supply the Initial Value (IV) for ciphering”)
# Split the data into blocks, crypting each one seperately
i = 0
dict = {}
result = []
#cached = 0
#lines = 0
while i < len(data):
# Test code for caching encryption results
#lines += 1
#if dict.has_key(data[i:i+8]):
#print “Cached result for: %s” % data[i:i+8]
# cached += 1
# result.append(dict[data[i:i+8]])
# i += 8
# continue
block = self.__String_to_BitList(data[i:i+8])
# Xor with IV if using CBC mode
if self.getMode() == CBC:
if crypt_type == des.ENCRYPT:
block = map(lambda x, y: x ^ y, block, iv)
#j = 0
#while j < len(block):
# block[j] = block[j] ^ iv[j]
# j += 1
processed_block = self.__des_crypt(block, crypt_type)
if crypt_type == des.DECRYPT:
processed_block = map(lambda x, y: x ^ y, processed_block, iv)
#j = 0
#while j < len(processed_block):
# processed_block[j] = processed_block[j] ^ iv[j]
# j += 1
iv = block
else:
iv = processed_block
else:
processed_block = self.__des_crypt(block, crypt_type)
# Add the resulting crypted block to our list
#d = self.__BitList_to_String(processed_block)
#result.append(d)
result.append(self.__BitList_to_String(processed_block))
#dict[data[i:i+8]] = d
i += 8
# print “Lines: %d, cached: %d” % (lines, cached)
# Remove the padding from the last block
if crypt_type == des.DECRYPT and self.getPadding():
#print “Removing decrypt pad”
s = result[-1]
while s[-1] == self.getPadding():
s = s[:-1]
result[-1] = s
# Return the full crypted string
return ‘’.join(result)
def encrypt(self, data, pad=’’):
```

def encrypt(self, data, pad=’’):
“”“encrypt(data, [pad]) -> string
data : String to be encrypted
pad : Optional argument for encryption padding. Must only be one byte
The data must be a multiple of 8 bytes and will be encrypted
with the already specified key. Data does not have to be a
multiple of 8 bytes if the padding character is supplied, the
data will then be padded to a multiple of 8 bytes with this
pad character.
“””
self.__padding = pad
return self.crypt(data, des.ENCRYPT)
def decrypt(self, data, pad=’’):
“”“decrypt(data, [pad]) -> string
data : String to be encrypted
pad : Optional argument for decryption padding. Must only be one byte
The data must be a multiple of 8 bytes and will be decrypted
with the already specified key. If the optional padding character
is supplied, then the un-encypted data will have the padding characters
removed from the end of the string. This pad removal only occurs on the
last 8 bytes of the data (last data block).
“””
self.__padding = pad
return self.crypt(data, des.DECRYPT)
##

Triple DES

class triple_des:
“”“Triple DES encryption/decrytpion class
This algorithm uses the DES-EDE3 (when a 24 byte key is supplied) or
the DES-EDE2 (when a 16 byte key is supplied) encryption methods.
Supports ECB (Electronic Code Book) and CBC (Cypher Block Chaining) modes.
pyDes.des(key, [mode], [IV])
key -> The encryption key string, must be either 16 or 24 bytes long
mode -> Optional argument for encryption type, can be either pyDes.ECB
(Electronic Code Book), pyDes.CBC (Cypher Block Chaining)
IV -> Optional string argument, must be supplied if using CBC mode.
Must be 8 bytes in length.
“””
def **init**(self, key, mode=ECB, IV=None):
self.block_size = 8
self.setMode(mode)
self.__padding = ‘’
self.__iv = IV
self.setKey(key)
def getKey(self):
“”“getKey() -> string”""
return self.__key
def setKey(self, key):
“”“Will set the crypting key for this object. Either 16 or 24 bytes long.”""
self.key_size = 24 # Use DES-EDE3 mode
if len(key) != self.key_size:
if len(key) == 16: # Use DES-EDE2 mode
self.key_size = 16
else:
raise ValueError(“Invalid triple DES key size. Key must be either 16 or 24 bytes long”)
if self.getMode() == CBC and (not self.getIV() or len(self.getIV()) != self.block_size):
raise ValueError(“Invalid IV, must be 8 bytes in length”) ## TODO: Check this
# modes get handled later, since CBC goes on top of the triple-des
self.__key1 = des(key[:8])

```
self.__key1 = des(key[:8])
self.__key2 = des(key[8:16])
if self.key_size == 16:
self.__key3 = self.__key1
else:
self.__key3 = des(key[16:])
self.__key = key
def getMode(self):
“”“getMode() -> pyDes.ECB or pyDes.CBC”””
return self.__mode
def setMode(self, mode):
“”“Sets the type of crypting mode, pyDes.ECB or pyDes.CBC”””
self.__mode = mode
def getIV(self):
“”“getIV() -> string”””
return self.__iv
def setIV(self, IV):
“”“Will set the Initial Value, used in conjunction with CBC mode”””
self.__iv = IV
def xorstr( self, x, y ):
“”“Returns the bitwise xor of the bytes in two strings”””
if len(x) != len(y):
raise “string lengths differ %d %d” % (len(x), len(y))
ret = ‘’
for i in range(len(x)):
ret += chr(ord(x[i]) ^ ord(y[i]))
return ret
def encrypt(self, data, pad=’’):
“”“encrypt(data, [pad]) -> string
data : String to be encrypted
pad : Optional argument for encryption padding. Must only be one byte
The data must be a multiple of 8 bytes and will be encrypted
with the already specified key. Data does not have to be a
multiple of 8 bytes if the padding character is supplied, the
data will then be padded to a multiple of 8 bytes with this
pad character.
“””
if self.getMode() == ECB:
# simple
data = self.__key1.encrypt(data, pad)
data = self.__key2.decrypt(data)
return self.__key3.encrypt(data)
if self.getMode() == CBC:
raise “This code hasn’t been tested yet”
if len(data) % self.block_size != 0:
raise “CBC mode needs datalen to be a multiple of blocksize (ignoring padding for now)”
# simple
lastblock = self.getIV()
retdata = ‘’
for i in range( 0, len(data), self.block_size ):
thisblock = data[ i:i+self.block_size ]
# the XOR for CBC
thisblock = self.xorstr( lastblock, thisblock )
thisblock = self.__key1.encrypt(thisblock)
thisblock = self.__key2.decrypt(thisblock)
lastblock = self.__key3.encrypt(thisblock)
retdata += lastblock
return retdata
raise “Not reached”
def decrypt(self, data, pad=’’):
```

```
“”“decrypt(data, [pad]) -> string
data : String to be encrypted
pad : Optional argument for decryption padding. Must only be one byte
The data must be a multiple of 8 bytes and will be decrypted
with the already specified key. If the optional padding character
is supplied, then the un-encypted data will have the padding characters
removed from the end of the string. This pad removal only occurs on the
last 8 bytes of the data (last data block).
“””
if self.getMode() == ECB:
# simple
data = self.__key3.decrypt(data)
data = self.__key2.encrypt(data)
return self.__key1.decrypt(data, pad)
if self.getMode() == CBC:
if len(data) % self.block_size != 0:
raise “Can only decrypt multiples of blocksize”
lastblock = self.getIV()
retdata = ‘’
for i in range( 0, len(data), self.block_size ):
# can I arrange this better? probably…
cipherchunk = data[ i:i+self.block_size ]
thisblock = self.__key3.decrypt(cipherchunk)
thisblock = self.__key2.encrypt(thisblock)
thisblock = self.__key1.decrypt(thisblock)
retdata += self.xorstr( lastblock, thisblock )
lastblock = cipherchunk
return retdata
raise “Not reached”
################################################################################
```

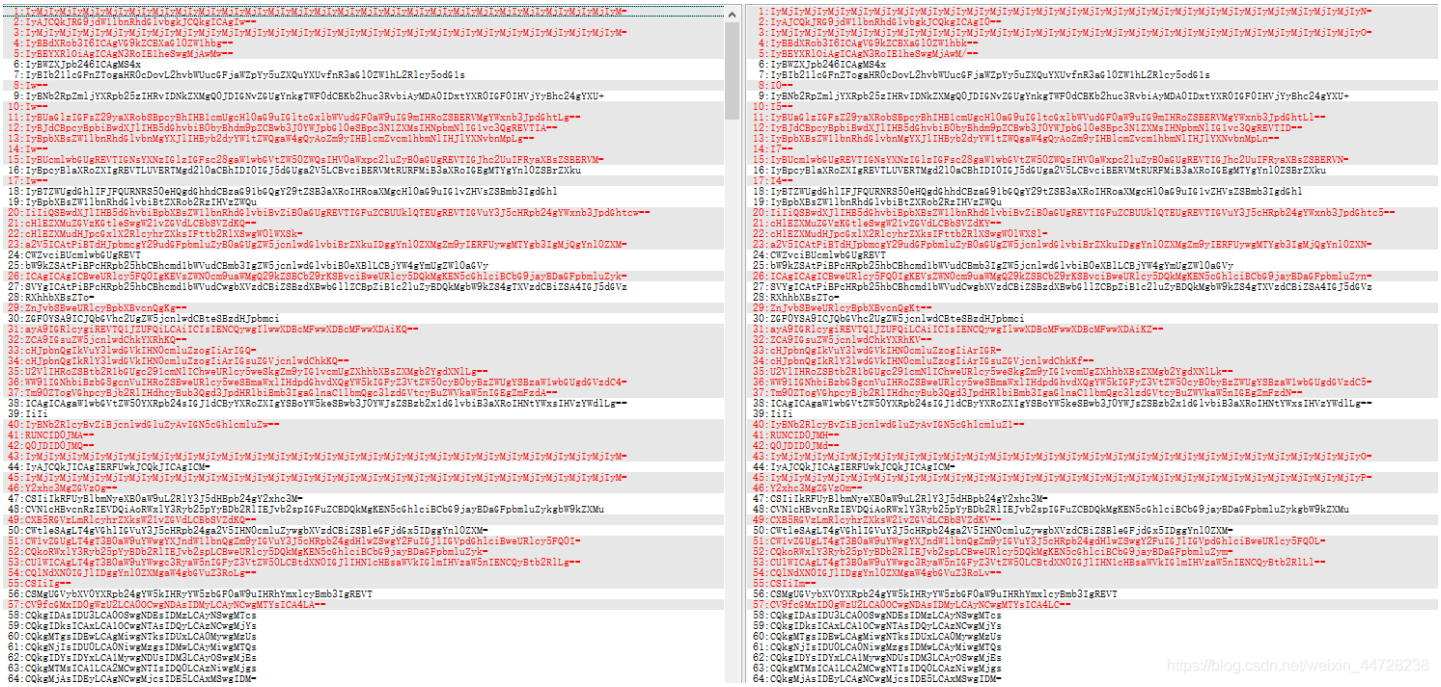
Examples

```
################################################################################ def example_triple_des():
from time import time
# Utility module
from binascii import unhexlify as unhex
# example shows triple-des encryption using the des class
print “Example of triple DES encryption in default ECB mode (DES-EDE3)\n”
print “Triple des using the des class (3 times)”
t = time()
k1 = des(unhex(“133457799BBCDFF1”))
k2 = des(unhex(“1122334455667788”))
k3 = des(unhex(“77661100DD223311”))
d = “Triple DES test string, to be encrypted and decrypted…”
print “Key1: %s” % k1.getKey()
print “Key2: %s” % k2.getKey()
print “Key3: %s” % k3.getKey()
print “Data: %s” % d
e1 = k1.encrypt(d)
e2 = k2.decrypt(e1)
e3 = k3.encrypt(e2)
print “Encrypted: " + e3
d3 = k3.decrypt(e3)
d2 = k2.encrypt(d3)
d1 = k1.decrypt(d2)
print “Decrypted: " + d1
print “DES time taken: %f (%d crypt operations)” % (time() - t, 6 * (len(d) / 8))
print “”
# Example below uses the triple-des class to achieve the same as above
```

```
print “Now using triple des class”
t = time()
t1 = triple_des(unhex(“133457799BBCDFF1112233445566778877661100DD223311”))
print “Key: %s” % t1.getKey()
print “Data: %s” % d
td1 = t1.encrypt(d)
print “Encrypted: " + td1
td2 = t1.decrypt(td1)
print “Decrypted: " + td2
print “Triple DES time taken: %f (%d crypt operations)” % (time() - t, 6 * (len(d) / 8)) def example_des():
from time import time
# example of DES encrypting in CBC mode with the IV of “\0\0\0\0\0\0\0\0”
print “Example of DES encryption using CBC mode\n”
t = time()
k = des(“DESCRYPT”, CBC, “\0\0\0\0\0\0\0\0”)
data = “DES encryption algorithm”
print “Key : " + k.getKey()
print “Data : " + data
d = k.encrypt(data)
print “Encrypted: " + d
d = k.decrypt(d)
print “Decrypted: " + d
print “DES time taken: %f (6 crypt operations)” % (time() - t)
print “” def test():
example_des()
example_triple_des() def fulltest():
# This should not produce any unexpected errors or exceptions
from binascii import unhexlify as unhex
from binascii import hexlify as dohex
test()
print “”
k = des(”\0\0\0\0\0\0\0\0”, CBC, “\0\0\0\0\0\0\0\0”)
d = k.encrypt(“DES encryption algorithm”)
if k.decrypt(d) != “DES encryption algorithm”:
print “Test 1 Error: Unencypted data block does not match start data”
k = des(”\0\0\0\0\0\0\0\0”, CBC, “\0\0\0\0\0\0\0\0”)
d = k.encrypt(“Default string of text”, '')
if k.decrypt(d, '') != “Default string of text”:
print “Test 2 Error: Unencypted data block does not match start data”
k = des(”\r\n\tABC\r\n”)
d = k.encrypt(“String to Pad”, '')
if k.decrypt(d) != "String to Pad**”:
print “'%s'” % k.decrypt(d)
print “Test 3 Error: Unencypted data block does not match start data”
k = des("\r\n\tABC\r\n")
d = k.encrypt(unhex(“000102030405060708FF8FDCB04080”), unhex(“44”))
if k.decrypt(d, unhex(“44”)) != unhex(“000102030405060708FF8FDCB04080”):
print “Test 4a Error: Unencypted data block does not match start data”
if k.decrypt(d) != unhex(“000102030405060708FF8FDCB0408044”):
print “Test 4b Error: Unencypted data block does not match start data”
k = triple_des(“MyDesKey\r\n\tABC\r\n0987543")
d =
k.encrypt(unhex(“000102030405060708FF8FDCB040800001020304050607
08FF8FDCB040800001020304050607
08FF8FDCB040800001020304050607
08FF8FDCB040800001020304050607
08FF8FDCB040800001020304050607
08FF8FDCB04080”))
if k.decrypt(d) !=
unhex(“000102030405060708FF8FDCB040800001020304050607
08FF8FDCB040800001020304050607
08FF8FDCB040800001020304050607
08FF8FDCB040800001020304050607
08FF8FDCB040800001020304050607
08FF8FDCB040
```

```
80”):
print “Test 5 Error: Unencypted data block does not match start data”
k = triple_des("\r\n\tABC\r\n0987543”)
d =
k.encrypt(unhex(“0001020304050607 08FF8FDCB04080000102030405060708FF8FDCB04080000102030405060708FF8FDCB04080
00010203040506070 8FF8FDCB04080000102030405060708FF
8FDCB04080000102030405060708FF8FDCB04080000102030405060708FF8FDCB04080000102030405060708FF8FDCB04080”))
if k.decrypt(d) !=
unhex(“0001020304050607 08FF8FDCB04080000102030405060708FF8FDCB04080000102030405060708FF8FDCB0408000010203
0405060708FF8FDCB0408000 0102030405060
708FF8FDCB04080000102030405060708FF8FDCB04080000102030405060708FF8FDCB04080000102030405060708FF8FDCB040
80”):
print “Test 6 Error: Unencypted data block does not match start data” def filetest():
from time import time
f = open(“pyDes.py”, “rb+”)
d = f.read()
f.close()
t = time()
k = des(“MyDESKey”)
d = k.encrypt(d, " ")
f = open(“pyDes.py.enc”, “wb+”)
f.write(d)
f.close()
d = k.decrypt(d, " ")
f = open(“pyDes.py.dec”, “wb+”)
f.write(d)
f.close()
print “DES file test time: %f” % (time() - t) def profile():
import profile
profile.run(‘fulltest()’)
#profile.run(‘filetest()’) if name == ‘main’:
test()
#fulltest()
#filetest()
#profile()
```

我们将原网页代码使用base64进行编码与我们得到的编码进行对比



可以看到二者之间还是有差距的，主要在于某一些字段的最后会出现不同。由于base64编码本身的编码方式，最后几位是否有填充出现时影响编码的重要原因。如图所示的4个填充实际上在解码过程中会忽略，也就是说，我们可以在这里放任何位，这是隐藏信息的好地方。

```
# coding=utf-8
import base64
import string
def tobin(data):#转化为二进制数
    #b64是小写字母、大写字母、数字和+/的组合
    b64table = string.ascii_uppercase + string.ascii_lowercase + string.igits + '+/'
    index = b64table.find(data)
    return format(index, '06b')
def toStr(bin):#转化为字符串
    binlen = len(bin)
    out = ''
    for i in range(0, binlen, 8):
        out += chr(int(bin[i:i+8], 2))
    return out
out = ''
for line in open('cip_d0283b2c5b4b87423e350f8640a0001e', 'rb'):
    line = line.strip()
    #如果最后是==，则说明填充了4位数
    if line.strip()[-2:] == '==':
        binstr = tobin(line[-3:-2])
        out += binstr[-4:]
        print binstr[-4:]
    #如果最后是=，说明填充了1个数
    elif line.strip()[-1:] == '=':
        binstr = tobin(line[-2:-1])
        out += binstr[-2:]
        print binstr[-2:]
print out
print toStr(out)
```

结果为:ROIS{base_GA_caN_b3_d1ffeR3nT}

题目15：wacky-agent-75
难度系数： 8.0
题目来源： asis-ctf-quals-2016
题目描述：暂无

解题：
首先我们可以知道这个文件为.xz文件，解压后是一个经过base64编码的文件，其中有两个文本块，一个是base64

暂时不会:)

题目16：cr3-what-is-this-encryption
难度系数：
题目来源： alexctf-2017
题目描述：Fady同学以为你是菜鸟，不怕你看到他发的东西。他以明文形式将下面这些东西发给了他的朋友
p=0xa6055ec186de51800ddd6fcbf0192384ff42d707a55f57af4fcfb0d1dc7bd97055e8275cd4b78ec63c5d592f567c66393a061324aa2e6a8d8fc2a910cbee1ed9
q=0xfa0f9463ea0a93b929c099320d31c277e0b0dbc65b189ed76124f5a1218f5d91fd0102a4c8de11f28be5e4d0ae91ab319f4537e97ed74bc663e972a4a9119307
e=0x6d1fdab4ce3217b3fc32c9ed480a31d067fd57d93a9ab52b472dc393ab7852fbcb11abbebfd6aaae8032db1316dc22d3f7c3d631e24df13ef23d3b381a1c3e04abcc745d402ee3a031ac2718fae63b240837b4f657f29ca4702da9af22a3a019d68904a969ddb01bcf941df70af042f4fae5cbeb9c2151b324f387e525094c41
c=0x7fe1a4f743675d1987d25d38111fae0f78bbea6852cba5beda47db76d119a3efe24cb04b9449f53becd43b0b46e269826a983f832abb53b7a7e24a43ad15378344ed5c20f51e268186d24c76050c1e73647523bd5f91d9b6ad3e86bbf9126588b1dee21e6997372e36c3e74284734748891829665086e0dc523ed23c386bb520 他严重低估了我们的解密能力

解题：
这道题直接用RSA的原理计算数字，并最终转化为字符串即可

```
import gmpy2
import binascii
from Crypto.PublicKey import RSA
p='0xa6055ec186de51800ddd6fcbf0192384ff42d707a55f57af4fcfb0d1dc7bd97055e8275cd4b78ec63c5d592f567c66393a061324aa2
e6a8d8fc2a910cbee1ed9'
q="0xfa0f9463ea0a93b929c099320d31c277e0b0dbc65b189ed76124f5a1218f5d91fd0102a4c8de11f28be5e4d0ae91ab319f4537e97ed
74bc663e972a4a9119307"
e='0x6d1fdab4ce3217b3fc32c9ed480a31d067fd57d93a9ab52b472dc393ab7852fbcb11abbebfd6aaae8032db1316dc22d3f7c3d631e24
df13ef23d3b381a1c3e04abcc745d402ee3a031ac2718fae63b240837b4f657f29ca4702da9af22a3a019d68904a969ddb01bcf941df70af
042f4fae5cbeb9c2151b324f387e525094c41'
c='0x7fe1a4f743675d1987d25d38111fae0f78bbea6852cba5beda47db76d119a3efe24cb04b9449f53becd43b0b46e269826a983f832ab
b53b7a7e24a43ad15378344ed5c20f51e268186d24c76050c1e73647523bd5f91d9b6ad3e86bbf9126588b1dee21e6997372e36c3e742847
34748891829665086e0dc523ed23c386bb520'

pnew=gmpy2.mpz(int(p,16))
qnew=gmpy2.mpz(int(q,16))
enew=gmpy2.mpz(int(e,16))
dnew = long(gmpy2.invert(enew,(pnew-1)*(qnew-1)))
nnew=long(gmpy2.mpz(pnew*qnew))
#pubkey=RSA.construct((long(nnew), long(enew)))
#privatekey=RSA.construct((long(nnew), long(enew), long(dnew), long(pnew), long(qnew)))
secrect=gmpy2.mpz(int(c,16))
plaintext=gmpy2.powmod(secrect,dnew,nnew)
#print binascii.unhexlify(hex(plaintext)[2:-1])
print binascii.a2b_hex(hex(plaintext)[2:])
```

题目17：flag_in_your_hand
难度系数：:9.0
题目来源： CISCN-2018-Quals
题目描述：暂无

解题:
可以看到一个网页，只要输入正确的token即可
我们查看网页的代码，可以看到，当ic为true的时候，就会返回正确的flag

```
function showFlag() {
    var t = document.getElementById("flagTitle");
    var f = document.getElementById("flag");
    t.innerText = !!ic ? "You got the flag below!!" : "Wrong!";
    t.className = !!ic ? "rightflag" : "wrongflag";
    f.innerText = fg;
}
```

于是去阅读js代码，可以看到，要向得到正确的ic，即ic的值比a的值小3，于是我们得到ic=security-xbu，最终输入后得到结果Renlbyd8Fgg5hawvQm7TDQ

```
,
function ck(s) {
    try {
        ic
    } catch (e) {
        return;
    }
    var a = [118, 104, 102, 120, 117, 108, 119, 124, 48,123,101,120];
    if (s.length == a.length) {
        for (i = 0; i < s.length; i++) {
            if (a[i] - s.charCodeAt(i) != 3)
                return ic = false;
        }
        return ic = true;
    }
    return ic = false;
}
```

题目18：Handicraft_RSA
难度系数： 9.0
题目来源： ASIS-CTF-Finals-2017
题目描述：有人正在他老房子的地下室里开发自己的RSA系统。 证明他这个RSA系统只在他的地下室有效！

解题：
通过以下方法可以获得文件的格式：

```
import magic
data=open("Handicraft_RSA").read()
print magic.from_buffer(data)
```

可以得知文件为.xz文件，于是我们使用winHex将其保存为.xz文件并解压，获得了一个可以阅读的代码文件，内容如下：

```
handicraft_rsa/
000755 000765 000024 00000000000 13154462657 015146 5
ustar 00factoreal staff
000000 000000
handicraft_rsa/._handicraft_rsa.py
000755 000765 000024 00000000317 13154462657 020711 0
ustar 00factoreal staff
```

000000 000000
Mac OS X 2 ⓪ ⓪
ATTR ⓪ ⓪ ⓪
com.macromates.selectionRange ⓪
com.macromates.visibleIndex 260
handicraft_rsa/handicraft_rsa.py
000755 000765 000024 00000001510 13154462657 020470 0
ustar 00factoreal staff
000000 000000
#!/usr/bin/python
from Crypto.Util.number import * from Crypto.PublicKey import RSA from
secret import s, FLAG
def gen_prime(s):
while True:
r = getPrime(s)
R = [r]
t = int(5 * s / 2) + 1
for i in range(0, t):
R.append(r + getRandomRange(0, 4 * s ** 2))
p = reduce(lambda a, b: a * b, R, 2) + 1
if isPrime§:
if len(bin§[2:]) == 1024:
return p
while True:
p = gen_prime(s)
q = gen_prime(s)
n = p * q
e = 65537
d = inverse(e, (p-1)*(q-1))
if len(bin(n)[2:]) == 2048:
break
msg = FLAG key = RSA.construct((long(n), long(e), long(d), long§,
long§)) for _ in xrange(s):
enc = key.encrypt(msg, 0)[0]
msg = enc
print key.publickey().exportKey() print '-' * 76 print
enc.encode('base64') print '-' * 76
handicraft_rsa/output.txt
000644 000765 000024 00000001673 13154462657 017256 0
ustar 00factoreal staff
000000 000000
-----BEGIN PUBLIC KEY----- MIIBIjANBgkqhkiG9w0BAQEFAAOCAQ8AMIIBCgKCAQEAq+m7iHurBa9G8ujEiTpZ
71aHOVNhQXpd6jCQNhwMN3hD6JHkv0HSxmJwfGe0EnXDtjRraWmS6OYzT4+LSrXs
z9IkWGzRlJ4IC7WHS8D3NWIWYHCP4TRt2N0TIWXWm9nFCrEXqQ3IWgYQpQvKzsds
etnIZJL1tf1wQzGE6rbkbvURIUBbzBSuidkmi0kY5Qxp2Jfb6OUl647zx2dPxJpD
ffSCNffVIDUYOvrgYxlhs5HmCF3XECC3VfaKtRceL5JM8R0qz5nVU2Ns8hPvSVP+
7/i7G447cjW151si0joB7RpBplu44Vk8TXXDAk0JZdW6KwJn7ITaX04AAAAAAAAA
AQIDAQAB
-----END PUBLIC KEY-----
---
eER0JNlcZYx/t+7InRvv8s8zyMw8dYspZlne0MQUatQNcnDL/wnHtkAoNdCalQkpcbnZeAz4qeMX
5GBmsO+BXyAKDueMA4uy3fw2k/dqFSsZFiB7I9M0oEkqUja52IMpkGDJ2eXGj9WHe4mqknilayS4
2o4p9b0Qlz754qqRgkuaKzPWkZPKynULAtFXF39zm6dPl/jUA2BEo5WBoPzsCzwRmdr6QmJXTsau
5BAQC5qdIkmCNq7+NLY1fjOmSEF/W+mdQvcwYPbe2zezroCiLiPNZnoABfmPbWAcASVU6M0YxvnX
sh2YjkyLFf4cJSgroM3Aw4fVz3PPSsAQyCFKBA==

这是一个RSA加密的过程，该过程随机生成了公钥和私钥并对flag进行了加密，我们只要对其进行解密即可。
首先我们需要获取公钥，从而来获得私钥，这可以使用openssl来完成

分解后得到p=
139457081371053313087662621808811891689477698775602541222732432884929677435971504758581219546068100871560676389156360422
970589688848020499752936702307974617390996217688749392344211044595211963580524376876607487048719085184308509979502505202
804812382023512342185380439620200563119485952705668730322944000000001
q=155617827023249833340719354421664777126919280716316528121008762838820577123085292134385394346751341309377546683859340593
439660968379640585296350265350950535158375685103003837903550191128377455111656903429282868722284520586387794090131818535
032744071918282383650099890243578253423157468632973312000000000000001
e=65537

利用RSAtool生成了一个私钥文件private.pem，并直接利用这个private.pem进行解密。（这道题中用RSA进行了多次的加密）

```
from Crypto.PublicKey import RSA
import base64
from Crypto.Cipher import PKCS1_v1_5 as Cipher_pkcs1_v1_5
from Crypto import Random
with open('private.pem') as f:
    p = f.read()
    rsakey = RSA.importKey(p)
    n=rsakey.n
    e=rsakey.e
    d=rsakey.d
    private_key = RSA.construct((long(n), long(e), long(d)))
    cipher = Cipher_pkcs1_v1_5.new(rsakey)
random_generator = Random.new().read
c= base64.b64decode("eER0JNIcZYx/t+7lnRvv8s8zyMw8dYspZlne0MQUatQNcnDL/wnHtkAoNdCalQkpcbnZeAz4qeMX5GBmsO+BXyAKDue
MA4uy3fw2k/dqFSsZFiB7I9M0oEkqUja52IMpkGDJ2eXGj9WHe4mqkniIayS42o4p9b0Qlz754qqRgkuaKzPWkZPKynULAtFXF39zm6dPI/jUA2B
Eo5WBoPzsCzwRmdr6QmJXTsau5BAQC5qdIkmCNq7+NLY1fjOmSEF/W+mdQvcwYPbe2zezroCiLiPNZnoABfmPbWAcASVU6M0YxvnXsh2YjkyLFf4
cJSgroM3Aw4fVz3PPSsAQyCFKBA==")
for s in range(18,23):
    msg = c
    for _ in range(s):#_只是一个循环的标志
        msg = private_key.decrypt(msg)
    print repr(msg)
```

题目20：in-plain-sight
难度系数： 9.0
题目来源： bsidessf-ctf-2017
题目描述：这次的挑战并不难：你只需要对隐藏的密文进行解密。为了让解密更加简单，我会给你除了HiddenCiphertext以外你需要的所有东西，你要做的就是自己将密文找出来！ 你需要： 算法：AES-256-CBC 密钥：c086e08ad8ee0ebe7c2320099cfec9eea9a346a108570a4f6494cfe7c2a30ee1 IV：0a0e176722a95a623f47fa17f02cc16a

此时就只需要找出密文即可，emmm这道题给出的信息就这么些所以自然想到把几个字符串挨个试了，当我们把HiddenCiphertext作为密文时，恰好可以解出
FLAG:1d010f248d\x01

```
import binascii
from Crypto.Cipher import AES
k = binascii.unhexlify('c086e08ad8ee0ebe7c2320099cfec9eea9a346a108570a4f6494cfe7c2a30ee1')
iv = binascii.unhexlify('0a0e176722a95a623f47fa17f02cc16a')
c = 'HiddenCiphertext'
aes = AES.new(k, AES.MODE_CBC, iv)
print aes.decrypt(c)
```

题目21：Decrypt-the-Message
难度系数：
题目来源： su-ctf-quals-2014
题目描述：解密这段信息！

解题：
这道题是poem_code，我们可以使用工具https://github.com/abpolym/crypto-tools/tree/master/poemcode，具体原理在https://wmbriggs.com/post/1001/有讲解

题目21：babyrsa
难度系数：
题目来源： XCTF 4th-QCTF-2018
题目描述：暂无

解题：
``

```
from pwn import *

context.log_level = 'WARN'

def num_to_bytes(n):
    b = hex(n)[2:].strip('L')
    b = '0' + b if len(b)%2 == 1 else b
    return b.decode('hex')

e=0x10001
n=0x0b765daa79117afe1a77da7ff8122872bbcbddb322bb078fe0786dc40c9033fadd639adc48c3f2627fb7cb59bb0658707fe516967464
439bdec2d6479fa3745f57c0a5ca255812f0884978b2a8aaeb750e0228cbe28a1e5a63bf0309b32a577eecea66f7610a9a4e720649129e9d
c2115db9d4f34dc17f8b0806213c035e22f2c5054ae584b440def00afbccd458d020cae5fd1138be6507bc0b1a10da7e75def484c5fc1fcb
13d11be691670cf38b487de9c4bde6c2c689be5adab08b486599b619a0790c0b2d70c9c461346966bcbae53c5007d0146fc520fa6e3106fb
fc89905220778870a7119831c17f98628563ca020652d18d72203529a784ca73716db
c=0x4f377296a19b3a25078d614e1c92ff632d3e3ded772c4445b75e468a9405de05d15c77532964120ae11f8655b68a630607df0568a743
9bc694486ae50b5c0c8507e5eecdea4654eeff3e75fb8396e505a36b0af40bd5011990663a7655b91c9e6ed2d770525e4698dec9455db17d
b38fa4b99b53438b9e09000187949327980ca903d0eef114afc42b771657ea5458a4cb399212e943d139b7ceb6d5721f546b75cd53d65e02
5f4df7eb8637152ecbb6725962c7f66b714556d754f41555c691a34a798515f1e2a69c129047cb29a9eef466c206a7f4dbc2cea1a46a39ad
3349a7db56c1c997dc181b1afcb76fa1bbbf118a4ab5c515e274ab2250dba1872be0

upper=n
lower=0
k=1
while True:
    io=remote('47.96.239.28',23333)
    io.recvuntil('You can input ciphertext(hexdecimal) now\n')
    power=pow(2,k,n)
    new_c=(pow(power,e,n)*c)%n
    new_c=hex(new_c)[2:].strip('L')
    io.sendline(new_c)
    data=io.recvline()[:-1]
    io.close()
    if data=="even":
        print 'Round %d: even' % k
        upper=(upper+lower)/2
    if data=="odd":
        print 'Round %d: odd' % k
        lower=(upper+lower)/2
    if data=="error": break
    if (upper-lower)<2: break
    k+=1

flag=num_to_bytes(upper)[:-1]+'}'
print flag
```