攻防世界Crypto第一页-进阶区

原创

人生若只如初见Crypto 于 2020-04-01 11:40:35 发布 1343 收藏 2

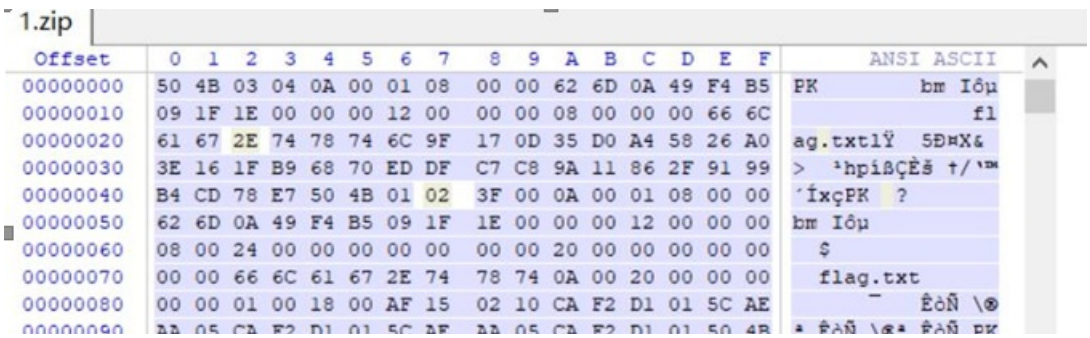
文章标签： 密码学

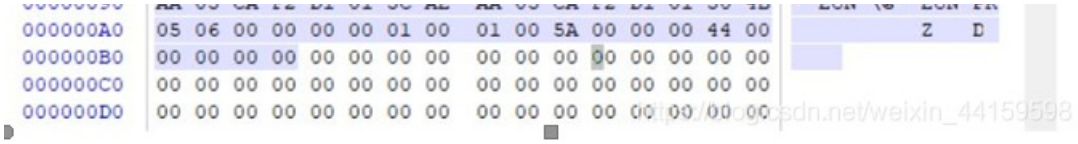
本文链接：https://blog.csdn.net/weixin_44159598/article/details/103346365

你猜猜

- 打开附件明显为 16 进制，查询得知为 zip 的文件开头

- 打开 Winhex 后，新建文件，将内容导入保存为 1.zip，打开压缩包



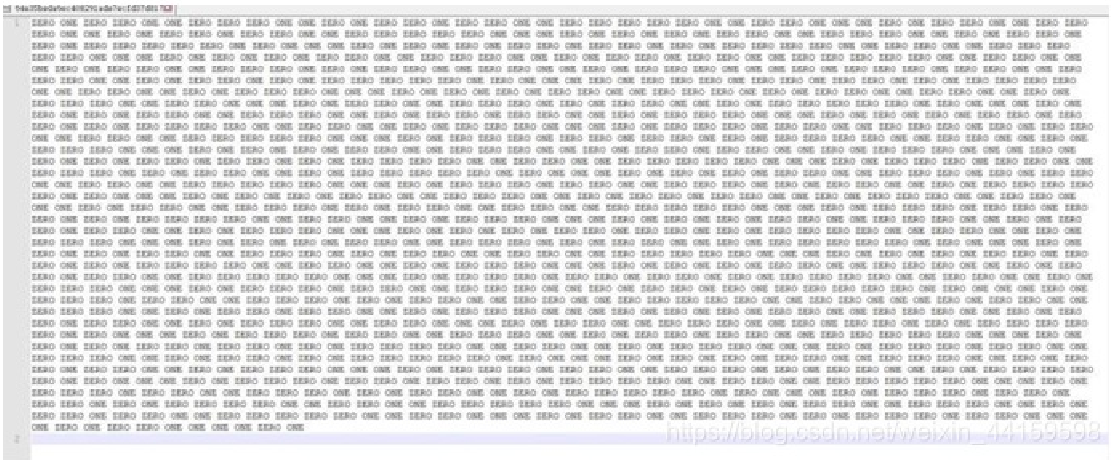


- 此时发现压缩包有密码，使用 ziperello 爆破解密

- 解出flag

enc


enc
最佳Writeup由admin提供
WP
建议
难度系数：
1.0
题目来源：暂无
题目描述：Fady不是很理解加密与编码的区别 所以他使用编码而不是加密，给他的朋友传了一些秘密的消息。
题目场景：暂无
题目附件：
附件1


https://blog.csdn.net/weixin_44159598

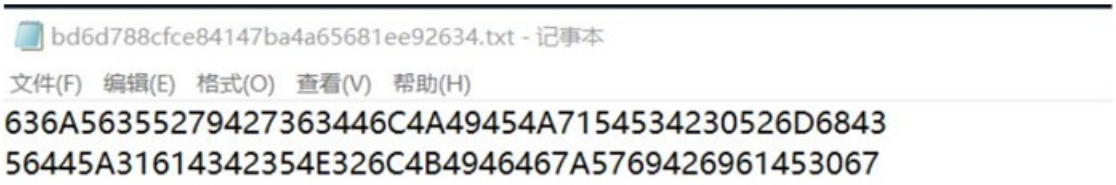
- 可以看到为 zero 和 one ， 明显为二进制，全部转换为 0 和 1,这时候可以利用 word 等专业文字编辑工具

- 将二进制转为字符串
- 明显的 base64 编码，base64 解密,摩斯密码

- Flag 即为 ALEXCTF{TH15_1S_5UP3R_5ECR3T_TXT},最后一步的小坑，将 O 替换为“—”

告诉你个秘密

附件为↵



猜测为 16 进制，转换为字符串↵

Base64 解码↵

发现直接分组，看到键盘上每组字母之间有规律，所以为键盘密码↵

解密后 flag 为 tongyuan↵

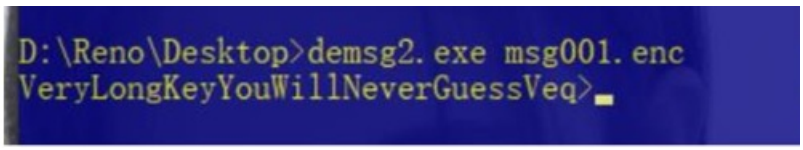
Easy——one

```
#include <stdlib.h>
#include <stdio.h>
#include <string.h>


int main(int argc, char **argv) {
if (argc != 3) {
printf("USAGE: %s INPUT OUTPUT\n", argv[0]);
return 0;
}
FILE* input    = fopen(argv[1], "rb");
FILE* output = fopen(argv[2], "wb");
if (!input || !output) {
printf("Error\n");
return 0;
}
char k[] = "CENSORED";
char c, p, t = 0;
int i = 0;
while ((p = fgetc(input)) != EOF) {
c = (p + (k[i % strlen(k)] ^ t) + i*i) & 0xff;
t = p;
i++;
fputc(c, output);
}
return 0
}
```

- 是使用 k[]和 input 经过加密算法后生成 output
- 文件 msg001.enc 和 msg001 是对应的，应该是用 k[] 和 msg001(input) 生成了msg001.enc(output)，在这个加密过程中，k[]好像是个常量，代码中给出的应该是个例子，不是真实的 k[]，对 msg001 加密后并不是 msg001.enc，看来我们需要先找到这个真实的 k[]了。
- 根据 msg001 和 msg001.enc，把算法逆过来算出 k[]：

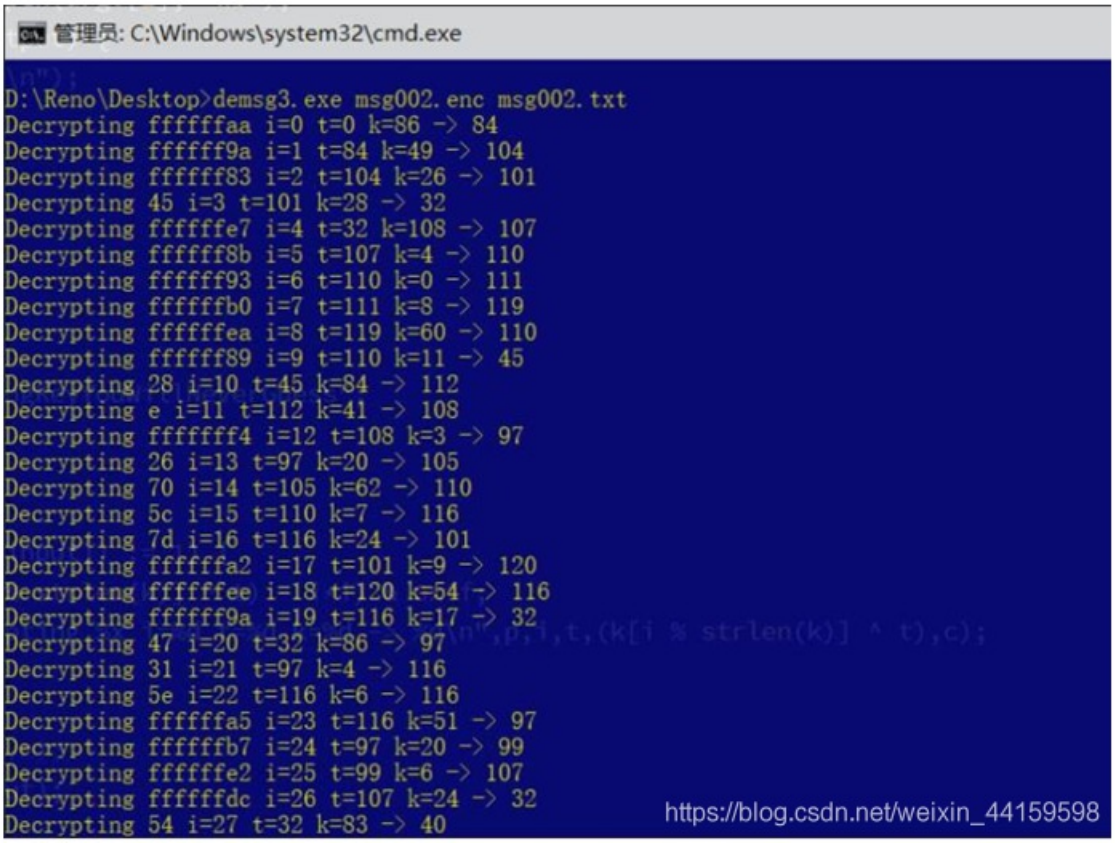
```
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
int main(int argc, char **argv) {
if (argc != 2) {
printf("USAGE: %s INPUT OUTPUT\n", argv[0]);
return 0;
}
//FILE* input    = fopen(argv[1], "rb");
FILE* input = fopen(argv[1], "rb");

}


if (!input) {
printf("Error\n");
return 0;
}
char c, p, t = 0;
int i = 0;
char w[] = "Hi! This is only test message\n";    //原来 input  中的值
unsigned int j = 0;
while ((p = fgetc(input)) != EOF) {
// printf("read %d", p);
for (j=31;j<125;j++) {
c = (p - (j ^ t) - i*i) & 0xff;
if (c == w[i]) {
printf("%c",j);
t = c;
i++;
break;
}
}
}
return 0;
}
```

编译之后对 main 传参：



int main(int argc, char **argv)argc 表示参数个数，即是后面 argv 数组的元素个数，不用输入，会根据传入参数计算，只需传递 argv 数组的元素即可。argv 数组的第一个元素（argv[0]）是程序名，传参的时候注意 argc 的值加上这个元素，其余的元素自行传递。这样就得到 k[]：VeryLongKeyYouWillNeverGuess,然后利用 k 再对 msg002.enc 进行解密，得到 msg002.enc 对应的明文:

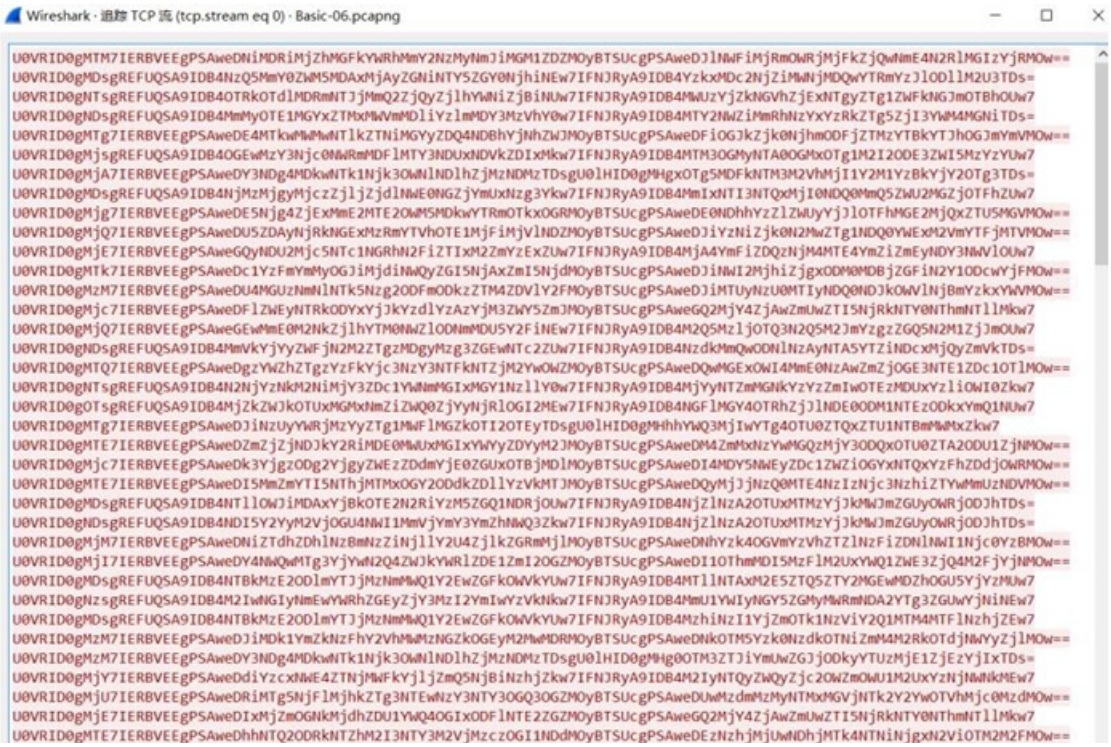
```
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
int main(int argc, char **argv) {
if (argc != 3) {
printf("USAGE: %s INPUT OUTPUT\n", argv[0]);


return 0;
}
FILE* input    = fopen(argv[1], "rb");
FILE* output = fopen(argv[2], "wb");
if (!input || !output) {
printf("Error\n");
return 0;
}
char c, p, t = 0;
int i = 0;
char k[] = "VeryLongKeyYouWillNeverGuess";
i = 0;
c, p, t = 0;
int g = 0;
while ((p = fgetc(input)) != EOF) {
c = (p - (k[i % strlen(k)] ^ t) - i*i) & 0xff;
printf("Decrypting %x i=%d t=%d k=%d -> %d\n",p,i,t,(k[i % strlen(k)] ^ t),c);
t = c;
i++;
//printf("%c",c);
fputc(c, output);
g++;
if (g>450) {break;}
}
return 0;
}
```



- 最终得到明文：The known-plaintext attack (KPA) is an attack model for cryptanalysis where the attacker has samples of both the plaintext (called a crib), and its encrypted version (ciphertext). These canbe used to reveal further secret information such as secret keys and code books. The term"crib" originated at Bletchley Park, the British World War Ⅱ decryption operation.The flag is CTF{6d5eba48508efb13dc87220879306619}鴐牖[?顆剝.}9 壺?F*損犄溦烇?5 蹈

说我作弊需要证据

- 使用 wireshark 打开所给流量包后，进行 tcp 流追踪



- 为 base64 加密后的数据，解密

```
SEQ = 13; DATA = 0x3b04b26a0adada2f67326bb0c5d6L; SIG =
0x2e5ab24f9dc21df406a87de0b3b4L; SEQ = 0; DATA =
0x7492f4ec9001202dcb569df468b4L; SIG = 0xc9107666b1cc040a4fc2e89e3e7L
SEQ = 5; DATA = 0x94d97e04f52c2d6f42f9aacbf0b5L; SIG =
0x1e3b6d4eaf11582e85ead4bf90a9L; SEQ = 4; DATA =
0x2c29150f1e311ef09bc9f06735acL; SIG = 0x1665fb2da761c4de89f27ac80cbL
SEQ = 18; DATA = 0x181901c059de3b0f2d4840ab3aebL; SIG =
0x1b8bdf9468f81ce33a0da2a8bfbeL; SEQ = 2; DATA =
0x8a03676745df01e16745145dd212L; SIG = 0x1378c25048c19853b6817eb9363aL
SEQ = 20; DATA = 0x674880905956979ce49af33433L; SIG =
0x198901d5373ea225cc5c0db66987L; SEQ = 0; DATA =
0x633282273f9cf7e5a44fcbe1787bL; SIG = 0x2b15275412244442d9ee60fc91aeL
SEQ = 28; DATA = 0x19688f112a61169c9090a4f9918dL; SIG =
0x1448ac6eee2b2e91a0a6241e590eL; SEQ = 24; DATA =
0x59d0264d4a134fa5a91521b25e46L; SIG = 0x2bc3bf947c0e85444aa13efa1c15L
SEQ = 21; DATA = 0xd24562795754da7abe213ffc11eL; SIG =
0x208babd43638118bfbfa24675ee9L; SEQ = 19; DATA =
0x75c1fbc28bb27b5d2db9601fb967L; SIG = 0x2b5b628bf8183400cdab7f5870b1L
SEQ = 33; DATA = 0x580e36ce59978681f893e38d5ecaL; SIG =
0x2b15275412244442d9ee60fc91aeL; SEQ = 27; DATA =
0x1eea254d861b2dc7ec03b37ef9fbL; SIG = 0xd6268f00fe0e2964d56458f59e2L
SEQ = 24; DATA = 0xa02a43cdf9aa345fe83f059cab4L; SIG =
0x3d939c9477d93bfc83dd97c5f2f9L; SEQ = 4; DATA =
0x2edb62eac7c6e83082387da0576eL; SIG = 0x77d2d083e702509a6b471242fedL
SEQ = 14; DATA = 0x83afae83c1db7776751d56c3f09fL; SIG =
0x400a19b82a4700ffc8a7515d7599L; SEQ = 5; DATA =
0x7ccc3d3cb267d75acf0b10f579ecL; SIG = 0x26256f0cdc63fb0913051c9b9b4fL
```

- 发现前面为编号，后面为传输的数据即 DATA，题目所给 RSA 参数，求出各自的 d,p,q即可解出 flag

```python
from Crypto.PublicKey import RSA
import gmpy2
import base64
#Alice's
A_n = 169620613905294892430494833347 4767
A_p = 38456719616722997
A_q = 44106885765559411
#Bob's
B_n = 3104649130901425335933838103517383
B_p = 49662237675630289
B_q = 62515288803124247

A_phi = (A_p - 1) * (A_q - 1)
B_phi = (B_p - 1) * (B_q - 1)

e = 65537

A_d = int(gmpy2.invert(e, A_phi))
B_d = int(gmpy2.invert(e, B_phi))

A_rsa = RSA.construct( (A_n, e, A_d) )
B_rsa = RSA.construct( (B_n, e, B_d) )
```

```
flag = {}
with open('zero_one') as f:
    for s in f.readlines():
        line = str(base64.b64decode(s), encoding = 'utf8')
        seq = int(line.split(';')[0].split(' ')[2])
        data = int(line.split('0x')[1].split('L;')[0], 16)
        sig = int(line.split('0x')[2].rstrip('L;\n'), 16)
        decry = B_rsa.decrypt(data)
        signcheck = A_rsa.sign(decry, '')[0]

        if signcheck == sig:
            flag[seq] = chr(decry)
dic = sorted(flag.items(), key = lambda item:item[0])  #对字典按键值进行排序，返回值为列表
print(dic)
f = ''
for i in dic:
    f += i[1]
print(f)
```


- Flag 为 flag{n0th1ng_t0_533_h3r3_m0v3_0n}

X_xor_md5

```
010  55 30 49 4C 56 D2 73 70  12 45 A8 BA 85 C0 3E 53  U0ILVÒsp E¨º…À>S
020  73 1B 78 2A 4B E9 77 26  5E 73 BF AA 85 9C 15 6F  s x*Kéw&^s¿ª…œ o
030  54 2C 73 1B 58 8A 66 48  5B 19 84 B0 80 CA 33 73  T,s XŠfH[ „°€Ê3s
040  5C 52 0C 4C 10 9E 32 37  12 0C FB BA CB 8F 6A 53  \R L ž27  û°Ë jS
050  01 78 0C 4C 10 9E 32 37  12 0C FB BA CB 8F 6A 53   x L ž27  û°Ë jS
060  01 78 0C 4C 10 9E 32 37  12 0C FB BA CB 8F 6A 53   x L ž27  û°Ë jS
070  01 78 0C 4C 10 9E 32 37  12 0C 89 D5 A2 FC         x L ž27  ‰Õ¢ü
```

由于题目名称中提示了 XOR，并且序列 01 78 0c 4c 10 9e 32 37 12 0c fb↵
ba cb 8f 6a 53重复出现，所以考虑 XOR KEY就是↵

bash 01 78 0c 4c 10 9e 32 37 12 0c fb ba cb 8f 6a 53↵

执行异或操作后得到 bash 00000000: 68 4d 4d 4d 0c 00 47 4f 4f 44 00 4a↵
4f 42 0c 2a hMMM..GOOD.JOB.* 00000010: 54 48 45 00 46 4c 41 47 00 49↵
53 00 4e 4f 54 00 THE.FLAG.IS.NOT. 00000020: 72 63 74 66 5b 77 45 11↵
4c 7f 44 10 4e 13 7f 3c rctf[wE.L.D.N..< 00000030: 55 54 7f 57 48 14↵
54 7f 49 15 7f 0a 4b 45 59 20 UT.WH.T.I...KEY 00000040: 5d 2a 00 00↵
00 00 00 00 00 00 00 00 00 00 00 00]*.............. 00000050: 00 00↵
00 00 00 00 00 00 00 00 00 00 00 00 00 00 .... 00000060:↵
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ....↵
00000070: 00 00 00 00 00 00 00 00 00 00 72 6f 69 73 ....rois↵

注意到 hMMM的大小写是反着的，并且 rctf后面应该是"{"，所以 XOR KEY↵
的每一个字节还应该再 xor 0x20，得到↵

bash 00000000: 48 6d 6d 6d 2c 20 67 6f 6f 64 20 6a 6f 62 2c↵
0a Hmmm,goodjob,. 00000010: 74 68 65 20 66 6c 61 67 20 69 73 20 6e 6f↵
74 20 theflagisnot 00000020: 52 43 54 46 7b 57 65 31 6c 5f 64 30 6e↵
33 5f 1c RCTF{We1l_d0n3_. 00000030: 75 74 5f 77 68 34 74 5f 69 35 5f↵
2a 6b 65 79 00 ut wh4t i5 *key. 00000040: 7d 0a 20 20 20 20 20 20 20↵

20 20 20 20 20 20 20 }. 00000050: 20 20 20 20 20 20 20 20 20 20 20↵
20 20 20 20 00000060: 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20↵
00000070: 20 20 20 20 20 20 20 20 20 52 4f 49 53 ROIS↵

FLAG中最后一列的字符还有非可显示字符，说明最后一列的 KEY不正确，尝↵
试调整后得到 bash 00000000: 48 6d 6d 6d 2c 20 67 6f 6f 64 20 6a 6f 62↵
2c 20 Hmmm,goodjob, 00000010: 74 68 65 20 66 6c 61 67 20 69 73 20 6e↵
6f 74 0a theflagisnot. 00000020: 52 43 54 46 7b 57 65 31 6c 5f 64 30↵
6e 33 5f 36 RCTF{We11_d0n3_6 00000030: 75 74 5f 77 68 34 74 5f 69 35↵
5f 2a 6b 65 79 2a ut_wh4t_i5_*key* 00000040: 7d 0a 20 20 20 20 20 20↵
20 20 20 20 20 20 20 0a }. 00000050: 20 20 20 20 20 20 20 20 20 20 20↵

20 20 20 20 0a 00000060: 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20↵
0a 00000070: 20 20 20 20 20 20 20 20 20 20 52 4f 49 53 . .. ROIS↵

此时的 XOR KEY=21582c6c30be1217322cdb9aebaf4a59，根据题目名称来看这是↵
个 MD5，在 https://hashkiller.co.uk/md5-decrypter.aspx查询到对应原文↵
是“that”↵

替换 FLAG中的 key得到最终 FLAG↵

RCTF{We11_d0n3_6ut_wh4t_i5_that}↵

Easy_Crypto

附件.txt - 记事本
文件(F) 编辑(E) 格式(O) 查看(V) 帮助(H)

```
get buf unsign s[256]

get buf t[256]

we have key:hello world

we have flag:????????????????????????????????

for i:0 to 256

set s[i]:i

for i:0 to 256
   set t[i]:key[(i)mod(key.lenth)]

for i:0 to 256
   set j:(j+s[i]+t[i])mod(256)
      swap:s[i],s[j]

for m:0 to 37
   set i:(i + 1)mod(256)
   set j:(j + S[i])mod(256)
   swap:s[i],s[j]
   set x:(s[i] + (s[j]mod(256))mod(256))
   set flag[m]:flag[m]^s[x]

fprint flagx to file
```


- 中间有的地方 i、j 没有给出值，统统初始化为 0 即可，解密脚本：

```
# -*- coding: utf-8 -*-
f = open('enc. txt', 'r', encoding= ' ISO- 8859-1' )
C = f.read()
t=[]
key = 'hello world'
ch =
j=0#初始化
s = list(range(256)) #创建有序列表
for i in range(256):
j =(j+ s[i] + ord(key[i % len(key)])) % 256
s[i],s[j] = s[j],s[i]
i=0#初始化
j=0#初始化
for r in C:
i=(i+1)%256
j=(j+s[i])% 256
s[i], s[j] = s[j], s[i]
x=(s[i]+(s[j]%256))%256
ch += chr(ord(r) ^ s[x])
print(ch)
```

- EIS{55a0a84f86a6ad40006f014619577ad3}

cr2-many-time-secrets

- 题目没有任何提示，查找writeup后发现利用One Time Pad的重用导致的攻击。我首先把密文直接放到了CyberChief里看看能不能解密。用了Magic模式并不能直接得到明文。因为OTP是利用明文XOR密钥得到密文的，我又尝试了XOR bruteforce，也不
  能迅速地得到明文

能迅速地得到明文。

- 对于OTP密码的重用，我们可以利用Crib dragging attack来破解。这是一种已知部分明文的攻击，counter mode的block cipher如果重用了IV或者counter也可以用这种攻击。具体的解释如下：http://travisdazell.blogspot.com/2012/11/many-time-pad-attack-crib-drag.html,实现这种攻击的脚本：https://github.com/SpiderLabs/cribdrag,利用这个脚本来破解题目中的密文,首先把原文件中有换行的密文合并成一行，把这行密文放入脚本中：

```
52a182239373d6f740a1e3c651f207f2c212a247f3d2e65262430791c263e203d63232f0f20653f2
07f332065262c31683137223679182f2f372133202f1426652126372222220733e383f2426386b
Your message is currently:
0       ________________________________________
40      ________________________________________
80      ________________________________________
120     ________________________________________
160     ________________________________________
200     ________________________________________
240     ________________________________________
280     ____
Your key is currently:
0       ________________________________________
40      ________________________________________
80      ________________________________________
120     ________________________________________
160     ________________________________________
200     ________________________________________
240     ________________________________________
280     ____
Please enter your crib:
```

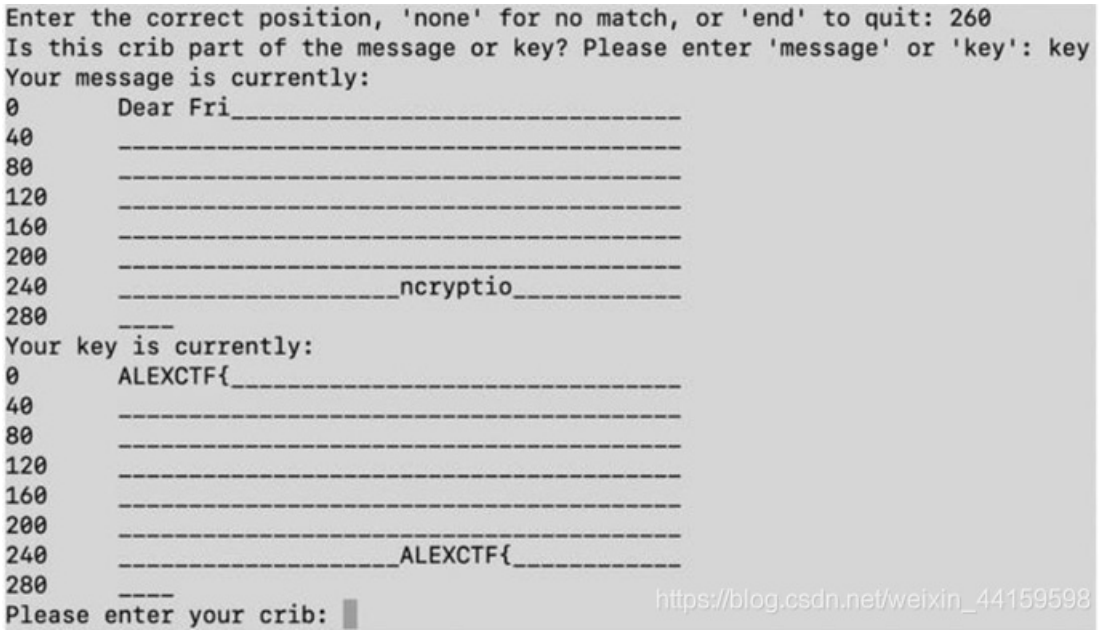

- 程序会提醒我们输入一个可能存在于明文或者密钥里的字符串，根据题目提示，flag 的开头是 ALEXCTF{，把这串字符输入：

```
Please enter your crib: ALEXCTF{
*** 0: "Dear Fri"
1: "hho;Q`TV"
2: "ef&JwFkP"
3: "k/WlQymM"
4: ""^qJnp"
5: "SxWuhb/"
```


- 可以看到 0 这个选项就是有意义的字符串。对于可能有意义的字符串，程序会在序号之前加上***。程序提示输入正确的位置，我们输入 0。程序又会提示我们输入我们的 crib 是明文中的还是密钥中的，假设 flag 是密钥，就输入 key：

```
Enter the correct position, 'none' for no match, or 'end' to quit: 0
Is this crib part of the message or key? Please enter 'message' or 'key': key
Your message is currently:
0       Dear Fri________________________________
40      ________________________________________
80      ________________________________________
120     ________________________________________
160     ________________________________________
200     ________________________________________
240     ________________________________________
280     ____
Your key is currently:
0       ALEXCTF{________________________________
40      ________________________________________
80      ________________________________________
```

- 这样程序就恢复了一部分明文。在刚才的结果中，不止 0 一个位置是有意义的， *** 260:"ncryptio"也同样有意义。再次输入 ALEXCTF{，输入 260 作为正确的位置。现在的结果如下：



- 回到明文开头，我们可以猜测这是一封信的开头，Fri 开头的单词很可能是 Friend。输入“Dear Friend,”作为 crib。得到 0: “ALEXCTF{HERE”。
- 根据 flag 的常见格式，可以猜测 HERE 之后是下划线。将“ALEXCTF{HERE_”作为 crib 输入,得到有意义的字符串有： *** 260: “ncryption sch”， *** 234: "gree with me "，*** 208: "cure,Let Me “，*** 182: “ever if the k”，*** 156: " proven to be”，*** 130: “hod that is m”，*** 104:“is the only e”，*** 78: “n scheme, I h”，*** 52: "sed One time "，*** 26: "nderstood my "
- 先看 260，可以猜测后面的单词是 scheme，输入"ncryption scheme "作为 crib：260: “ALEXCTF{HERE_GOES”,将新的 key 后面加上下划线输入：*** 260: “ncryption scheme a”， *** 234: “gree with me to us”，*** 208: “cure, Let Me know”，*** 182: “ever if the key is”，*** 156: " proven to be not "，*** 130: “hod that is mathem”，*** 104: “is the only encryp”，*** 78: "n scheme, I heard "，*** 52: “sed One time pad e”，***26: “nderstood my mista”，*** 0: "Dear Friend, This “52 的后面几乎可以确定是 encryption，而且这样填充的字母多，所以这次输入"sed One time pad encryption”:52: "ALEXCTF{HERE_GOES_THE_KEY}

OldDriver

打开附件，发现为 RSA 的模型，应用费马定理解密↵

[{"c":
736606757474117146172206513324291608049550591366325033008274746538367689397041147655074839484143741810531235397109500342432267961694037112302898218950204 2, "e": 10, "n":
251625070523397144218396888737345961777511240367238310033009597611378114907152057429417384065481502408617793017841336521659082279174154831375853889862748 03},
{"c":
219628253233004691517959202898868865627909427715468585008421798065664357671038039788851487721393054843196882493689995037844415073834760959462580113179514 61, "e": 10, "n":
239768595899044197983208120976818586523254737918912327104319972028978195806349370709006252132180953307668771902124180232973417328088394883085511264099831 93},
{"c":
556968942027406695783598339058358528657008761904811014118770058419379269523540507781154435515929038235714937410707640608615410335189789079359899768705398 3, "e": 10, "n":
185037828368585400439745580356016546109489155056452198201502510623051201487455459065675486501918320908234828526043464783353537845010767619226053618487036 23},
{"c":
450824616804451351845249388271353639063674154155180582179033897379761597127186724858437981311412547819528469269592866894655362548317963326605712296754705 2, "e": 10, "n":
233830874785455122187131579329347461107217068190774234180602200836577134285035828019098071423026473679942897750155951005411683670830975061938094513650107 23},
{"c":
229661056702912823355888430182441615527644863731179428659669040761911223374355425532767439383176867295547143154948189227538801989458972224221372684276116 72, "e": 10, "n":
317756490898614286710579090761441528707967225281125804794420733650539160125072734330284517554369870547224960577497317584759583011640827550031956320053084 93}

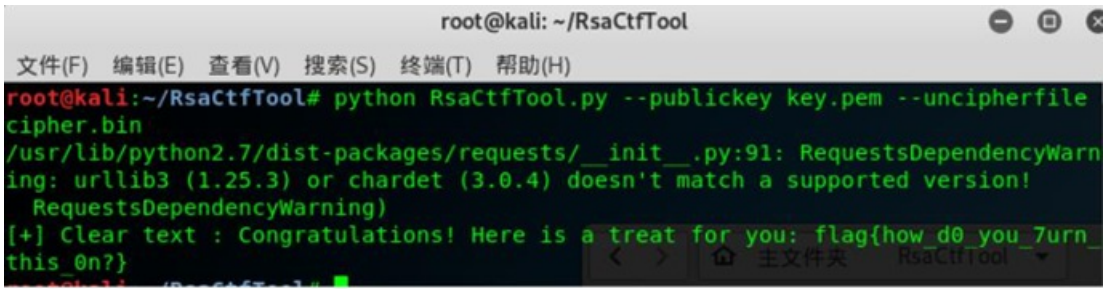
- 脚本如下：

```
import gmpy2

def broadcast(n1,n2,n3,n4,n5,n6,n7,n8,n9,n10,c1,c2,c3,c4,c5,c6,c7,c8,c9,c10):
    n = [n1,n2,n3,n4,n5,n6,n7,n8,n9,n10]
    c = [c1,c2,c3,c4,c5,c6,c7,c8,c9,c10]
    N = 1
  for i in n:
        N *= i

    Ni = []
  for i in n:
        Ni.append(N/i)

    T =[]
  for i in xrange(10):
        T.append(long(gmpy2.invert(Ni[i],n[i])))

    X =0
  for i in xrange(10):
        X += c[i] * Ni[i] * T[i]

    m3 = X%N
    m = gmpy2.iroot(m3,10)
  print m


c1=736606757474117146172206513324291608049550591366325033008274746538367689397041147655074839484143741810531235397109500342432267961694837112302898218950204 2
n1=251625070523397144218396808737345961777511240367238310033009597611378114907152057429417384065481502408617793017841336521659082279174154831375853889862748 03
c2=219628253233004691517959202898688865627990427715468585008421798065664357671038803978885148772139365484319688249368999503784441507383476089594625801131795 1461
n2=239768595899044197983208120976818506523254737918912327104319972020978619580863493707090062521321809533076688771902124180232973417320888394883885511126409983 193
c3=656968942027406695785359033905635052865708076190481101411877005841937926952354058778115443551692903823571493741870764606861541033518979907935989976870539 83
n3=185037828368585480439745580356816546109489155056452198201502510623851201487554590656754865019183209082348285260434647833535378450107676192266536164487036 23
c4=450824616804451351845249388271353639063674154155180582179033897379761597127186724858437981311412547819520469269592866894655362548317963326605712296754705 2
n4=233830074785455122187131579329347461107217068190774234188002208836577134285835820019098071428026473679942897758155951005411683670830975061938694513650107 23
c5=229661656702912823355880443818244161552764486373117942865966904076191122337455542553276743938817686729554714315494881922753880198945897222422137268427611672
n5=317756498898614286710579090761441528707967225281125804794420733650539160125072733433828451755436987054722496057749731758475958301164082755083195632005388493
c6=179633313063405045742968136916219038352135561785389534381262979264585397089844470879023686505405355160998533122970239261072026689217153126649390825646712087
n6=222463420229434323820696190444155666528992837865384117263228322788817449540224863306101061557264212658459116375033891921394564607483382390552164302587959349
c7=165241753470902945038057065397370532090811767959756387302260314088050748256848294831013154094822779704550539033314619158674926924954816824731640407401 4639
n7=253954611426706312681561061360283257443933584366175286779672493473535249246550011518495440222017725000332808223726613443526074343786960851779095736547813043
c8=155857717344483518394566313944040497759568679429510619219766191978000767536174185929049073245111264877664812677975936042820519466472151649702629098178623935 2
n8=326565088927441849012894132877280398913838323115486068141088227787632675367415412477513277692048193537018475675678510754078163257629533046873826817316780904 7
c9=896512342163769465004421684523379163347478029124815032832813225058732558524239660648746204684140746788823681886010577342254841014594570067467985642359797
n9=528497662695418274728189428822064857416253995958539599226164980998743574226302855105006426889033339287717357281169159984125315046021998681796446197073655 3
c10=135609457565430230085293981084469488471378530384376952445730358885312885777078029065666632006939789839484848484703032101891563838183935558095834271998987247
n10=304159848003075789329463999875590889683556383543448233593972044191912418027217724994866156616990809985024399015855739586890479185379066878407250054962386 21
broadcast(n1,n2,n3,n4,n5,n6,n7,n8,n9,n10,c1,c2,c3,c4,c5,c6,c7,c8,c9,c10)
```


- 改为 16 进制后转字符串即为 flag

wtc_rsa_bbq

- 附件 winhex 打开后为压缩包开头，修改后缀，有两个文件：cirpher.bin 和 key,pem，直接使用 RsaCtftool 工具得到 flag



cr4-poor-rsa

- 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
```


```
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 = 96544530432699819479828222884248473243845717059599952342690 1
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/qI
WDjSi+w=")


print key.decrypt(secret)
```

- 结果为：ALEXCTF{SMALL_PRIMES_ARE_BAD}

flag_in_your_hand1

所给页面提交数据会有反馈↵

此时在所给 js 文件中找到提交数据所经过的函数

```
}
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;
}
```

说明字符串应为 a 中数字减 3,python 脚本为:

```
File  Edit  Format  Run  Options  Window  Help
a =[115, 101, 99, 117, 114, 105, 116, 121, 45, 120, 98, 117]
s = ""
for i in a:
    s += chr(i)

print(s)
```

```
================ RESTART: C:\Users\Administrator\Desktop\1.py ================
security-xbu
```

输入得到 flag

safer-than-rot13

cry100

```
XMVZGC RGC AMG RVMG HGFGMQYCD VT VWM BYNO, NSVWDS NSGO RAO XG UWFN AF
HACDGMVWF. AIRVFN AII AMG JVRRVC-XVMC, FYRBIG TVIZ ESV SAH CGQGM XGGC
RVMG NSAC A RYIG TMVR NSG SVWFG ESGMG NSGO EGMG XVMC WCNYI NSG HAO
FVRG IVMH JARG MVWCH NV NAZG NSGR VTT NV EAM. OVWM TIAD YF "CV NSYF
YF CVN JMOBNV RO HGAM", YC IVEGMJAFG, EYNS WCHGMFJVMGF YCFNGAH VT
FBAJGF, FWMMVWCHGH XO NSG WFWAI "TIAD" NAD ACH JWMIO XMAJGF. GCUVO.
```

文字大批量，所以进行词频分析

```
0   -1.258  BROKEN MEN ARE MORE DESERVING OF OUR PITY, THOUGH THEY MAY BE JUST AS DANGEROUS. ALMOST ALL ARE COMMON-BORN, SIMPLE FOLK WHO HAD
            NEVER BEEN MORE THAN A MILE FROM THE HOUSE WHERE THEY WERE BORN UNTIL THE DAY SOME LORD CAME ROUND TO TAKE THEM OFF TO WAR. YOUR
            FLAG IS "NO THIS IS NOT CRYPTO MY DEAR", IN LOWERCASE, WITH UNDERSCORES INSTEAD OF SPACES, SURROUNDED BY THE USUAL "FLAG" TAG AND
            CURLY BRACES. ENJOY.

1   -1.542  BROKEN MEN ARE MORE DESERVING OF OUR PITY, THOUGH THEY MAY BE JUST AS DANGEROUS. ALMOST ALL ARE COMMON- BORN, SIMPLE FOLK WHO HAD
            NEVER BEEN MORE THAN A MILE FROM THE HOUSE WHERE THEY WERE BORN UNTIL THE DAY SOME LORD CAME ROUND TO TAKE THEM OFF TO WAR. YOUR
            FLAG IS" NO THIS IS NOT CRYPTOMY DEAR", IN LOWER CASE, WITH UNDERSCORES INSTEAD OF SPACES, SURROUNDED BY THE USUAL" FLAG" TAG AND
            CURLY BRACES. ENJOY.

2   -3.128  CRAVEL DEL URE DARE FENERGOLK AS AIR ZOTY, THAIKH THEY DUY CE PINT UN FULKERAIN. UMDANT UMM URE BADDAL-CARL, NODZME SAMV WHA HUF
            LEGER CEEL DARE THUL U DOME SRAD THE HAINE WHERE THEY WERE CARL ILTOM THE FUY NADE MARF BUDE RAILF TA TUVE THED ASS TA WUR. YAIR
            SMUK ON "LA THON ON LAT BRYZTA DY FEUR", OL MAWERBUNE, WOTH ILFERNBAREN OLNTEUF AS NZUBEN, NIRRAILFEF CY THE INIUM "SMUK" TUK ULF
            BIRMY CRUBEN. ELPAY.

3   -3.312  BLAXEN MEN OLE MALE DEKELVING AT AUL WIPS, PRAUGR PRES MOS BE YUKP OK DONGELAUK. OZMAKP OZZ OLE HAMMAN-BALN, KIMWZE TAZX CRA ROD
            NEVEL BEEN MALE PRON O MIZE TLAM PRE RAUKE CRELE PRES CELE BALN UNPIZ PRE DOS KAME ZALD HOME LAUND PA POXE PREM ATT PA COL. SAUL
            TZOG IK "NA PRIK IK NAP HLSWPA MS DEOL", IN ZACELHOKE, CIPR UNDELKHALEK INKPEOD AT KWOHEK, KULLAUNDED BS PRE UKUOZ "TZOG" POG OND
            HULZS BLOHEK. ENYAS.

4   -3.320  HRAVED MED ORE MARE BENERGUDP AS AIR ZUCY, CLAIPL CLEY MOY HE WINC ON BODPERAIN. OTMANC OTT ORE KAMMAD-HARD, NUMZTE SATV FLA LOB
            DEGER HEED MARE CLOD O MUTE SRAM CLE LAINE FLERE CLEY FERE HARD IDCUT CLE BOY NAME TARB KOME RAIDB CA COVE CLEM ASS CA FOR. YAIR
            STOP UN "DA CLUN UN DAC KRYZCA MY BEOR", UD TAFERKONE, FUCL IDBERNKAREN UDNCEOB AS NZOKEN, NIRRAIDBEB HY CLE INIOT "STOP" COP ODB
            KIRTY HROKEN. EDWAY.

5   -3.355  GNEJAS BAS INA BENA MARANZOSC EV EUN FOTY, THEUCH THAY BIY GA PURT IR MISCANEUR. ILBERT ILL INA DEBBES-GENS, ROBFLA VELJ KHE HIM
            SAZAN GAAS BENA THIS I BOLA VNEB THA HEURA KHANA THAY KANA GENS USTOL THA MIY REBA LENM DIBA NEUSM TE TIJA THAB EVV TE KIN. YEUN
            VLIC OR "SE THOR OR SET DNYFTE BY MAIN", OS LEKANDIRA, KOTH USMANRDENAR OSRTAIM EV RFIDAR, RUNNEUSMAM GY THA URUIL "VLIC" TIC ISM
            DUNLY GNIDAR. ASPEY.

6   -3.395  BNOMAD KAD INA KONA SATANGEDP OF OLN UECY, CHOLPH CHAY KIY BA VLTC IT SIDPANOLT. IRKOTC IRR INA JOKKOD-BOND, TEKURA FORM WHO HIS
            DAGAN BAAD KONA CHID I KERA FNOK CHA HOLTA WHANA CHAY WANA BOND LDCER CHA SIY TOKA RONS JIKA NOLDS CO CIMA CHAK OFF CO WIN. YOLN
            FRIP ET "DO CHET ET DOC JNYUCO KY SAIN", ED ROWANJITA, WECH LDSANTJONAT EDICAIS OF IUIJAT, TLNNOLDSAS BY CHA LTLIR "FRIF" CIP IDS
            JLNRY BNIJAT. ADVOY.
```

- 提交flag,将空格替换为下划线“_”

Decode_The_File


```
IyMjIyMjIyMjIyMjIyMjIyMjIyMjIyMjIyMjIyMjIyMjIyMjIyMjIyMjIyMjIyMjIyMjIyMjIyMjIyMjIyMjIyMjIyMjIyMjIyMjIyN=
IyAJCQkJRG9jdW1lbnRhdGlvbgkJCQkgICAgI0==
IyMjIyMjIyMjIyMjIyMjIyMjIyMjIyMjIyMjIyMjIyMjIyMjIyMjIyMjIyMjIyMjIyMjIyMjIyMjIyMjIyMjIyMjIyMjIyMjIyMjIyO=
IyBBdXRob3I6ICAgVG9kZCBXaGl0ZW1hbk==
IyBEYXRlOiAgICAgN3RoIE1heSwgMjAwM/==
IyBWZXJpb246ICAgMS4x
IyBIb21lcGFnZTogaHR0cDovL2hvbWUucGFjaWZpYy5uZXQuYXUvfnR3aGl0ZW1hL2Rlcy5odG1s
I0==
IyBNb2RpZmljYXRpb25zIHRvIDNkZXMgQ0JDIGNvZGUgYnkgTWF0dCBKb2huc3RvbiAyMDA0IDxtYXR0IGF0IHVjYyBhc24gYXU+
I5==
IyBUaGlzIGFsZ29yaXRobSBpcyBhIHB1cmUgcHl0aG9uIGltcGxlbWVudGF0aW9uIG9mIHRoZSBERVMgYWxnb3JpdGhtLl==
IyBJdCBpcyBpbiBwdXJlIHB5dGhvbiB0byBhdm9pZCBwb3J0YWJpbGl0eSBpc3N1ZXMsIHNpbmNlIG1vc3QgREVTID==
IyBpbXBsZW1lbnRhdGlvbnMgYXJlIHByb2dyYW1tZWQgaW4gQyAoZm9yIHBlcmZvcm1hbmNlIHJlYXNvbnMpLn==
I7==
IyBUcmlwbGUgREVTIGNsYXNzIGlzIGFsc28gaW1wbGVtZW50ZWQsIHV0aWxpc2luZyB0aGUgREVTIGJhc2UuIFRyaXBsZSBERVN=
IyBpcyBlaXRoZXIgREVTLUVERTMgd2l0aCBhIDI0IGJ5dGUga2V5LCBvciBERVMtRURFMiB3aXRoIGEgMTYgYnl0ZSBrZXku
I4==
IyBTZWUgdGhlIFJFQURNRS50eHQgdGhhdCBzaG91bGQgY29tZSB3aXRoIHRoaXMgcHl0aG9uIG1vZHVsZSBmb3IgdGhl
IyBpbXBsZW1lbnRhdGlvbiBtZXRob2RzIHVzZWQu
IiIiQSBwdXJlIHB5dGhvbiBpbXBsZW1lbnRhdGlvbiBvZiB0aGUgREVTIGFuZCBUUklQTEUgREVTIGVuY3J5cHRpb24gYWxnb3JpdGhtcy5==
```


- 显然，这里的数据是通过 Base64 算法编码的，让我们对其进行解码：

```
#############################################################################
#                       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 <matt at ucc asn au>
#
# 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.
```


- https://github.com/n0fate/chainbreaker/blob/master/pyDes.py,将解码后的文件与从上述链接下载的文件进行了比较，我们找不到添加到解码后文件的任何其他信息。因此，必须在 Base64字符串中隐藏一些秘密，但是它们是什么？让我们使用标准的 Base64算法对内容进行重新编码，以查看是否存在任何差异：

- 如果你熟悉 Base64，我认为你可以轻松找出原因。众所周知，对于 Base64算法，原始数据将被分成 3个字节的组，如果最后一组仅包含 1个或 2个字节，它将在末尾添加一些填充并使用 1个或 2个“ =”表示最后一组中有多少个原始字节。这是最后一组中 1个字节的示例：

Text content	M	null	null
ASCII	77 (0x4d)	0 (0x00)	0 (0x00)
Bit pattern	0 1 0 0 1 1 0 1	0 0 0 0 - - - -	- - - - - - - -
Index	19	22	null	null
Base64-encoded	T	W	=	=

4 paddings

- 实际上，此处的 4个填充将被解码例程忽略，也就是说，我们可以在此处放置任何位，这是隐藏信息的好地方！理解这一点，解决这个挑战将没有困难，以下脚本是我用来提取隐藏信息的工具：

```
import base64
import string

def tobin(data):
    b64table = string.ascii_uppercase + string.ascii_lowercase + string.digits +
'+/'
    index = b64table.find(data)
    return format(index, '06b')

def toStr(bin):
    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()
    if line.strip()[-2:] == '==':
        binstr = tobin(line[-3:-2])
        out += binstr[-4:]
        print binstr[-4:]
    elif line.strip()[-1:] == '=':
        binstr = tobin(line[-2:-1])
        out += binstr[-2:]
        print binstr[-2:]

print out
print toStr(out)
```


- Flag: ROIS{base_GA_caN_b3_d1ffeR3nT}