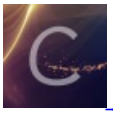
ISCC 2018 Reverse WriteUp

原创

lacoucou 于 2018-05-22 14:26:59 发布 4242 收藏
分类专栏： ctf 文章标签： ISCC 2018 WriteUp Reverse



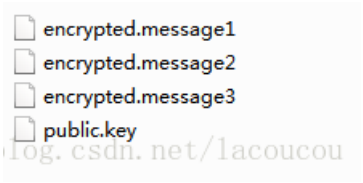
1.RSA256 [分值:100]

问题描述：

RSA

题目已说明是RSA算法。

附件中共有四个文件。



3个加密后的信息，一个公钥,公钥内容如下：

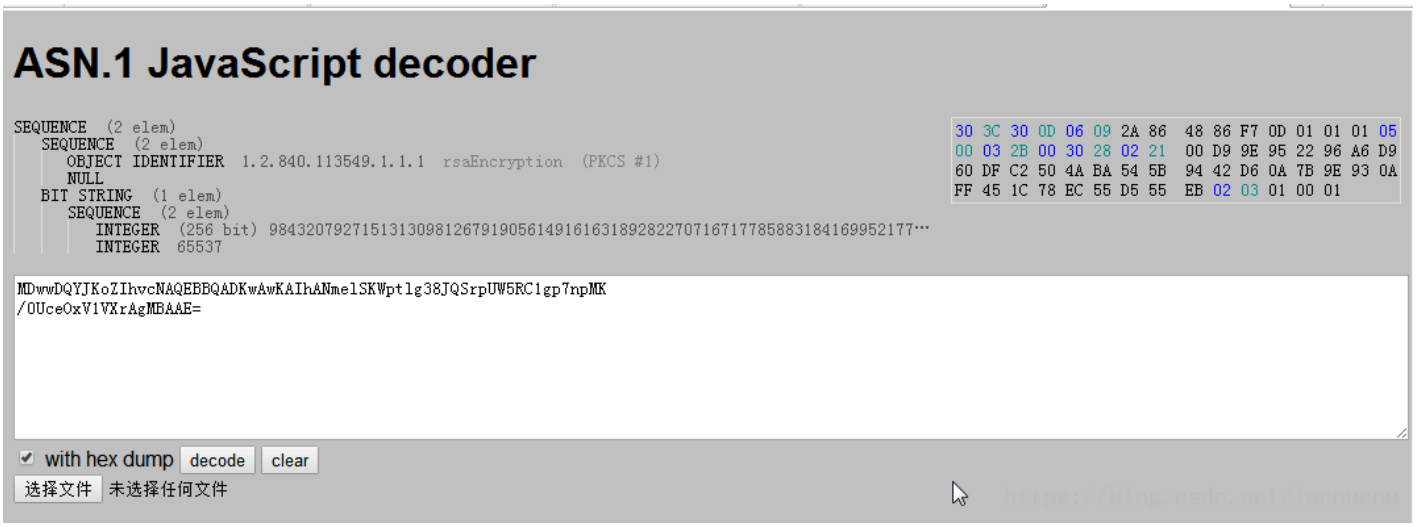
```
-----BEGIN PUBLIC KEY-----
MDwwDQYJKoZIhvcNAQEBBQADKwAwKAIhANmelSKWptlg38JQSrpUW5RC1gp7npMK
/0UceOxV1VXrAgMBAAE=
-----END PUBLIC KEY-----
```

结果搜索，确定公钥应该是PKCS#8格式。

```
https://blog.csdn.net/yue7603835/article/details/72575262
Public Key file (PKCS#8)
Because RSA is not used exclusively inside X509 and SSL/TLS, a more generic key format is available in the

It starts and ends with the tags:

-----BEGIN PUBLIC KEY-----
BASE64 ENCODED DATA
-----END PUBLIC KEY-----
Within the base64 encoded data the following DER structure is present:

PublicKeyInfo ::= SEQUENCE {
  algorithm       AlgorithmIdentifier,
  PublicKey       BIT STRING
}

AlgorithmIdentifier ::= SEQUENCE {
  algorithm       OBJECT IDENTIFIER,
  parameters      ANY DEFINED BY algorithm OPTIONAL
}
So for an RSA public key, the OID is 1.2.840.113549.1.1.1 and there is a RSAPublicKey as the PublicKey key
```

```
RSAPublicKey ::= SEQUENCE {
    modulus           INTEGER,  -- n
    publicExponent    INTEGER   -- e
}
```

key的格式是ASN.1 在线解析网站http://lapo.it/asn1js/



获取到数据如下：(10进制)

N=984320792715131309812679190561491616318928227071671778588318416995217743108 91

E=65537

要想解密数据需要私钥D,要计算私钥D，则需要分解N，得到素数P，Q。

很幸运的是这个N已经被分解过，在网站之耳机可以查询：

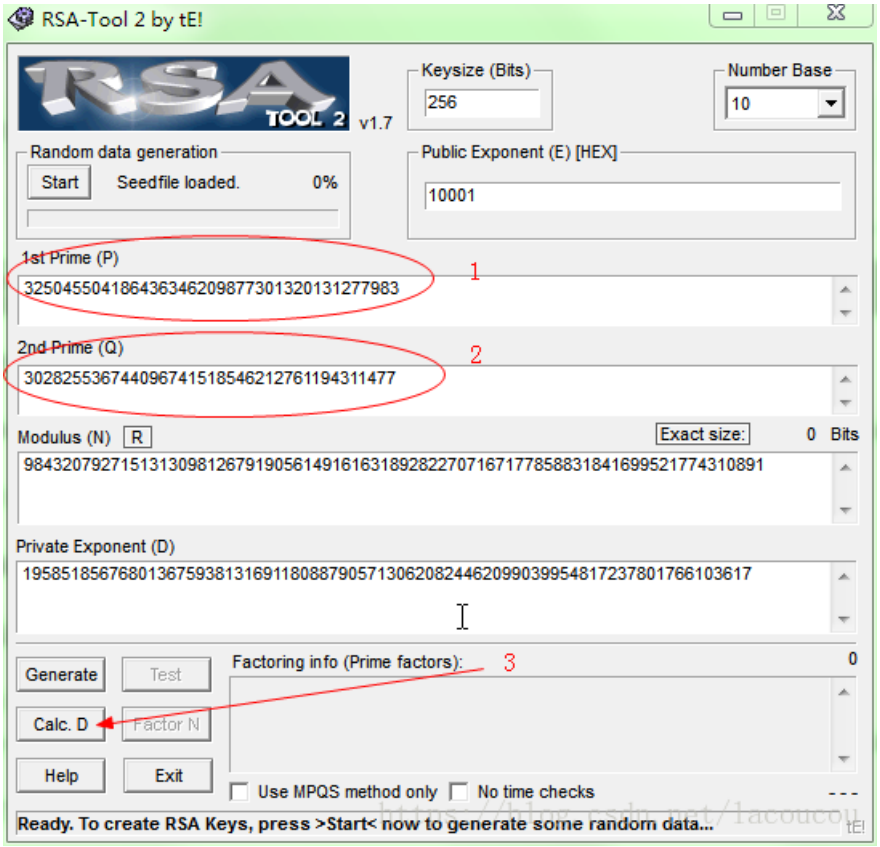
http://factordb.com/index.php?query=9843207927151313098126791905614916163189282270716717785883184169952177431089 1

分解之后：

P=3028255367440967415185462127611943114 77

Q=3250455041864363462098773013201312779 83

接着用工具计算出D，输入P，Q点击Calc.D 立即就能出结果。



当然也可以用此工具分解N，但是速度有点慢。

知道了以上信息就可以解密了。

网上找了一段python代码：

https://my.oschina.net/KFeC5dSwgN1q/blog/1623311

```
def __multi(array, bin_array):
    result = 1
    for index in range(len(array)):
        a = array[index]
        if not int(bin_array[index]):
            continue
        result *= a
    return result

def exp_mode(base, exponent, n):
    bin_array = bin(exponent)[2:][::-1]
    r = len(bin_array)
    base_array = []

    pre_base = base
    base_array.append(pre_base)

    for _ in range(r - 1):
        next_base = (pre_base * pre_base) % n
        base_array.append(next_base)
        pre_base = next_base

    a_w_b = __multi(base_array, bin_array)
    return a_w_b % n

# 加密 m是被加密的信息 加密成为c
def encrypt(m, n,e):
    c = exp_mode(m, e, n)
    return c

# 解密 c是密文，解密为明文m
def decrypt(c, n,d):
    m = exp_mode(c, d, n)
    return m
```

解密：

```
P=0xE3D213B0A3C9551F9FB1EB8D7C3DAF35
Q=0xF4897CAABA80236BDC1B59385C4BF49F
N=0xD99E952296A6D960DFC2504ABA545B9442D60A7B9E930AFF451C78EC55D555EB
D=0x04547B732CBC3527104CB57C4728D6899B44C4994FAE2713D6B594BC0F522A41
E=0x10001
```

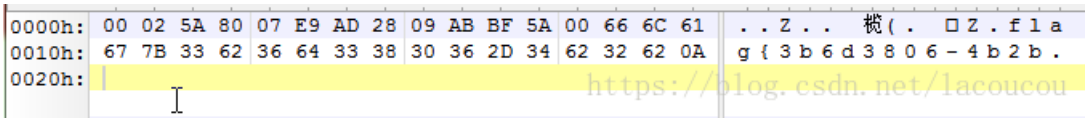
```
//x1 x2 x3 分别对应encrypted.message1，2,3  大端序
x1=0x0B39CC1B6127D3BBED2BC045148C911D467985A94B147EDE80750F95A360D47A
x2=0x9AFB1CDC1986D3BB53A3425B396C83618EFAAA81C14C965C813415E5C54FCE4B
x3=0x0F04B3B67EF230F80BB518D26DED38AF84B6C8D87BA80C09EBF1D865123082FA

//x1 x2 x3 分别对应encrypted.message1，2,3  小端序
x11=0x7ad460a3950f7580de7e144ba98579461d918c1445c02bedbbd327611bcc390b
x12=0x4BCE4FC5E51534815C964CC181AAFA8E61836C395B42A353BBD38619DC1CFB9A
x13=0xfa82301265d8f1eb090ca87bd8c8b684af38ed6dd218b50bf830f27eb6b3040f

num1=decrypt(x1,N,D)
num2=decrypt(x2,N,D)
num3=decrypt(x3,N,D)

print hex(num1)
print hex(num2)
print hex(num3)
```

这里num1,2,3就是解出来之后的明文，这里有个坑，密文32字节，解出来也应该是32字节，但是数字的话，最高位为0可以省略。

hex(num1) 值为0x25a8007e9ad2809abbf5a00666c61677b33623664333830362d346232620aL，补上几个0

00025a8007e9ad2809abbf5a00666c61677b33623664333830362d346232620a

16进制数据转字符。



如此，三个解完整就可以得到key.

2.My math is bad [分值:150]

问题描述：

```
I think the math problem is too difficult for me.
flag：flag{XXXX}
```

程序是一个ELF程序，x64的，用IDA打开：

判断逻辑很明显：

```
_int64 __fastcall main(__int64 a1, char **a2, char **a3)
{
  puts("======================================");
  puts("= Welcome to the flag access machine! =");
  puts("=   Input the password to login ...   =");
  puts("======================================");
  __isoc99_scanf("%s", s);
  if ( (unsigned int)sub_400766() )
  {
    puts("Congratulations! You should get the flag...");
    sub_400B16();
  }
  else
  {
    puts("Wrong password!");
  }
  return 0LL;
}
```

主要验证函数：

```
sub_400766
```

```
signed __int64 sub_400766()
{
  signed __int64 result; // rax
  __int64 v1_22; // ST40_8
  __int64 v2_39; // ST48_8
  __int64 x1; // [rsp+20h] [rbp-60h]
  __int64 x2; // [rsp+28h] [rbp-58h]
  __int64 x3; // [rsp+30h] [rbp-50h]
  __int64 x4; // [rsp+38h] [rbp-48h]
  __int64 v7_45; // [rsp+50h] [rbp-30h]
  __int64 v8_45; // [rsp+58h] [rbp-28h]
  __int64 v9_35; // [rsp+60h] [rbp-20h]
  __int64 v10_41; // [rsp+68h] [rbp-18h]
  __int64 v11_13; // [rsp+70h] [rbp-10h]
  __int64 v12_36; // [rsp+78h] [rbp-8h]

  if ( strlen(s) != 32 )
    // A1 A2 A3 A4 A5 A6 A7 A8
    return 0LL;
  x1 = *(signed int *)&s[16];
  x2 = *(signed int *)&s[20];
  x3 = *(signed int *)&s[24];
  x4 = *(signed int *)&s[28];
  if ( *(signed int *)&s[4] * (signed __int64)*(signed int *)s
     - *(signed int *)&s[12] * (signed __int64)*(signed int *)&s[8] != 0x24CDF2E7C953DA56LL )
    goto LABEL_15;
  if ( 3LL * *(signed int *)&s[8] + 4LL * *(signed int *)&s[12] - *(signed int *)&s[4] - 2LL * *(signed int
    goto LABEL_15;
  if ( 3 * *(signed int *)s * (signed __int64)*(signed int *)&s[12]
     - *(signed int *)&s[8] * (signed __int64)*(signed int *)&s[4] != 0x2E6E497E6415CF3ELL )
    goto LABEL_15;
  if ( 27LL * *(signed int *)&s[4] + *(signed int *)s - 11LL * *(signed int *)&s[12] - *(signed int *)&s[8]
```

```
    goto LABEL_15;
  // s=A1
  // s[4]=A2
  // S[8]=A3
  // S[12]=A4

  // A1*A2-A4*A3=0x24CDF2E7C953DA56LL
  // 3*A3+4*A4-A2-2*A1=0x
  // 3*A1*A4-A3*A2=

  // 27*A2+A1-11*A4-a3=
  srand(*(_DWORD *)&s[8] ^ *(_DWORD *)&s[4] ^ *(_DWORD *)s ^ *(_DWORD *)&s[12]);
  v1_22 = rand() % 50;
  v2_39 = rand() % 50;
  v7_45 = rand() % 50;
  v8_45 = rand() % 50;
  v9_35 = rand() % 50;
  v10_41 = rand() % 50;
  v11_13 = rand() % 50;
  // 0x16 0x27 0x2d 0x2d 0x23 0x29 0xd 0x24
  // 22 39 45 45 35 41 13 36
  v12_36 = rand() % 50;
  if ( x4 * v2_39 + x1 * v1_22 - x2 - x3 != 61799700179LL
    || x4 + x1 + x3 * v8_45 - x2 * v7_45 != 48753725643LL
    || x1 * v9_35 + x2 * v10_41 - x3 - x4 != 59322698861LL
    || x3 * v12_36 + x1 - x2 - x4 * v11_13 != 51664230587LL )
  {
LABEL_15:
    result = 0LL;
  }
  else
  {
    result = 1LL;
  }
  return result;
}
```

简单的说就是输入一个32个字符组成的字符串，会分割成8个int型的值，分别记为x1-x8

首先前4个值x1-x4满足方程：

x1*x2-x3*x4-2652042832920173142=0

3*x3+4*x4-x2-2*x1-397958918=0

3*x1*x4-x3*x2-3345692380376715070=0

27*x2+x1-11*x4-x3-40179413815=0

解方程使用scypi:

```
from scipy.optimize import fsolve
import struct

def func(x):
    x1,x2,x3,x4 = x.tolist()
    return [(x2*x1)-(x4*x3)-0x24CDF2E7C953DA56,
            3*x3+4*x4-x2-2*x1-0x17B85F06,
            (3*x1*x4)-(x3*x2)-0x2E6E497E6415CF3E,
            27*x2+x1-11*x4-x3-0x95AE13337]
initial_x = [0x88880000,0x8888ffff,0x8888ffff,0x8888ffff]
result = fsolve(func, initial_x)

strxx=""
print result

x1,x2,x3,x4 = result.tolist()
print x1*x2-x3*x4-2652042832920173142
print 3*x3+4*x4-x2-2*x1-397958918
print 3*x1*x4-x3*x2-3345692380376715070
print 27*x2+x1-11*x4-x3-40179413815
for x in result:
    print hex(x),x,int(x)
    strxx+=struct.pack("I",int(x)&0xffffffff)

print strxx
```

上边的代码有一些问题，似乎是受到初始值的影响，计算出来的值的最后一个字节是错的，错的，....

找到bug了，python float转int 有问题。

比如：811816014.0 转int 会变成 811816013

后边四个值要满足方程：

```
v1=0x0000000007779C28%50
v2=0x000000002E27FDB5%50
v7=0x00000000528857D7%50
v8=0x000000000941FAD3%50
v9=0x00000000129422D7%50
v10=0x00000000724E98F7%50
v11=0x00000000263FD923%50
v12=0x0000000052AB8CB2%50
print hex(v1),hex(v2),hex(v7),hex(v8),hex(v9),hex(v10),hex(v11),hex(v12)
print (v1),(v2),(v7),(v8),(v9),(v10),(v11),(v12)
def func(x):
    x1,x2,x3,x4 = x.tolist()
    return [x4 * v2 + x1 * v1 - x2 - x3-0xE638C96D3,
            x4 + x1 + x3 * v8 - x2 * v7 -0xB59F2D0CB,
            x1 * v9 + x2 * v10 - x3 - x4- 0xDCFE88C6D,
            x3 * v12 + x1 - x2 - x4 * v11-0xC076D98BB ]
initial_x = [0xffffffff,0xffffffff,0xffffffff,0xffffffff]
result = fsolve(func, initial_x)
strxx=""
for x in result:
    print hex(x),x,int(x)
    strxx+=struct.pack("I",int(x)&0xffffffff)
print strxx
```

这个也有点问题，第一个值差了1，。。。。。。。。

不知道为啥我的科学计算全是错的（科学计算没有错，是python float转int 与期望不太一样）。

这个网站所出来是对的。

http://www.yunsuanzi.com/cgi-bin/linear_system.py

算出来正确字符串输入即可得到答案。

贴一个完整脚本：

```
from scipy.optimize import fsolve
import struct

def float2Int(fx):
    text=str(fx)
    n=text.find(".")
    return int(text[0:n])

def func(x):
    x1,x2,x3,x4 = x.tolist()
    return [(x2*x1)-(x4*x3)-0x24CDF2E7C953DA56,
            3*x3+4*x4-x2-2*x1-0x17B85F06,
            (3*x1*x4)-(x3*x2)-0x2E6E497E6415CF3E,
            27*x2+x1-11*x4-x3-0x95AE13337]
initial_x = [0xffffffff,0x8888ffff,0x8888ffff,0x8888ffff]
result = fsolve(func, initial_x)
strxx=""
for x in result:
    #print hex(x),x,int(x)
    strxx+=struct.pack("I",float2Int(x))
print "first: ",strxx

v1=0x0000000007779C28%50
v2=0x000000002E27FDB5%50
v7=0x00000000528857D7%50
v8=0x000000000941FAD3%50
v9=0x00000000129422D7%50
v10=0x00000000724E98F7%50
v11=0x00000000263FD923%50
v12=0x0000000052AB8CB2%50
def func(x):
    x1,x2,x3,x4 = x.tolist()
    return [x4 * v2 + x1 * v1 - x2 - x3-0xE638C96D3,
            x4 + x1 + x3 * v8 - x2 * v7 -0xB59F2D0CB,
            x1 * v9 + x2 * v10 - x3 - x4- 0xDCFE88C6D,
            x3 * v12 + x1 - x2 - x4 * v11-0xC076D98BB ]
initial_x = [0xffffffff,0xffffffff,0xffffffff,0xffffffff]
result = fsolve(func, initial_x)
strxx=""
for x in result:
    strxx+=struct.pack("I",float2Int(x))
print "second: ",strxx
```

3.obfuscation and encode [分值:250]

题目描述：

You know what to do.

附件是一个x64的elf。

主函数：

```
int __cdecl main(int argc, const char **argv, const char **envp)
{
  signed __int64 v3; // rax
  char s1; // [rsp+60h] [rbp-270h]
  char v6; // [rsp+160h] [rbp-170h]
  char s; // [rsp+260h] [rbp-70h]
  int v8; // [rsp+2CCh] [rbp-4h]

  v8 = 0;
  memset(&s, 0, 0x64uLL);
  printf("flag:", 0LL);
  __isoc99_scanf("%s", &s);
  memset(&v6, 0, 0xFFuLL);
  memset(&s1, 0, 0xFFuLL);
  if ( (unsigned int)fencode(&s, (__int64)&v6) )
  {
    v3 = strlen(&v6);
    encode((__int64)&v6, v3, (__int64)&s1);
    if ( !strcmp(&s1, (const char *)(unsigned int)"lUFBuT7hADvItXEGn7KgTEjqw8U5VQUq") )
      printf("correct");
    else
      printf("error");
  }
  else
  {
    printf("error", &v6);
    v8 = 0;
  }
  return v8;
}
```

主要就是两个函数：

1.fencode --这个是一个矩阵的计算 (4元一次方程组)

代码有乱序膨胀主要代码也不多：

```
.text:000000000040065A                mov     rdi, [rbp+s]    ; s
.text:000000000040065E                call    _strlen
.text:00000000004007E3                mov     rdx, [rbp+var_8] ; len
.text:00000000004007E7                cmp     rdx, 18h        ; 注册码长度0x18

//矩阵乘法
.text:0000000000400894                mov     rax, offset m
.text:000000000040089E                movsxd  rcx, [rbp+var_2C]
.text:00000000004008A2                shl     rcx, 4
.text:00000000004008A6                add     rax, rcx
.text:00000000004008A9                movsxd  rcx, [rbp+var_34_i]
.text:00000000004008AD                mov     edx, [rax+rcx*4]
.text:00000000004008B0                mov     rax, [rbp+s]
.text:00000000004008B4                mov     esi, [rbp+var_34_i]
.text:00000000004008B7                mov     edi, [rbp+var_28]
.text:00000000004008BA                shl     edi, 2
.text:00000000004008BD                add     esi, edi
.text:00000000004008BF                movsxd  rcx, esi
.text:00000000004008C2                movsx   esi, byte ptr [rax+rcx]
.text:00000000004008C6                imul    edx, esi
.text:00000000004008C9                add     edx, [rbp+var_30_sum]
.text:00000000004008CC                mov     [rbp+var_30_sum], edx

//计算结果
.text:00000000004008F0                mov     eax, 7Fh
.text:00000000004008F5                mov     ecx, [rbp+var_30_sum]
.text:00000000004008F8                mov     dl, cl
.text:00000000004008FA                movsx   ecx, dl
.text:00000000004008FD                mov     [rbp+var_80], eax
.text:0000000000400900                mov     eax, ecx
.text:0000000000400902                cdq
.text:0000000000400903                mov     ecx, [rbp+var_80]
.text:0000000000400906                idiv    ecx
.text:0000000000400908                mov     sil, dl
.text:000000000040090B                mov     rdi, [rbp+var_20]
.text:000000000040090F                mov     edx, [rbp+var_24_i]
.text:0000000000400912                mov     r8d, edx
.text:0000000000400915                add     r8d, 1
.text:0000000000400919                mov     [rbp+var_24_i], r8d
.text:000000000040091D                movsxd  r9, edx
.text:0000000000400920                mov     [rdi+r9], sil
```

函数的主要功能类似以下C代码：

```
int m[][4]={{2,2,4,-5},{1,1,3,-3},{-1,-2,-3,4},{-1,0,-2,2}};
char * pInput="flag{11111111111111111}"; //自己输入的
unsigned char szOut[24]={0};              //这里边的内容会进入下边的运算
for(int k=0;k<6;k++)
{
    for(int i=0;i<4;i++)
    {
        char nSum=0;
        for(int j=0;j<4;j++)
        {
            nSum+=*(pInput+k*4+j)*m[i][j];
        }
        szOut[k*4+i]=(nSum%0x7f);
    }
}
QString xxOut;
for(int i=0;i<24;i++)
{
    xxOut+=QString("%1 ").arg(szOut[i],2,16,QLatin1Char( '0' ));
}
qDebug()<<xxOut;
```

2.encode 是一个变形的base64_encode 算法，初始的字符表被改。

修改后的字符表：FeVYKw6a0lDlOsnZQ5EAf2MvjS1GUiLWPTtH4JqRgu3dbC8hrcNo9/mxzpXBky7+

所以要还原输入的flag

1.将字符串lUFBuT7hADvltXEGn7KgTEjqw8U5VQUq 进行base64解密

解密后内容为：

```
"25 c0 3b a6 1f af 4c a5 cb 8b a4 9b 3b e1 28 85 26 26 16 e7 11 09 07 26 "
```

2.解方程：

四个字符一组，设为x1,x2,x3,x4

则有方程

```
(2*x1+2*x2+4*x3+(-5)*x4)&0xff %0x7f==25
(1*x1+1*x2+2*x3+(-3)*x4)&0xff %0x7f==c0
(-1*x1+(-2)*x2+(-3)*x3+(4)*x4)&0xff %0x7f==3b
(-1*x1+0*x2+(-2)*x3+(2)*x4)&0xff %0x7f==a6
```

带取模运算的不好接。但是根据以往输入的数据计算结果显示：

```
348   221   -232  -109
276   189   -184  -85
397   264   -270  -143
245   146   -147  -97
161   105   -105  -56
-233  -130  206   103
```

第一个结果>第二个结果>（第三个结果>第四个结果），前两个多数为正数，后两个多为负数。

因此可将结果调整为：

```
25 c0 3b a6
1f af 4c a5
cb 8b a4 9b
3b e1 28 85
26 26 16 e7
11 09 07 26


293  192  -197  -90        {25->125  c0->c0     3b->ff3b            ffffffffffffffa6}
287  175  -180  -91        {1f->11f  ffffffffffffffaf->af 4C->ff4C  ffffffffffffffa5}
203  139  -92  -101        {cb      8b  ffa4  ff9b}
315  225  -216 -123        {3b->13b  ffffffffffffffe1_>e1  28-->ff28 ffffffffffffff85}
38   38   22   -25         {26->26  26->26  16-->16 ffffffffffffffe7}
17   9    7     38
```

http://www.yunsuanzi.com/matrixcomputations/solvelinearsystems.html

在线解方程转字符即可得到答案。

4.leftleftrightright [分值:150]

题目描述:

leftleftrightright.....Aha, here is the flag!

附件是一个win32控制台程序，加有upx壳。

程序逻辑：

```
int main_401170()
{
  int v0; // edx@1
  int v1; // ebx@1
  int v2; // edi@1
  void **v3; // esi@1
  int v4; // ecx@1
  void **v5; // eax@2
  void **v6; // edx@3
  void *v7; // eax@5
  char *v8; // ecx@6
  unsigned int v9; // eax@8
  void *v10; // eax@16
  unsigned int v11; // eax@19
  unsigned int v12; // edi@26
  unsigned int v13; // eax@26
  const char *v14; // edx@30
  int v15; // eax@32
  int v16; // ecx@32
  unsigned int v17; // eax@36
  unsigned int v18; // ecx@38
  unsigned int v19; // eax@48
  unsigned int v20; // esi@50
```

```
int v22; // [sp+10h] [bp-5Ch]@1
void *Memory; // [sp+14h] [bp-58h]@5
unsigned int v24; // [sp+28h] [bp-44h]@5
void *v25; // [sp+2Ch] [bp-40h]@1
int v26; // [sp+3Ch] [bp-30h]@1
unsigned int v27; // [sp+40h] [bp-2Ch]@1
void *Dst; // [sp+44h] [bp-28h]@1
unsigned int v29; // [sp+54h] [bp-18h]@1
unsigned int v30; // [sp+58h] [bp-14h]@1
int v31; // [sp+68h] [bp-4h]@1

v27 = 15;
v26 = 0;
LOBYTE(v25) = 0;
v31 = 0;
v30 = 15;
v29 = 0;
LOBYTE(Dst) = 0;
LOBYTE(v31) = 1;
input_401F60(std::cin, (int)&v25);
v1 = v26;
v2 = 0;
v3 = (void **)v25;
v4 = v26 - 1;
v22 = v26 - 1;
while ( v1 > 0 )
{
  --v1;
  v5 = &v25;
  if ( v1 & 2 )
  {
    v6 = &Dst;
    if ( v27 >= 0x10 )
      v5 = v3;
    LOBYTE(v6) = *((_BYTE *)v5 + v4);
    v7 = (void *)string_xx_401AD0((int)&Memory, (int)v6, &Dst);
    string_xxx_401480(&Dst, v7);
    if ( v24 >= 0x10 )
    {
      v8 = (char *)Memory;
      if ( v24 + 1 >= 0x1000 )
      {
        if ( (unsigned __int8)Memory & 0x1F )
          goto LABEL_35;
        v9 = *((_DWORD *)Memory - 1);
        if ( v9 >= (unsigned int)Memory )
          goto LABEL_35;
        v8 = (char *)Memory - v9;
        // 长度  4-0x23
        if ( (char *)Memory - v9 < (char *)4 || (unsigned int)v8 > 0x23 )
          goto LABEL_35;
        v8 = (char *)*((_DWORD *)Memory - 1);
      }
      j_free(v8);
    }
    v4 = v22-- - 1;
  }
  else
  {
```

```
      if ( v27 >= 0x10 )
        v5 = v3;
      LOBYTE(v0) = *((_BYTE *)v5 + v2);
      v10 = (void *)string_xx_401AD0((int)&Memory, v0, &Dst);
      string_xxx_401480(&Dst, v10);
      if ( v24 >= 0x10 )
      {
        v8 = (char *)Memory;
        if ( v24 + 1 >= 0x1000 )
        {
          if ( (unsigned __int8)Memory & 0x1F )
            goto LABEL_35;
          v11 = *((_DWORD *)Memory - 1);
          if ( v11 >= (unsigned int)Memory )
            goto LABEL_35;
          v8 = (char *)Memory - v11;
          if ( (char *)Memory - v11 < (char *)4 || (unsigned int)v8 > 0x23 )
            goto LABEL_35;
          v8 = (char *)*((_DWORD *)Memory - 1);
        }
        j_free(v8);
      }
      v4 = v22;
      ++v2;
    }
  }
  v12 = v29;
  v13 = 0x1D;
  if ( v29 < 0x1D )
    v13 = v29;
  if ( sub_401090(v13) || v12 < 0x1D || (v14 = "flag is right!", v12 > 0x1D) )
    v14 = "try again!";
  v15 = sub_401BF0(std::cout, v14);
  std::basic_ostream<char,std::char_traits<char>>::operator<<(v15, sub_401E30);
  system("pause");
  if ( v30 >= 0x10 )
  {
    v8 = (char *)Dst;
    if ( v30 + 1 >= 0x1000 )
    {
      if ( (unsigned __int8)Dst & 0x1F )
LABEL_35:
        invalid_parameter_noinfo_noreturn(v8);
      v17 = *((_DWORD *)v8 - 1);
      if ( v17 >= (unsigned int)v8 )
        v17 = invalid_parameter_noinfo_noreturn(v8);
      v18 = (unsigned int)&v8[-v17];
      if ( v18 < 4 )
        v17 = invalid_parameter_noinfo_noreturn(v18);
      if ( v18 > 0x23 )
        v17 = invalid_parameter_noinfo_noreturn(v18);
      v8 = (char *)v17;
    }
    j_free(v8);
  }
  v30 = 15;
  v29 = 0;
  LOBYTE(Dst) = 0;
  if ( v27 >= 0x10 )
  {
```

```
  {
    if ( v27 + 1 >= 0x1000 )
    {
      if ( (unsigned __int8)v25 & 0x1F )
        invalid_parameter_noinfo_noreturn(v16);
      v19 = (unsigned int)*(v3 - 1);
      if ( v19 >= (unsigned int)v3 )
        v19 = invalid_parameter_noinfo_noreturn(v16);
      v20 = (unsigned int)v3 - v19;
      if ( v20 < 4 )
        v19 = invalid_parameter_noinfo_noreturn(v16);
      if ( v20 > 0x23 )
        v19 = invalid_parameter_noinfo_noreturn(v16);
      v3 = (void **)v19;
    }
    j_free(v3);
  }
  return 0;
}
```

比较乱，也懒得看。

大致流程就是检查长度，然后打乱输入字符串，最后与字符串s_imsaplw_e_siishtnt{g_ialt}F比较，相等即成功。

既然给了最后的字符串，而且算法只是简单的移位的话，说明要输入的字符串长度必定与此字符串相等，由此可知字符串长度为29.

构造一个29位的字符串：0123456789abcdefghiklmnopqrst

在最后比较的地方查看移位后的字符串：edfgcbhia9kl87mn65op43qr21st0

有了这几个字符串，对照着还原即可：

```
str_end="s_imsaplw_e_siishtnt{g_ialt}F"

str_in="0123456789abcdefghiklmnopqrst"
str_out="edfgcbhia9kl87mn65op43qr21st0"

strxx=""
for x in str_in:
    index=str_out.find(x)
    strxx+=str_end[index]

print strxx
```