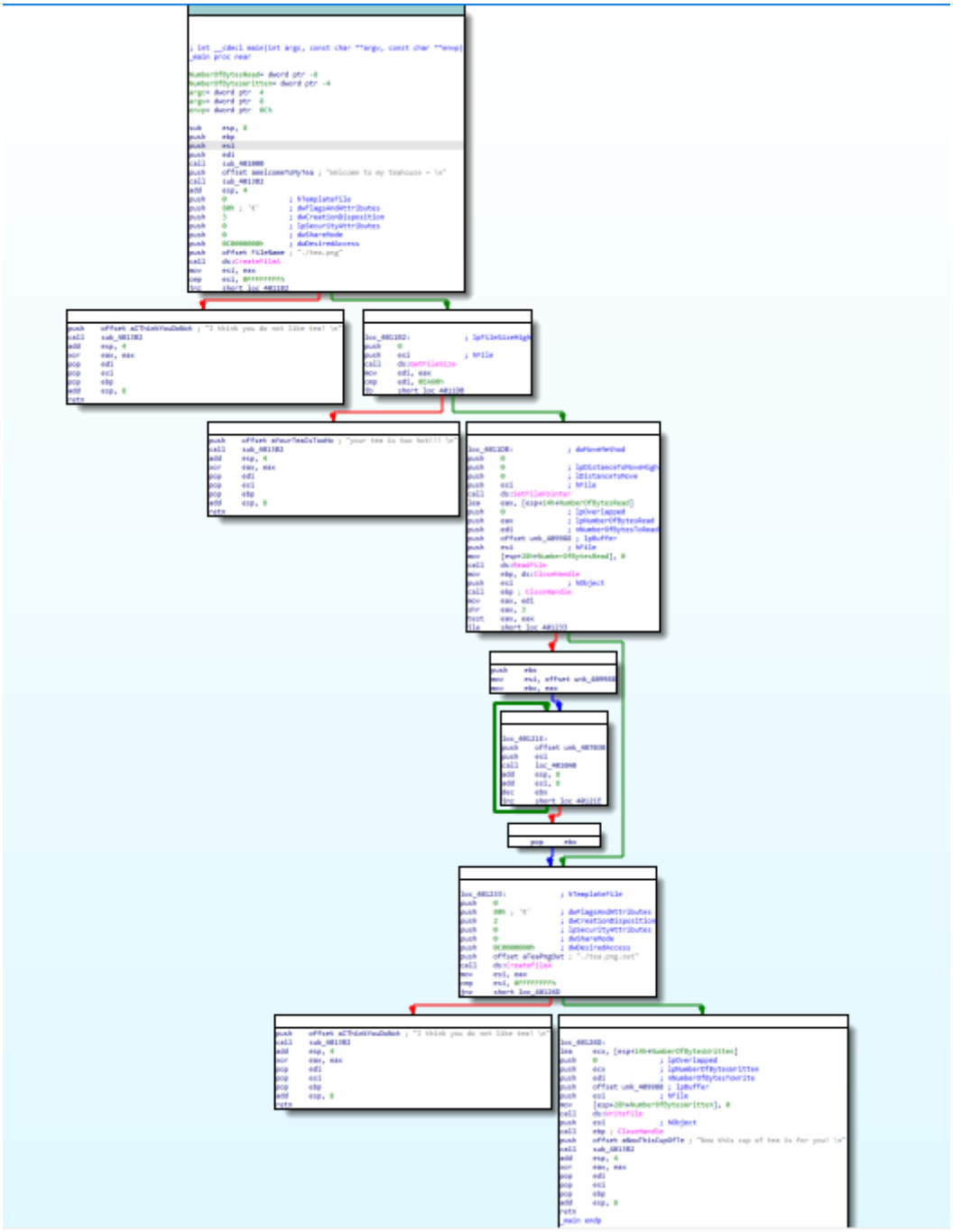
method 1th

首先用IDA载入，



流程很简单，我们看到main（我按了一下反斜杠键，看着清爽些）

```
int __cdecl main(int argc, const char **argv, const char **envp)
{
  HANDLE v3; // eax
  void *v4; // esi
  int result; // eax
  DWORD v6; // edi
  char *v7; // esi
  DWORD v8; // ebx
  HANDLE v9; // eax
```

```
  void *v10; // esi
  DWORD NumberOfBytesRead; // [esp+Ch] [ebp-8h] BYREF
  DWORD NumberOfBytesWritten; // [esp+10h] [ebp-4h] BYREF

  sub_401000();
  sub_4013B2(aWelcomeToMyTea);
  v3 = CreateFileA(FileName, 0xC0000000, 0, 0, 3u, 0x80u, 0);
  v4 = v3;
  if ( v3 == -1 )
  {
    sub_4013B2(aIThinkYouDoNot);
    result = 0;
  }
  else
  {
    v6 = GetFileSize(v3, 0);
    if ( v6 < 0xEA60 )
    {
      SetFilePointer(v4, 0, 0, 0);
      NumberOfBytesRead = 0;
      ReadFile(v4, &unk_409988, v6, &NumberOfBytesRead, 0);
      CloseHandle(v4);
      if ( v6 >> 3 )
      {
        v7 = &unk_409988;
        v8 = v6 >> 3;
        do
        {
          (loc_4010A0)(v7, &unk_407030);
          v7 += 8;
          --v8;
        }
        while ( v8 );
      }
      v9 = CreateFileA(aTeaPngOut, 0xC0000000, 0, 0, 2u, 0x80u, 0);
      v10 = v9;
      if ( v9 == -1 )
      {
        sub_4013B2(aIThinkYouDoNot);
      }
      else
      {
        NumberOfBytesWritten = 0;
        WriteFile(v9, &unk_409988, v6, &NumberOfBytesWritten, 0);
        CloseHandle(v10);
        sub_4013B2(aNowThisCupOfTe);
      }
      result = 0;
    }
    else
    {
      sub_4013B2(aYourTeaIsTooHo);
      result = 0;
    }
  }
  return result;
}
```

逻辑很简单：

首先读入一个文件tea.png【判断是否读入成功，否则退出】

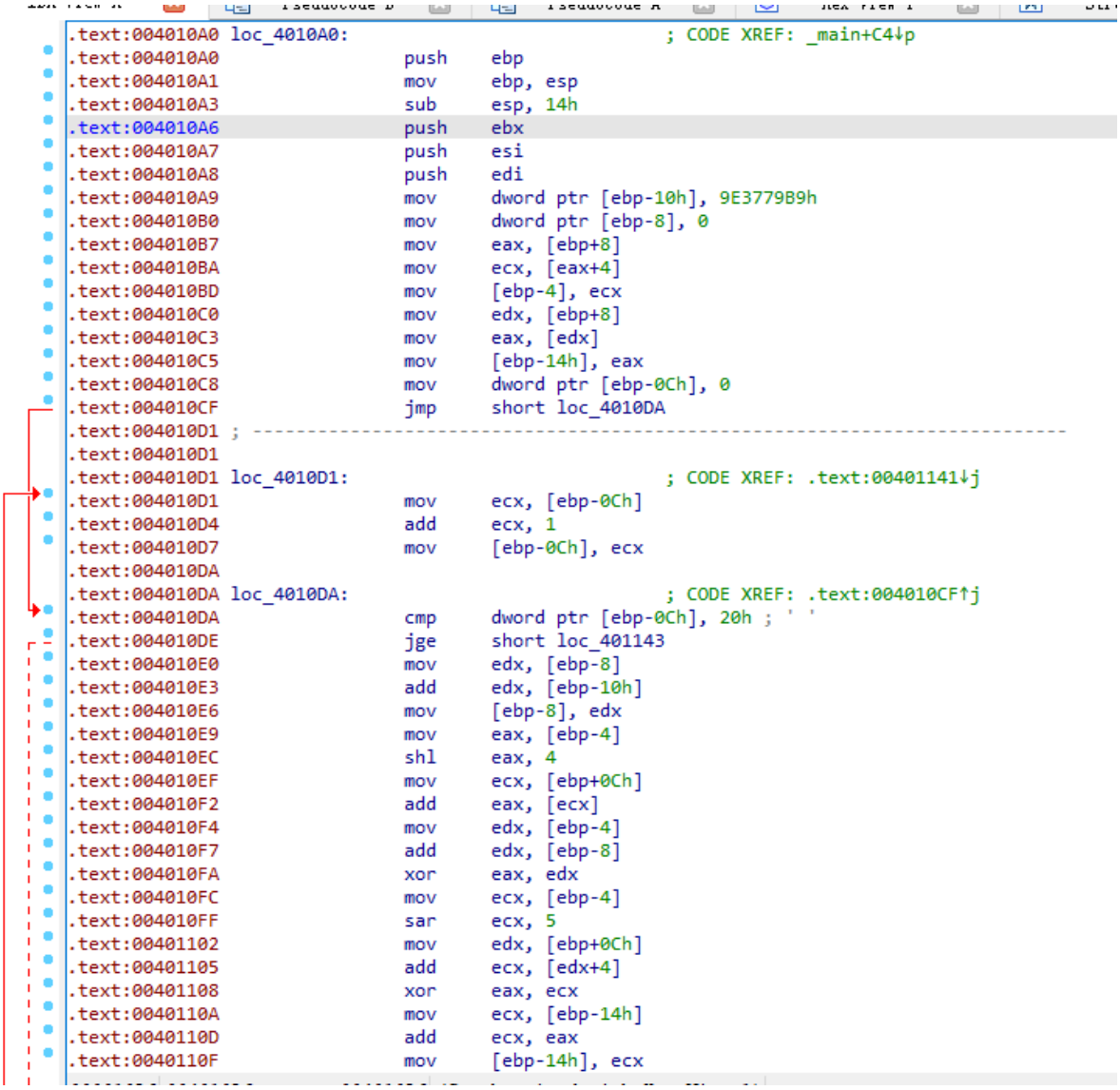
获取文件大小【如果文件大于等于0xEA60字节，退出】

然后将文件大小除以8，作为循环躺数（以8字节为单位对文件内容进行加密）

【加密函数】（文件地址，&unk_407030（**key**））

最后写入一个新的文件tea.png.out。

那我们就来看看这个加密函数



```
.text:004010A0 loc_4010A0:                             ; CODE XREF: _main+C4↓p
.text:004010A0                 push    ebp
.text:004010A1                 mov     ebp, esp
.text:004010A3                 sub     esp, 14h
.text:004010A6                 push    ebx
.text:004010A7                 push    esi
.text:004010A8                 push    edi
.text:004010A9                 mov     dword ptr [ebp-10h], 9E3779B9h
.text:004010B0                 mov     dword ptr [ebp-8], 0
.text:004010B7                 mov     eax, [ebp+8]
.text:004010BA                 mov     ecx, [eax+4]
.text:004010BD                 mov     [ebp-4], ecx
.text:004010C0                 mov     edx, [ebp+8]
.text:004010C3                 mov     eax, [edx]
.text:004010C5                 mov     [ebp-14h], eax
.text:004010C8                 mov     dword ptr [ebp-0Ch], 0
.text:004010CF                 jmp     short loc_4010DA
.text:004010D1 ; ---------------------------------------------------------------------------
.text:004010D1
.text:004010D1 loc_4010D1:                             ; CODE XREF: .text:00401141↓j
.text:004010D1                 mov     ecx, [ebp-0Ch]
.text:004010D4                 add     ecx, 1
.text:004010D7                 mov     [ebp-0Ch], ecx
.text:004010DA
.text:004010DA loc_4010DA:                             ; CODE XREF: .text:004010CF↑j
.text:004010DA                 cmp     dword ptr [ebp-0Ch], 20h ; ' '
.text:004010DE                 jge     short loc_401143
.text:004010E0                 mov     edx, [ebp-8]
.text:004010E3                 add     edx, [ebp-10h]
.text:004010E6                 mov     [ebp-8], edx
.text:004010E9                 mov     eax, [ebp-4]
.text:004010EC                 shl     eax, 4
.text:004010EF                 mov     ecx, [ebp+0Ch]
.text:004010F2                 add     eax, [ecx]
.text:004010F4                 mov     edx, [ebp-4]
.text:004010F7                 add     edx, [ebp-8]
.text:004010FA                 xor     eax, edx
.text:004010FC                 mov     ecx, [ebp-4]
.text:004010FF                 sar     ecx, 5
.text:00401102                 mov     edx, [ebp+0Ch]
.text:00401105                 add     ecx, [edx+4]
.text:00401108                 xor     eax, ecx
.text:0040110A                 mov     ecx, [ebp-14h]
.text:0040110D                 add     ecx, eax
.text:0040110F                 mov     [ebp-14h], ecx
```

这里发现IDA反编译失败了，此时，要么就去啃汇编，要么就去找到原因

继续往下拉会发现

```
.text:0040110A                 mov     ecx, [ebp-14h]
.text:0040110D                 add     ecx, eax
.text:0040110F                 mov     [ebp-14h], ecx
.text:00401112                 jz      short near ptr loc_401116+1
.text:00401114                 jnz     short near ptr loc_401116+1
.text:00401116
.text:00401116 loc_401116:                             ; CODE XREF: .text:00401112↑j
.text:00401116                                         ; .text:00401114↑j
.text:00401116                 call    near ptr 0C22C66A6h
.text:0040111B                 loop    near ptr loc_401120+1
.text:0040111D                 mov     eax, [ebp+0Ch]
.text:00401120
.text:00401120 loc_401120:                             ; CODE XREF: .text:0040111B↑j
.text:00401120                 add     edx, [eax+8]
.text:00401123                 mov     ecx, [ebp-14h]
.text:00401126                 add     ecx, [ebp-8]
.text:00401129                 xor     edx, ecx
.text:0040112B                 mov     eax, [ebp-14h]
.text:0040112E                 sar     eax, 5
.text:00401131                 mov     ecx, [ebp+0Ch]
.text:00401134                 add     eax, [ecx+0Ch]
```

这里有个神奇的指令，看不懂，据说是什么花指令（这个坑以后填叭），void教我的，在这里先按U（undefined），然后把00401116处 的值改为90（在hex-view里，F2，改掉，F2保存），也就是nop掉，

然后往上找push ebp【函数起点】，按p，创建函数

```
.text:004010A0
.text:004010A0 loc_4010A0:                             ; CODE XREF: _main+C4↓p
.text:004010A0                 push    ebp
.text:004010A1                 mov     ebp, esp
.text:004010A3                 sub     esp, 14h
.text:004010A6                 push    ebx
.text:004010A7                 push    esi
.text:004010A8                 push    edi
.text:004010A9                 mov     dword ptr [ebp-10h], 9E3779B9h
.text:004010B0                 mov     dword ptr [ebp-8], 0
.text:004010B7                 mov     eax, [ebp+8]
.text:004010BA                 mov     ecx, [eax+4]
.text:004010BD                 mov     [ebp-4], ecx
.text:004010C0                 mov     edx, [ebp+8]
.text:004010C3                 mov     eax, [edx]
.text:004010C5                 mov     [ebp-14h], eax
.text:004010C8                 mov     dword ptr [ebp-0Ch], 0
.text:004010CF                 jmp     short loc_4010DA
.text:004010CF ; ---------------------------------------------------------------------------
.text:004010D1                 db  8Bh
.text:004010D2                 db  4Dh ; M
.text:004010D3                 db 0F4h
.text:004010D4                 db  83h
.text:004010D5                 db 0C1h
.text:004010D6                 db    1
.text:004010D7                 db  89h
.text:004010D8                 db  4Dh ; M
.text:004010D9                 db 0F4h
.text:004010DA ; ---------------------------------------------------------------------------
.text:004010DA
.text:004010DA loc_4010DA:                             ; CODE XREF: .text:004010CF↑j
.text:004010DA                 cmp     dword ptr [ebp-0Ch], 20h ; ' '
.text:004010DE                 jge     short loc_401143
.text:004010E0                 mov     edx, [ebp-8]
.text:004010E3                 add     edx, [ebp-10h]
.text:004010E6                 mov     [ebp-8], edx
.text:004010E9                 mov     eax, [ebp-4]
.text:004010EC                 shl     eax, 4
.text:004010EF                 mov     ecx, [ebp+0Ch]
.text:004010F2                 add     eax, [ecx]
.text:004010F4                 mov     edx, [ebp-4]
```

然后就可以快乐F5了

```
IDA View-A    Pseudocode-B    Pseudocode-A
int *__cdecl sub_4010A0(int *a1, _DWORD *a2)
{
  int *result; // eax
  int v3; // [esp+Ch] [ebp-14h]
  int i; // [esp+14h] [ebp-Ch]
  int v5; //int v3; // [esp+Ch] [ebp-14h]
  int v6; // [esp+1Ch] [ebp-4h]

  v5 = 0;
  v6 = a1[1];
  v3 = *a1;
  for ( i = 0; i < 32; ++i )
  {
    v5 -= 1640531527;
    v3 += (a2[1] + (v6 >> 5)) ^ (v5 + v6) ^ (*a2 + 16 * v6);
    v6 += (a2[3] + (v3 >> 5)) ^ (v5 + v3) ^ (a2[2] + 16 * v3);
  }
  a1[1] = v6;
  result = a1;
  *a1 = v3;
  return result;
}
```

可以发现这个函数的两个参数，一个是int类型的指针（4字节），一个是_DWORD类型的指针（4字节）

并且在这里的操作中，我们可以看到，对第一个参数只用了a1[0],a1[1]这两个参数，对第二个参数，是用到了四个值

我们在思考一下这两个参数原本的意义，一个是8字节的文件，另一个是key，去看一下key

```
.data:00407030 key             db  66h ; f             ; DATA XREF: _main:loc_40121E↑o
.data:00407031                 db  6Ch ; l
.data:00407032                 db  61h ; a
.data:00407033                 db  67h ; g
.data:00407034                 db  7Bh ; {
.data:00407035                 db  66h ; f
.data:00407036                 db  61h ; a
.data:00407037                 db  6Bh ; k
.data:00407038                 db  65h ; e
.data:00407039                 db  5Fh ; _
.data:0040703A                 db  66h ; f
.data:0040703B                 db  6Ch ; l
.data:0040703C                 db  61h ; a
.data:0040703D                 db  67h ; g
.data:0040703E                 db  21h ; !
.data:0040703F                 db  7Dh ; }
```

发现是个字符串，数一数是有16个字节的，所以估计是4个字节为一组作为一个_DWORD，那么第一个参数就是4个字符为一组作为一个int。然后在加密函数里进行操作。

那么字符怎么转换为数字呢？在.data块按d键就可以转化类型了，转化之后是

```
.data:00407030 key             dd 67616C66h            ; DATA XREF: _main:loc_40121E↑o
.data:00407034                 dd 6B61667Bh
.data:00407038                 dd 6C665F65h
.data:0040703C                 dd 7D216761h
.data:00407040 aGetOut         db 'get out! ',0Ah,0    ; DATA XREF: sub_401000:loc_401087↑o
.data:0040704B                 align 4
```

可以发现这个顺序怪怪的，是小端序，所以对于文件的数据而言，也是用的小端序了。

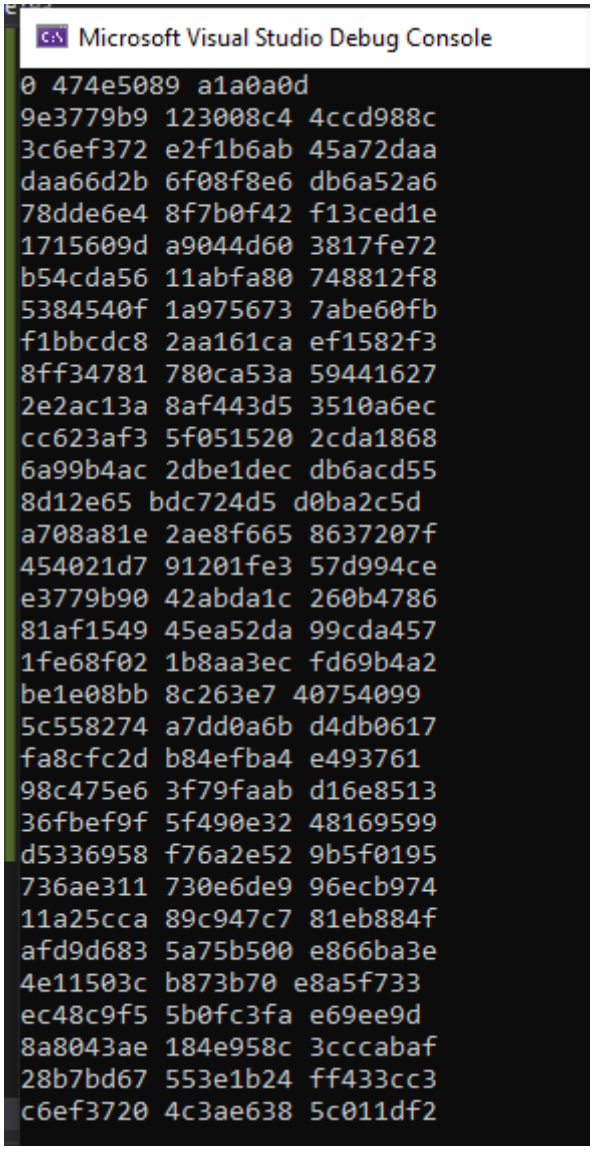
这里是对于前八字节的加密代码（我们能确定png文件的前八字节）

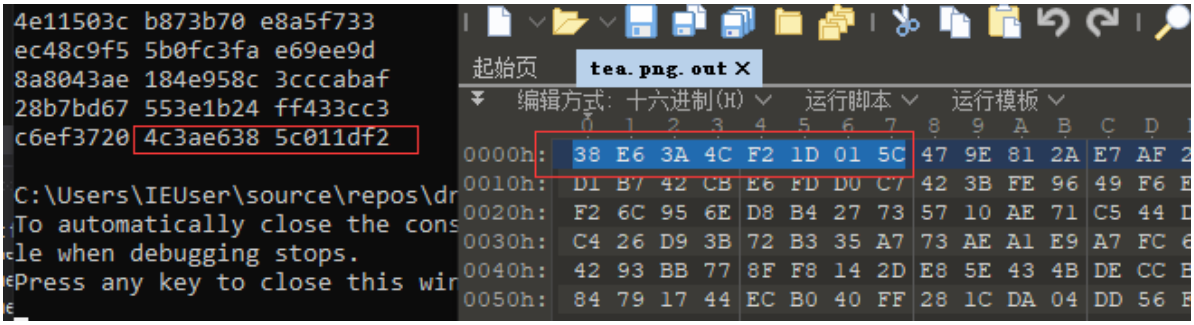
```
int main() {
    int key[4] = { 0x67616C66, 0x6b61667b, 0x6c665f65, 0x7d216761 };
```

```
    int v3 = 0; // [esp+Ch] [ebp-14h]
    int i; // [esp+14h] [ebp-Ch]
    int v5; // [esp+18h] [ebp-8h]
    int v6; // [esp+1Ch] [ebp-4h]
    int m[2] = { 0x474e5089, 0x0a1a0a0d };
    v5 = 0;
    v6 = m[1];
    v3 = m[0];
    for (i = 0; i < 32; ++i)
    {
        printf("%x %x %x\n", v5, v3, v6);
        v5 -= 0x61C88647;
        v3 += (key[1] + (v6 >> 5)) ^ (v5 + v6) ^ (key[0] + 16 * v6);
        v6 += (key[3] + (v3 >> 5)) ^ (v5 + v3) ^ (key[2] + 16 * v3);

    }
    m[1] = v6;
    m[0] = v3;
    printf("%x %x %x\n", v5, v3, v6);
    return 0;
}
```

运行结果





可以看到，前八字节的密文与他给的加密文件的头八字节是一致的。那么至此我们已经可以选择写这个算法的解密算法了。

这就是方法一，解密脚本暂时没有，因为C语言快忘光了，不怎么会操作，python写起来又不太对，python复现加密

```python
#enc
from Crypto.Util.number import *
v5=0
key = [0x67616C66, 0x6b61667b, 0x6c665f65, 0x7d216761]
m = [0x474e5089, 0x0a1a0a0d]
v3 = m[0]
v6 = m[1]
for j in range(32):
    print(hex(v5),hex(v3),hex(v6))
    v5 += 0x9E3779B9
    v5 &= 0xffffffff
    v3 += (key[1] + (v6 >> 5) ^ (v5 + v6)) ^ (key[0] + 16 * v6)
    v3 &= 0xffffffff
    v6 += (key[3] + (v3 >> 5) ^ (v5 + v3)) ^ (key[2] + 16 * v3)
    v6 &= 0xffffffff

print(hex(v3),hex(v6))


```

结果

```
('0x0', '0x474e5089', '0xa1a0a0d')
('0x9e3779b9L', '0x123008c4L', '0x4ccd988cL')
('0x3c6ef372L', '0xe2f1b6abL', '0x4da72daaL')
('0xdaa66d2bL', '0xef748f8e6L', '0xf2e852a6L')
('0x78dde6e4L', '0x7420ff42L', '0xb5332d9eL')
('0x1715609dL', '0x7bf736dcL', '0xef07a8e9L')
('0xb54cda56L', '0xfd4e10e7L', '0x65ddf9e9L')
('0x5384540fL', '0x1001222bL', '0xd83a7e46L')
('0xf1bbcdc8L', '0x60dd48d0L', '0x80c629a0L')
('0x8ff34781L', '0x6cf8295fL', '0xc850d6beL')
('0x2e2ac13aL', '0xd8aecbedL', '0x3abd2590L')
('0xcc623af3L', '0x51cab62fL', '0x234fb3f5L')
('0x6a99b4acL', '0xcf98dd3cL', '0xffa8407cL')
('0x8d12e65L', '0xea5b8ff5L', '0x64efe38bL')
('0xa708a81eL', '0xbd8b5b1dL', '0x775b3c2L')
('0x454021d7L', '0xb71fc423L', '0xa85cd5afL')
('0xe3779b90L', '0xcd7ebf64L', '0x1f8627bcL')
('0x81af1549L', '0x602a01ffL', '0x2e840729L')
('0x1fe68f02L', '0xcd48d068L', '0x5cdb9e94L')
('0xbe1e08bbL', '0xef381eeL', '0x47f29630L')
('0x5c558274L', '0x3e55aedcL', '0xfc70fed2L')
('0xfa8cfc2dL', '0xea1dfbe4L', '0x6b0fb1c6L')
('0x98c475e6L', '0x5f4f4747L', '0x84624129L')
('0x36fbef9fL', '0xd8ae7e01L', '0xff64caadL')
('0xd5336958L', '0xd3192de4L', '0xb57464f6L')
('0x736ae311L', '0xba8f2847L', '0x6fcb6d24L')
('0x11a25ccaL', '0x46395ef3L', '0x57403494L')
('0xafd9d683L', '0xf89f35a1L', '0x320708f3L')
('0x4e11503cL', '0x63d7721cL', '0xca7c6a7fL')
('0xec48c9f5L', '0x2c2fdc08L', '0x141ac6d8L')
('0x8a8043aeL', '0x87c4cbd9L', '0x8fc465a5L')
('0x28b7bd67L', '0x3bc937ebL', '0xc24b000cL')
('0xb0221c5cL', '0x5d860e26L')
```

第三行的v6开始出错，估计是sar的问题，也就是那个算术位移操作【带符号的】

所以就放弃了，我选择patch这个程序，让他从加密程序，变成解密程序。

method 2nd

他的加密逻辑很简单，

先加v5

然后处理v3【v3与v5和v6有关】

最后处理v6【v6与v5和已经被处理的v3有关】

那么解密逻辑也不难，

先处理v6【最后的v6与最后的v5，v3的值有关，且这两个值已知，所以v6可以还原为上一轮的值】

然后处理v3【最后的v3与最后v5和上一轮的v6值有关，上一轮的v6值我们已经处理出来了，最后的v5我们也知道，因此v3也可以还原为为上一轮的值】

然后v5减去那个常数

所以我们可以选择从底层（汇编）改他的程序流程

但是比赛的时候我又失败了，改流程得稍微处理下，最后我是在原程序的基础上，patch了程序处理的值

method 3rd

把v5的初始值改为v5的最终值，然后把v5的add改为sub，那么程序一开始就是sub一下，意味着v5的值一开始就会被还原为上一轮，这是我们不期望的，我们还没用它呢，所以我们得在v5的最终值的基础上，在加一轮先【也就是改成第33轮的v5】，方便程序去sub

然后就是把v3 patch成v6，把v6 patch成v3。

与此同时我们也要把他们用到的key调换一下，但调换的时候会遇到问题，好像是代码的长度对不上，那么我们就直接在.data段【hex-view】把key前后两段的顺序换一下，这样效果还是一样的。

结果为

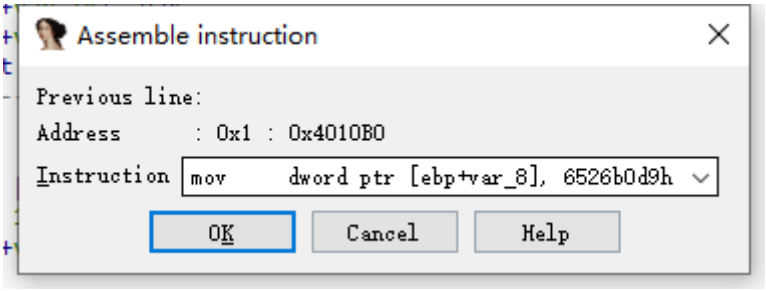
```
int *__cdecl sub_4010A0(int *a1, _DWORD *a2)
{
  int *result; // eax
  int v3; // [esp+Ch] [ebp-14h]
  int i; // [esp+14h] [ebp-Ch]
  int v5; // [esp+18h] [ebp-8h]
  int v1; // [esp+1Ch] [ebp-4h]

  v5 = 0x6526B0D9;
  v1 = a1[1];
  v3 = *a1;
  for ( i = 0; i < 32; ++i )
  {
    v5 += 0x61C88647;
    v1 -= (a2[1] + (v3 >> 5)) ^ (v5 + v3) ^ (*a2 + 16 * v3);
    v3 -= (a2[3] + (v1 >> 5)) ^ (v5 + v1) ^ (a2[2] + 16 * v1);
  }
  a1[1] = v1;
  result = a1;
  *a1 = v3;
  return result;
}
```

```
.data:0040702C                  align 10h
.data:00407030 dword_407030     dd 'lf_e'               ; DATA XREF: _main:loc_40121E↑o
.data:00407034                  dd '}!ga'
.data:00407038                  dd 'galf'
.data:0040703C                  dd 'kaf{'
.data:00407040 aGetOut          db 'get out! ',0Ah,0    ; DATA XREF: sub_401000:loc_401087↑o
.data:0040704B                  align 4
```

注：patch：【Edit-Patch Program-Asemble】

patch的时候给v5赋值的时候要用dword ptr，不然会报错



patch完了保存【Edit-Patch Program-Apply patches to input file】

这样子这个程序就变成了解密程序了，将加密图片的后缀去掉，点击这个程序，然后就会生成一个tea.png.out文件，再把这个文件后缀去掉，就是flag的图片了。

patch好的程序也一起塞着这里了

【回头补一下前两个方法的做法】