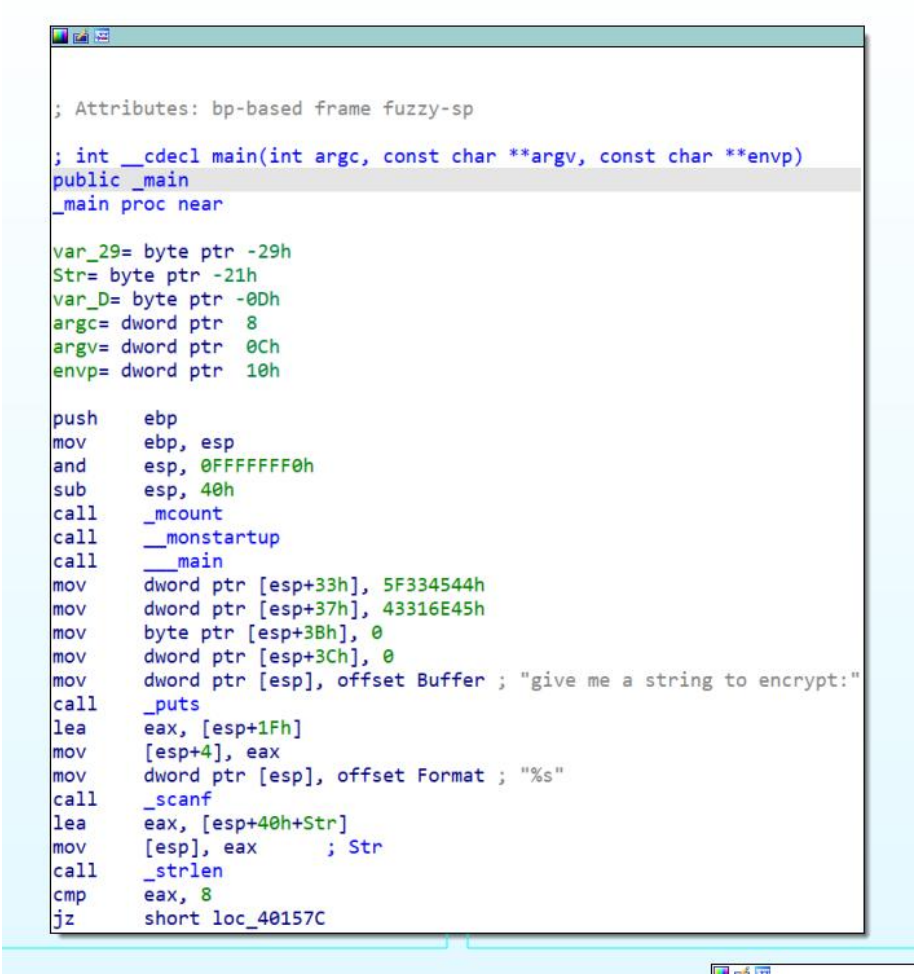
汇编语言与逆向工程第二次作业-DES 算法逆向分析

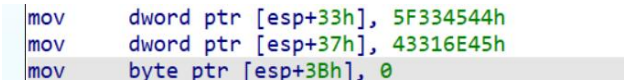
首先编译 DESEnc.cpp 得到 DESEnc.exe，用 IDA 打开

来到 main 函数

先来看第一块：



```
; Attributes: bp-based frame fuzzy-sp

; int __cdecl main(int argc, const char **argv, const char **envp)
public _main
_main proc near

var_29= byte ptr -29h
Str= byte ptr -21h
var_D= byte ptr -0Dh
argc= dword ptr  8
argv= dword ptr  0Ch
envp= dword ptr  10h

push    ebp
mov     ebp, esp
and     esp, 0FFFFFFF0h
sub     esp, 40h
call    _mcount
call    __monstartup
call    ___main
mov     dword ptr [esp+33h], 5F334544h
mov     dword ptr [esp+37h], 43316E45h
mov     byte ptr [esp+3Bh], 0
mov     dword ptr [esp+3Ch], 0
mov     dword ptr [esp], offset Buffer ; "give me a string to encrypt:"
call    _puts
lea     eax, [esp+1Fh]
mov     [esp+4], eax
mov     dword ptr [esp], offset Format ; "%s"
call    _scanf
lea     eax, [esp+40h+Str]
mov     [esp], eax      ; Str
call    _strlen
cmp     eax, 8
jz      short loc_40157C
```



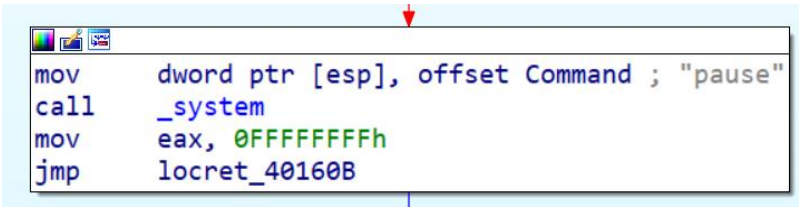
```
mov     dword ptr [esp+33h], 5F334544h
mov     dword ptr [esp+37h], 43316E45h
mov     byte ptr [esp+3Bh], 0
```

这三行给出了密钥是“DE3_En1C”

_scanf 函数表示输入密码，存到从 esp+1Fh 开始的地址上

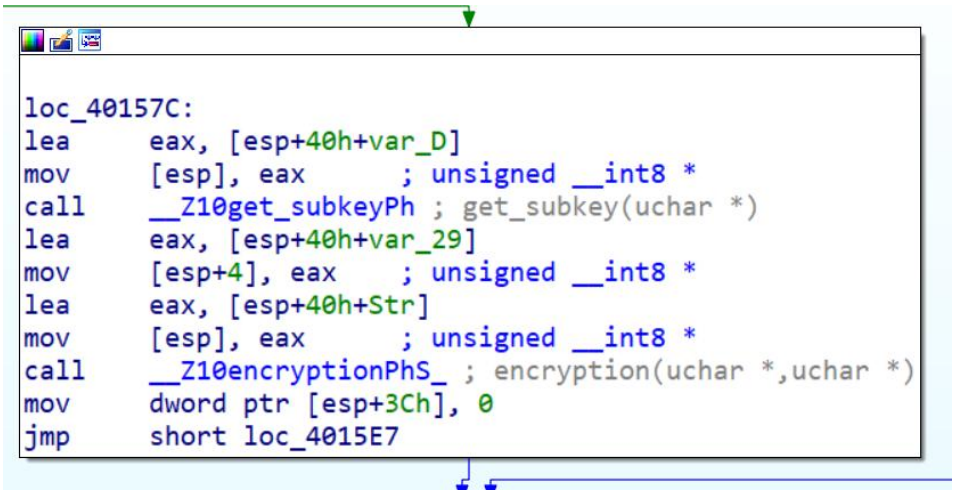
_strlen 求输入的长度，存入 eax

输入长度和 8 比较，如果不相等就向左跳转，



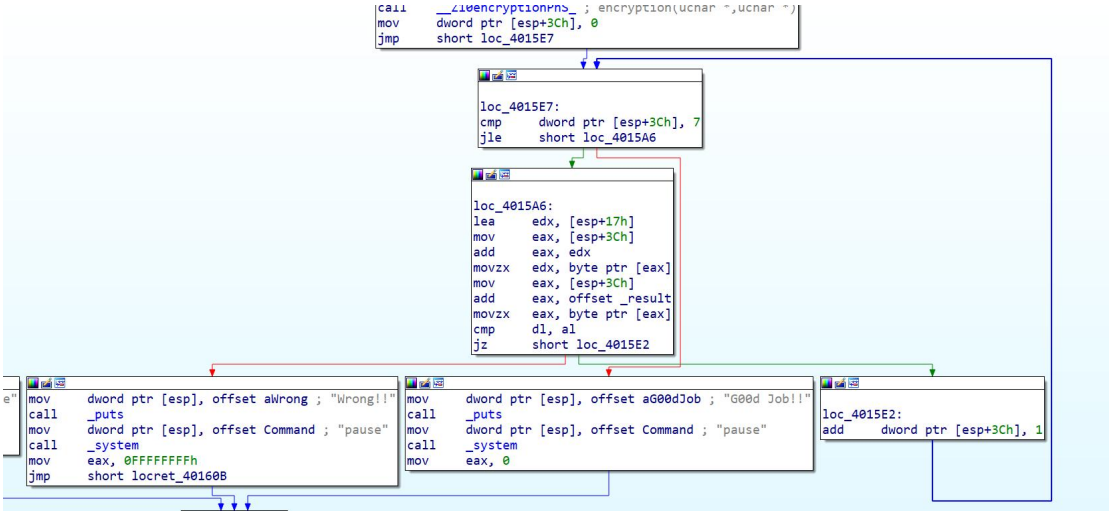
会调用_system 函数终止程序运行；

如果相等向右跳转，



转，可以猜测这个框就是加密过程

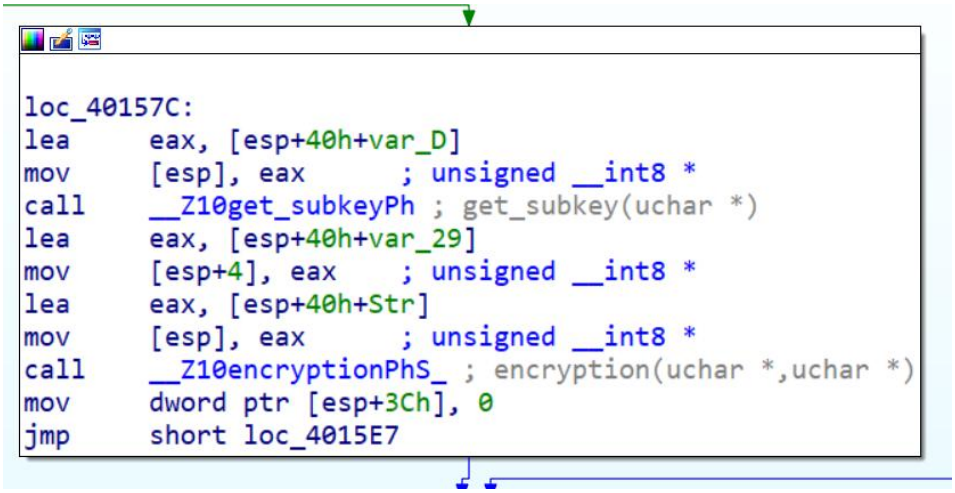
先往下看：



由把[esp+3Ch]置零，然后和 7 比较，可知这里是一位一位和密文比较，若错误则输出“Wrong!!”，正确则循环到 7，然后跳转到输出“G00d Job!!”

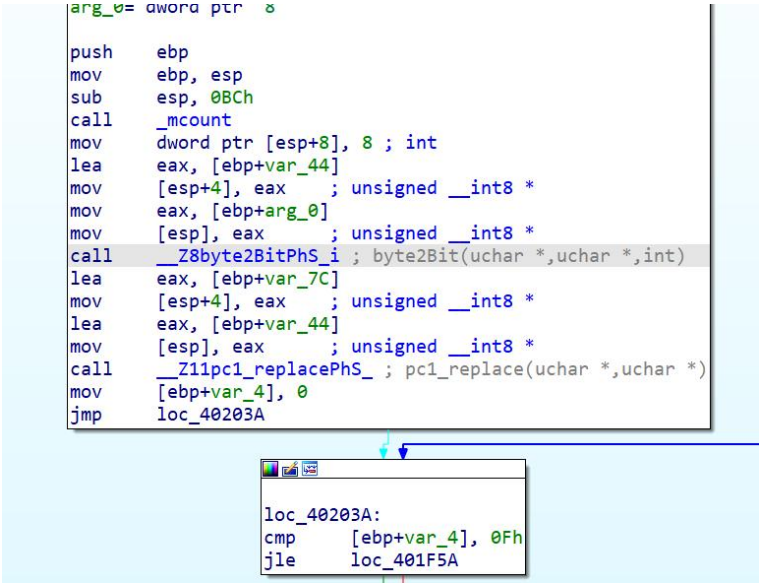
从中间那个块，我们可以得到密文存放的位置是_result，点进去得到密文是“EFh, 34h, D4h, A3h, C6h, 84h, E4h, 23h”

好，我们已经得到了密文，然后再去看加密过程那个块：



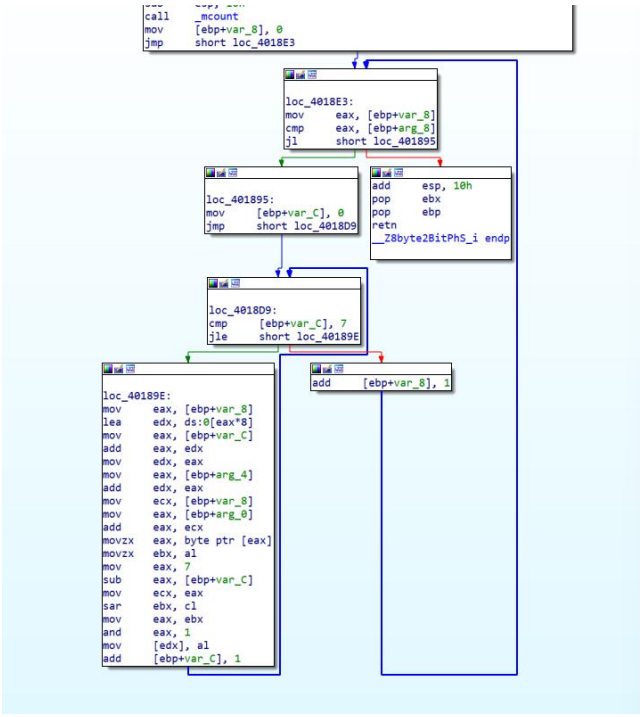
第一个函数_Z10get_subkeyPh，只有一个参数，就是存放密钥的位置，所以这个函数应该是生成 16 轮子密钥的函数，

点进去



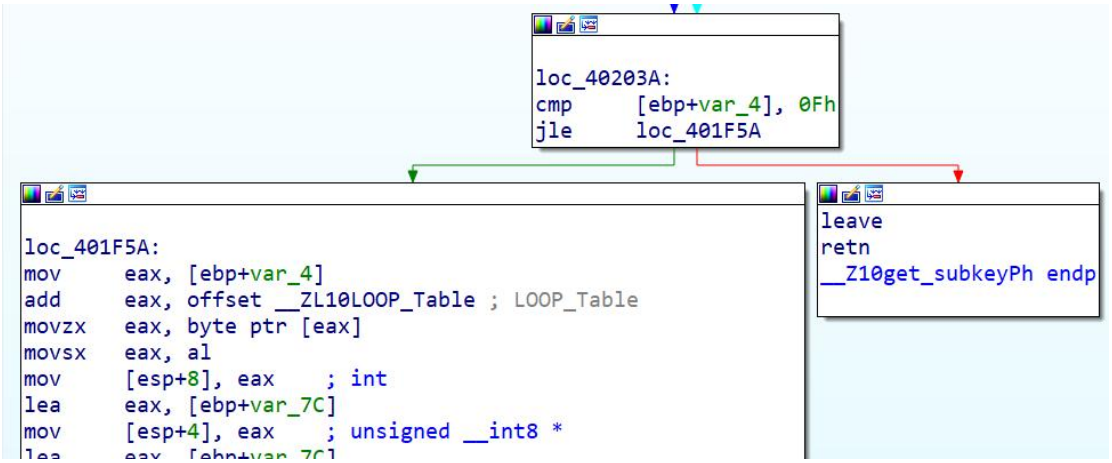
观察到_Z8byte2BitPhS_i 函数，参数有密钥，进入，

观察到两层循环结构都是循环八次，可知，他是把密钥的 8 个字节转化为 64 位二进制数，存放到另一个参数 var_44 中。



参数 var_44 又作为函数__Z11pc1_replacePhS_的输入，由右侧 pc1_replace(uchar*,uchar*)可知，该函数是对 64 位密钥进行 PC1 置换，

下一块



var_4 被置零，和 15 比较，可以猜测，是生成 16 轮密钥的循环

在左侧代码块中找到，行移位和 PC2 置换的函数，加以证明这个函数就是生成 16 轮密钥的函数。

```
mov     [esp], eax      ; unsigned __int8 *
call    __Z10shift_leftPhS_i ; shift_left(uchar *,uchar *,int)
mov     eax, [ebp+var_4]
add     eax, offset __ZL10LOOP_Table ; LOOP_Table
movzx   eax, byte ptr [eax]
movsx   eax, al
mov     [esp+8], eax    ; int
lea     eax, [ebp+var_7C]
add     eax, 1Ch
mov     [esp+4], eax    ; unsigned __int8 *
lea     eax, [ebp+var_7C]
add     eax, 1Ch
mov     [esp], eax      ; unsigned __int8 *
call    __Z10shift_leftPhS_i ; shift_left(uchar *,uchar *,int)
lea     eax, [ebp+var_AC]
mov     [esp+4], eax    ; unsigned __int8 *
lea     eax, [ebp+var_7C]
mov     [esp], eax      ; unsigned __int8 *
call    __Z11pc2_replacePhS_ ; pc2_replace(uchar *,uchar *)
mov     edx, [ebp+var_4]
mov     eax, edx
add     eax, eax
```

我们再回到上一层 main 函数，

```
call    __Z10get_subkeyPh ; get_subkey(uchar *)
lea     eax, [esp+40h+var_29]
mov     [esp+4], eax    ; unsigned __int8 *
lea     eax, [esp+40h+Str]
mov     [esp], eax      ; unsigned __int8 *
call    __Z10encryptionPhS_ ; encryption(uchar *,uchar *)
mov     dword ptr [esp+3Ch], 0
jmp     short loc_4015E7
```

下一个函数以我们输入的字符串和 8 个空地址为参数，应该是加密函数，加密我们输入的字符串，存放在空地址当中；

点进去分析：

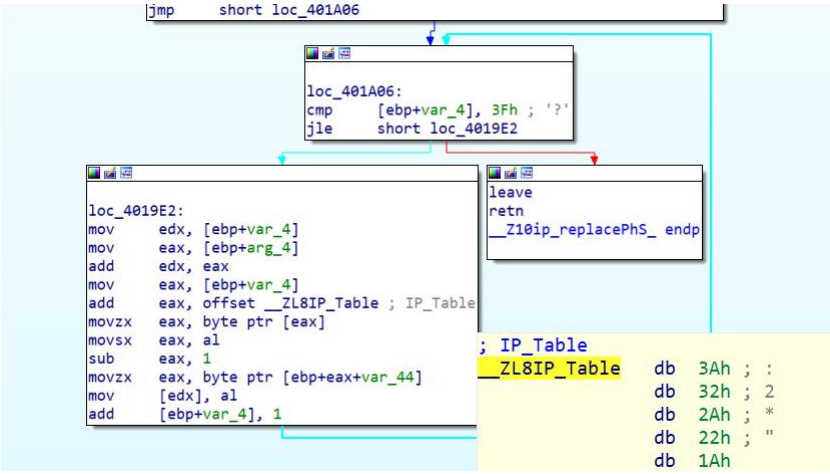
```
arg_0= dword ptr  8
arg_4= dword ptr  0Ch

push    ebp
mov     ebp, esp
sub     esp, 0C8h
call    _mcount
mov     dword ptr [esp+8], 8 ; int
lea     eax, [ebp+var_4C]
mov     [esp+4], eax    ; unsigned __int8 *
mov     eax, [ebp+arg_0]
mov     [esp], eax      ; unsigned __int8 *
call    __Z8byte2BitPhS_i ; byte2Bit(uchar *,uchar *,int)
lea     eax, [ebp+var_4C]
mov     [esp+4], eax    ; unsigned __int8 *
lea     eax, [ebp+var_4C]
mov     [esp], eax      ; unsigned __int8 *
call    __Z10ip_replacePhS_ ; ip_replace(uchar *,uchar *)
mov     eax, dword ptr [ebp+var_4C]
mov     dword ptr [ebp+var_8C], eax
mov     eax, [ebp+var_48]
mov     [ebp+var_88], eax
mov     eax, [ebp+var_44]
mov     [ebp+var_84], eax
mov     eax, [ebp+var_40]
mov     [ebp+var_80], eax
mov     eax, [ebp+var_3C]
mov     [ebp+var_7C], eax
mov     eax, [ebp+var_38]
mov     [ebp+var_78], eax
mov     eax, [ebp+var_34]
mov     [ebp+var_74], eax
mov     eax, [ebp+var_30]
mov     [ebp+var_70], eax
lea     eax, [ebp+var_4C]
add     eax, 20h ; ' '
```

arg_0 应该是我们输入的字符串，arg_4 为存放地址

首先调用__Z8byte2BitPhS_i 函数把我们输入的字符串变成 64 位 2 进制数，

然后压入两个空地址，调用__Z10ip_replacePhS_ 函数，

进入观察



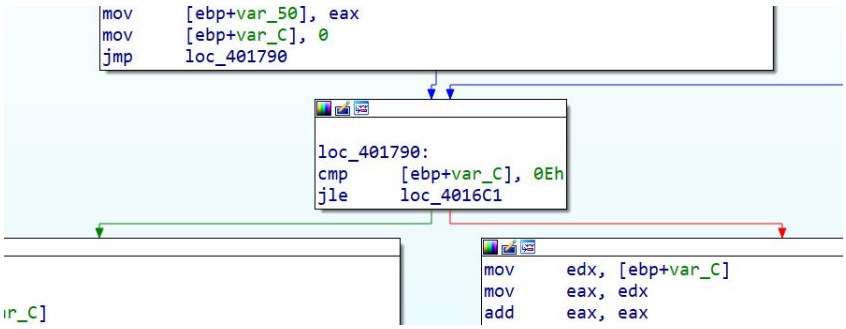
这里拿 var_4 和 63 比较循环，下面有 IP 表，可知为初始 IP 置换函数，

回到加密函数；

调用__Z10ip_replacePhS_ 函数之后进行了多次移动，可知是将 64 位数据分成 L0 和 R0，方便进行后续 16 轮加密，L0 存放的位置是 ebp-8C 到 ebp-70，R0 存放的位置是 ebp-6C 到 ebp-50，

```
call    __Z10ip_replacePhS_ ; ip_replace(uc
mov     eax, dword ptr [ebp+var_4C]
mov     dword ptr [ebp+var_8C], eax
mov     eax, [ebp+var_48]
mov     [ebp+var_88], eax
mov     eax, [ebp+var_44]
mov     [ebp+var_84], eax
mov     eax, [ebp+var_40]
mov     [ebp+var_80], eax
mov     eax, [ebp+var_3C]
mov     [ebp+var_7C], eax
mov     eax, [ebp+var_38]
mov     [ebp+var_78], eax
mov     eax, [ebp+var_34]
mov     [ebp+var_74], eax
mov     eax, [ebp+var_30]
mov     [ebp+var_70], eax
lea     eax, [ebp+var_4C]
add     eax, 20h ; ' '
mov     edx, [eax]
mov     dword ptr [ebp+var_6C], edx
mov     edx, [eax+4]
mov     [ebp+var_68], edx
mov     edx, [eax+8]
mov     [ebp+var_64], edx
mov     edx, [eax+0Ch]
mov     [ebp+var_60], edx
mov     edx, [eax+10h]
mov     [ebp+var_5C], edx
mov     edx, [eax+14h]
mov     [ebp+var_58], edx
mov     edx, [eax+18h]
mov     [ebp+var_54], edx
mov     eax, [eax+1Ch]
mov     [ebp+var_50], eax
mov     [ebp+var_C], 0
jmp     loc_401790
```

接着往下看：



这里把 var-C 置零然后和 15 比较，共比较 16 次才跳右边，可知是 16 轮加密，左边就是轮加密

```
loc_4016C1:
mov     edx, [ebp+var_C]
mov     eax, edx
add     eax, eax
add     eax, edx
shl     eax, 4
add     eax, offset _subkey
mov     [esp+8], eax    ; unsigned __int8 *
lea     eax, [ebp+var_AC]
mov     [esp+4], eax    ; unsigned __int8 *
lea     eax, [ebp+var_6C]
mov     [esp], eax      ; unsigned __int8 *
call    __Z6f_funcPhS_S_ ; f_func(uchar *,uchar *,uchar *)
mov     dword ptr [esp+8], 20h ; ' ' ; int
lea     eax, [ebp+var_8C]
mov     [esp+4], eax    ; unsigned __int8 *
lea     eax, [ebp+var_AC]
mov     [esp], eax      ; unsigned __int8 *
call    __Z7byteXORPhS_i ; byteXOR(uchar *,uchar *,int)
mov     eax, dword ptr [ebp+var_6C]
mov     dword ptr [ebp+var_8C], eax
mov     eax, [ebp+var_68]
mov     [ebp+var_88], eax
mov     eax, [ebp+var_64]
```

_Z6f_funcPhS_S 函数有三个参数，分别是 R0，一个地址，和子密钥，可见其为轮函数中的 Li=Li-1^F（K，Ri-1）中的 F 函数

我们点进去

```
push    ebp
mov     ebp, esp
sub     esp, 5Ch
call    _mcount
lea     eax, [ebp+var_30]
mov     [esp+4], eax    ; unsigned __int8 *
mov     eax, [ebp+arg_0]
mov     [esp], eax      ; unsigned __int8 *
call    __Z8e_expandPhS_ ; e_expand(uchar *,uchar *)
mov     dword ptr [esp+8], 30h ; '0' ; int
mov     eax, [ebp+arg_8]
mov     [esp+4], eax    ; unsigned __int8 *
lea     eax, [ebp+var_30]
mov     [esp], eax      ; unsigned __int8 *
call    __Z7byteXORPhS_i ; byteXOR(uchar *,uchar *,int)
lea     eax, [ebp+var_50]
mov     [esp+4], eax    ; unsigned __int8 *
lea     eax, [ebp+var_30]
mov     [esp], eax      ; unsigned __int8 *
call    __Z9s_replacePhS_ ; s_replace(uchar *,uchar *)
lea     eax, [ebp+var_50]
mov     [esp+4], eax    ; unsigned __int8 *
lea     eax, [ebp+var_50]
mov     [esp], eax      ; unsigned __int8 *
call    __Z9p_replacePhS_ ; p_replace(uchar *,uchar *)
mov     eax, [ebp+arg_4]
mov     edx, dword ptr [ebp+var_50]
mov     [eax], edx
mov     edx, [ebp+var_4C]
```

发现_Z6f_funcPhS_S 函数里面有四个与加密有关的函数，分别是__Z8e_expandPhS_（应该是对 Ri-1 从 32 位扩展到 48 位的函数），__Z7byteXORPhS_i （将 48 位和轮密钥按位异或的函数），__Z9s_replacePhS_（将 48 位按 S 盒选择字节压缩到 32 位的函数），

__Z9p_replacePhS_（最后对 32 位数据进行 P 置换）

回到加密函数：

在轮函数之后又调用了一次__Z7byteXORPhS_i 函数，参数是[ebp+var_8C]（L0 的首地址）和[ebp+var_AC]（_Z6f_funcPhS_S 轮函数的结果）故该函数得到的结果应该是 L1，

```
mov     eax, dword ptr [ebp+var_8C]
mov     dword ptr [ebp+var_8C], eax
mov     eax, [ebp+var_68]
mov     [ebp+var_88], eax
mov     eax, [ebp+var_64]
mov     [ebp+var_84], eax
mov     eax, [ebp+var_60]
mov     [ebp+var_80], eax
mov     eax, [ebp+var_5C]
mov     [ebp+var_7C], eax
mov     eax, [ebp+var_58]
mov     [ebp+var_78], eax
mov     eax, [ebp+var_54]
mov     [ebp+var_74], eax
mov     eax, [ebp+var_50]
mov     [ebp+var_70], eax
mov     eax, dword ptr [ebp+var_AC]
mov     dword ptr [ebp+var_6C], eax
mov     eax, [ebp+var_A8]
mov     [ebp+var_68], eax
mov     eax, [ebp+var_A4]
mov     [ebp+var_64], eax
mov     eax, [ebp+var_A0]
mov     [ebp+var_60], eax
mov     eax, [ebp+var_9C]
mov     [ebp+var_5C], eax
mov     eax, [ebp+var_98]
mov     [ebp+var_58], eax
mov     eax, [ebp+var_94]
mov     [ebp+var_54], eax
mov     eax, [ebp+var_90]
mov     [ebp+var_50], eax
add     [ebp+var_C], 1
```

下面的一系列置换可知为将 R0 移入 L1，完成一轮加密

当[ebp+var_C]=16 时跳转到右边

```
mov     edx, [ebp+var_C]
mov     eax, edx
add     eax, eax
add     eax, edx
shl     eax, 4
add     eax, offset _subkey
mov     [esp+8], eax    ; unsigned __int8 *
lea     eax, [ebp+var_AC]
mov     [esp+4], eax    ; unsigned __int8 *
lea     eax, [ebp+var_6C]
mov     [esp], eax      ; unsigned __int8 *
call    __Z6f_funcPhS_S_ ; f_func(uchar *,uchar *,uchar *)
mov     dword ptr [esp+8], 20h ; ' ' ; int
lea     eax, [ebp+var_AC]
mov     [esp+4], eax    ; unsigned __int8 *
lea     eax, [ebp+var_8C]
mov     [esp], eax      ; unsigned __int8 *
call    __Z7byteXORPhS_i ; byteXOR(uchar *,uchar *,int)
mov     eax, dword ptr [ebp+var_8C]
mov     dword ptr [ebp+var_4C], eax
```

同样先对 R 进行轮函数__Z6f_funcPhS_S_ ，然后用__Z7byteXORPhS_i 函数，得到异或结果，

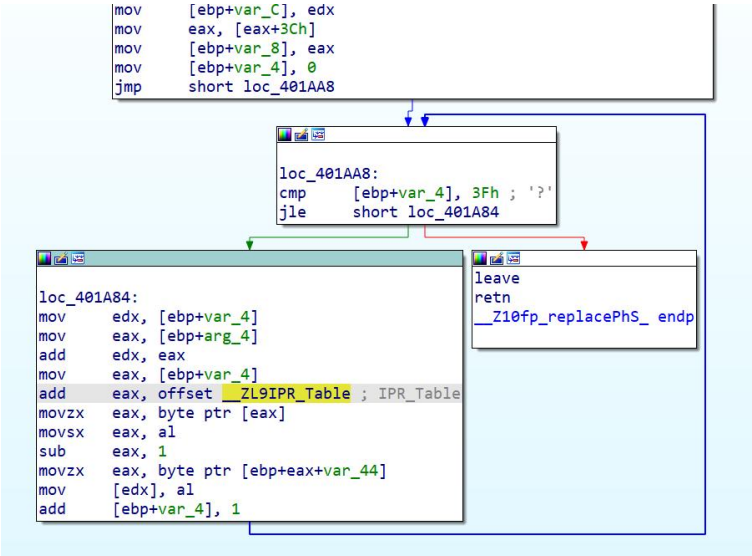
```
mov     eax, dword ptr [ebp+var_8C]
mov     dword ptr [ebp+var_4C], eax
mov     eax, [ebp+var_88]
mov     [ebp+var_48], eax
mov     eax, [ebp+var_84]
mov     [ebp+var_44], eax
mov     eax, [ebp+var_80]
mov     [ebp+var_40], eax
mov     eax, [ebp+var_7C]
mov     [ebp+var_3C], eax
mov     eax, [ebp+var_78]
mov     [ebp+var_38], eax
mov     eax, [ebp+var_74]
mov     [ebp+var_34], eax
mov     eax, [ebp+var_70]
mov     [ebp+var_30], eax
lea     eax, [ebp+var_4C]
add     eax, 20h ; ' '
mov     edx, dword ptr [ebp+var_6C]
mov     [eax], edx
mov     edx, [ebp+var_68]
mov     [eax+4], edx
mov     edx, [ebp+var_64]
mov     [eax+8], edx
mov     edx, [ebp+var_60]
mov     [eax+0Ch], edx
mov     edx, [ebp+var_5C]
mov     [eax+10h], edx
mov     edx, [ebp+var_58]
mov     [eax+14h], edx
mov     edx, [ebp+var_54]
mov     [eax+18h], edx
mov     edx, [ebp+var_50]
mov     [eax+1Ch], edx
lea     eax, [ebp+var_4C]
mov     [esp+4], eax    ; unsigned __int8 *
lea     eax, [ebp+var_4C]
mov     [esp], eax      ; unsigned __int8 *
call    __Z10fp_replacePhS_ ; fp_replace(uchar *,uchar *)
```

但这次将 L15 移入 L16，以及将运算结果移入 R16，没有经过交换，

```
lea     eax, [ebp+var_4C]
mov     [esp+4], eax    ; unsigned __int8 *
lea     eax, [ebp+var_4C]
mov     [esp], eax      ; unsigned __int8 *
call    __Z10fp_replacePhS_ ; fp_replace(uchar *,uchar *)
mov     dword ptr [esp+8], 8 ; int
mov     eax, [ebp+arg_4]
mov     [esp+4], eax    ; unsigned __int8 *
lea     eax, [ebp+var_4C]
mov     [esp], eax      ; unsigned __int8 *
call    __Z8bit2BytePhS_i ; bit2Byte(uchar *,uchar *,int)
leave
retn
__Z10encryptionPhS_ endp
```

然后调用__Z10fp_replacePhS_函数，可猜测是完成 IP 逆置换，

可以进入函数，



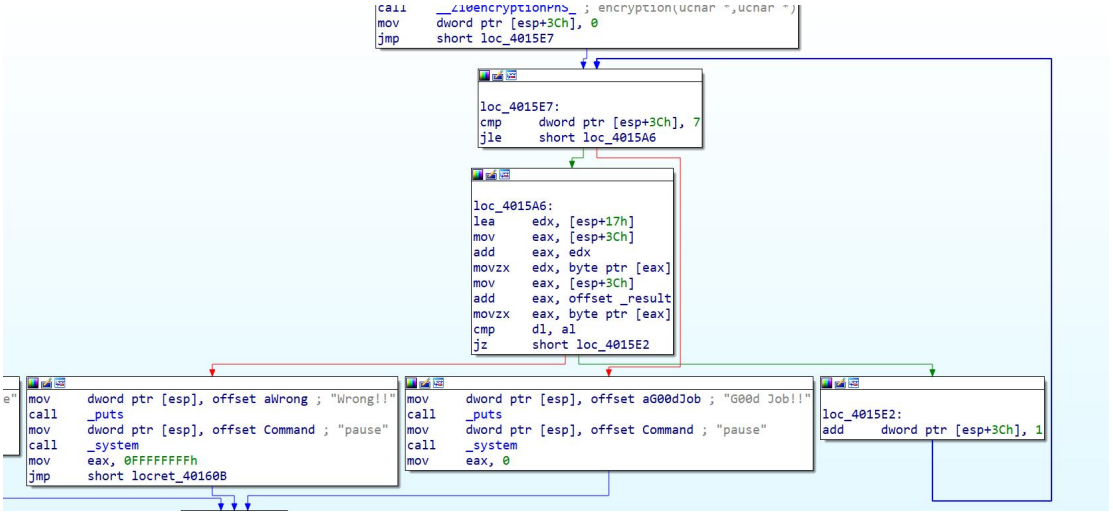
发现与 IP 置换函数相似的结构，点击查看 offset __ZL9IPR_Table，

```
.rdata:00405080 ; IPR_Table
.rdata:00405080 __ZL9IPR_Table  db  28h ; (
.rdata:00405081                 db   8
.rdata:00405082                 db  30h ; 0
.rdata:00405083                 db  10h
.rdata:00405084                 db  38h ; 8
.rdata:00405085                 db  18h
.rdata:00405086                 db  40h ; @
.rdata:00405087                 db  20h
.rdata:00405088                 db  27h ; '
.rdata:00405089                 db   7
.rdata:0040508A                 db  2Fh ; /
.rdata:0040508B                 db  0Fh
.rdata:0040508C                 db  37h ; 7
.rdata:0040508D                 db  17h
.rdata:0040508E                 db  3Fh ; ?
.rdata:0040508F                 db  1Fh
.rdata:00405090                 db  26h ; &
.rdata:00405091                 db   6
.rdata:00405092                 db  2Eh ; .
.rdata:00405093                 db  0Eh
.rdata:00405094                 db  36h ; 6
.rdata:00405095                 db  16h
.rdata:00405096                 db  3Eh ; >
.rdata:00405097                 db  1Eh
.rdata:00405098                 db  25h ; %
.rdata:00405099                 db   5
.rdata:0040509A                 db  2Dh ; -
.rdata:0040509B                 db  0Dh
.rdata:0040509C                 db  35h ; 5
.rdata:0040509D                 db  15h
.rdata:0040509E                 db  3Dh ; =
.rdata:0040509F                 db  1Dh
.rdata:004050A0                 db  24h ; $
```

找到 IP 逆置换表，故__Z10fp_replacePhS_函数是 IP 逆置换函数，

最后调用__Z8bit2BytePhS_i 函数，将 8 字节转换为 64 位二进制数，

存储到[ebp+arg4]（最后要存储的地址）中完成加密，然后回到 main 函数，

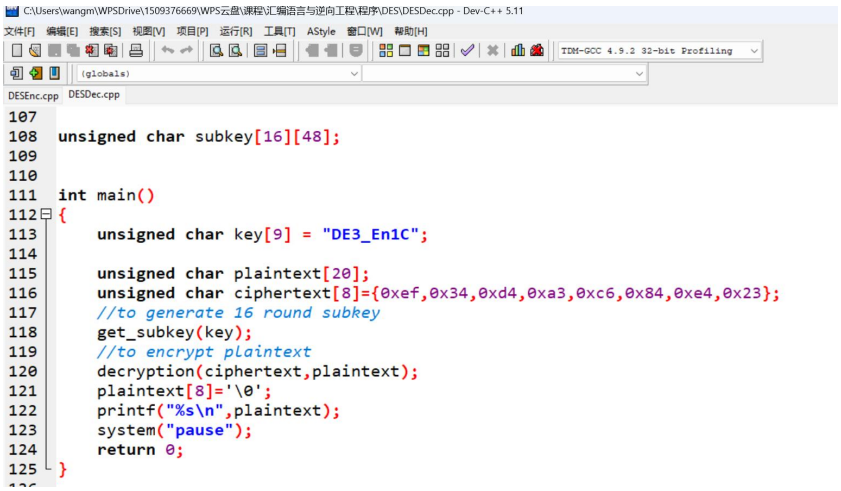


把[esp+3Ch]置零，然后和 7 比较，一位一位和密文比较，若错误则输出“Wrong!!”，正确则循环到 7，然后跳转到输出“G00d Job!!”

再中间那个块，我们可以得到密文存放的位置是_result，点进去得到密文是“EFh, 34h, D4h, A3h, C6h, 84h, E4h, 23h”

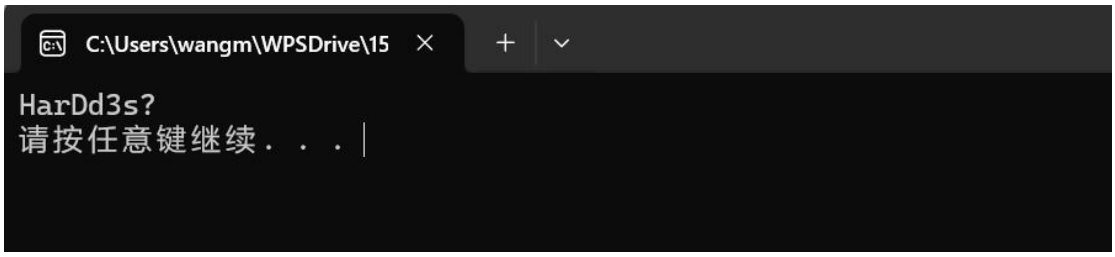
运行 DESDec. cpp，将要解密的数组替换成

{0xef, 0x34, 0xd4, 0xa3, 0xc6, 0x84, 0xe4, 0x23}

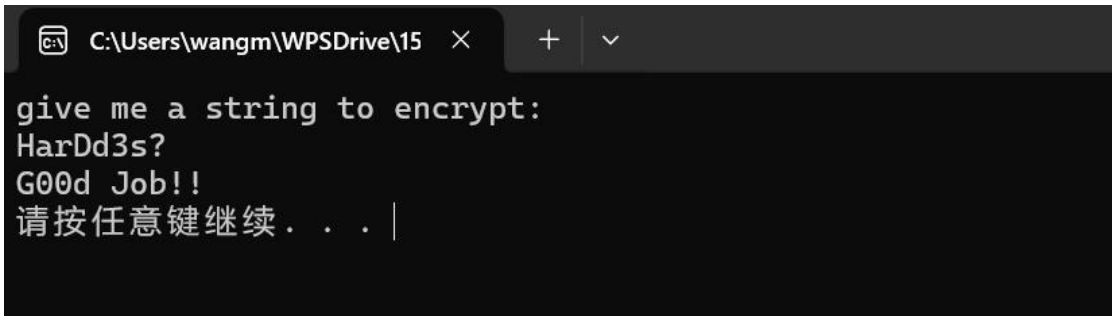


运行，得到解密结果为

HarDd3s?



结我们运行 DESEnc. exe, 验证一下



输出“G00d Job!!”，结果正确，完成 DES 加密算法的逆向分析。