北京郵電大學

Beijing University of Posts and Telecommunications

SHA256 分析报告

题　　目	SHA256 分析
姓　　名	[redacted]
学　　号	
班　　级	
指导教师	付俊松

目录

SHA256 分析

Main 函数

```
push    ebp
mov     ebp, esp
sub     esp, 4B8h
push    ebx
push    esi
push    edi
lea     edi, [ebp+var_4B8]
mov     ecx, 12Eh
mov     eax, 0CCCCCCCCh
rep stosd
mov     [ebp+first], 0
mov     ecx, 10h
xor     eax, eax
lea     edi, [ebp+var_43]
rep stosd
mov     [ebp+Str], 0
mov     ecx, 0FFh
xor     eax, eax
lea     edi, [ebp+var_443]
rep stosd
stosw
stosb
mov     [ebp+Dest], 0
xor     eax, eax
```

首先进入 main 函数，first、Str、Dest 都置零。

```
mov     [ebp+var_456], 2
mov     [ebp+var_455], 4
push    offset Format   ; "Please input your flag:\n"
call    _printf
add     esp, 4
lea     ecx, [ebp+Str]
push    ecx
push    offset aS       ; "%s"
call    _scanf
add     esp, 8
lea     edx, [ebp+Str]
push    edx             ; Str
call    _strlen
add     esp, 4
mov     [ebp+var_478], eax
cmp     [ebp+var_478], 5
jnb     short loc_401236
```

Str 保存输入的字符串。字符串长度保存在 eax。

```
call    _strlen
add     esp, 4
mov     [ebp+var_478], eax
cmp     [ebp+var_478], 5
jnb     short loc_401236
```

长度>=5 时 cf = 0，则跳转。我们可以看出输入的 flag 应该大于 5。否则直接 wrong。长度大于 5 跳转到 loc_401236。

```
loc_401236:             ; Count
push    4
lea     eax, [ebp+Str]
push    eax             ; Source
lea     ecx, [ebp+Dest]
push    ecx             ; Dest
call    _strncpy
add     esp, 0Ch
lea     edx, [ebp+Dest]
push    edx             ; Str
call    _strlen
add     esp, 4
push    eax             ; unsigned int
lea     eax, [ebp+Dest]
push    eax             ; char *
lea     ecx, [ebp+first]
push    ecx             ; Dest
call    j_?sha_calc@@YAXPADPBDI@Z ; sha_calc(char *,char const *,uint)
add     esp, 0Ch
mov     [ebp+var_450], 0
jmp     short loc_40128C
```

复制 str 到 dest，当 str 的长度小于 4 时，dest 的剩余部分将用空字节填充。eax 保存输入字符串长度，push dest、字符串长度，first。猜测 first 应该时保存八个初始哈希值，以及 64 个常量，进行哈希加密。Call sha_calc 函数。

sha_calc

```
rep stosd
push    40h ; '@'       ; Size
call    _malloc
add     esp, 4
mov     [ebp+Memory], eax
lea     eax, [ebp+var_24]
push    eax             ; char *
lea     ecx, [ebp+var_20]
push    ecx             ; char *
lea     edx, [ebp+var_1C]
push    edx             ; char *
lea     eax, [ebp+var_18]
push    eax             ; char *
lea     ecx, [ebp+var_14]
push    ecx             ; char *
lea     edx, [ebp+var_10]
push    edx             ; char *
lea     eax, [ebp+var_C]
push    eax             ; char *
lea     ecx, [ebp+dst]
push    ecx             ; dst
call    j_?sha_init@@YAXPAI0000000@Z ; sha_init(uint *,uint *,uint *,uint *,uint *,uint *,uint *,uint *)
```

进入 shacalc 函数，申请 40 字节大小的空间，并把首地址传给[ebp+Memory]。然后传入七个参数，以及字符串 dest。然后 call sha_init 函数。

sha_init

```
mov     [ebp+src], 67h ; 'g'
mov     [ebp+var_1F], 0E6h
mov     [ebp+var_1E], 9
mov     [ebp+var_1D], 6Ah ; 'j'
mov     [ebp+var_1C], 85h
mov     [ebp+var_1B], 0AEh
mov     [ebp+var_1A], 67h ; 'g'
mov     [ebp+var_19], 0BBh
mov     [ebp+var_18], 72h ; 'r'
mov     [ebp+var_17], 0F3h
mov     [ebp+var_16], 6Eh ; 'n'
mov     [ebp+var_15], 3Ch ; '<'
mov     [ebp+var_14], 3Ah ; ':'
mov     [ebp+var_13], 0F5h
mov     [ebp+var_12], 4Fh ; 'O'
mov     [ebp+var_11], 0A5h
mov     [ebp+var_10], 7Fh
mov     [ebp+var_F], 52h ; 'R'
mov     [ebp+var_E], 0Eh
mov     [ebp+var_D], 51h ; 'Q'
mov     [ebp+var_C], 8Ch
mov     [ebp+var_B], 68h ; 'h'
mov     [ebp+var_A], 5
mov     [ebp+var_9], 9Bh
mov     [ebp+var_8], 0ABh
mov     [ebp+var_7], 0D9h
mov     [ebp+var_6], 83h
mov     [ebp+var_5], 1Fh
mov     [ebp+var_4], 19h
mov     [ebp+var_3], 0CDh
mov     [ebp+var_2], 0E0h
mov     [ebp+var_1], 5Bh ; '['
push    4               ; count
lea     eax, [ebp+src]
push    eax             ; src
mov     ecx, [ebp+dst]
push    ecx             ; dst
call    _memcpy
```

```
11
12  src[0] = 103;
13  src[1] = -26;
14  src[2] = 9;
15  src[3] = 106;
16  v9[0] = -123;
17  v9[1] = -82;
18  v9[2] = 103;
19  v9[3] = -69;
20  v10[0] = 114;
21  v10[1] = -13;
22  v10[2] = 110;
23  v10[3] = 60;
24  v11[0] = 58;
25  v11[1] = -11;
26  v11[2] = 79;
27  v11[3] = -91;
28  v12[0] = 127;
29  v12[1] = 82;
30  v12[2] = 14;
31  v12[3] = 81;
32  v13[0] = -116;
33  v13[1] = 104;
34  v13[2] = 5;
35  v13[3] = -101;
36  v14[0] = -85;
37  v14[1] = -39;
38  v14[2] = -125;
39  v14[3] = 31;
40  v15[0] = 25;
41  v15[1] = -51;
42  v15[2] = -32;
43  v15[3] = 91;
44  memcpy(dst, src, 4u);
45  memcpy(a2, v9, 4u);
46  memcpy(a3, v10, 4u);
47  memcpy(a4, v11, 4u);
48  memcpy(a5, v12, 4u);
49  memcpy(a6, v13, 4u);
50  memcpy(a7, v14, 4u);
51  memcpy(a8, v15, 4u);
52 }
```

进入 sha_init 函数，F5 查看函数，将 67 赋值给 src，对 32 个变量赋值。即传入 8 个哈希初值。32 个变量，4 个变量保存一个初始哈希值。然后 push 4，src，dst，call memcpy，即：将 src 的 4 个字节（h0）复制到 dst，依次把哈希值的初值 h1-h7 复制到传入的七个参数。所以我们可以知道这个函数时将八个哈希初始值初始化。然后返回 sha_calc 函数。

```
call    j_?sha_init@@YAXPAI0000000@Z ; sha_init(uint *,uint *,uint *,uint *,uint *,uint *,uint *,uint *)
add     esp, 20h
mov     edx, [ebp+count]
push    edx             ; count
mov     eax, [ebp+src]
push    eax             ; src
lea     ecx, [ebp+var_2C]
push    ecx             ; unsigned __int8 **
call    j_?sha_update@@YAIPAPAEPBDI@Z ; sha_update(uchar * *,char const *,uint)
```

Push src 就是输入的字符串，count 字符串长度，var_2C 保存返回的结果。Call sha_update 函数

sha_update

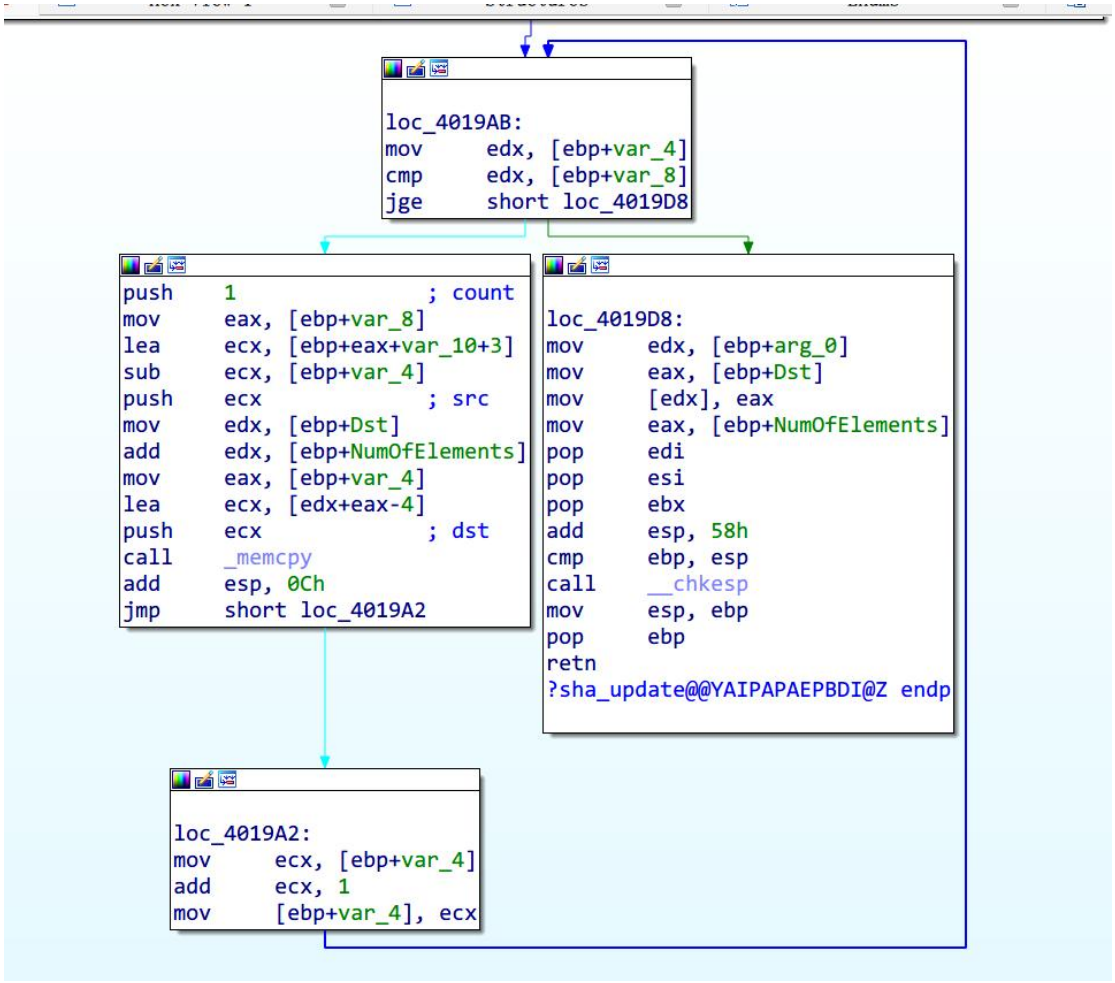
```
push    ebp
mov     ebp, esp
sub     esp, 58h
push    ebx
push    esi
push    edi
lea     edi, [ebp+var_58]
mov     ecx, 16h
mov     eax, 0CCCCCCCCh
rep stosd
mov     eax, [ebp+count]
shl     eax, 3
mov     [ebp+var_C], eax
mov     eax, [ebp+count]
xor     edx, edx
mov     ecx, 40h ; '@'
div     ecx
mov     eax, 38h ; '8'
sub     eax, edx
mov     [ebp+var_10], eax
mov     ecx, [ebp+var_10]
mov     edx, [ebp+count]
lea     eax, [edx+ecx+8]
mov     [ebp+NumOfElements], eax
push    1               ; SizeOfElements
mov     ecx, [ebp+NumOfElements]
push    ecx             ; NumOfElements
call    _calloc
```

进入 sha_update，eax 保存字符串长度，把字符串的长度 count*8 就是字符串的 bit 数保存到[ebp+var_c],用 edx 保存字符串长度除以 64 的余数。将 eax 赋值为 56，然后给 eax 赋值为 56-edx。eax = 56-字符串长度模 64 的余数。Edx 为字符串长度 L。eax 赋值为 edx+ecx+8。然后请求 eax 个大小为 1 字节的空间。[ebp+var_10]保存 56-字符串长度模 64 的余数

```
call    _calloc
add     esp, 8
mov     [ebp+Dst], eax
push    4               ; Size
push    0               ; Val
mov     edx, [ebp+Dst]
push    edx             ; Dst
call    _memset
add     esp, 0Ch
mov     eax, [ebp+count]
push    eax             ; count
mov     ecx, [ebp+src]
push    ecx             ; src
mov     edx, [ebp+Dst]
push    edx             ; dst
call    _memcpy
add     esp, 0Ch
mov     eax, [ebp+Dst]
add     eax, [ebp+count]
mov     byte ptr [eax], 80h ; '€'
mov     [ebp+var_8], 4
mov     [ebp+var_4], 0
jmp     short loc_4019AB
```

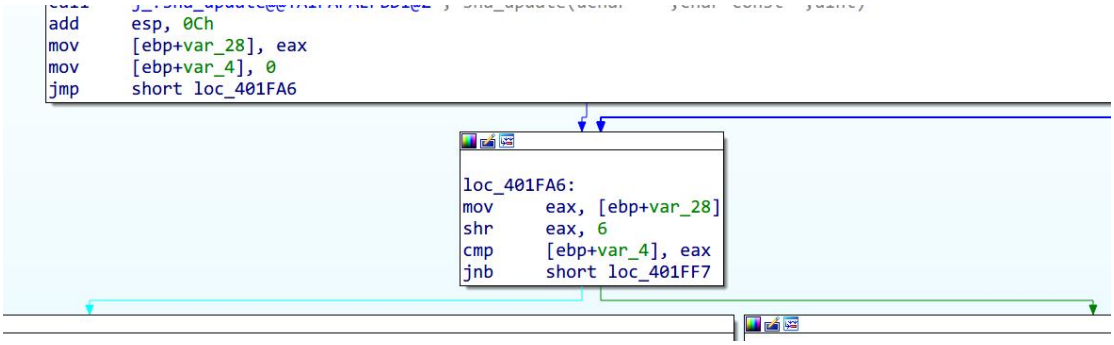
将申请空间的首地址赋值给 Dst. 然后 push 4、0、Dst。Call memset 函数。Mov byte ptr [eax] 80h,将字符串的 bit 位后补个 1。

复制字符 src（一个无符号字符）到参数 Dst 所指向的字符串的前 4 个字符。（填充 0）。从存储区 **src** 复制 **count（字符串长度）**个字节到存储区 **Dst**。然后将 80h 赋值给存储区的[Dst+count]的位置。

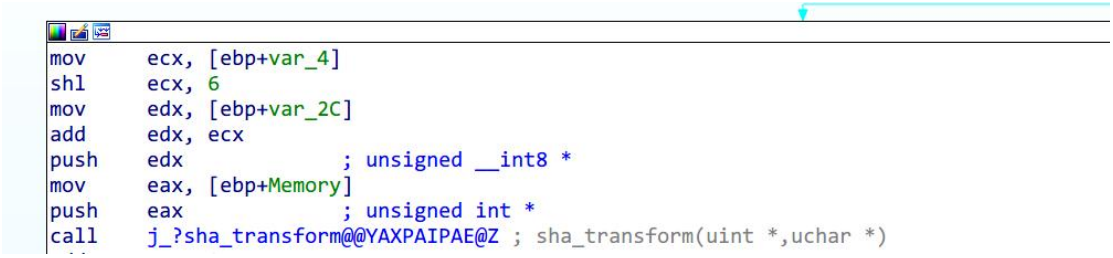



```
1 void *__cdecl sha_update(unsigned __int8 **a1, const char *src, unsigned int count)
2 {
3   void *v4[22]; // [esp+Ch] [ebp-58h] BYREF
4
5   memset(v4, 0xCCu, sizeof(v4));
6   v4[19] = (void *)(8 * count);
7   v4[18] = (void *)(56 - count % 0x40);
8   v4[17] = (char *)v4[18] + count + 8;
9   v4[16] = calloc((size_t)v4[17], 1u);
10  memset(v4[16], 0, 4u);
11  memcpy((char *)v4[16], (char *)src, count);
12  *((_BYTE *)v4[16] + count) = 0x80;
13  v4[20] = (void *)4;
14  for ( v4[21] = 0; (int)v4[21] < (int)v4[20]; ++v4[21] )
15    memcpy(
16      (char *)v4[17] + (unsigned int)v4[16] + (unsigned int)v4[21] - 4,
17      (char *)((char *)&v4[18] + (unsigned int)v4[20] + 3 - (char *)v4[21]),
18      sizeof(char));
19  *a1 = (unsigned __int8 *)v4[16];
20  return v4[17];
21 }
```

循环四次，依次复制一个字节一个字节将[ebp+var_10+3-var_4+var_8]复制到[ebp+Dst+NumofElement+var_4-4]，然后将 Dst 的地址复制到传入的变量 argc_0 存储空间中。然后将 NumofElement 赋值给 eax。（即 NumofElement 保存加密的块数，有多少个 512bit 数据）。返回分组好的数据首地址。返回 sha_caloc 函数。Var_2C 保存预处理后的数据。

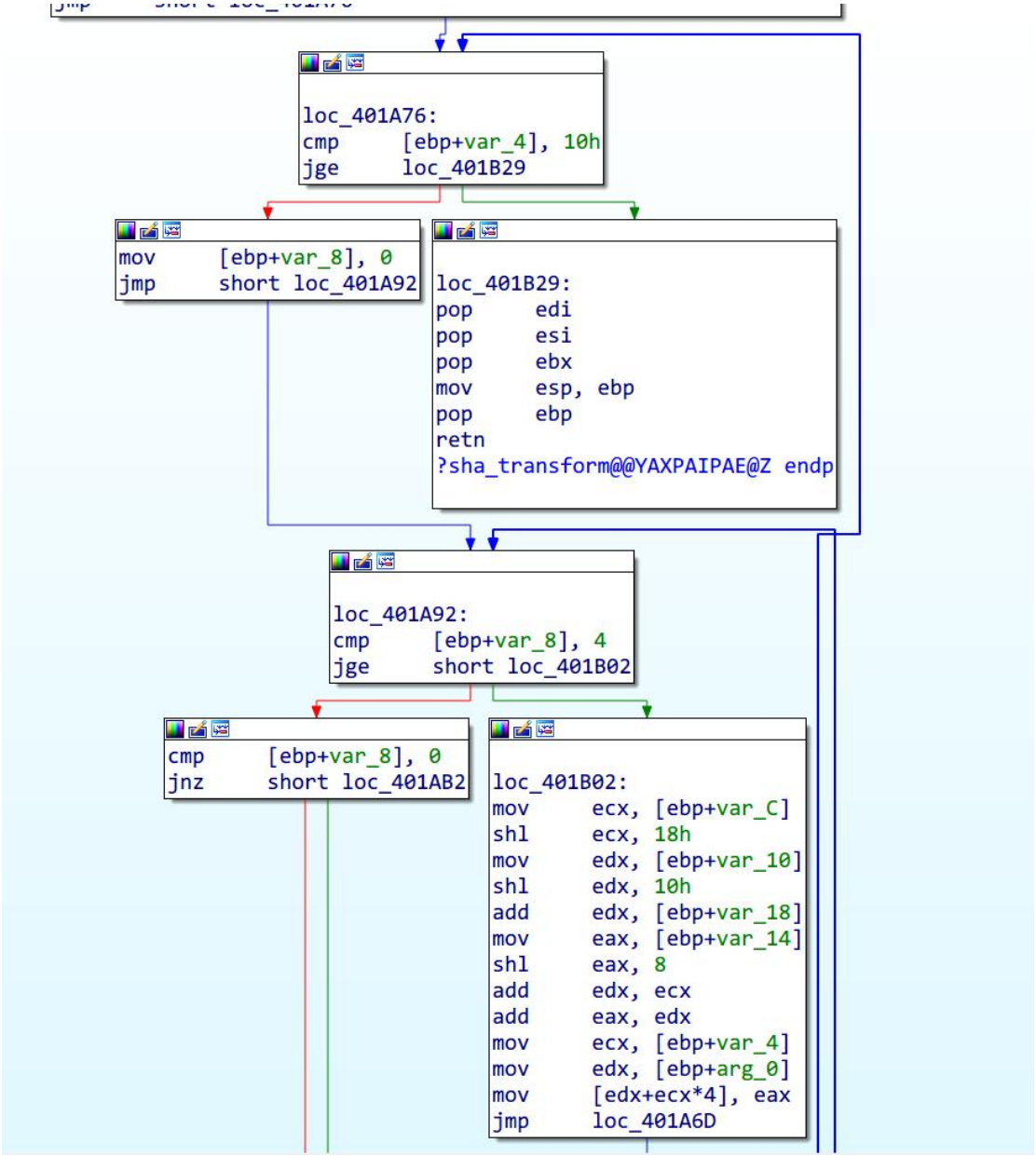


[ebp+var_28]记录的是有字符串填充后多少个字节。[ebp+var_4]置零，做循环。将[ebp+var_28]赋值给 eax，右移 6 位，就是/64。所以就是把字符串填充后多少个字节除以 64（）分为每 521bit 一块。所以这个循环要循环的 512bit 块数。[ebp+var_4] 小于 eax，所以不跳转。

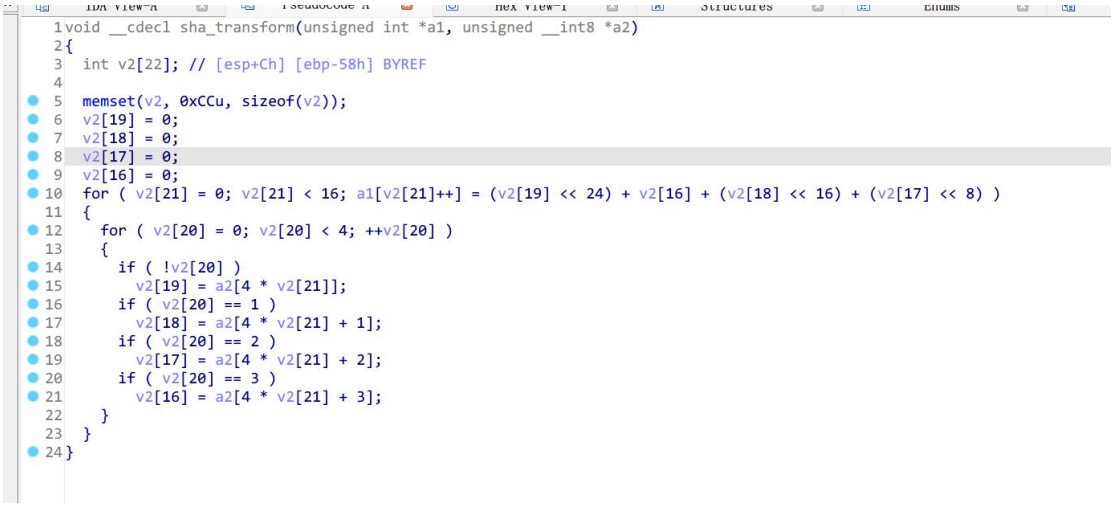


将变量[ebp+var_4]左移 64 位，就是*64，移动对要对应操作的 512bit 的位置。push 预处理的 512bit（64 字节）消息摘要+对应循环要处理的第 var_4 块。以及请求分配 64 字节资源的首地址[ebp+memory]，然后进入 sha_transform 函数。

sha_transform

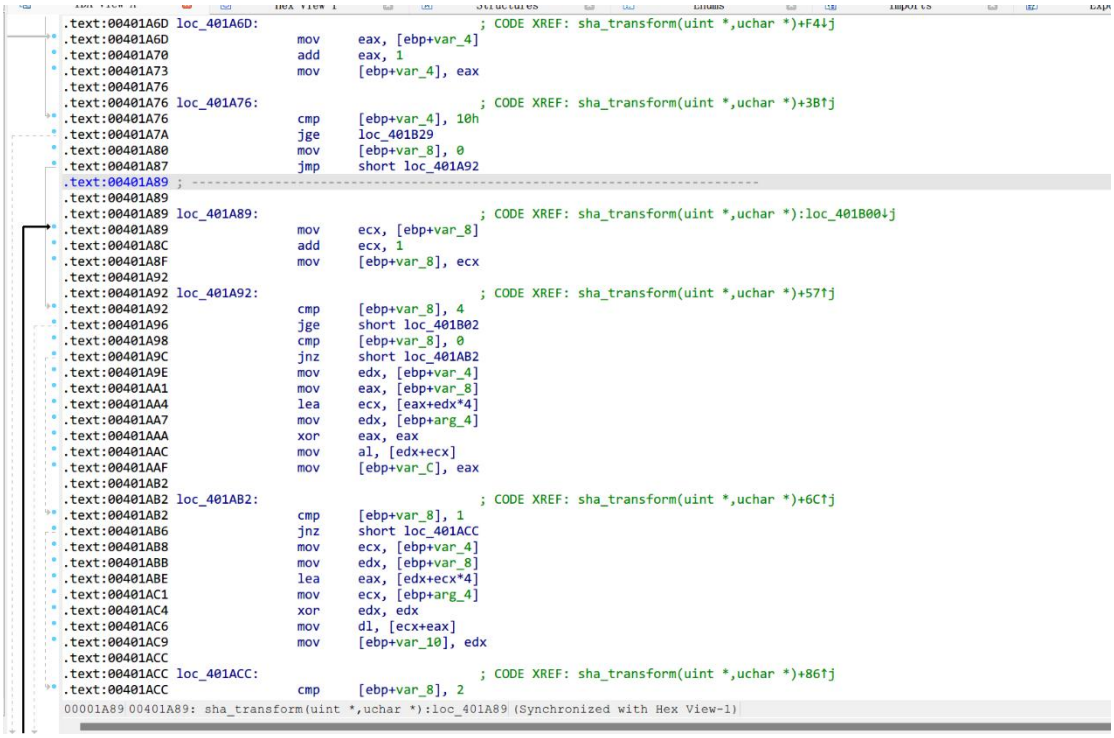


进入 sha_transfrom 函数，可以看出有一个 16*4 的循环，应该就是把分组好的 512bit（64 字节）的块，再分成 16 块，每块 4 字节（32bit）放入[ebp+memory]中。

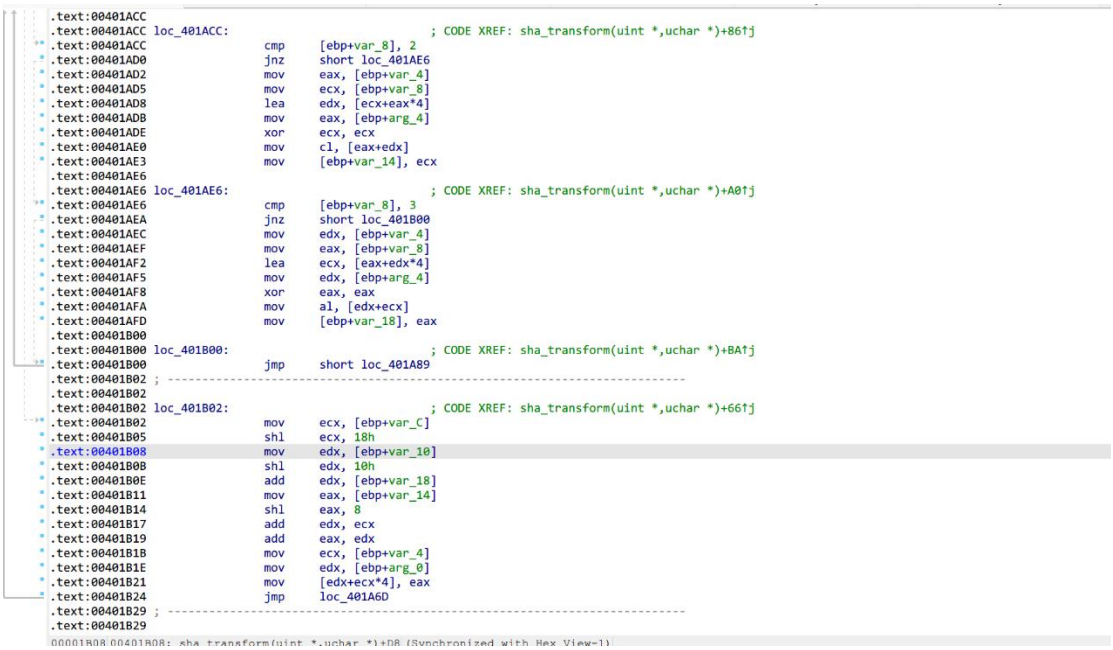


```
1void __cdecl sha_transform(unsigned int *a1, unsigned __int8 *a2)
2{
3  int v2[22]; // [esp+Ch] [ebp-58h] BYREF
4
5  memset(v2, 0xCCu, sizeof(v2));
6  v2[19] = 0;
7  v2[18] = 0;
8  v2[17] = 0;
9  v2[16] = 0;
10 for ( v2[21] = 0; v2[21] < 16; a1[v2[21]++] = (v2[19] << 24) + v2[16] + (v2[18] << 16) + (v2[17] << 8) )
11 {
12   for ( v2[20] = 0; v2[20] < 4; ++v2[20] )
13   {
14     if ( !v2[20] )
15       v2[19] = a2[4 * v2[21]];
16     if ( v2[20] == 1 )
17       v2[18] = a2[4 * v2[21] + 1];
18     if ( v2[20] == 2 )
19       v2[17] = a2[4 * v2[21] + 2];
20     if ( v2[20] == 3 )
21       v2[16] = a2[4 * v2[21] + 3];
22   }
23 }
24}
```

F5 查看函数，发现确实如此，16 的大循环还有 4 的小循环。将处理好的快块逐字节复制到 v2 中，再把 4 字节（32bit），4 字节 4 字节的赋值给 a1（就是[ebp+momery]中）。

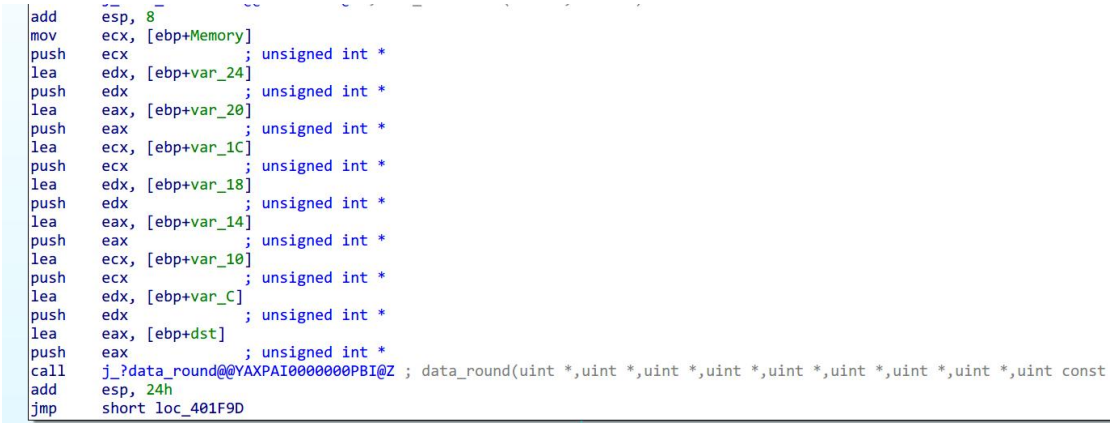


```
.text:00401A6D loc_401A6D:                             ; CODE XREF: sha_transform(uint *,uchar *)+F4↓j
.text:00401A6D                 mov     eax, [ebp+var_4]
.text:00401A70                 add     eax, 1
.text:00401A73                 mov     [ebp+var_4], eax
.text:00401A76
.text:00401A76 loc_401A76:                             ; CODE XREF: sha_transform(uint *,uchar *)+3B↑j
.text:00401A76                 cmp     [ebp+var_4], 10h
.text:00401A7A                 jge     loc_401B29
.text:00401A80                 mov     [ebp+var_8], 0
.text:00401A87                 jmp     short loc_401A92
.text:00401A89 ; ---------------------------------------------------------------------------
.text:00401A89
.text:00401A89 loc_401A89:                             ; CODE XREF: sha_transform(uint *,uchar *):loc_401B00↓j
.text:00401A89                 mov     ecx, [ebp+var_8]
.text:00401A8C                 add     ecx, 1
.text:00401A8F                 mov     [ebp+var_8], ecx
.text:00401A92
.text:00401A92 loc_401A92:                             ; CODE XREF: sha_transform(uint *,uchar *)+57↑j
.text:00401A92                 cmp     [ebp+var_8], 4
.text:00401A96                 jge     short loc_401B02
.text:00401A98                 cmp     [ebp+var_8], 0
.text:00401A9C                 jnz     short loc_401AB2
.text:00401A9E                 mov     edx, [ebp+var_4]
.text:00401AA1                 mov     eax, [ebp+var_8]
.text:00401AA4                 lea     ecx, [eax+edx*4]
.text:00401AA7                 mov     edx, [ebp+arg_4]
.text:00401AAA                 xor     eax, eax
.text:00401AAC                 mov     al, [edx+ecx]
.text:00401AAF                 mov     [ebp+var_C], eax
.text:00401AB2
.text:00401AB2 loc_401AB2:                             ; CODE XREF: sha_transform(uint *,uchar *)+6C↑j
.text:00401AB2                 cmp     [ebp+var_8], 1
.text:00401AB6                 jnz     short loc_401ACC
.text:00401AB8                 mov     ecx, [ebp+var_4]
.text:00401ABB                 mov     edx, [ebp+var_8]
.text:00401ABE                 lea     eax, [edx+ecx*4]
.text:00401AC1                 mov     ecx, [ebp+arg_4]
.text:00401AC4                 xor     edx, edx
.text:00401AC6                 mov     dl, [ecx+eax]
.text:00401AC9                 mov     [ebp+var_10], edx
.text:00401ACC
.text:00401ACC loc_401ACC:                             ; CODE XREF: sha_transform(uint *,uchar *)+86↑j
.text:00401ACC                 cmp     [ebp+var_8], 2
00001A89 00401A89: sha_transform(uint *,uchar *):loc_401A89 (Synchronized with Hex View-1)
```



```
.text:00401ACC
.text:00401ACC loc_401ACC:                             ; CODE XREF: sha_transform(uint *,uchar *)+86↑j
.text:00401ACC                 cmp     [ebp+var_8], 2
.text:00401AD0                 jnz     short loc_401AE6
.text:00401AD2                 mov     eax, [ebp+var_4]
.text:00401AD5                 mov     ecx, [ebp+var_8]
.text:00401AD8                 lea     edx, [ecx+eax*4]
.text:00401ADB                 mov     eax, [ebp+arg_4]
.text:00401ADE                 xor     ecx, ecx
.text:00401AE0                 mov     cl, [eax+edx]
.text:00401AE3                 mov     [ebp+var_14], ecx
.text:00401AE6
.text:00401AE6 loc_401AE6:                             ; CODE XREF: sha_transform(uint *,uchar *)+A0↑j
.text:00401AE6                 cmp     [ebp+var_8], 3
.text:00401AEA                 jnz     short loc_401B00
.text:00401AEC                 mov     edx, [ebp+var_4]
.text:00401AEF                 mov     eax, [ebp+var_8]
.text:00401AF2                 lea     ecx, [eax+edx*4]
.text:00401AF5                 mov     edx, [ebp+arg_4]
.text:00401AF8                 xor     eax, eax
.text:00401AFA                 mov     al, [edx+ecx]
.text:00401AFD                 mov     [ebp+var_18], eax
.text:00401B00
.text:00401B00 loc_401B00:                             ; CODE XREF: sha_transform(uint *,uchar *)+BA↑j
.text:00401B00                 jmp     short loc_401A89
.text:00401B02 ; ---------------------------------------------------------------------------
.text:00401B02
.text:00401B02 loc_401B02:                             ; CODE XREF: sha_transform(uint *,uchar *)+66↑j
.text:00401B02                 mov     ecx, [ebp+var_C]
.text:00401B05                 shl     ecx, 18h
.text:00401B08                 mov     edx, [ebp+var_10]
.text:00401B0B                 shl     edx, 10h
.text:00401B0E                 add     edx, [ebp+var_18]
.text:00401B11                 mov     eax, [ebp+var_14]
.text:00401B14                 shl     eax, 8
.text:00401B17                 add     edx, ecx
.text:00401B19                 add     eax, edx
.text:00401B1B                 mov     ecx, [ebp+var_4]
.text:00401B1E                 mov     edx, [ebp+arg_0]
.text:00401B21                 mov     [edx+ecx*4], eax
.text:00401B24                 jmp     loc_401A6D
.text:00401B29 ; ---------------------------------------------------------------------------
.text:00401B29
00001B08 00401B08: sha_transform(uint *,uchar *)+D8 (Synchronized with Hex View-1)
```

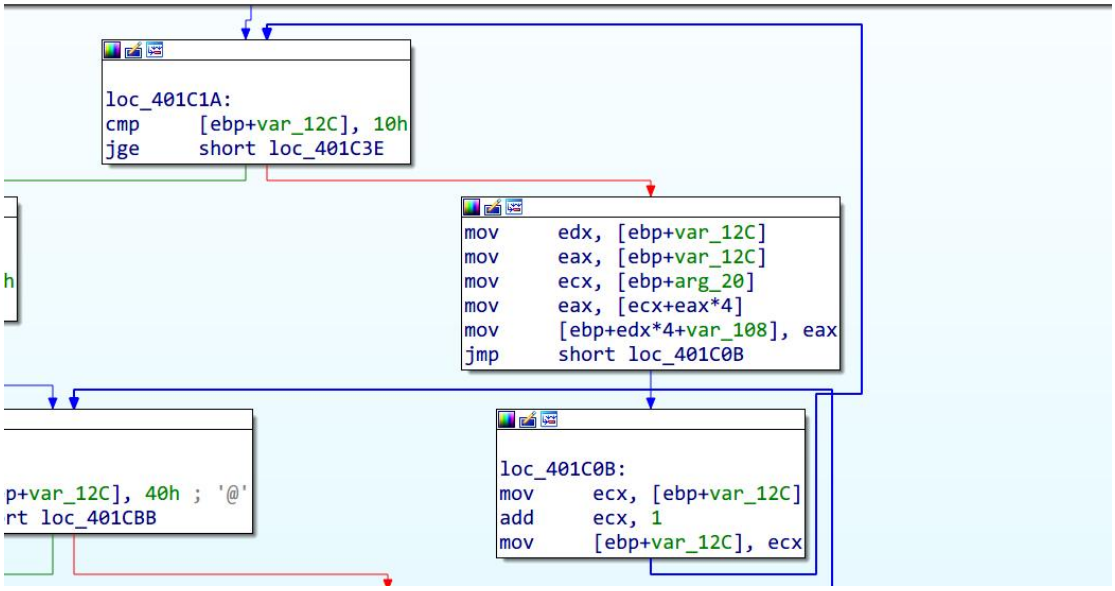
从汇编中看得出来，也确实如此。处理前 16 个字后，返回 sha_caloch 函数。



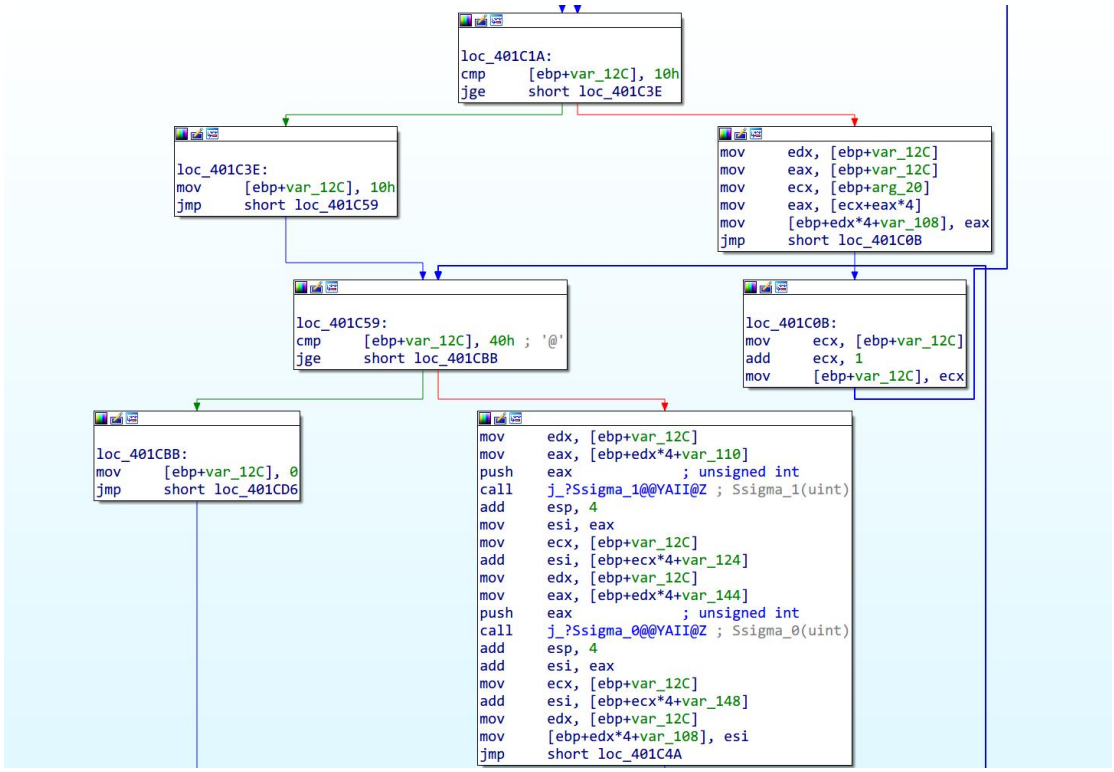
Push 处理好的保存 16 个字的[ebp+memory],再 push8 个 32 位的哈希初始值。Call data_round 函数。

data_round

取出 8 个哈希值放到 8 个变量中，并把[ebp+var_12C]置零

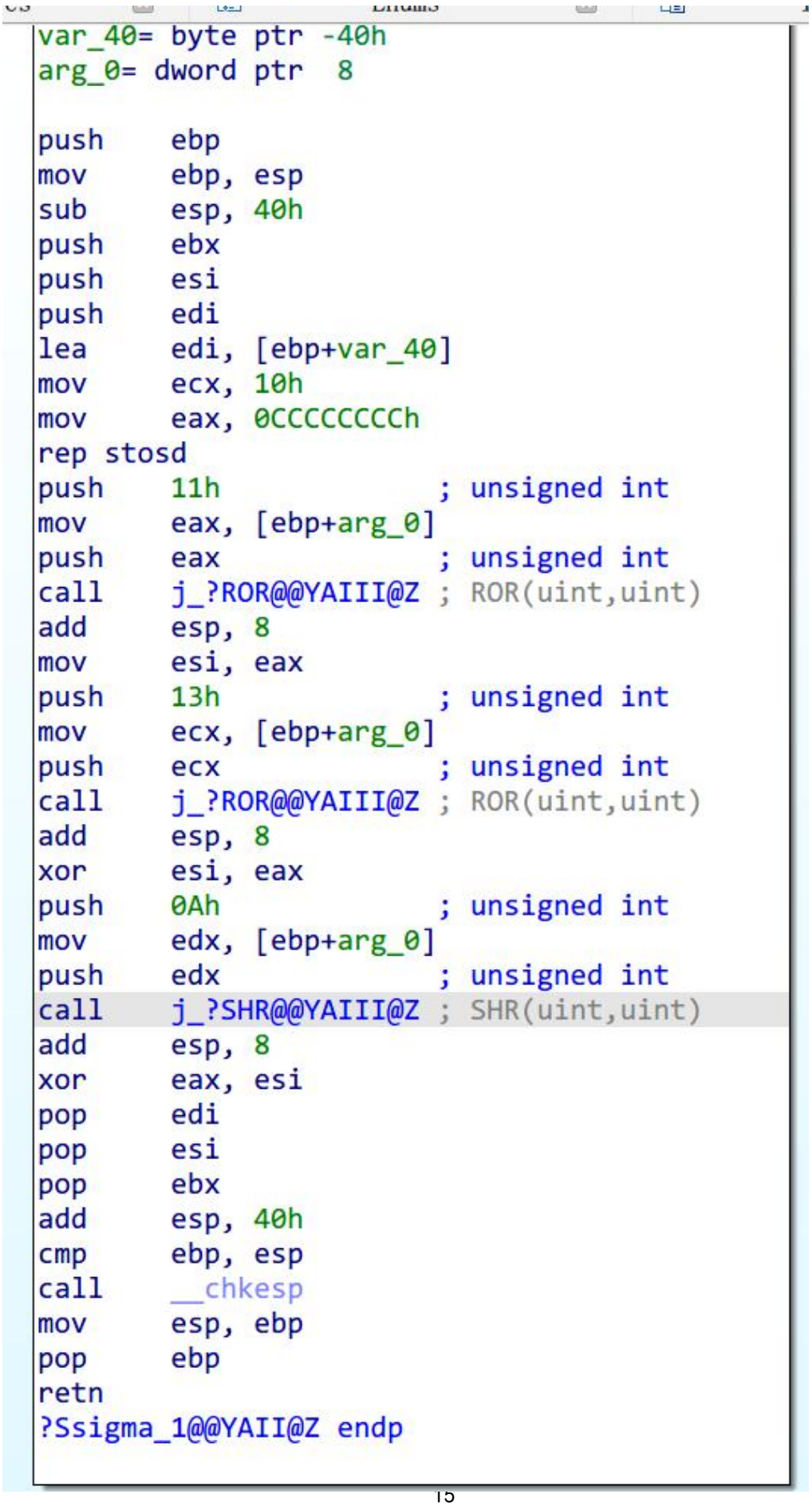


令 i = var_12C 先前已经将[ebp+var_108]置零。然后将[ebp+arg_20+4*i] 传入[ebp+var_108 +4*i]，即将之前的预处理后的 16 个字（16*32bit）传到依次[ebp+var_108 +4*i]。循环 16 次完成。



然后跳转到 loc_401c3e,然后给 var_12C 赋值 16，进入 loc_401c59。循环 64-16 次，所以就是生成剩下的 w[16]-w[63]。Push [ebp+4*i+var_110],实际上就是上一轮[ebp+var_108 +4*i]的前两个 4 字节（就是 W[i-2]）。然后 call Ssigma_1 函数。

Ssigma_1



```
var_40= byte ptr -40h
arg_0= dword ptr  8

push    ebp
mov     ebp, esp
sub     esp, 40h
push    ebx
push    esi
push    edi
lea     edi, [ebp+var_40]
mov     ecx, 10h
mov     eax, 0CCCCCCCCh
rep stosd
push    11h             ; unsigned int
mov     eax, [ebp+arg_0]
push    eax             ; unsigned int
call    j_?ROR@@YAIII@Z ; ROR(uint,uint)
add     esp, 8
mov     esi, eax
push    13h             ; unsigned int
mov     ecx, [ebp+arg_0]
push    ecx             ; unsigned int
call    j_?ROR@@YAIII@Z ; ROR(uint,uint)
add     esp, 8
xor     esi, eax
push    0Ah             ; unsigned int
mov     edx, [ebp+arg_0]
push    edx             ; unsigned int
call    j_?SHR@@YAIII@Z ; SHR(uint,uint)
add     esp, 8
xor     eax, esi
pop     edi
pop     esi
pop     ebx
add     esp, 40h
cmp     ebp, esp
call    __chkesp
mov     esp, ebp
pop     ebp
retn
?Ssigma_1@@YAII@Z endp
```

进入 Ssigma_1 函数，push 11h（17）、传入的 w[i-12] 调用 ROR 函数。

ROR

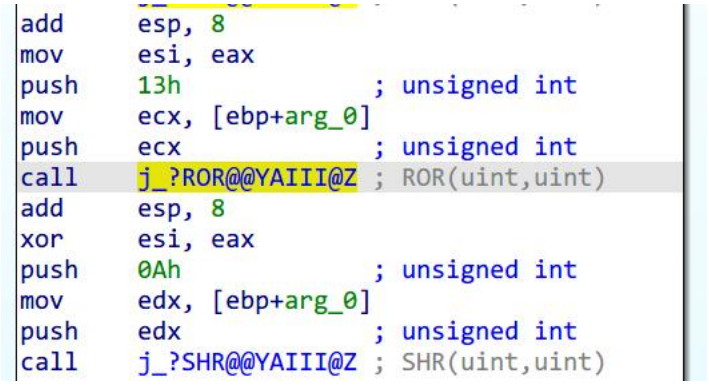
```
; Attributes: bp-based frame

; unsigned int __cdecl ROR(unsigned int, unsigned int)
?ROR@@YAIII@Z proc near

var_40= byte ptr -40h
arg_0= dword ptr  8
arg_4= dword ptr  0Ch

push    ebp
mov     ebp, esp
sub     esp, 40h
push    ebx
push    esi
push    edi
lea     edi, [ebp+var_40]
mov     ecx, 10h
mov     eax, 0CCCCCCCCh
rep stosd
mov     eax, [ebp+arg_0]
mov     ecx, [ebp+arg_4]
shr     eax, cl
mov     ecx, 20h ; ' '
sub     ecx, [ebp+arg_4]
mov     edx, [ebp+arg_0]
shl     edx, cl
or      eax, edx
pop     edi
pop     esi
pop     ebx
mov     esp, ebp
pop     ebp
retn
?ROR@@YAIII@Z endp
```

将 w[i-2]右移 17 位保存到 eax，然后用 32-17=15 赋值给 ecx。然后将 w[i-2]左移 15 位保存到 edx，然后 edx 与 eax 异或。得到了循环右移 17 位后的 w[i-2]。



Push 13h，同理，得到循环右移 19 位后的 w[i-2]。然后再 push 0AH(10), w[i-2],CALL SHR 函数。

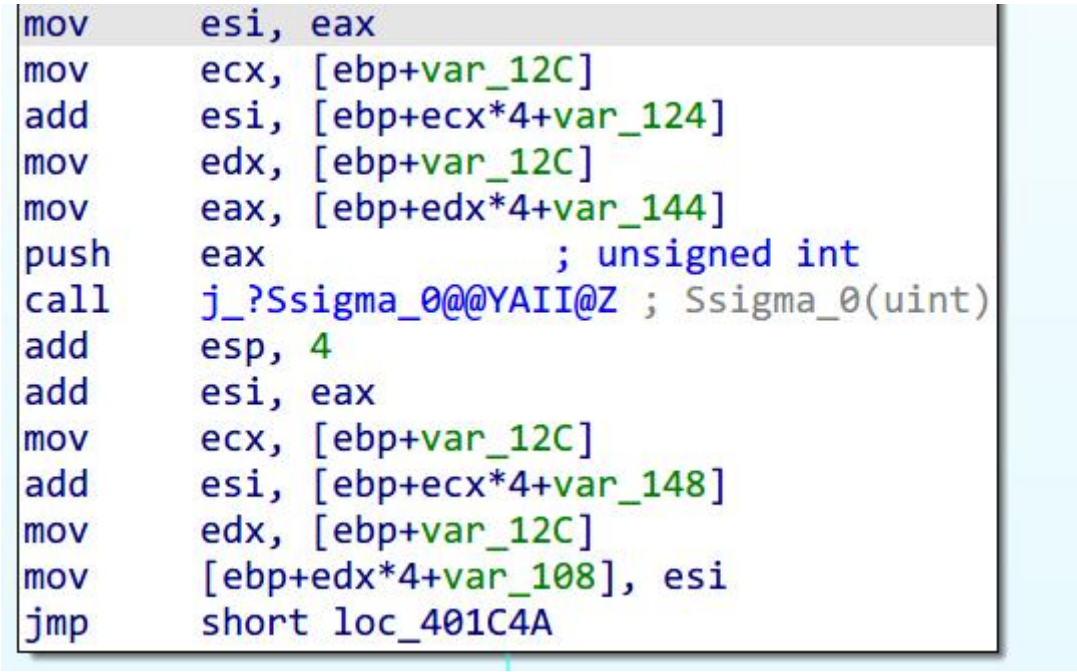
SHR

```
push    ebp
mov     ebp, esp
sub     esp, 40h
push    ebx
push    esi
push    edi
lea     edi, [ebp+var_40]
mov     ecx, 10h
mov     eax, 0CCCCCCCCh
rep stosd
mov     eax, [ebp+arg_0]
mov     ecx, [ebp+arg_4]
shr     eax, cl
pop     edi
pop     esi
pop     ebx
mov     esp, ebp
pop     ebp
retn
?SHR@@YAIII@Z endp
```

进入 SHR 函数，w[i-12]传给 eax，整数赋值给 ecx，eax 右移整数值（10）的位数.然后结束。返回。

```
mov     eax, [ebp+arg_0]
push    eax             ; unsigned int
call    j_?ROR@@YAIII@Z ; ROR(uint,uint)
add     esp, 8
mov     esi, eax
push    13h             ; unsigned int
mov     ecx, [ebp+arg_0]
push    ecx             ; unsigned int
call    j_?ROR@@YAIII@Z ; ROR(uint,uint)
add     esp, 8
xor     esi, eax
push    0Ah             ; unsigned int
mov     edx, [ebp+arg_0]
push    edx             ; unsigned int
call    j_?SHR@@YAIII@Z ; SHR(uint,uint)
add     esp, 8
xor     eax, esi
pop     edi
pop     esi
pop     ebx
add     esp, 40h
cmp     ebp, esp
call    __chkesp
```

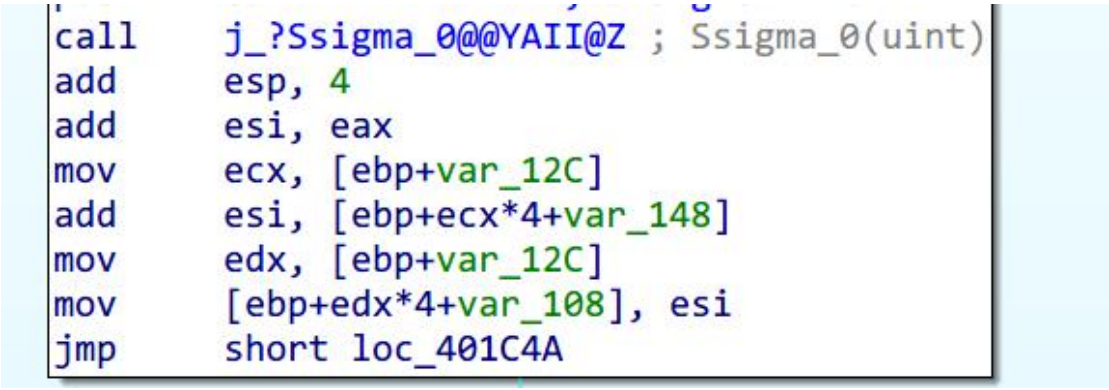
通过 xor，将以上计算的值传到 eax。（保存 sigma_1 函数的结果）。然后返回 dataround 函数。



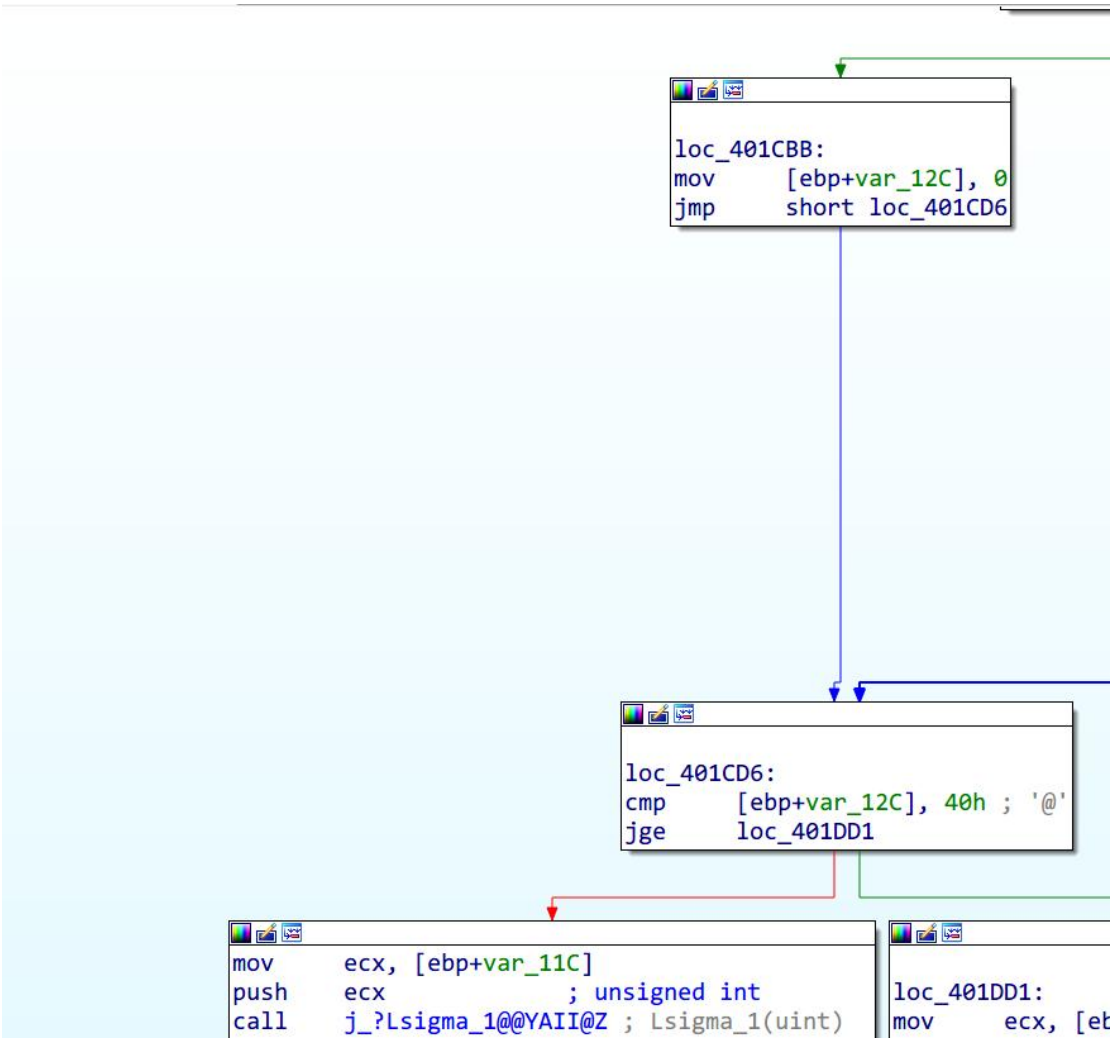
将 eax 的值传给 esi，然后 esi 加上 w[i-7]。然后 push w[i-15]。Call Ssigma_0 函数。

Ssigma_0

与 Ssigma_1 相似，只是移位的位数不一样，就不一一解释。



然后将 esi 加上 Ssigma_0 的结果再加上 w[i-16]。然后将 esi 的值传给 [ebp+i*4+var_108].循环 56 次，结束循环，完成对 w[16] -w[63]的生成。



[ebp+var_12]置零，然后 cmp[ebp+var_12] 40h（64），jge 小于 不跳转。循环 64，即是 64 次的加密循环。然后 push[ebp+var_11c] 用 ABCDEFGH,表示 H0-H7(哈希初始值)，所以就是 push E。call Lsigma_1 函数。

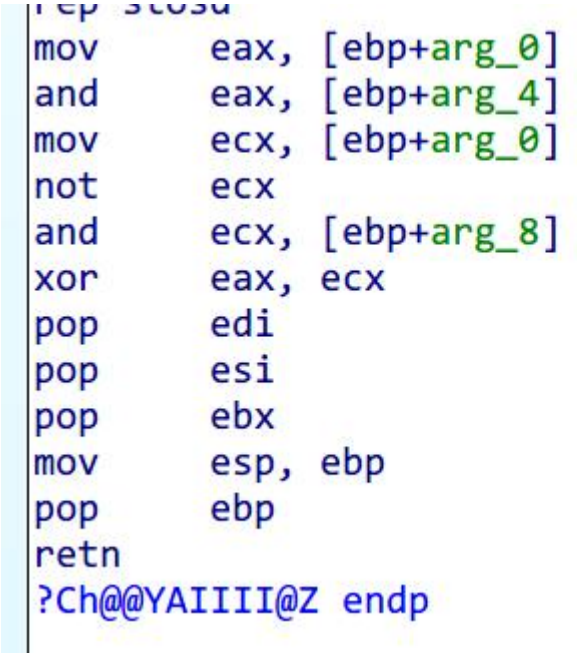
Lsigma_1

```
push    ebp
mov     ebp, esp
sub     esp, 40h
push    ebx
push    esi
push    edi
lea     edi, [ebp+var_40]
mov     ecx, 10h
mov     eax, 0CCCCCCCCh
rep stosd
push    6               ; unsigned int
mov     eax, [ebp+arg_0]
push    eax             ; unsigned int
call    j_?ROR@@YAIII@Z ; ROR(uint,uint)
add     esp, 8
mov     esi, eax
push    0Bh             ; unsigned int
mov     ecx, [ebp+arg_0]
push    ecx             ; unsigned int
call    j_?ROR@@YAIII@Z ; ROR(uint,uint)
add     esp, 8
xor     esi, eax
push    19h             ; unsigned int
mov     edx, [ebp+arg_0]
push    edx             ; unsigned int
call    j_?ROR@@YAIII@Z ; ROR(uint,uint)
add     esp, 8
xor     eax, esi
pop     edi
pop     esi
pop     ebx
add     esp, 40h
cmp     ebp, esp
call    __chkesp
mov     esp, ebp
pop     ebp
retn
?Lsigma_1@@YAII@Z endp
```

进入函数，发现与之前的 Ssigma_1 与 Ssigma_0 类似，只是，Lsigma_1 都只循环右移 6 位，11 位，25 位，依次相加。然后将结果传给 eax。返回 data_round 函数。

先将 Lsigma_1 返回结果 eax，然后将 H 传给 esi，然后 esi+=eax。然后 push E、F,G call CH 函数

CH 函数



将传入的第三个参数[ebp+arg_0]与[ebp+arg_4]与运算结果保存在 eax，然后将[ebp+arg_0]取反与[ebp+arg_8]与运算，结果保存在 ecx，然后 eax 与 ecx 异或结果保存在 eax。即：CH(E,F,G).。然后返回 data_round。

```
call    j_?Ch@@YAIIII@Z ; Ch(uint,uint,uint)
add     esp, 0Ch
add     esi, eax
mov     edx, [ebp+var_12C]
add     esi, ds:K[edx*4]
mov     eax, [ebp+var_12C]
add     esi, [ebp+eax*4+var_108]
mov     [ebp+var_4], esi
mov     ecx, [ebp+var_10C]
push    ecx             ; unsigned int
call    j_?Lsigma_0@@YAII@Z ; Lsigma_0(uint)
```

然后 esi+CH 函数返回的结果。然后 esi 再加上 K[i](var_12c = i，32bit)。esi 再加上[ebp+4*i+var_108]（w[i]）。然后将 esi 传给[ebp+var_4]。然后 push[ebp+var_10c](就是 A) 然后 call Lsigma_0 函数。

Lsigma_0

与 Lsigma_1 类似，只是，循环右移次数不相同。就不一一介绍。然后将计算结果返回 eax。

```
call    j_?Lsigma_0@@YAII@Z ; Lsigma_0(uint)
add     esp, 4
mov     esi, eax
mov     edx, [ebp+var_114]
push    edx             ; unsigned int
mov     eax, [ebp+var_110]
push    eax             ; unsigned int
mov     ecx, [ebp+var_10C]
push    ecx             ; unsigned int
call    j_?Maj@@YAIIII@Z ; Maj(uint,uint,uint)
```

Lsigma_0 的结果传给 esi。然后 push[ebp+var_114], [ebp+var_110], [ebp+var_10c](C,B,A) CALL Maj 函数。

Maj

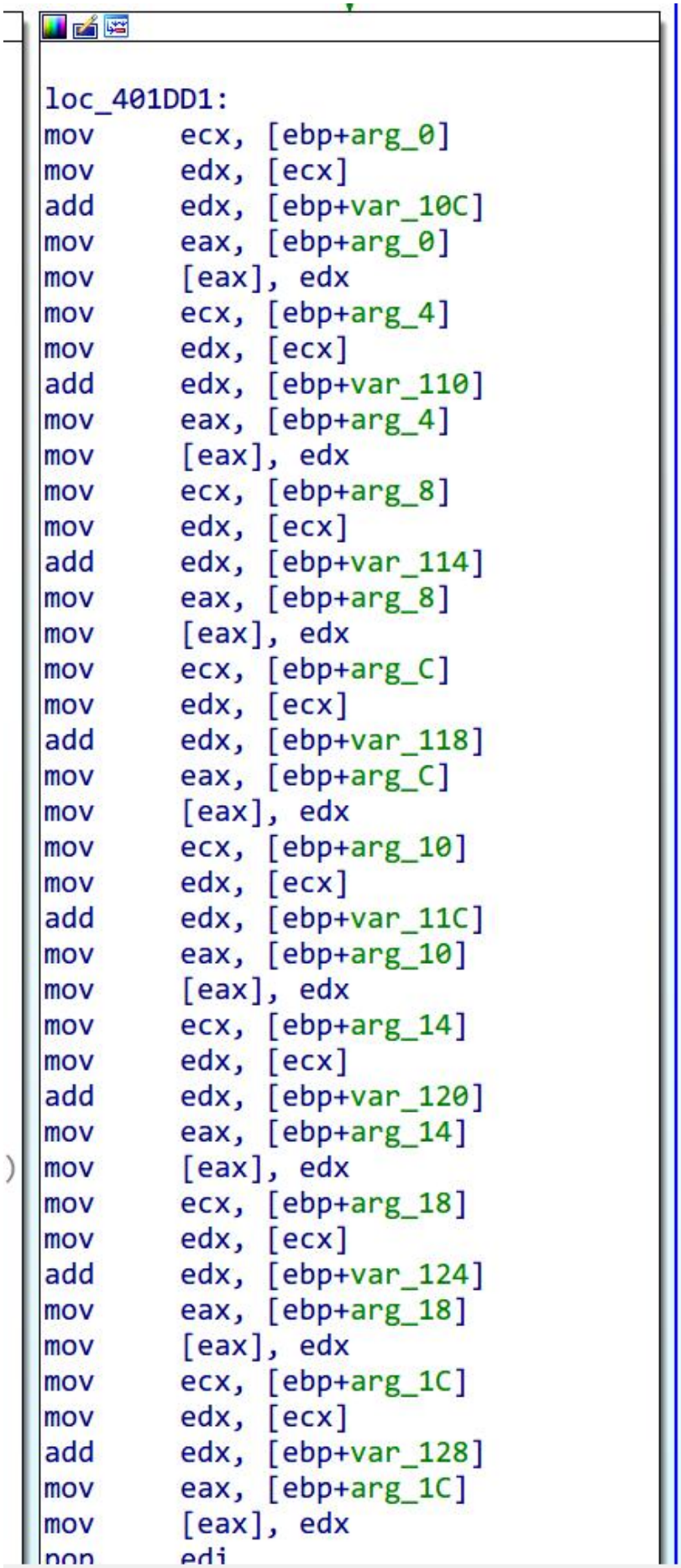
```
mov     eax, [ebp+arg_0]
and     eax, [ebp+arg_4]
mov     ecx, [ebp+arg_0]
and     ecx, [ebp+arg_8]
xor     eax, ecx
mov     edx, [ebp+arg_4]
and     edx, [ebp+arg_8]
xor     eax, edx
pop     edi
pop     esi
pop     ebx
mov     esp, ebp
pop     ebp
retn
?Maj@@YAIIII@Z endp
```

[ebp+arg_0]与[ebp+arg_4]与运算，[ebp+arg_0]与[ebp+arg_8]与运算，然后两个直接异或运算结果保存到 eax，然后[ebp+arg_8]与[ebp+arg_4]与运算再与

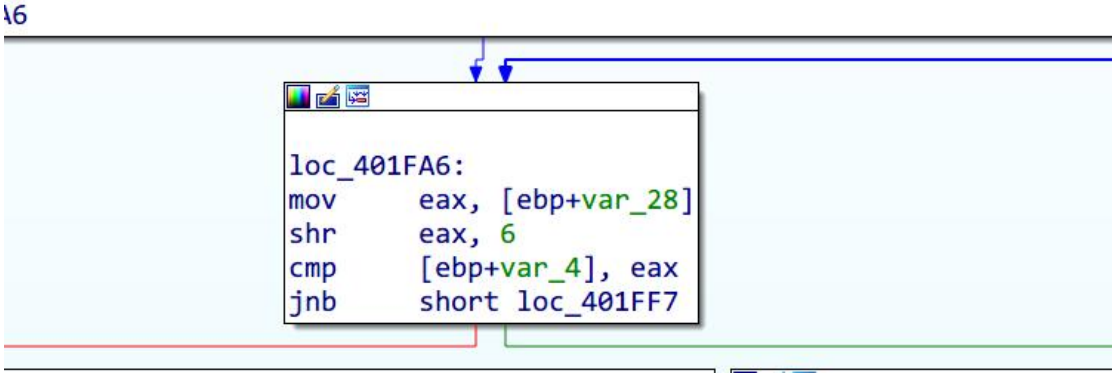
eax 异或结果保存到 eax。然后退出，返回 data_round 函数。

```
call    j_?Maj@@YAIIII@Z ; Maj(uint,uint,uint)
add     esp, 0Ch
add     esi, eax
mov     [ebp+var_8], esi
mov     edx, [ebp+var_124]
mov     [ebp+var_128], edx
mov     eax, [ebp+var_120]
mov     [ebp+var_124], eax
mov     ecx, [ebp+var_11C]
mov     [ebp+var_120], ecx
mov     edx, [ebp+var_118]
add     edx, [ebp+var_4]
mov     [ebp+var_11C], edx
mov     eax, [ebp+var_114]
mov     [ebp+var_118], eax
mov     ecx, [ebp+var_110]
mov     [ebp+var_114], ecx
mov     edx, [ebp+var_10C]
mov     [ebp+var_110], edx
mov     eax, [ebp+var_4]
add     eax, [ebp+var_8]
mov     [ebp+var_10C], eax
jmp     loc_401CC7
```

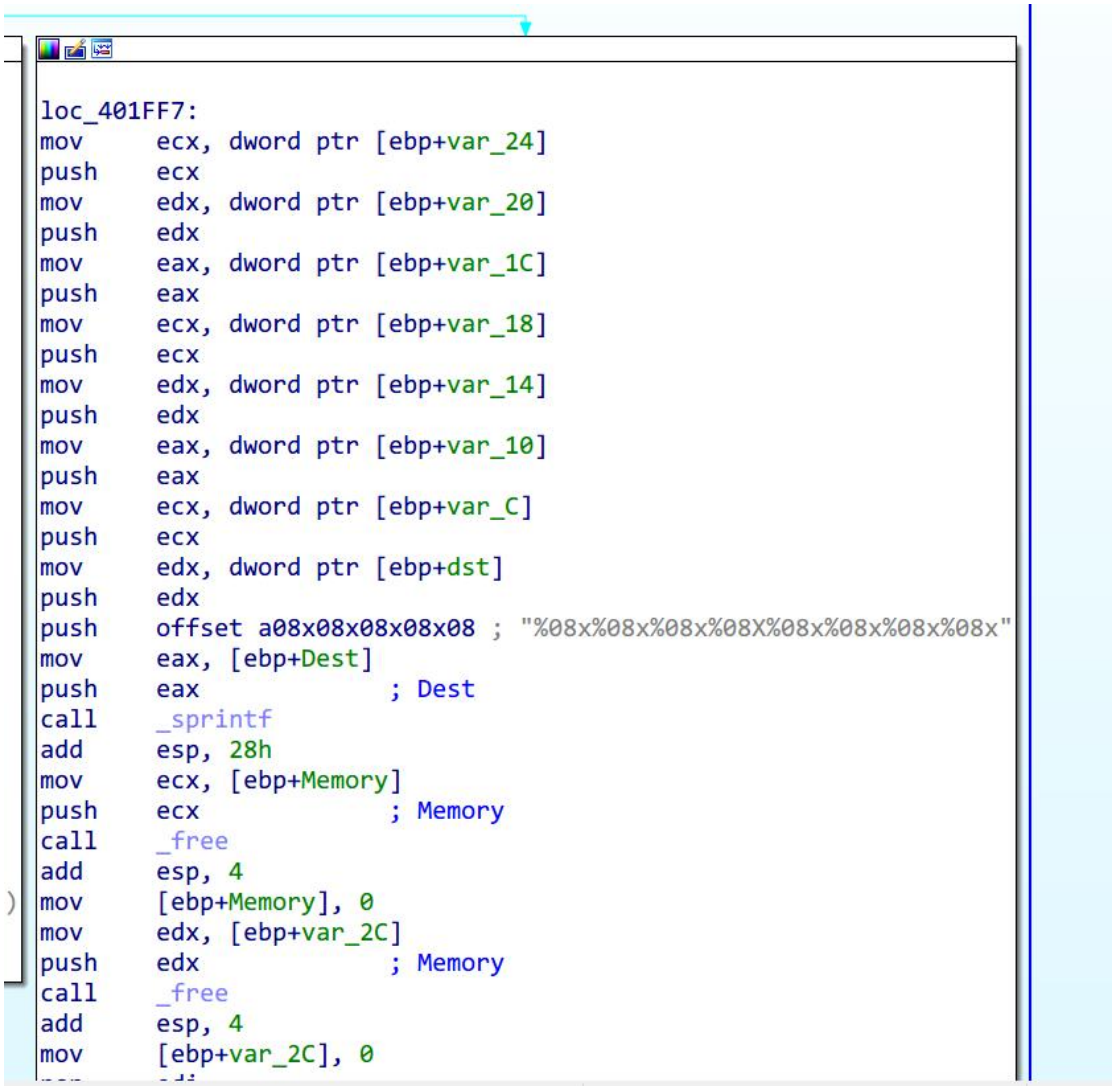
然后 eax 加上 esi 保存到[ebp+var_8]就是 Lsigma_0 的结果加上 Maj 的结果。然后进行三个传值操作，即：将 E、F、G 的值依次赋值给 F、G、H。然后将 D 加上[ebp+var_4](就是 H 加上 CH+W[i]+k[i]+H 的值)传给 E。然后再将 A、B、C 的值依次赋值给 B、C、D。最后将[ebp+var_4] (Maj+ Lsigma_0 的结果的值)加上[ebp+var_8](就是 H 加上 CH+W[i]+k[i]+H+Lsigma_1 的结果的值)然后传给 A。然后完成了一次迭代。共进行 64 次这样的迭代。


loc_401DD1:
mov ecx, [ebp+arg_0]
mov edx, [ecx]
add edx, [ebp+var_10C]
mov eax, [ebp+arg_0]
mov [eax], edx
mov ecx, [ebp+arg_4]
mov edx, [ecx]
add edx, [ebp+var_110]
mov eax, [ebp+arg_4]
mov [eax], edx
mov ecx, [ebp+arg_8]
mov edx, [ecx]
add edx, [ebp+var_114]
mov eax, [ebp+arg_8]
mov [eax], edx
mov ecx, [ebp+arg_C]
mov edx, [ecx]
add edx, [ebp+var_118]
mov eax, [ebp+arg_C]
mov [eax], edx
mov ecx, [ebp+arg_10]
mov edx, [ecx]
add edx, [ebp+var_11C]
mov eax, [ebp+arg_10]
mov [eax], edx
mov ecx, [ebp+arg_14]
mov edx, [ecx]
add edx, [ebp+var_120]
mov eax, [ebp+arg_14]
mov [eax], edx
mov ecx, [ebp+arg_18]
mov edx, [ecx]
add edx, [ebp+var_124]
mov eax, [ebp+arg_18]
mov [eax], edx
mov ecx, [ebp+arg_1C]
mov edx, [ecx]
add edx, [ebp+var_128]
mov eax, [ebp+arg_1C]
mov [eax], edx
pop edi

然后进入 loc_401DD1，依次将原来哈希初始值，赋值给的变量，将对应的变量传回原来的哈希初始值的位置。完成一次 512bit 块的迭代。（完成哈希初始值的迭代）。然后退出 data_round。返回 sha_calc 函数。



依次完成对有的几个 512bit 块的迭代后，跳转到 loc_401ff7。

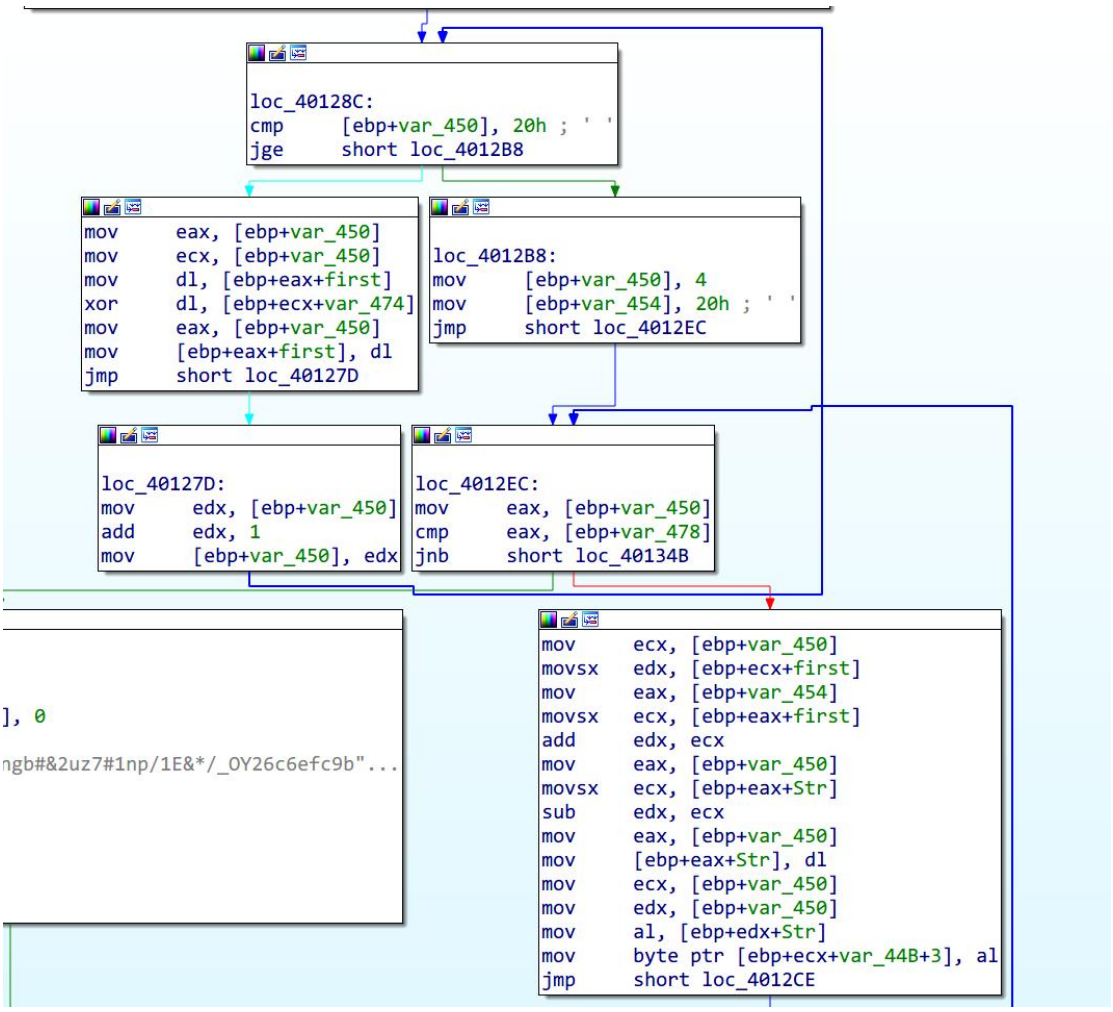


然后 push 哈希初始值迭代完成后的 8 个值，再 push dst，作为函数的返回结果。Call sprintf。将哈希初始值迭代完成后的 8 个值格式化输出到 dst 字符串。然后 free 释放掉以前 malloc 以及 calloc 申请的内存空间。结束 sha_calc 函数。回到 main 函数。sha_calc 函数的结果保存在[ebp+first]（就是 sha-256 的结

果）。

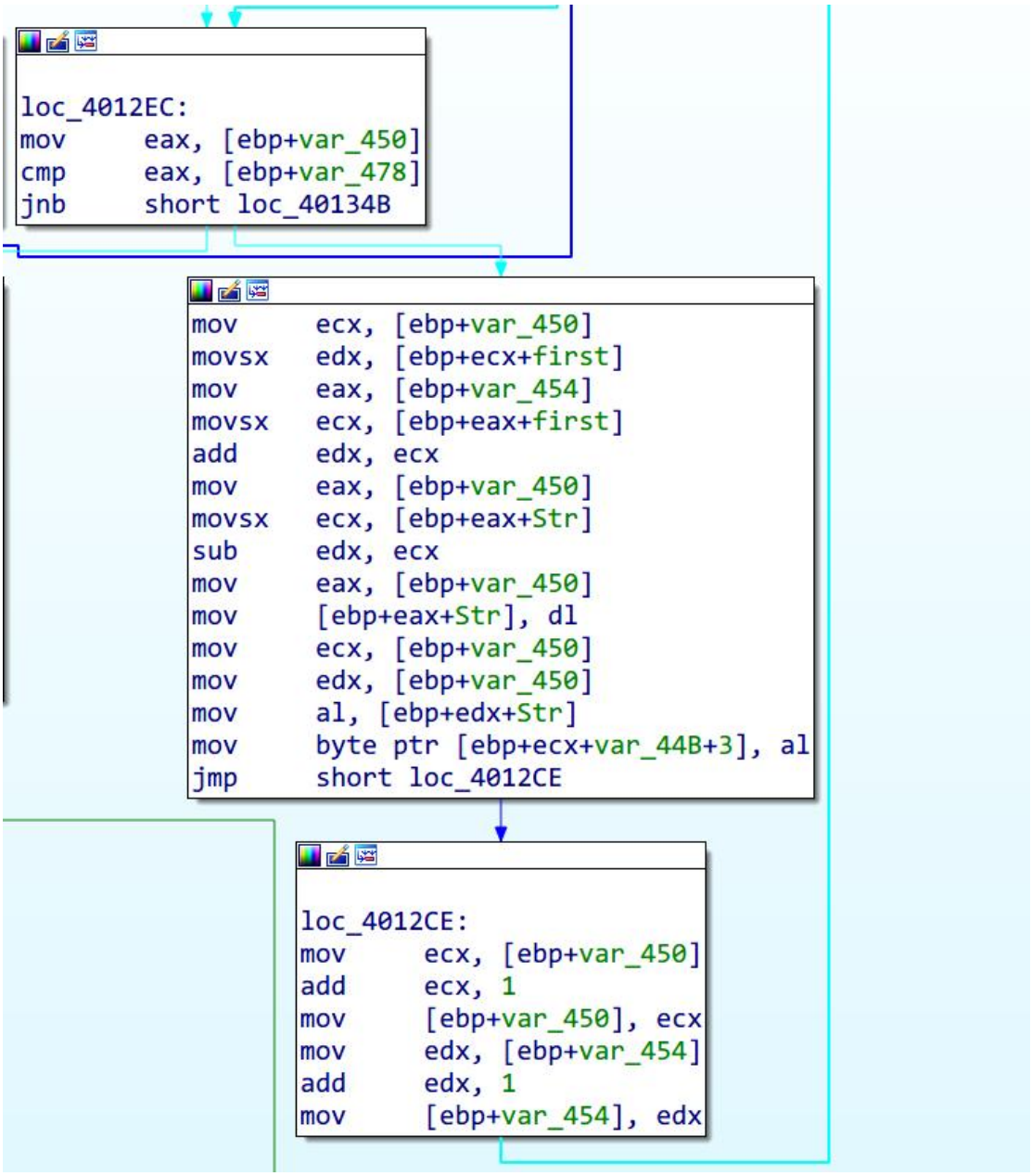
```
call    j_?sha_calc@@YAXPADPBDI@Z ; sha_calc(char *,char const *,uint)
add     esp, 0Ch
mov     [ebp+var_450], 0
jmp     short loc_40128C
```

然后将[ebp+var_450]置零，jmp loc_10128c



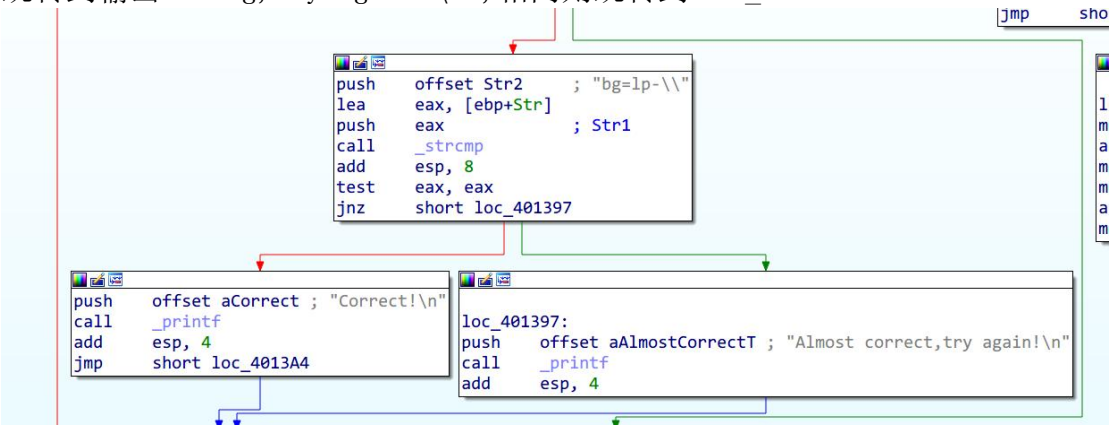
进入 loc_10128c，cmp[ebp+var_450] 20h(32)，就是循环 32 次。Jge 小于 不跳转。令 var_450 = i 然后将 sha_calc 函数的结果（ebp+first+i）取出一个字节，放在 edx 的低八位，然后与保存好的 sha_calc 函数的结果（[ebp+var_474+i]）进行异或。若与保存的 sha256 的结果相同，dl 为 0，并把结果传回[ebp+first+i]。循环 64 次（就是将输入的字符串生成的 sha256 与保存在函数中正确的 sha256 结果对比。结果传回[ebp+first+i]）然后结束循环进入 loc_4012b8。将

[ebp+var_450]赋值 4，[ebp+var_454]赋值 32.进入 loc_4012ec。



将[ebp+var_450]赋值给 eax，与[ebp+var_478]（字符串的长度）比较，若 4>=字符串长度则跳转 loc_40134b。若不跳转则，（var_450 = i，var_454 = j）。将 sha256 结果的第 i 个字节加上 sha256 结果的第 j 个字符再减去 字符串的第 i 的字符，然后将这个结果赋值回输入的字符串第 i 个字符。还有然后再将输入的字符串第 i 个字符传给输入的字符串第 i-4 个字符。最后 i++，j++。完成依次循环，完成字符串长度-4 的循环 然后结束。跳转到 loc_40134b。

将字符串长度-4 的字符[ebp+len+var_44b+3]赋值为 0。然后 push sha256 生成的 64 个字符与结果对比，结果保存在 eax。Test eax，若两个字符串不相同则跳转到输出"Wrong,try again!\n",相同则跳转到 loc_4013a6。



进入 loc_4013a6，比较输入的字符串（或者说，输入字符串的地址的值，已经被更改过了其实），与 bg=lp-/ 是否相同，相同则输出"Correct!\n"，不相同则输出"Almost correct,try again!\n"。至此 sha_256 逆向分析完成。

总结

当我成功地完成对 sha_256 的汇编逆向分析时，我感到非常兴奋和自豪。这是一个非常复杂的加密算法，我深入了解计算机的工作原理、指令集和寄存器的知识，以及对调试工具的熟悉。在这个过程中我最终收获了不少。

我不仅学会了如何理解复杂的加密算法。sha_256 是一个由多个轮次组成的算法，每个轮次包含数百条指令，需要对每个细节都进行仔细分析。通过逆向工程和分析汇编代码，我不仅深入理解了 sha_256 的工作原理，还学会了如何分析和解决其他类似复杂的算法问题。

总的来说， sha_256 的汇编逆向分析是一个非常有价值的经历。可以让我更加自信和有信心去面对其他的编程和算法问题。同时也为我的网安学习，逆向学习带来了知识与收获。写起来还是挺开心的，遇到了不少困难。但也不算太难，有一点点难，但是最后完成了有一种丰收的喜悦。

计算一下 flag 应该是 sha256_hash。

```c
#include <stdio.h>

int main() {
    char resultBuf[65] = "ov7s4g7nngb#&2uz7#1np/1E&*/_OY26c6efc9bc8a1ca1a8f8110b0535dc97a6";
    char input[12] = "bg=lp-\\lp-\\";
    int len = 11;
    int i, j;

    for (i = len -1, j = len + 28 - 1; i >= 4; i--, j--) {
        input[i] = resultBuf[i] + resultBuf[j] - input[i - 4];
        input[i - 4] = input[i];
    }

    printf("%s\n", input);

    return 0;
}
```

根据这段代码算出后七位为 56_hash，然后用爆破爆破出前 4 位，最后就是 sha256_hash