

汇编语言与逆向工程

北京邮电大学
崔宝江

北邮网安学院 崔宝江



第五章 常见加密算法逆向分析

- @5.1 简单加密算法逆向分析
- @5.2 对称加密算法逆向分析
- @5.3 单向散列算法逆向分析
- @5.4 其他算法逆向分析



5.3 单向散列算法逆向分析

- ① 1. MD5算法
- ② 2. SHA 算法



1. MD5算法

- ❑ (1) 算法原理
- ❑ (2) 逆向分析



MD系列哈希函数

- **Ron Rivest设计的系列哈希函数系列:**
 - **MD5 是MD4的改进型 [RFC1321]**
 - **MD4 [RFC1320]**
 - **MD2 [RFC1319], 已被Rogier等于1995 年攻破**
- **较早被标准化组织IETF接纳, 并已获得广泛应用**
- **Hash值长度为128bits**



MD5 算法逻辑

- ④ 输入：任意长度的消息
- ④ 输出：**128**位消息摘要
- ④ 处理：以**512**位输入数据块为单位

MD5 (RFC 1321) developed by Ron Rivest at MIT
北邮网安学院 崔宝江



$$L \times 512 \text{ bits} = N \times 32 \text{ bits}$$

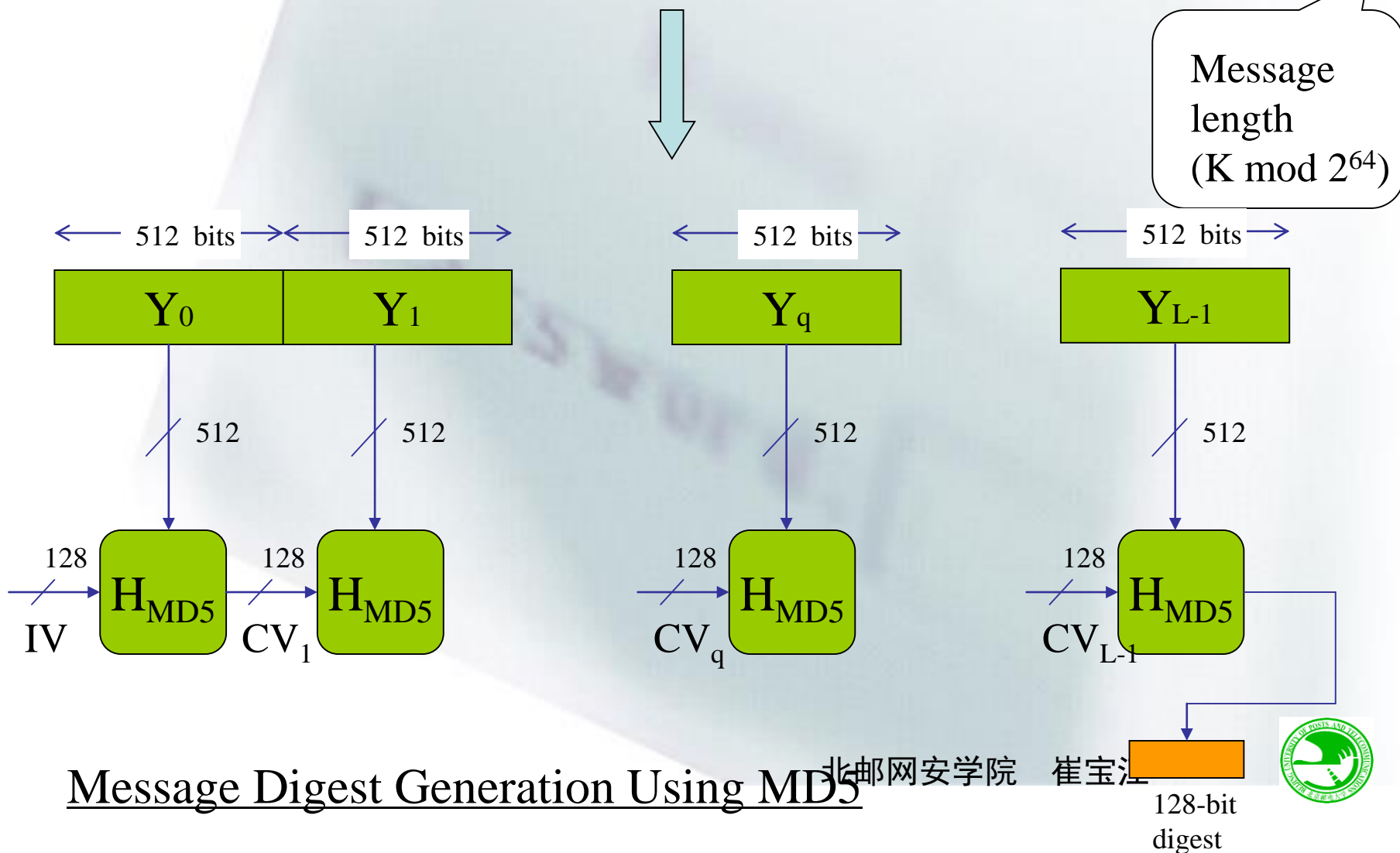
K bits

Padding
(1 to 512
bits)

Message

100...0

Message
length
($K \bmod 2^{64}$)



Message Digest Generation Using MD5

北邮网安学院

崔宝江

128-bit
digest



MD5 算法逻辑

MD5 Logic

步骤1：分组和填充：把明文消息按512位分组，最后填充一定长度的1000...使得每个消息的长度满足 $\text{length} \equiv 448 \pmod{512}$ 。填充的方法是先将比特“1”添加到消息的末尾，再添加k个零。

步骤2：附加消息：最后加上64位的消息摘要长度字段，整个明文恰好为512的整数倍。

步骤3：初始化MD缓冲区。一个128位MD缓冲区用以保存中间和最终散列函数的结果。置4个32比特长的缓冲区ABCD分别为

A: 01 23 45 67

B: 89 AB CD EF

C: FE DC BA 98

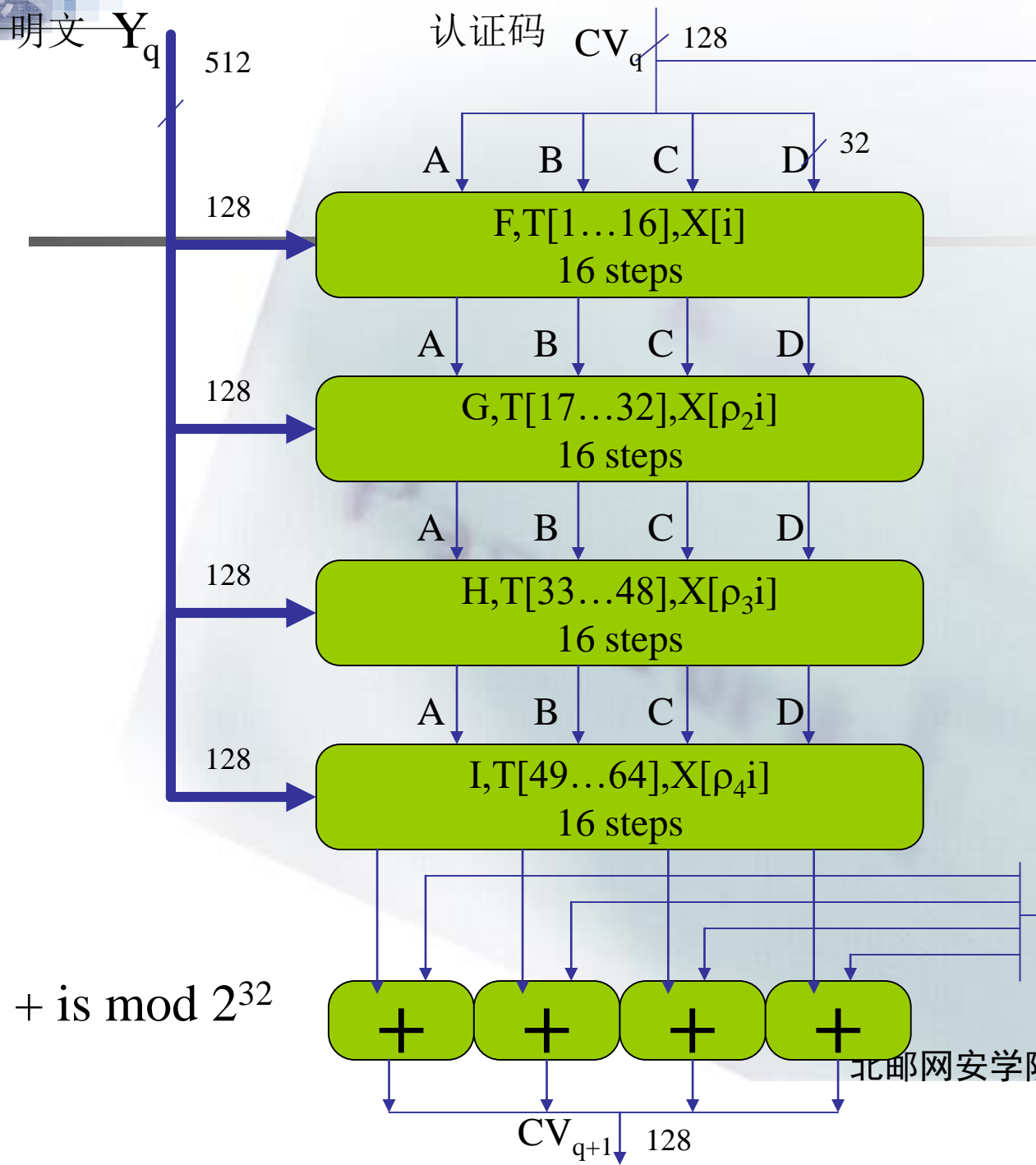
D: 76 54 32 10

步骤4：处理消息块（512位 = 16个32位字）。一个压缩函数是本算法的核心(H_{MD5})。它包括4轮处理。四轮处理具有相似的结构，但每次使用不同的基本逻辑函数，记为F,G,H,I。



明文 Y_q

认证码 CV_q



每一轮以当前的512位数据块(Y_q)和128位缓冲值ABCD作为输入，并修改缓冲值的内容。每次使用64元素表 $T[1 \dots 64]$ 中的四分之一， $T[]$ 由正弦函数 \sin 构造而成。 T 的第 i 个元素表示为 $T[i]$ ，其值等于 $2^{32} \times \text{abs}(\sin(i))$ ，其中 i 是弧度。由于 $\text{abs}(\sin(i))$ 是一个0到1之间的数， T 的每一个元素是一个可以表示成32位的整数。 T 表提供了随机化的32位模板，消除了输入数据中的任何规律性的特征。



MD5 算法逻辑

步骤5：输出结果。所有L个512位数据块处理完毕后，最后的结果就是128位消息摘要。

$$CV_0 = IV$$

$$CV_{q+1} = \text{SUM}_{32}(CV_q, RF_I[Y_q, RF_H[Y_q, RF_G[Y_q, RF_F[Y_q, CV_q]]]])$$

$$MD = CV_L$$

其中：IV = ABCD的初始值（见步骤3）

Y_q = 消息的第q个512位数据块

L = 消息中数据块数；

CV_q = 链接变量，用于第q个数据块的处理

RF_x = 使用基本逻辑函数x的一轮功能函数。

MD = 最终消息摘要结果

SUM_{32} = 分别按32位字计算的模 2^{32} 加法结果。



MD5 Compression Function

每一轮包含对缓冲区ABCD的16步操作所组成的一个序列。

$$a \leftarrow b + ((a + g(b,c,d) + X[k] + T[i]) \lll s)$$

其中，

a, b, c, d = 缓冲区的四个字，以一个给定的次序排列；

g = 基本逻辑函数F,G,H,I之一；

$\lll s$ = 对32位字循环左移s位

$X[k]$ = $M[q \times 16 + k]$ = 在第q个512位数据块中的第k个32位字

$T[i]$ = 表T中的第i个32位字；

$+$ = 模 2^{32} 的加；



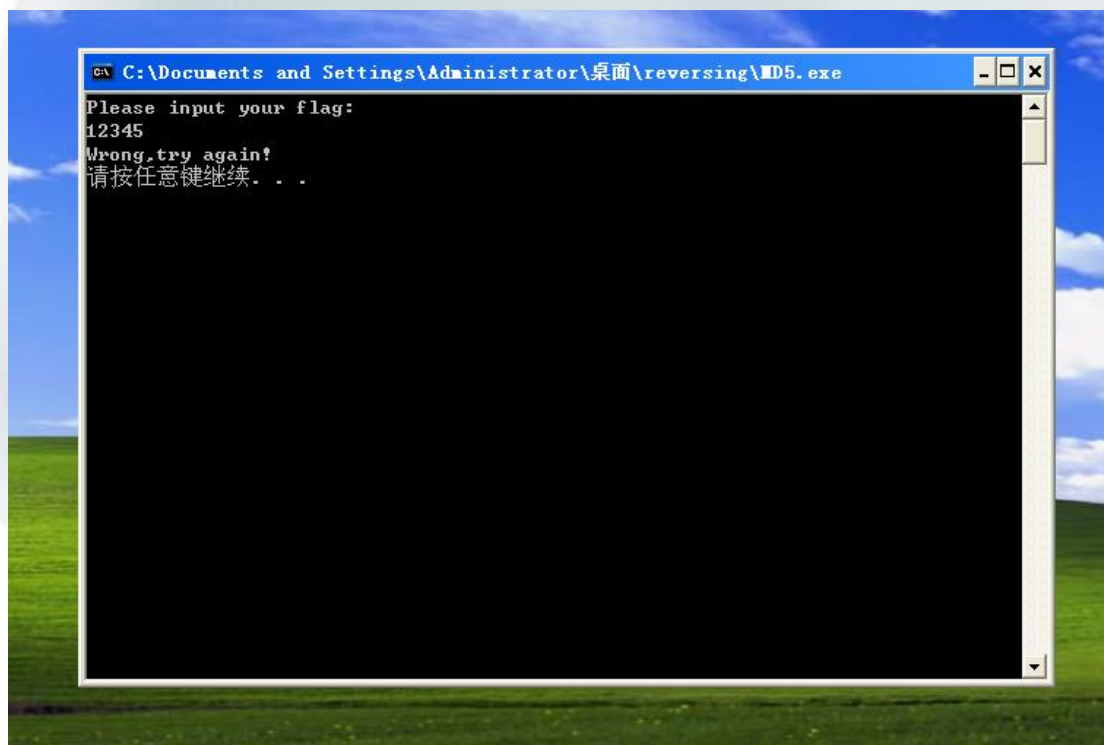
1. MD5算法

- ❑ (1) 算法原理
- ❑ (2) 逆向分析



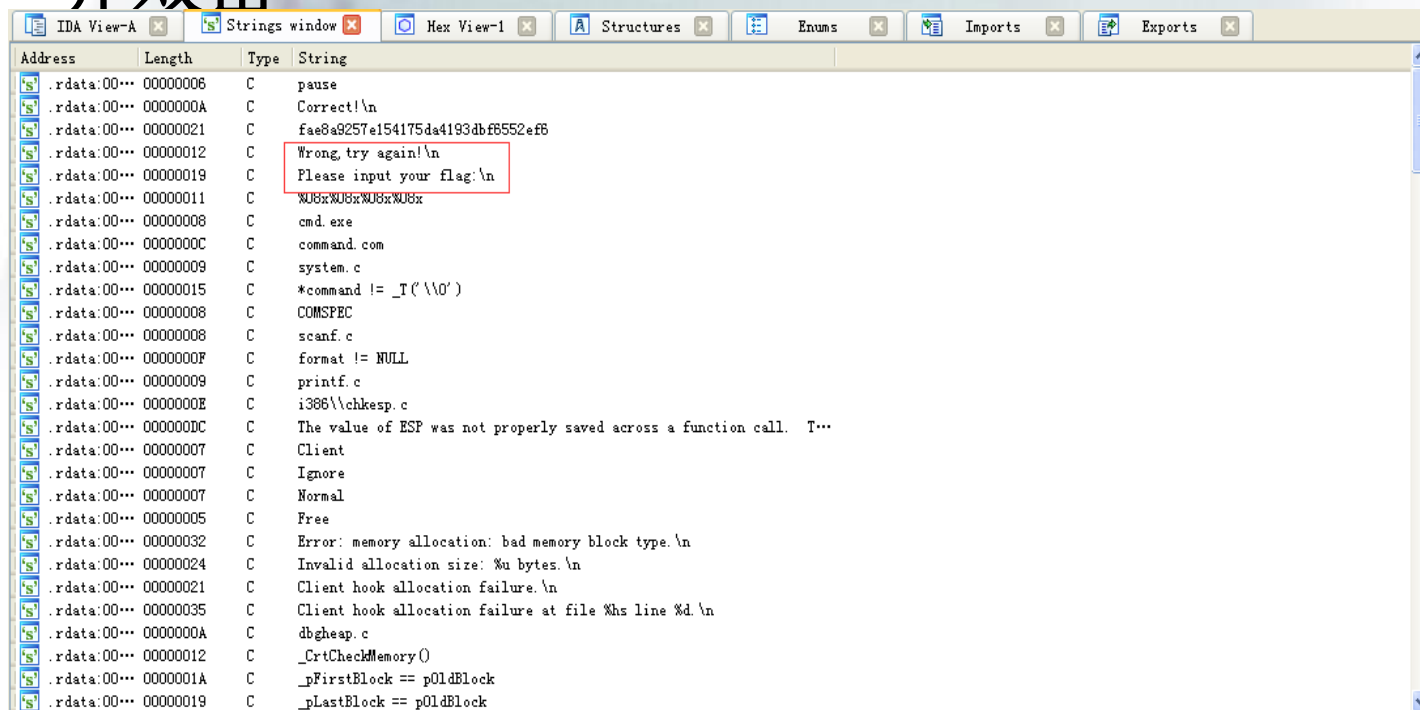
1. MD5算法

□ 运行示例程序MD5.exe，了解程序基本流程

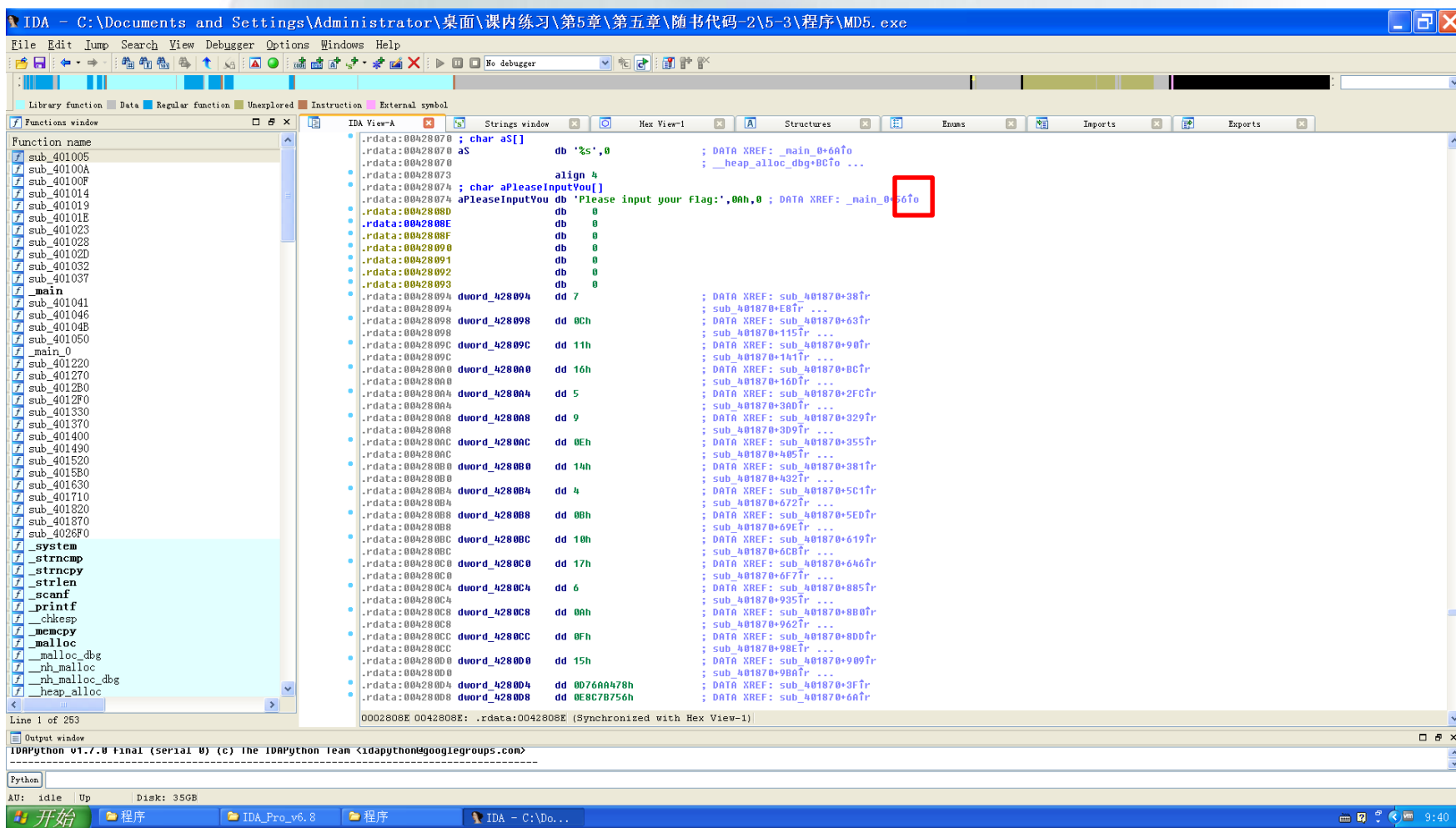


1. MD5算法

- ❑ 使用IDA打开MD5.exe
- ❑ 使用快捷键Shift+F12，找到Strings window并双击

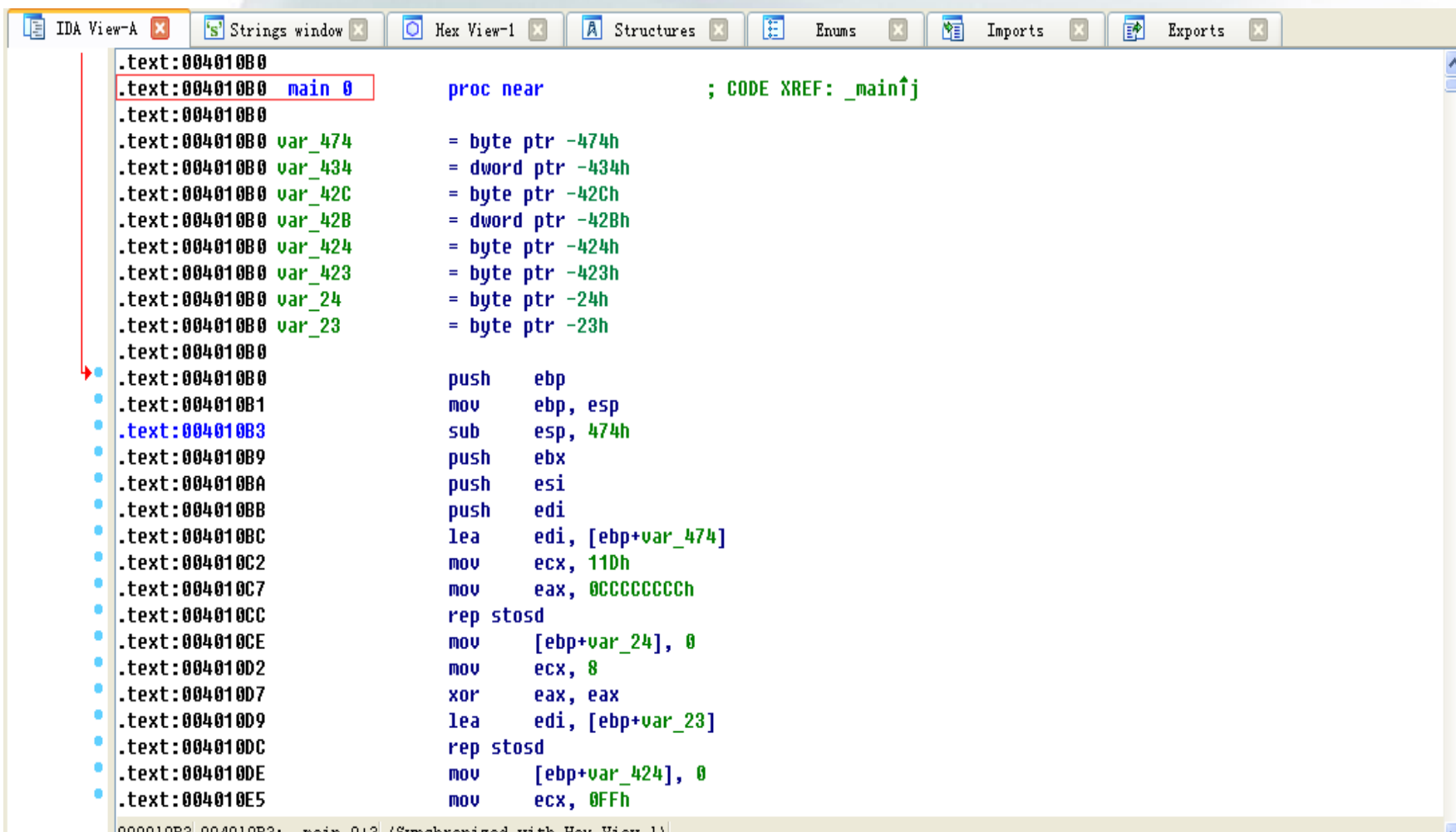


❑点字符串的交叉引用，就是后面的蓝色的箭头



1. MD5算法

□这样就根据关键字字符串定位到main函数

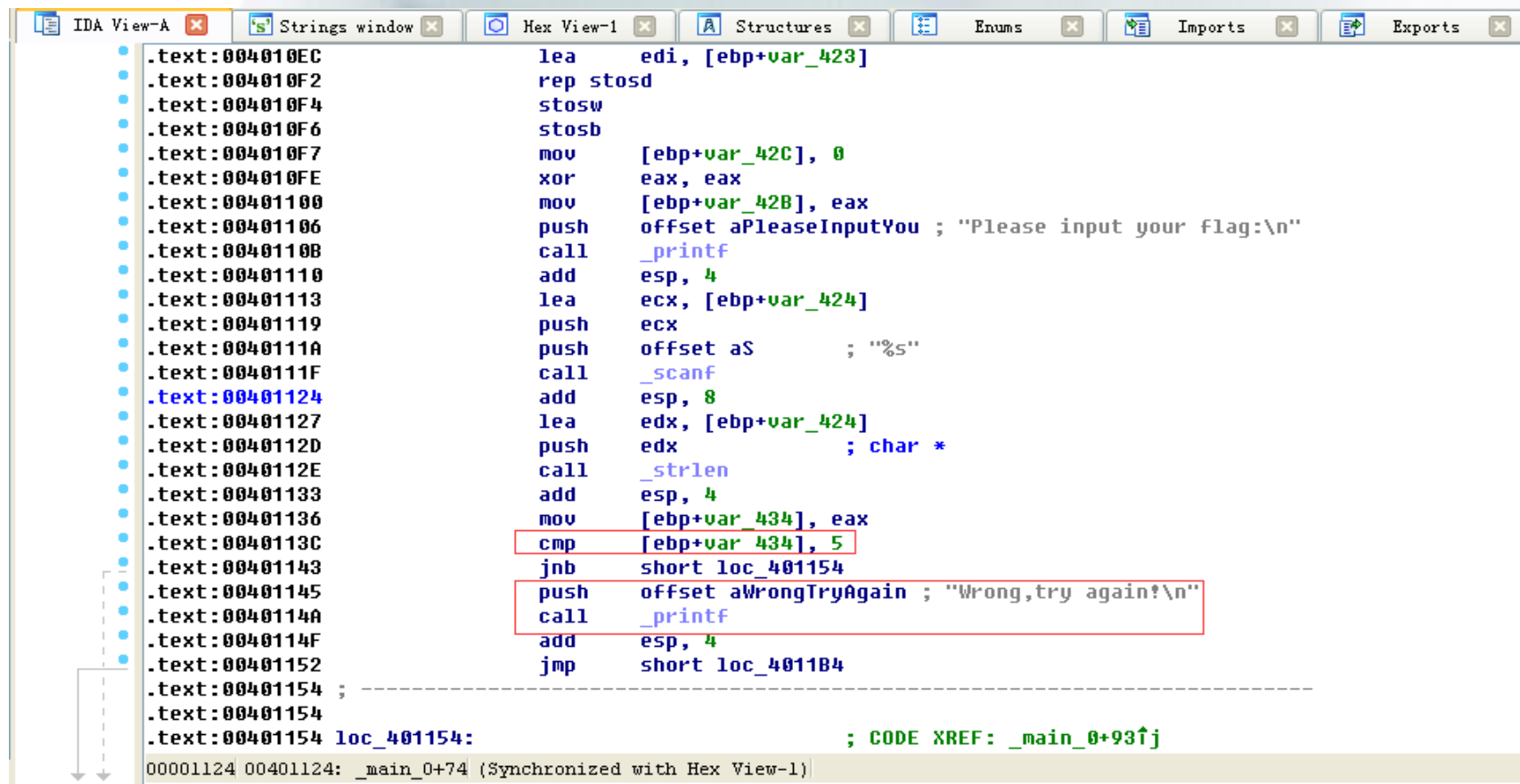


The screenshot shows the IDA Pro interface with the 'main' function selected. The 'Strings window' is open, showing the string 'main 0' at address 004010B0. The 'Hex View-1' window shows the assembly code for the function. The code starts with a 'proc near' directive and a cross-reference to '_main'. It then defines several local variables (var_474 to var_23) as pointers to specific memory locations. The function body begins with a 'push ebp' instruction, followed by 'mov ebp, esp' and 'sub esp, 474h'. It then pushes 'ebx', 'esi', and 'edi' onto the stack. A 'lea edi, [ebp+var_474]' instruction is followed by 'mov ecx, 110h' and 'mov eax, 0CCCCCCCCh'. The function then enters a loop with 'rep stosd' instructions, moving data from memory locations [ebp+var_24] and [ebp+var_424] to the stack. The function ends with 'mov ecx, 0FFh'.

```
.text:004010B0
.text:004010B0 main 0
.text:004010B0
.text:004010B0 var_474 = byte ptr -474h
.text:004010B0 var_434 = dword ptr -434h
.text:004010B0 var_42C = byte ptr -42Ch
.text:004010B0 var_42B = dword ptr -42Bh
.text:004010B0 var_424 = byte ptr -424h
.text:004010B0 var_423 = byte ptr -423h
.text:004010B0 var_24 = byte ptr -24h
.text:004010B0 var_23 = byte ptr -23h
.text:004010B0
.text:004010B0 push ebp
.text:004010B1 mov ebp, esp
.text:004010B3 sub esp, 474h
.text:004010B9 push ebx
.text:004010BA push esi
.text:004010BB push edi
.text:004010BC lea edi, [ebp+var_474]
.text:004010C2 mov ecx, 110h
.text:004010C7 mov eax, 0CCCCCCCCh
.text:004010CC rep stosd
.text:004010CE mov [ebp+var_24], 0
.text:004010D2 mov ecx, 8
.text:004010D7 xor eax, eax
.text:004010D9 lea edi, [ebp+var_23]
.text:004010DC rep stosd
.text:004010DE mov [ebp+var_424], 0
.text:004010E5 mov ecx, 0FFh
```

1. MD5算法

❑ 程序首先对输入进行了判断，若输入的长度小于5，则输出Wrong，程序退出

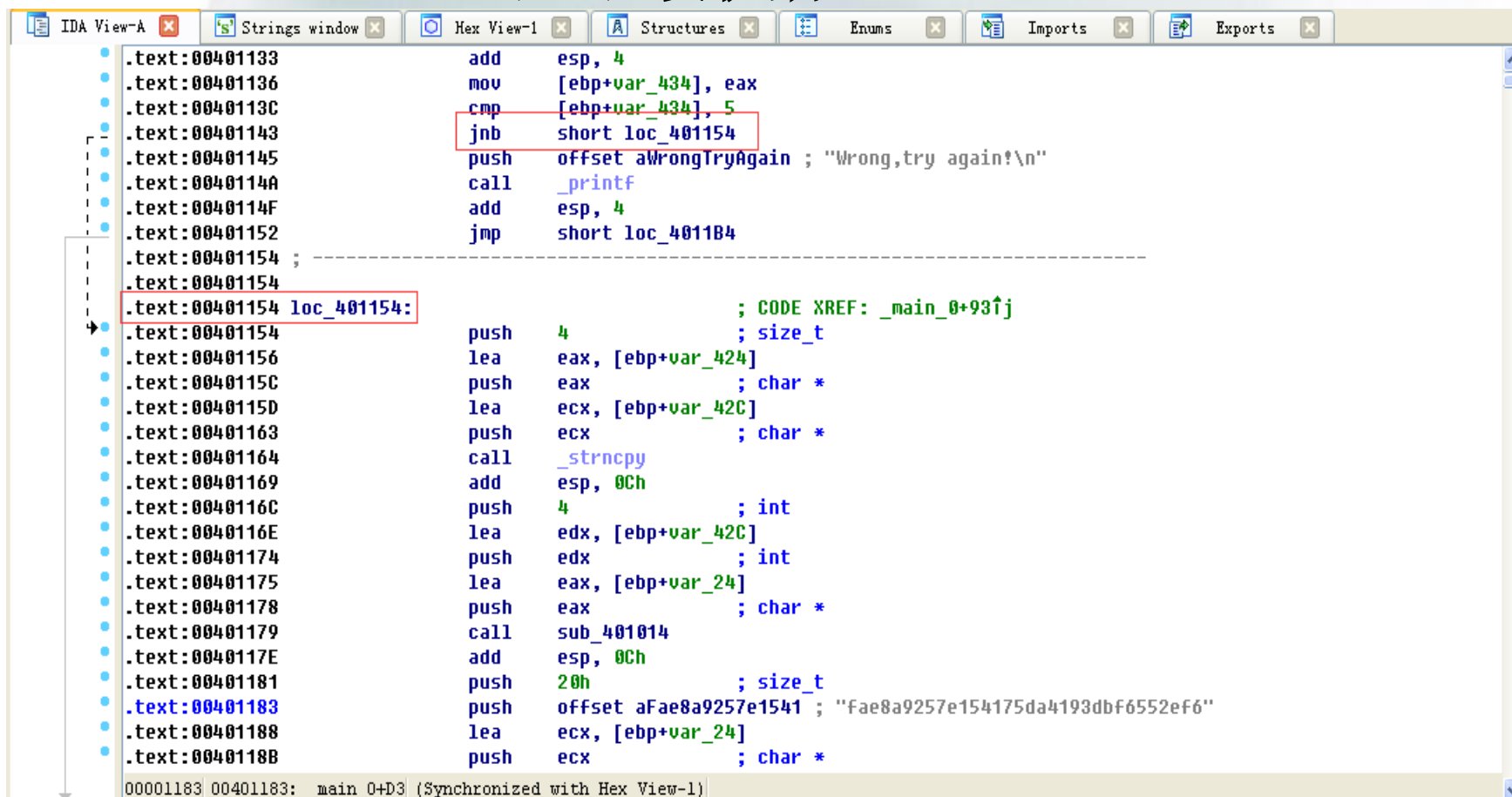


The screenshot shows the IDA Pro interface with the assembly view. The code is for a function that checks the length of input before processing it. The input is stored in memory at [ebp+var_423]. The code pushes the address of a prompt string "Please input your flag:\n" and calls _printf. Then it pushes the address of a format string "%5" and calls _scanf. It then calculates the length of the input using _strlen and compares it with 5. If the length is less than 5, it pushes the address of a string "Wrong, try again!\n" and calls _printf. The code is written in x86 assembly, and the IDA Pro interface shows various windows like Strings window, Hex View-1, Structures, Enums, Imports, and Exports.

```
.text:004010EC      lea     edi, [ebp+var_423]
.text:004010F2      rep stosd
.text:004010F4      stosw
.text:004010F6      stosb
.text:004010F7      mov     [ebp+var_42C], 0
.text:004010FE      xor     eax, eax
.text:00401100      mov     [ebp+var_42B], eax
.text:00401106      push    offset aPleaseInputYou ; "Please input your flag:\n"
.text:00401108      call    _printf
.text:00401110      add     esp, 4
.text:00401113      lea     ecx, [ebp+var_424]
.text:00401119      push    ecx
.text:0040111A      push    offset aS ; "%5"
.text:0040111F      call    _scanf
.text:00401124      add     esp, 8
.text:00401127      lea     edx, [ebp+var_424]
.text:0040112D      push    edx ; char *
.text:0040112E      call    _strlen
.text:00401133      add     esp, 4
.text:00401136      mov     [ebp+var_434], eax
.text:0040113C      cmp     [ebp+var_434], 5
.text:00401143      jnb     short loc_401154
.text:00401145      push    offset aWrongTryAgain ; "Wrong, try again!\n"
.text:0040114A      call    _printf
.text:0040114F      add     esp, 4
.text:00401152      jmp     short loc_4011B4
.text:00401154 ; -----
.text:00401154
.text:00401154 loc_401154: ; CODE XREF: _main_0+93↑j
00001124 00401124: _main_0+74 (Synchronized with Hex View-1)
```

1. MD5算法

❑ 若输入的长度大于或等于5，程序跳转到0x00401154处继续执行



```
.text:00401133      add     esp, 4
.text:00401136      mov     [ebp+var_434], eax
.text:0040113C      cmp     [ebp+var_434], 5
.text:00401143      jnb     short loc_401154
.text:00401145      push    offset aWrongTryAgain ; "Wrong,try again!\n"
.text:0040114A      call    _printf
.text:0040114F      add     esp, 4
.text:00401152      jmp     short loc_4011B4
.text:00401154      ; -----
.text:00401154      loc_401154:
.text:00401154      push    4 ; CODE XREF: _main_0+93↑j ; size_t
.text:00401156      lea     eax, [ebp+var_424]
.text:0040115C      push    eax ; char *
.text:0040115D      lea     ecx, [ebp+var_42C]
.text:00401163      push    ecx ; char *
.text:00401164      call    _strncpy
.text:00401169      add     esp, 0Ch
.text:0040116C      push    4 ; int
.text:0040116E      lea     edx, [ebp+var_42C]
.text:00401174      push    edx ; int
.text:00401175      lea     eax, [ebp+var_24]
.text:00401178      push    eax ; char *
.text:00401179      call    sub_401014
.text:0040117E      add     esp, 0Ch
.text:00401181      push    20h ; size_t
.text:00401183      push    offset aFae8a9257e1541 ; "Fae8a9257e154175da4193dbf6552ef6"
.text:00401188      lea     ecx, [ebp+var_24]
.text:0040118B      push    ecx ; char *
```

00001183 00401183: _main_0+D3 (Synchronized with Hex View-1)

1. MD5算法

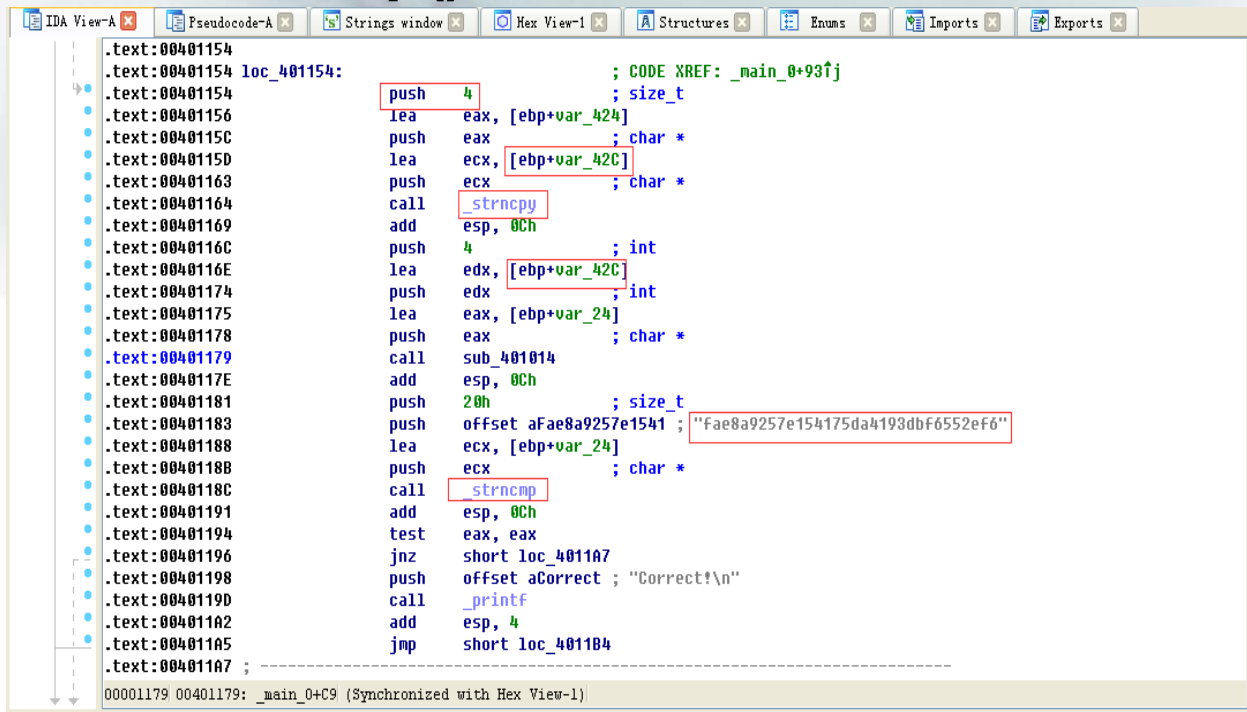
❑ 若输入的长度大于或等于5，程序跳转到0x00401154处继续执行

```
.text:00401133      add     esp, 4
.text:00401136      mov     [ebp+var_434], eax
.text:0040113C      cmp     [ebp+var_434], 5
.text:00401143      jnb     short loc_401154
.text:00401145      push    offset aWrongTryAgain ; "Wrong,try again!\n"
.text:0040114A      call    _printf
.text:0040114F      add     esp, 4
.text:00401152      jmp     short loc_401184
.text:00401154      ; -----
.text:00401154      loc_401154:
.text:00401154      push    4 ; CODE XREF: _main_0+93↑j ; size_t
.text:00401156      lea     eax, [ebp+var_424]
.text:0040115C      push    eax ; char *
.text:0040115D      lea     ecx, [ebp+var_42C]
.text:00401163      push    ecx ; char *
.text:00401164      call    _strncpy
.text:00401169      add     esp, 0Ch
.text:0040116C      push    4 ; int
.text:0040116E      lea     edx, [ebp+var_42C]
.text:00401174      push    edx ; int
.text:00401175      lea     eax, [ebp+var_24]
.text:00401178      push    eax ; char *
.text:00401179      call    sub_401014
.text:0040117E      add     esp, 0Ch
.text:00401181      push    20h ; size_t
.text:00401183      push    offset aFae8a9257e154175da4193dbf6552ef6 ; "Fae8a9257e154175da4193dbf6552ef6"
.text:00401188      lea     ecx, [ebp+var_24]
.text:0040118B      push    ecx ; char *
```



1. MD5算法

- 程序首先调用**strncpy**取出输入的前4字节并作为函数**sub_401014()**的第二个参数，最后将**sub_401014()**的返回结果与字符串“**fae8a9257e154175da4193dbf6552ef6**”进行比较，根据**strncmp()**的比较结果输出 **Correct**或**Wrong**。

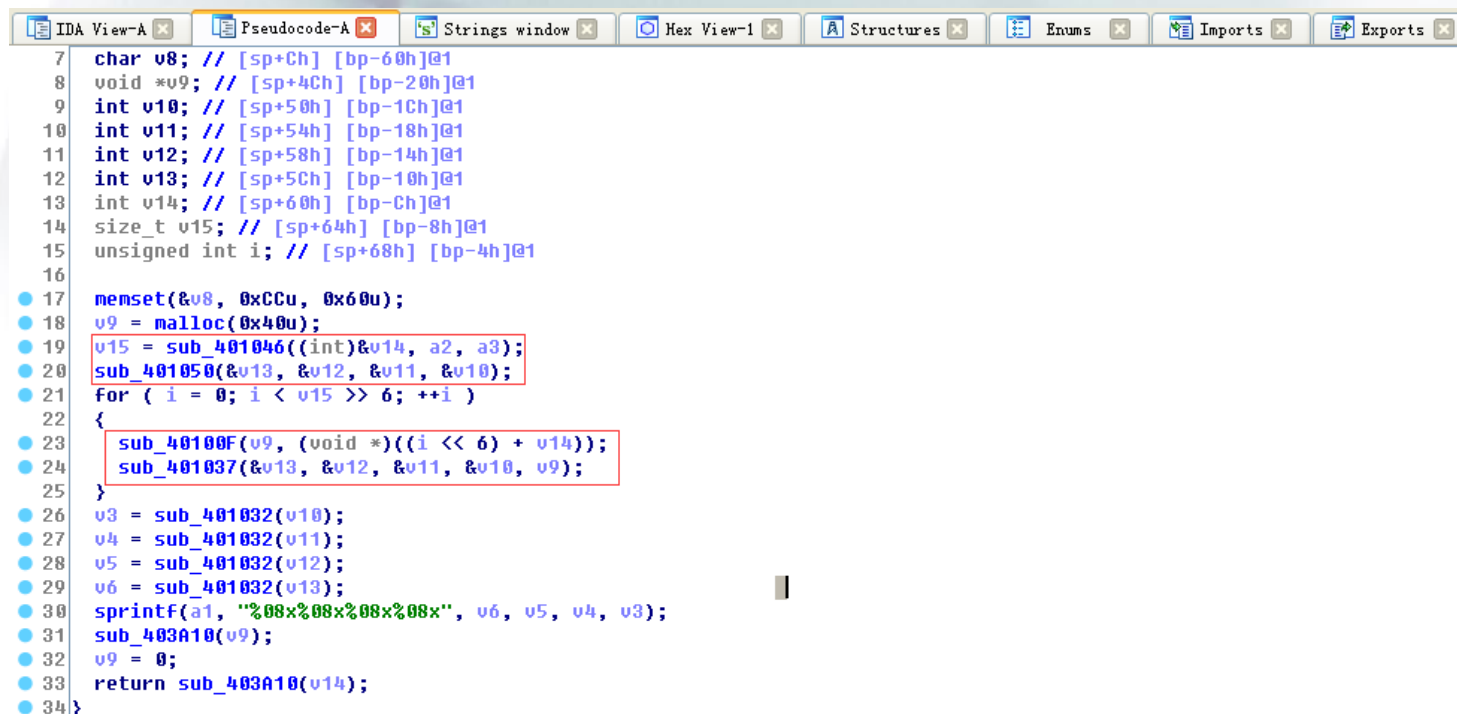


```
.text:00401154
.text:00401154 loc_401154:                ; CODE XREF: _main_0+93↑j
.text:00401154                push     4                ; size_t
.text:00401156                lea     eax, [ebp+var_424]
.text:0040115C                push     eax              ; char *
.text:0040115D                lea     ecx, [ebp+var_42C]
.text:00401163                push     ecx              ; char *
.text:00401164                call    _strncpy
.text:00401169                add     esp, 0Ch
.text:0040116C                push     4                ; int
.text:0040116E                lea     edx, [ebp+var_42C]
.text:00401174                push     edx              ; int
.text:00401175                lea     eax, [ebp+var_24]
.text:00401178                push     eax              ; char *
.text:00401179                call    sub_401014
.text:0040117E                add     esp, 0Ch
.text:00401181                push     20h              ; size_t
.text:00401183                push     offset aFae8a9257e1541 ; "fae8a9257e154175da4193dbf6552ef6"
.text:00401188                lea     ecx, [ebp+var_24]
.text:0040118B                push     ecx              ; char *
.text:0040118C                call    _strncmp
.text:00401191                add     esp, 0Ch
.text:00401194                test    eax, eax
.text:00401196                jnz     short loc_4011A7
.text:00401198                push     offset aCorrect ; "Correct!\n"
.text:0040119D                call    _printf
.text:004011A2                add     esp, 4
.text:004011A5                jmp     short loc_4011B4
.text:004011A7 ; -----
00001179 00401179: _main_0+C9 (Synchronized with Hex View-1)
```



1. MD5算法

- ❑ 对函数sub_401014()进行详细分析，使用快捷键F5对sub_401014()进行反编译
- ❑ 此函数的执行流程涉及到四个函数：sub_401046()、sub_401050()、sub_40100F()、和sub_401037()



```
7 char v8; // [sp+Ch] [bp-60h]@1
8 void *u9; // [sp+4Ch] [bp-20h]@1
9 int v10; // [sp+50h] [bp-1Ch]@1
10 int v11; // [sp+54h] [bp-18h]@1
11 int v12; // [sp+58h] [bp-14h]@1
12 int v13; // [sp+5Ch] [bp-10h]@1
13 int v14; // [sp+60h] [bp-Ch]@1
14 size_t v15; // [sp+64h] [bp-8h]@1
15 unsigned int i; // [sp+68h] [bp-4h]@1
16
17 memset(&v8, 0xCCu, 0x60u);
18 v9 = malloc(0x40u);
19 v15 = sub_401046((int)&v14, a2, a3);
20 sub_401050(&v13, &v12, &v11, &v10);
21 for ( i = 0; i < v15 >> 6; ++i )
22 {
23     sub_40100F(v9, (void *)((i << 6) + v14));
24     sub_401037(&v13, &v12, &v11, &v10, v9);
25 }
26 v3 = sub_401032(v10);
27 v4 = sub_401032(v11);
28 v5 = sub_401032(v12);
29 v6 = sub_401032(v13);
30 sprintf(a1, "%08x%08x%08x%08x", v6, v5, v4, v3);
31 sub_403A10(v9);
32 v9 = 0;
33 return sub_403A10(v14);
34 }
```



1. MD5算法

- ❑ 对于函数 `sub_401046()`，调用了 `sub_401710()`

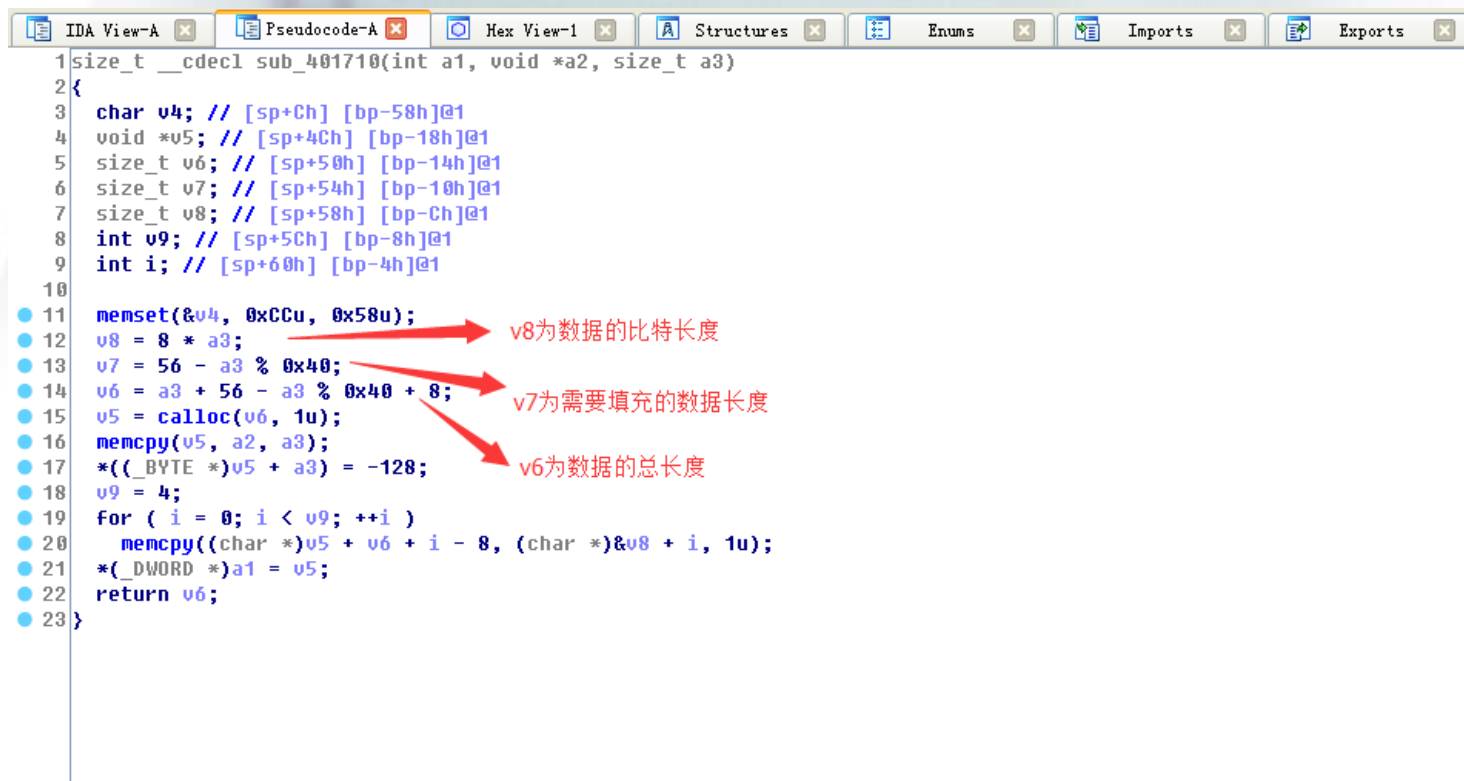
```
1 size_t __cdecl sub_401046(int a1, void *a2, size_t a3)
2 {
3     return sub_401710(a1, a2, a3);
4 }
```

- ❑ 根据 `sub_401710()` 一些特殊的语句，推测出这个函数的作用可能是数据填充
- ❑ 因为 `sub_401710()` 函数中出现了 56、64 以及模 64 这些数值和运算，联想到某些算法的数据填充规则，数据填充后使得数据的比特长度对 512 取模等于 448，换算为字节运算，即填充后的长度对 64 取模等于 56。



1. MD5算法

- ❑ 联系到：填充的方法是先将被“1”添加到数据的末尾，再添加若干0。填充完毕后再添加一个64比特长的块来存储消息长度，其值等于填充前消息长度的二进制表示。



```
1 size_t __cdecl sub_401710(int a1, void *a2, size_t a3)
2 {
3     char v4; // [sp+Ch] [bp-58h]@1
4     void *v5; // [sp+4Ch] [bp-18h]@1
5     size_t v6; // [sp+50h] [bp-14h]@1
6     size_t v7; // [sp+54h] [bp-10h]@1
7     size_t v8; // [sp+58h] [bp-Ch]@1
8     int v9; // [sp+5Ch] [bp-8h]@1
9     int i; // [sp+60h] [bp-4h]@1
10
11     memset(&v4, 0xCCu, 0x58u);
12     v8 = 8 * a3;
13     v7 = 56 - a3 % 0x40;
14     v6 = a3 + 56 - a3 % 0x40 + 8;
15     v5 = calloc(v6, 1u);
16     memcpy(v5, a2, a3);
17     *((_BYTE *)v5 + a3) = -128;
18     v9 = 4;
19     for ( i = 0; i < v9; ++i )
20         memcpy((char *)v5 + v6 + i - 8, (char *)&v8 + i, 1u);
21     *(_DWORD *)a1 = v5;
22     return v6;
23 }
```

v8为数据的比特长度

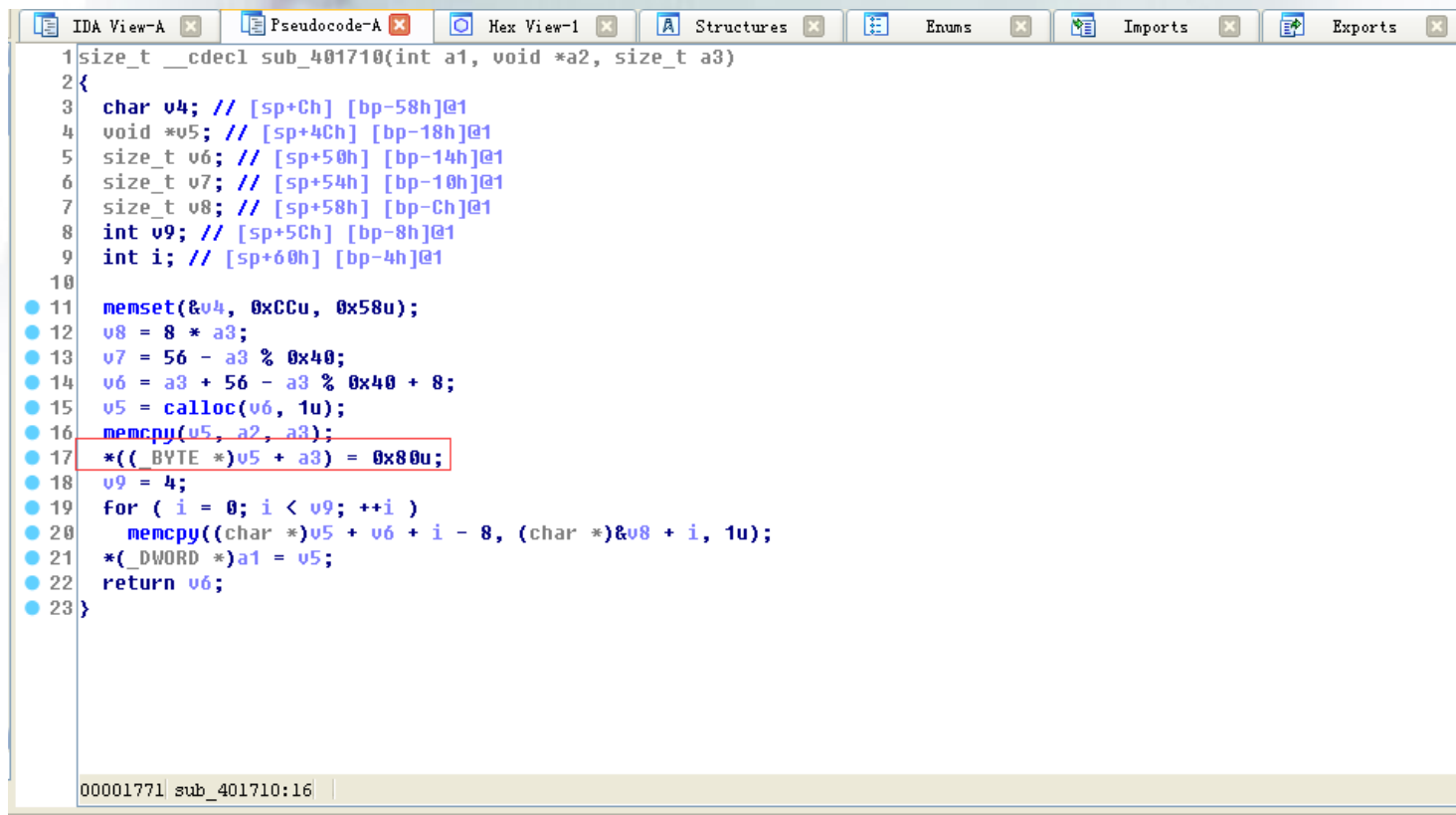
v7为需要填充的数据长度

v6为数据的总长度



1. MD5算法

❑ 另外0x80其实是比特串“10000000”，与数据填充规则中追加一个比特1，再填充0的填充规则相符合。



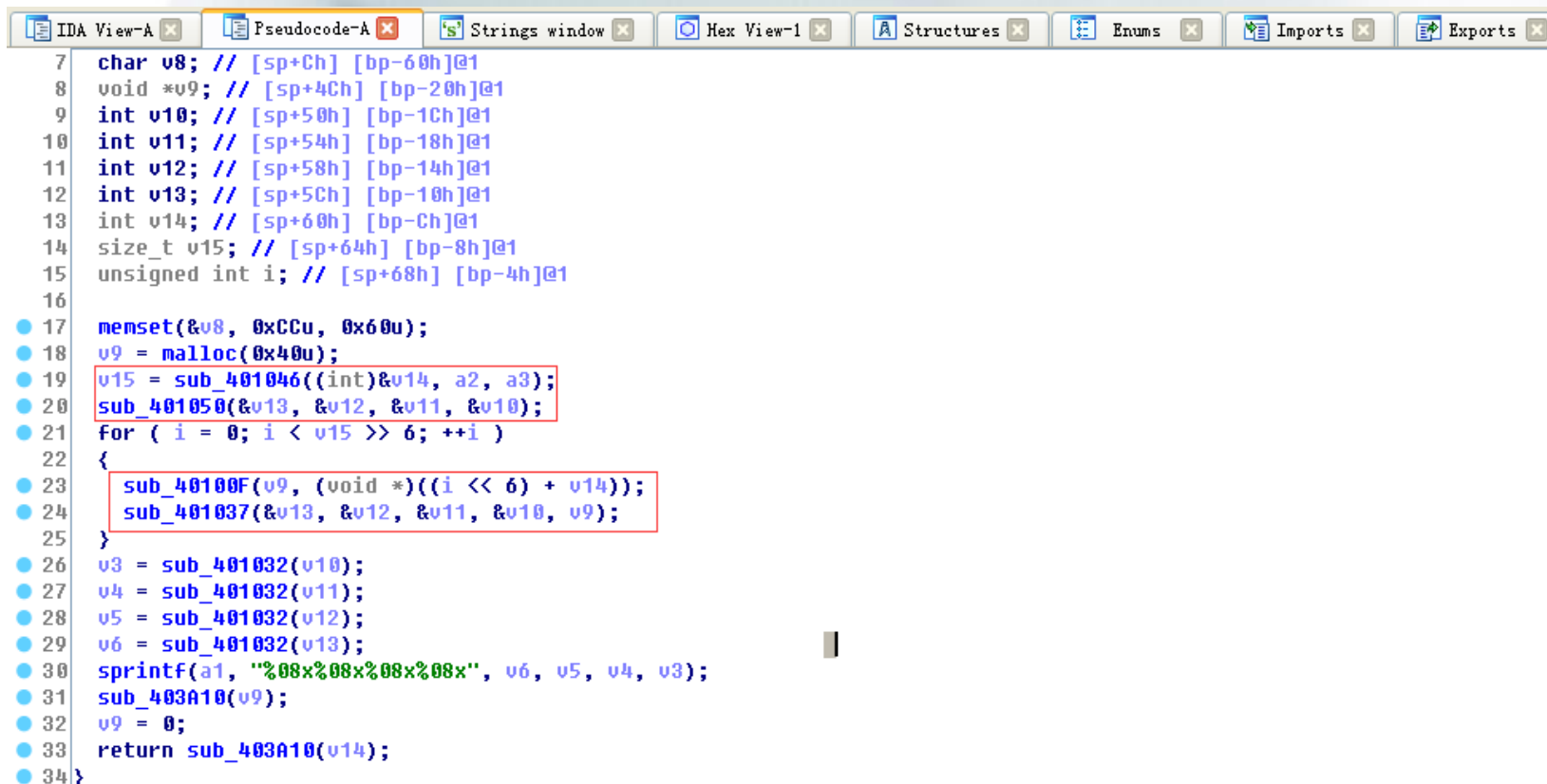
```
1 size_t __cdecl sub_401710(int a1, void *a2, size_t a3)
2 {
3     char v4; // [sp+Ch] [bp-58h]@1
4     void *v5; // [sp+4Ch] [bp-18h]@1
5     size_t v6; // [sp+50h] [bp-14h]@1
6     size_t v7; // [sp+54h] [bp-10h]@1
7     size_t v8; // [sp+58h] [bp-Ch]@1
8     int v9; // [sp+5Ch] [bp-8h]@1
9     int i; // [sp+60h] [bp-4h]@1
10
11     memset(&v4, 0xCCu, 0x58u);
12     v8 = 8 * a3;
13     v7 = 56 - a3 % 0x40;
14     v6 = a3 + 56 - a3 % 0x40 + 8;
15     v5 = calloc(v6, 1u);
16     memcpy(v5, a2, a3);
17     *((BYTE *)v5 + a3) = 0x80u;
18     v9 = 4;
19     for ( i = 0; i < v9; ++i )
20         memcpy((char *)v5 + v6 + i - 8, (char *)&v4 + i, 1u);
21     *(_DWORD *)a1 = v5;
22     return v6;
23 }
```

00001771 sub_401710:16



1. MD5算法

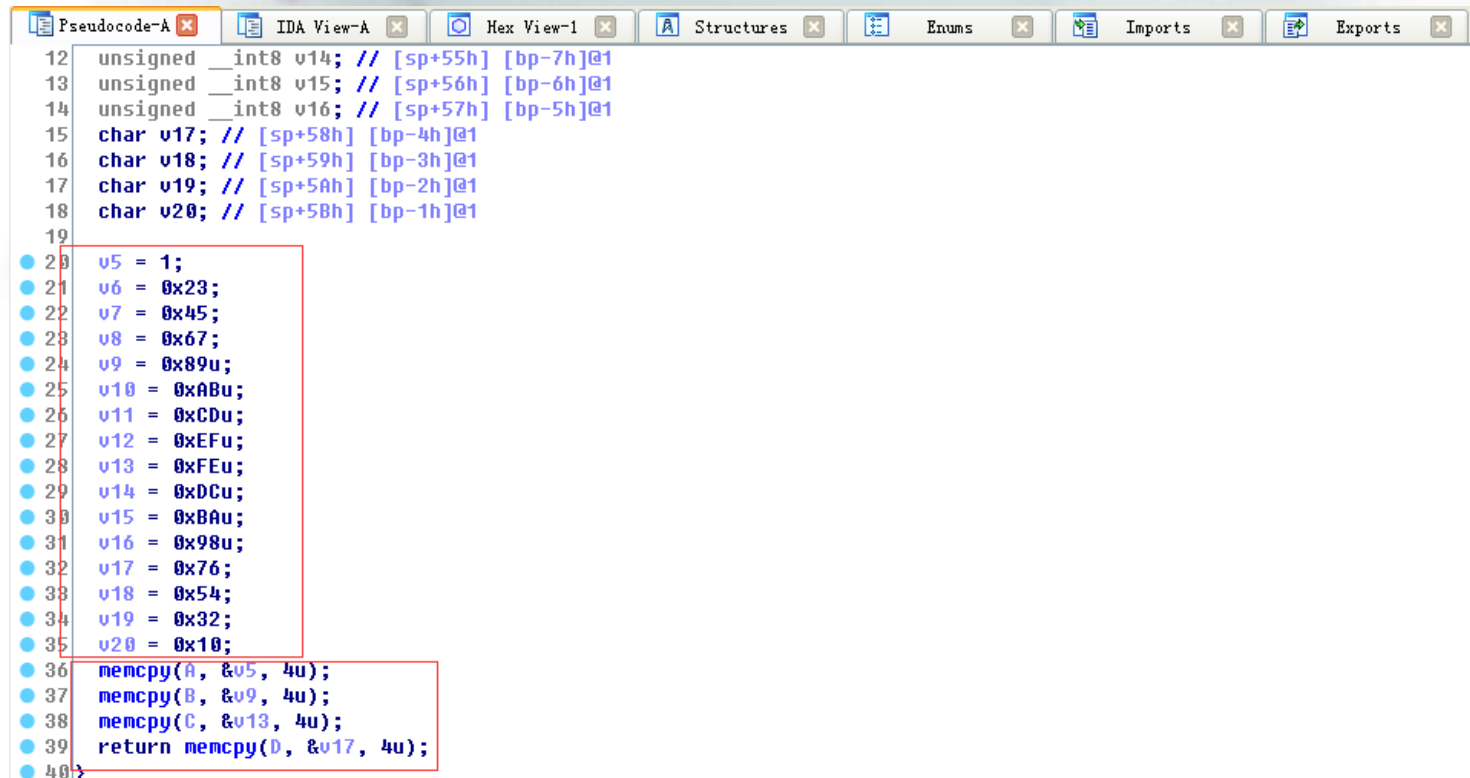
□ 结合上文的分析，sub_401046()函数完成了数据填充

The image shows a screenshot of the IDA Pro interface, specifically the Pseudocode-A window. The window title bar includes 'IDA View-A', 'Pseudocode-A', 'Strings window', 'Hex View-1', 'Structures', 'Enums', 'Imports', and 'Exports'. The pseudocode is for a function named 'sub_401046'. It starts with several local variable declarations: 'char v8', 'void *v9', and integers 'v10' through 'v14'. Then, it calls 'memset' on 'v8' and 'malloc' for 'v9'. The core of the function is a loop where 'v15' is passed to 'sub_401046', and then 'v13', 'v12', 'v11', and 'v10' are passed to 'sub_401050'. Inside the loop, 'sub_40100F' and 'sub_401037' are called with 'v9' and various bit-shifted values of 'i'. After the loop, 'v3' through 'v6' are calculated by calling 'sub_401032' on 'v10' through 'v13'. Finally, 'sprintf' is used to format 'v6', 'v5', 'v4', and 'v3' into 'a1', 'sub_403A10' is called with 'v9', 'v9' is set to 0, and the function returns 'sub_403A10(v14)'. Lines 19, 20, 21, 23, 24, and 25 are highlighted with a red box. Line numbers 7 through 34 are visible on the left margin.

```
7 char v8; // [sp+Ch] [bp-60h]@1
8 void *v9; // [sp+4Ch] [bp-20h]@1
9 int v10; // [sp+50h] [bp-1Ch]@1
10 int v11; // [sp+54h] [bp-18h]@1
11 int v12; // [sp+58h] [bp-14h]@1
12 int v13; // [sp+5Ch] [bp-10h]@1
13 int v14; // [sp+60h] [bp-Ch]@1
14 size_t v15; // [sp+64h] [bp-8h]@1
15 unsigned int i; // [sp+68h] [bp-4h]@1
16
17 memset(&v8, 0xCCu, 0x60u);
18 v9 = malloc(0x40u);
19 v15 = sub_401046((int)&v14, a2, a3);
20 sub_401050(&v13, &v12, &v11, &v10);
21 for ( i = 0; i < v15 >> 6; ++i )
22 {
23     sub_40100F(v9, (void *)((i << 6) + v14));
24     sub_401037(&v13, &v12, &v11, &v10, v9);
25 }
26 v3 = sub_401032(v10);
27 v4 = sub_401032(v11);
28 v5 = sub_401032(v12);
29 v6 = sub_401032(v13);
30 sprintf(a1, "%08x%08x%08x%08x", v6, v5, v4, v3);
31 sub_403A10(v9);
32 v9 = 0;
33 return sub_403A10(v14);
34 }
```

1. MD5算法

- 接下来分析函数 `sub_401050()`，它将 `v5~v20` 这16长度为1个字节的变量赋值给四个整型变量，分别设为 `A,B,C,D`，使用快捷键 `N` 修改变量名，使用快捷键 `H` 可以将 `v5~v20` 修改为16进制显示。



```
Pseudocode-A x IDA View-A x Hex View-1 x Structures x Enums x Imports x Exports x
12 unsigned __int8 v14; // [sp+55h] [bp-7h]@1
13 unsigned __int8 v15; // [sp+56h] [bp-6h]@1
14 unsigned __int8 v16; // [sp+57h] [bp-5h]@1
15 char v17; // [sp+58h] [bp-4h]@1
16 char v18; // [sp+59h] [bp-3h]@1
17 char v19; // [sp+5Ah] [bp-2h]@1
18 char v20; // [sp+5Bh] [bp-1h]@1
19
20 v5 = 1;
21 v6 = 0x23;
22 v7 = 0x45;
23 v8 = 0x67;
24 v9 = 0x89u;
25 v10 = 0xABu;
26 v11 = 0xCDu;
27 v12 = 0xEFu;
28 v13 = 0xFEu;
29 v14 = 0xDCu;
30 v15 = 0xBAu;
31 v16 = 0x98u;
32 v17 = 0x76;
33 v18 = 0x54;
34 v19 = 0x32;
35 v20 = 0x10;
36 memcpy(A, &v5, 4u);
37 memcpy(B, &v9, 4u);
38 memcpy(C, &v13, 4u);
39 return memcpy(D, &v17, 4u);
40 }
```



MD5 算法逻辑

MD5 Logic

步骤1：分组和填充：把明文消息按512位分组，最后填充一定长度的1000...使得每个消息的长度满足 $\text{length} \equiv 448 \pmod{512}$ 。填充的方法是先将比特“1”添加到消息的末尾，再添加k个零。

步骤2：附加消息：最后加上64位的消息摘要长度字段，整个明文恰好为512的整数倍。

步骤3：初始化MD缓冲区。一个128位MD缓冲区用以保存中间和最终散列函数的结果。置4个32比特长的缓冲区ABCD分别为

A: 01 23 45 67

B: 89 AB CD EF

C: FE DC BA 98

D: 76 54 32 10

步骤4：处理消息块（512位 = 16个32位字）。一个压缩函数是本算法的核心(H_{MD5})。它包括4轮处理。四轮处理具有相似的结构，但每次使用不同的基本逻辑函数，记为F,G,H,I。



1. MD5算法

□ 经过函数sub_401050()后，设这A~D这4个变量在内存中的存储为

Startup		Untitled1*																
▼ Edit As: Hex ▼		Run Script ▼		Run Template ▼														
		0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	0123456789ABCDEF
0000h:		01	23	45	67	89	AB	CD	EF	FE	DC	BA	98	76	54	32	10	.#Eg%«İipÜ°~vT2.
0010h:																		

□ 由于计算机为小端存储模式，因此A~D真正的数值为

A = 0x67452301;

B = 0xefcdab89;

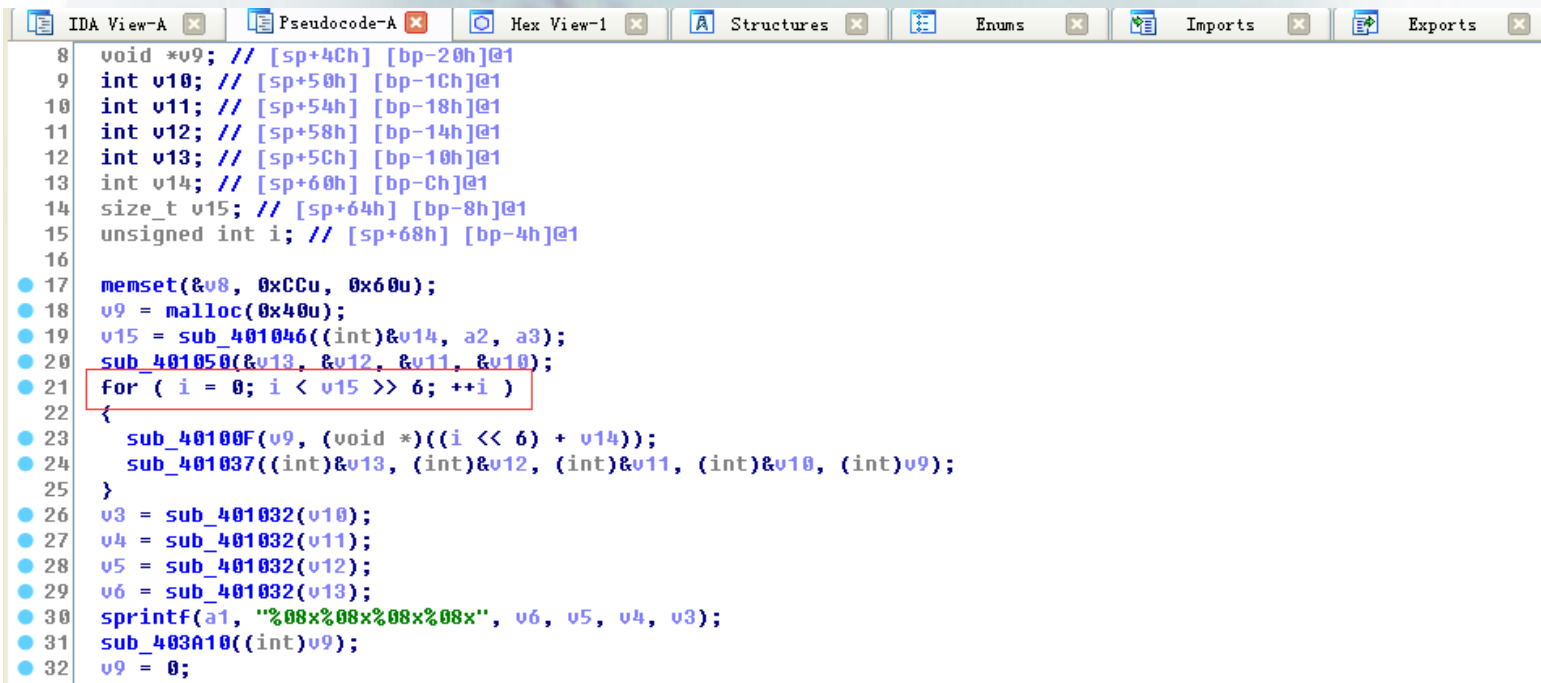
C = 0x98badcfe;

D = 0x10325476;



1. MD5算法

- ❑ 完成赋值操作后，函数进入for循环，循环次数为 **sub_401046()** 的返回值右移6位(相当于除以64，就是对每个分组进行加密)，上文分析得到 **sub_401046()** 函数的返回值为整个数据长度，这里可以看到数据的分组长度是64个字节（512bit）



```
8 void *v9; // [sp+4Ch] [bp-20h]@1
9 int v10; // [sp+50h] [bp-1Ch]@1
10 int v11; // [sp+54h] [bp-18h]@1
11 int v12; // [sp+58h] [bp-14h]@1
12 int v13; // [sp+5Ch] [bp-10h]@1
13 int v14; // [sp+60h] [bp-Ch]@1
14 size_t v15; // [sp+64h] [bp-8h]@1
15 unsigned int i; // [sp+68h] [bp-4h]@1
16
17 memset(&v8, 0xCCu, 0x60u);
18 v9 = malloc(0x40u);
19 v15 = sub_401046((int)&v14, a2, a3);
20 sub_401050(&v13, &v12, &v11, &v10);
21 for ( i = 0; i < v15 >> 6; ++i )
22 {
23     sub_40100F(v9, (void *)((i << 6) + v14));
24     sub_401037((int)&v13, (int)&v12, (int)&v11, (int)&v10, (int)v9);
25 }
26 v3 = sub_401032(v10);
27 v4 = sub_401032(v11);
28 v5 = sub_401032(v12);
29 v6 = sub_401032(v13);
30 sprintf(a1, "%08x%08x%08x%08x", v6, v5, v4, v3);
31 sub_403A10((int)v9);
32 v9 = 0;
```

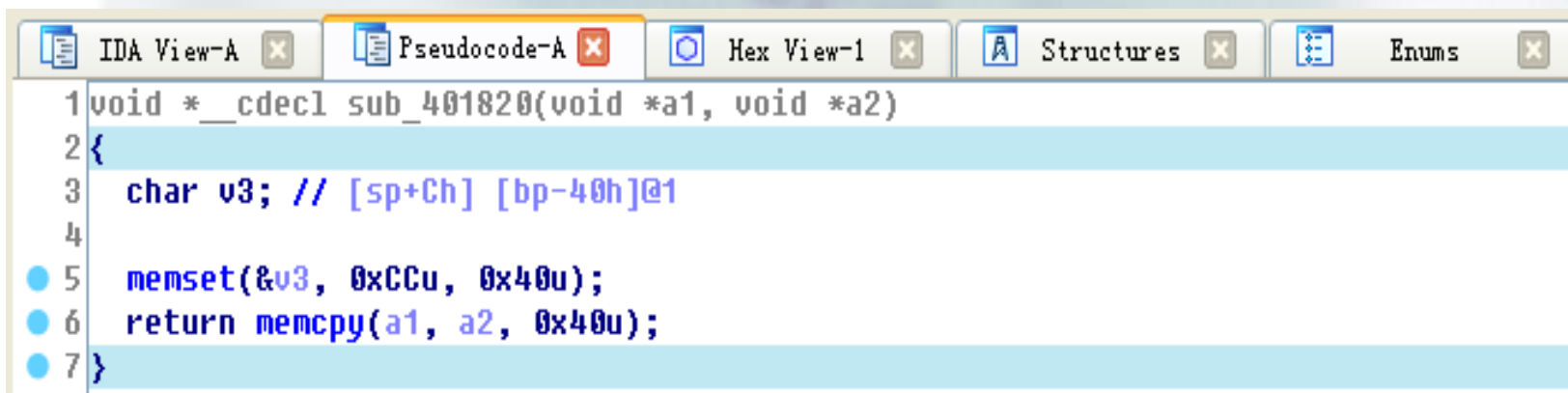


1. MD5算法

□ 首先分析for循环中的第一个函数 **sub_40100F()**。该函数比较简单，调用了 **memcpy()** 函数，将 **input** 复制到变量 **v9** 中

```
23 | sub_40100F(v9, (void *)((i << 6) + v14));
```

```
1 void *__cdecl sub_40100F(void *a1, void *a2)
2 {
3     return sub_401820(a1, a2);
4 }
```



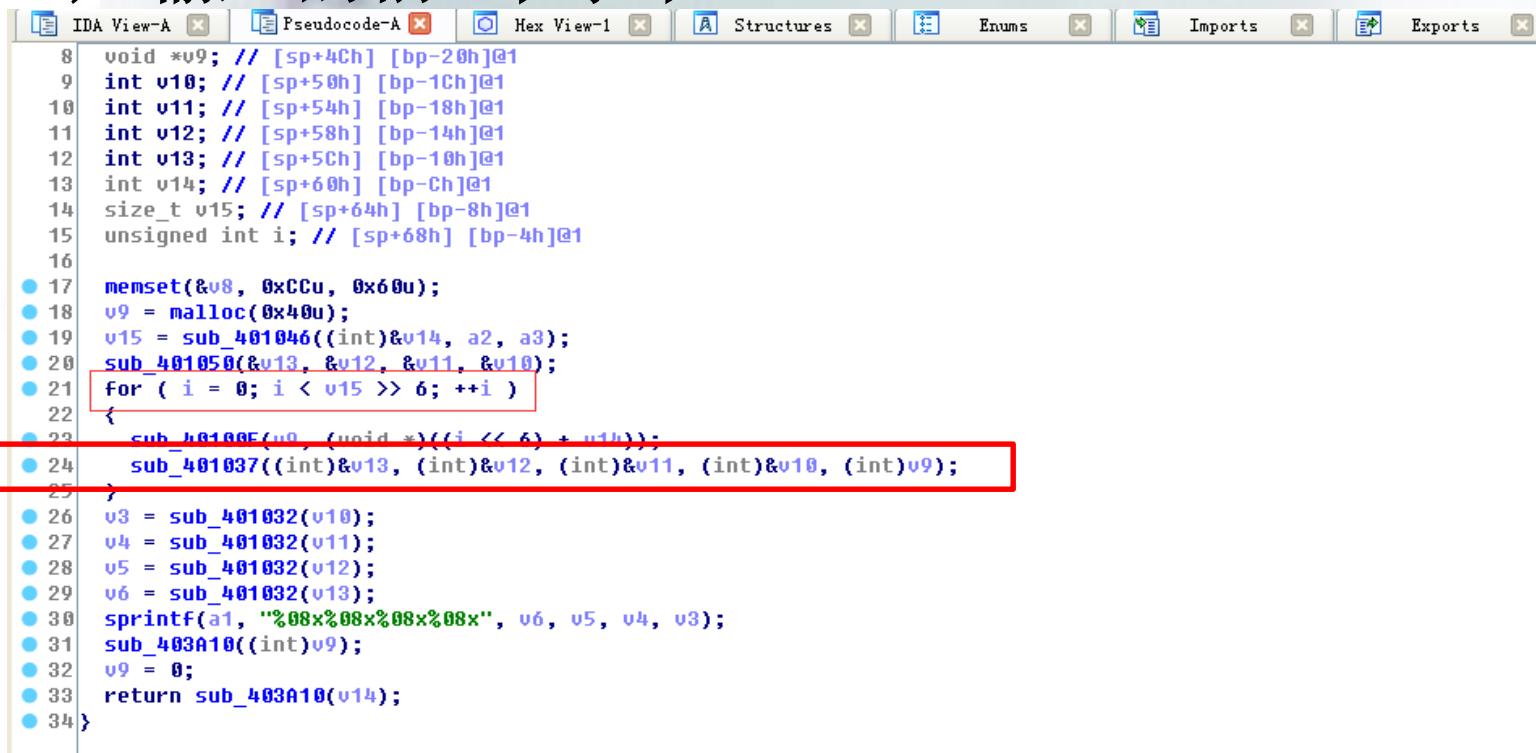
The screenshot shows the IDA Pro interface with the Pseudocode-A window active. The function sub_401820 is displayed with the following pseudocode:

```
1 void *__cdecl sub_401820(void *a1, void *a2)
2 {
3     char v3; // [sp+Ch] [bp-40h]@1
4
5     memset(&v3, 0xCCu, 0x40u);
6     return memcpy(a1, a2, 0x40u);
7 }
```



1. MD5算法

- ❑ 重点分析一下sub_401037()这个函数
- ❑ 该函数的参数为前面被赋值的4个变量以及用户输入的前4个字节



```
8 void *v9; // [sp+4Ch] [bp-20h]@1
9 int v10; // [sp+50h] [bp-1Ch]@1
10 int v11; // [sp+54h] [bp-18h]@1
11 int v12; // [sp+58h] [bp-14h]@1
12 int v13; // [sp+5Ch] [bp-10h]@1
13 int v14; // [sp+60h] [bp-Ch]@1
14 size_t v15; // [sp+64h] [bp-8h]@1
15 unsigned int i; // [sp+68h] [bp-4h]@1
16
17 memset(&v8, 0xCCu, 0x60u);
18 v9 = malloc(0x40u);
19 v15 = sub_401046((int)&v14, a2, a3);
20 sub_401050(&v13, &v12, &v11, &v10);
21 for ( i = 0; i < v15 >> 6; ++i )
22 {
23     sub_40100E(v8, (void *)((i << 6) + v14));
24     sub_401037((int)&v13, (int)&v12, (int)&v11, (int)&v10, (int)v9);
25 }
26 v3 = sub_401032(v10);
27 v4 = sub_401032(v11);
28 v5 = sub_401032(v12);
29 v6 = sub_401032(v13);
30 sprintf(a1, "%08x%08x%08x%08x", v6, v5, v4, v3);
31 sub_403A10((int)v9);
32 v9 = 0;
33 return sub_403A10(v14);
34 }
```



1. MD5算法

- ❑ 根据反编译结果可以看到，该函数对变量A~D进行了**64**轮的轮函数
 - 1~16轮调用了函数sub_401028()
 - 17~32轮调用了函数sub_401005()
 - 33~48轮调用了函数sub_40104B()
 - 49~64轮调用了函数sub_40101E()。
- ❑ 经过**64**轮的轮函数之后，变量A~D分别与初始值相加，函数返回。

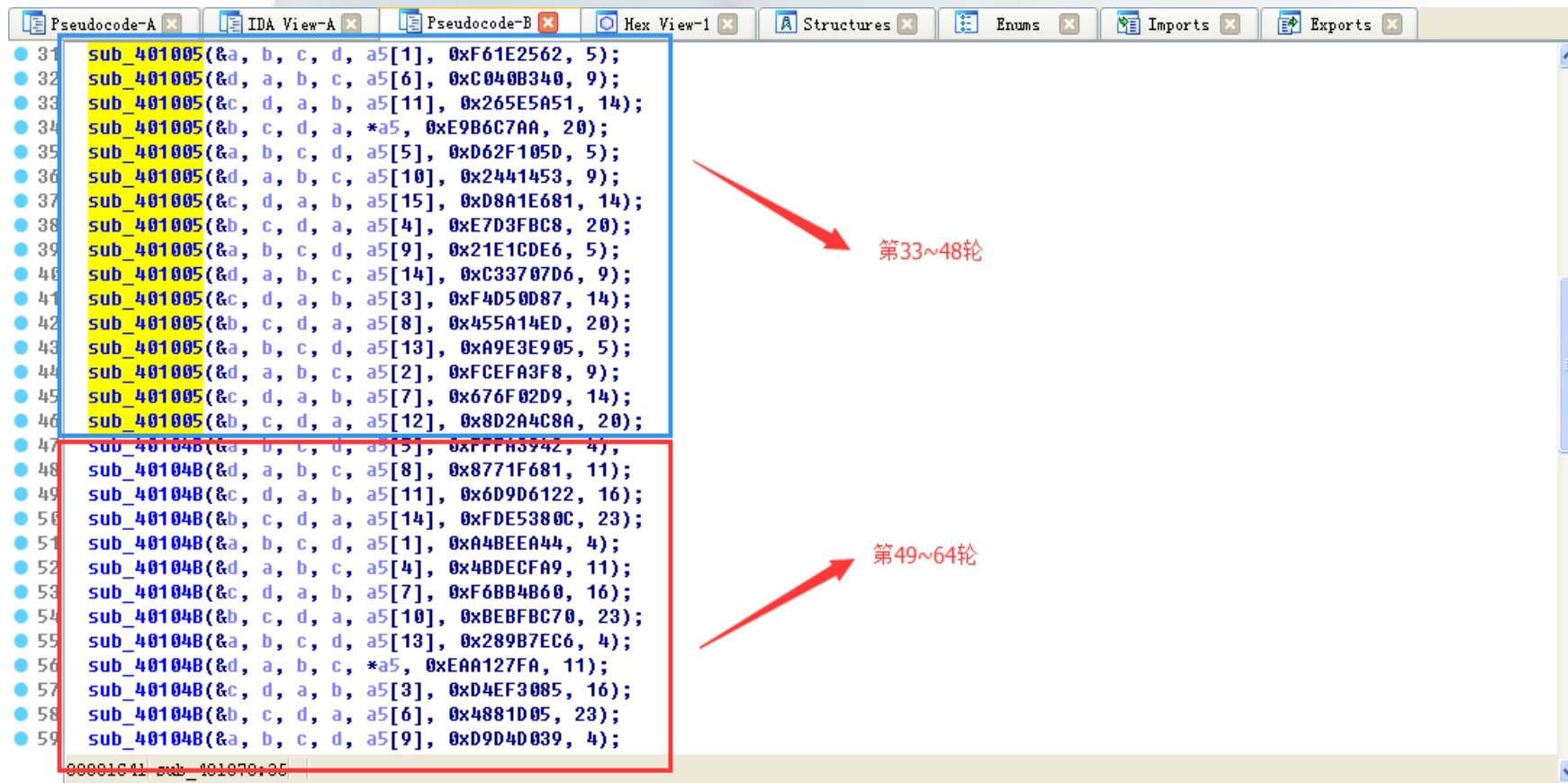


1. MD5算法

```
Pseudocode-A x Pseudocode-B x Hex View-1 x Structures x Enums x Imports x Exports x
16 sub_401028((int)&d, a, b, c, a5[1], 0xE8C7B756, 12);
17 sub_401028((int)&c, d, a, b, a5[2], 0x242070DB, 17);
18 sub_401028((int)&b, c, d, a, a5[3], 0xC1BDCEEE, 22);
19 sub_401028((int)&a, b, c, d, a5[4], 0xF57C0FAF, 7);
20 sub_401028((int)&d, a, b, c, a5[5], 0x4787C62A, 12);
21 sub_401028((int)&c, d, a, b, a5[6], 0xA8304613, 17);
22 sub_401028((int)&b, c, d, a, a5[7], 0xFD469501, 22);
23 sub_401028((int)&a, b, c, d, a5[8], 0x69809808, 7);
24 sub_401028((int)&d, a, b, c, a5[9], 0x8B44F7AF, 12);
25 sub_401028((int)&c, d, a, b, a5[10], 0xFFFF5BB1, 17);
26 sub_401028((int)&b, c, d, a, a5[11], 0x895CD7BE, 22);
27 sub_401028((int)&a, b, c, d, a5[12], 0x6B901122, 7);
28 sub_401028((int)&d, a, b, c, a5[13], 0xFD987193, 12);
29 sub_401028((int)&c, d, a, b, a5[14], 0xA679438E, 17);
30 sub_401028((int)&b, c, d, a, a5[15], 0x49B40821, 22);
31 sub_401005(&a, b, c, d, a5[1], 0x5A122582, 5);
32 sub_401005(&d, a, b, c, a5[6], 0xC040B340, 9);
33 sub_401005(&c, d, a, b, a5[11], 0x265E5A51, 14);
34 sub_401005(&b, c, d, a, *a5, 0xE9B6C7AA, 20);
35 sub_401005(&a, b, c, d, a5[5], 0xD62F105D, 5);
36 sub_401005(&d, a, b, c, a5[10], 0x2441453, 9);
37 sub_401005(&c, d, a, b, a5[15], 0xD8A1E681, 14);
38 sub_401005(&b, c, d, a, a5[4], 0xE7D3FBC8, 20);
39 sub_401005(&a, b, c, d, a5[9], 0x21E1CDE6, 5);
40 sub_401005(&d, a, b, c, a5[14], 0xC33707D6, 9);
41 sub_401005(&c, d, a, b, a5[3], 0xF4D50D87, 14);
42 sub_401005(&b, c, d, a, a5[8], 0x455A14ED, 20);
43 sub_401005(&a, b, c, d, a5[13], 0xA9E3E905, 5);
44 sub_401005(&d, a, b, c, a5[2], 0xFCF8A3F8, 9);
00001A2E sub_401070:23
```



1. MD5算法



```
31 sub_401005(&a, b, c, d, a5[1], 0xF61E2562, 5);
32 sub_401005(&d, a, b, c, a5[6], 0xC040B340, 9);
33 sub_401005(&c, d, a, b, a5[11], 0x265E5A51, 14);
34 sub_401005(&b, c, d, a, *a5, 0xE9B6C7AA, 20);
35 sub_401005(&a, b, c, d, a5[5], 0xD62F105D, 5);
36 sub_401005(&d, a, b, c, a5[10], 0x2441453, 9);
37 sub_401005(&c, d, a, b, a5[15], 0xD8A1E681, 14);
38 sub_401005(&b, c, d, a, a5[4], 0xE7D3FBC8, 20);
39 sub_401005(&a, b, c, d, a5[9], 0x21E1CDE6, 5);
40 sub_401005(&d, a, b, c, a5[14], 0xC33707D6, 9);
41 sub_401005(&c, d, a, b, a5[3], 0xF4D50D87, 14);
42 sub_401005(&b, c, d, a, a5[8], 0x455A14ED, 20);
43 sub_401005(&a, b, c, d, a5[13], 0xA9E3E905, 5);
44 sub_401005(&d, a, b, c, a5[2], 0xFCEFA3F8, 9);
45 sub_401005(&c, d, a, b, a5[7], 0x676F02D9, 14);
46 sub_401005(&b, c, d, a, a5[12], 0x8D2A4C8A, 20);
47 sub_401048(&a, b, c, d, a5[5], 0xFFFFA3942, 4);
48 sub_401048(&d, a, b, c, a5[8], 0x8771F681, 11);
49 sub_401048(&c, d, a, b, a5[11], 0x6D9D6122, 16);
50 sub_401048(&b, c, d, a, a5[14], 0xFDE5380C, 23);
51 sub_401048(&a, b, c, d, a5[1], 0xA4BEEA44, 4);
52 sub_401048(&d, a, b, c, a5[4], 0x4BDECF A9, 11);
53 sub_401048(&c, d, a, b, a5[7], 0xF6BB4B60, 16);
54 sub_401048(&b, c, d, a, a5[10], 0xBEBFC70, 23);
55 sub_401048(&a, b, c, d, a5[13], 0x289B7EC6, 4);
56 sub_401048(&d, a, b, c, *a5, 0xEAA127FA, 11);
57 sub_401048(&c, d, a, b, a5[3], 0xD4EF3085, 16);
58 sub_401048(&b, c, d, a, a5[6], 0x4881D05, 23);
59 sub_401048(&a, b, c, d, a5[9], 0xD9D4D039, 4);
```

第33~48轮

第49~64轮



1. MD5算法

- ❑ 每一轮轮函数的第6个参数均为常量值，经对比发现，这64个常量值与MD5算法中的64个加法常数相对应，第7个参数和MD5算法的移位的位数相对应。

```
31 sub_401005(&a, b, c, d, a5[1], 0xF61E2562, 5);
32 sub_401005(&d, a, b, c, a5[6], 0xC040B340, 9);
33 sub_401005(&c, d, a, b, a5[11], 0x265E5A51, 14);
34 sub_401005(&b, c, d, a, *a5, 0xE9B6C7AA, 20);
35 sub_401005(&a, b, c, d, a5[5], 0xD62F105D, 5);
36 sub_401005(&d, a, b, c, a5[10], 0x2441453, 9);
37 sub_401005(&c, d, a, b, a5[15], 0xD8A1E681, 14);
38 sub_401005(&b, c, d, a, a5[4], 0xE7D3FBC8, 20);
39 sub_401005(&a, b, c, d, a5[9], 0x21E1CDE6, 5);
40 sub_401005(&d, a, b, c, a5[14], 0xC33707D6, 9);
41 sub_401005(&c, d, a, b, a5[3], 0xF4D50D87, 14);
42 sub_401005(&b, c, d, a, a5[8], 0x455A14ED, 20);
43 sub_401005(&a, b, c, d, a5[13], 0xA9E3E905, 5);
44 sub_401005(&d, a, b, c, a5[2], 0xFCE5A2F8, 9);
45 sub_401005(&c, d, a, b, a5[7], 0x676F02D9, 14);
46 sub_401005(&b, c, d, a, a5[12], 0x8D2A4C8A, 20);
47 sub_401048(&a, b, c, d, a5[3], 0xFFFA3942, 4);
48 sub_401048(&d, a, b, c, a5[8], 0x8771F681, 11);
49 sub_401048(&c, d, a, b, a5[11], 0x6D9D6122, 16);
50 sub_401048(&b, c, d, a, a5[14], 0xFDE5380C, 23);
51 sub_401048(&a, b, c, d, a5[1], 0xA4BEEA44, 4);
52 sub_401048(&d, a, b, c, a5[4], 0x4BDECF09, 11);
53 sub_401048(&c, d, a, b, a5[7], 0xF6BB4B60, 16);
54 sub_401048(&b, c, d, a, a5[10], 0xBEBFBC70, 23);
55 sub_401048(&a, b, c, d, a5[13], 0x289B7EC6, 4);
56 sub_401048(&d, a, b, c, *a5, 0xEAA127FA, 11);
57 sub_401048(&c, d, a, b, a5[3], 0xD4EF3085, 16);
58 sub_401048(&b, c, d, a, a5[6], 0x4881D05, 23);
59 sub_401048(&a, b, c, d, a5[9], 0xD9D4D039, 4);
```

第33~48轮

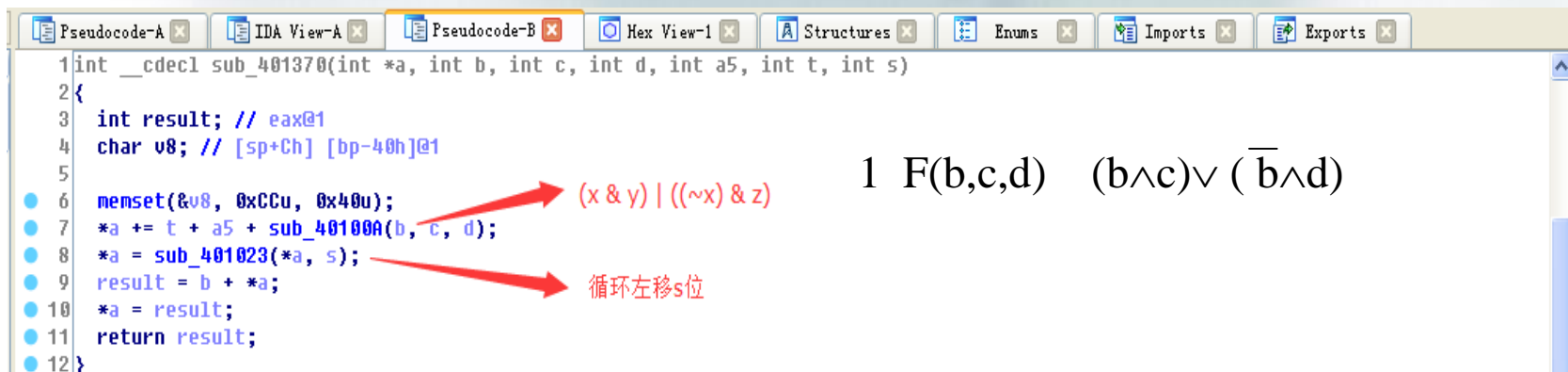
第49~64轮

1. MD5算法

□ 为了进一步验证该程序的核心算法是MD5算法的猜想，可以选择一个轮函数进行分析，比如 **sub_401028** 函数，与MD5算法中循环运算中的第一个轮函数相对应

FF(a,b,c,d,Mj,Ti,s)表示 $a = b + ((a + F(b,c,d) + Mj + Ti) \ll s)$

```
1 int __cdecl sub_401028(int a1, int a2, int a3, int a4, int a5, int a6, int a7)
2 {
3     return sub_401370(a1, a2, a3, a4, a5, a6, a7);
4 }
```



```
1 int __cdecl sub_401370(int *a, int b, int c, int d, int a5, int t, int s)
2 {
3     int result; // eax@1
4     char v8; // [sp+Ch] [bp-40h]@1
5
6     memset(&v8, 0xCCu, 0x40u);
7     *a += t + a5 + sub_40100A(b, c, d);
8     *a = sub_401023(*a, s);
9     result = b + *a;
10    *a = result;
11    return result;
12 }
```

1 $F(b,c,d) \quad (b \wedge c) \vee (\bar{b} \wedge d)$

循环左移s位

MD5 Compression Function

每一轮包含对缓冲区ABCD的16步操作所组成的一个序列。

$$a \leftarrow b + ((a + g(b,c,d) + X[k] + T[i]) \lll s)$$

其中，

a, b, c, d = 缓冲区的四个字，以一个给定的次序排列；

g = 基本逻辑函数F,G,H,I之一；

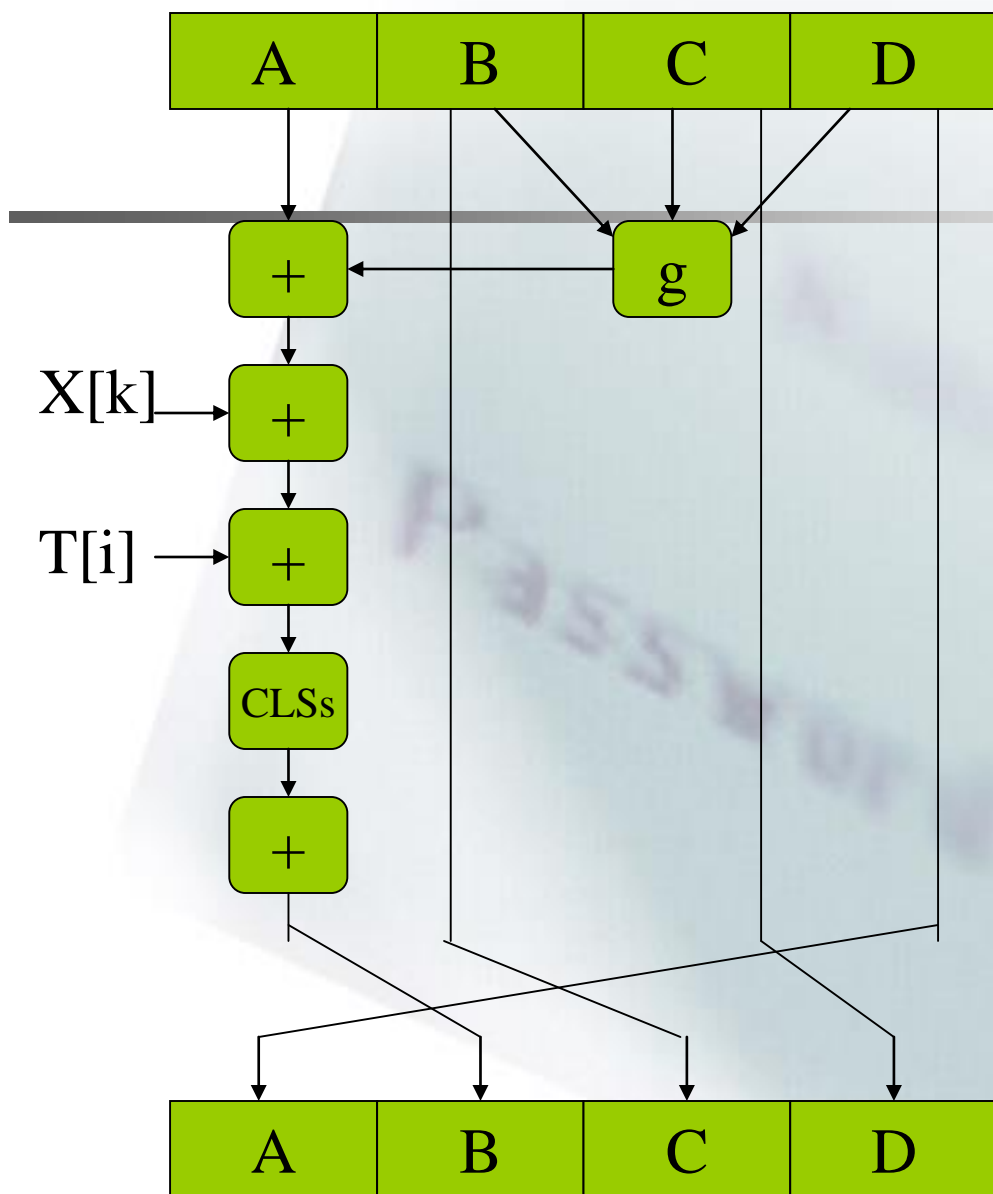
$\lll s$ = 对32位字循环左移s位

$X[k]$ = $M[q \times 16 + k]$ = 在第q个512位数据块中的第k个32位字

$T[i]$ = 表T中的第i个32位字；

$+$ = 模 2^{32} 的加；





Function g	$g(b,c,d)$
1 F(b,c,d)	$(b \wedge c) \vee \overline{(b \wedge d)}$
2 G(b,c,d)	$(b \wedge d) \vee (\overline{c} \wedge d)$
3 H(b,c,d)	$b \oplus c \oplus d$
4 I(b,c,d)	$c \oplus (b \vee d)$

$$\rho_2 i = (1 + 5i) \bmod 16$$

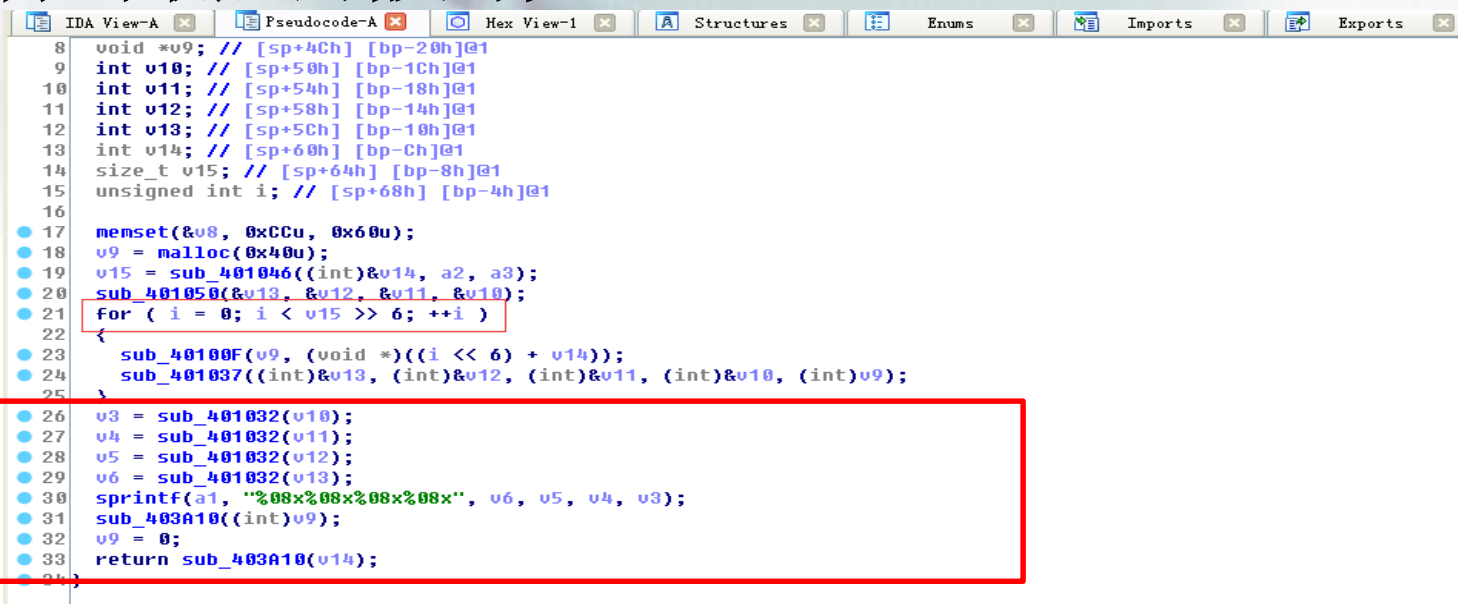
$$\rho_3 i = (5 + 3i) \bmod 16$$

$$\rho_2 i = 7i \bmod 16$$



1. MD5算法

□ 分析至此，无论是数据填充还是轮函数，可以确定改程序的核心算法是MD5算法，完成轮函数的计算后，程序又调用了函数sub_401032()对4个变量进行了移位操作，然后再进行链接得到最后的散列值。



The screenshot shows the IDA Pro interface with the Pseudocode-A view. The code defines a function that initializes variables v9 through v15, then enters a loop where it calls sub_40100F and sub_401037. After the loop, it calls sub_401032 on v10, v11, v12, and v13, then prints a string with these variables, and finally returns sub_403A10(v14). A red box highlights the sub_401032 calls and the final return statement.

```
8 void *v9; // [sp+4Ch] [bp-20h]@1
9 int v10; // [sp+50h] [bp-1Ch]@1
10 int v11; // [sp+54h] [bp-18h]@1
11 int v12; // [sp+58h] [bp-14h]@1
12 int v13; // [sp+5Ch] [bp-10h]@1
13 int v14; // [sp+60h] [bp-Ch]@1
14 size_t v15; // [sp+64h] [bp-8h]@1
15 unsigned int i; // [sp+68h] [bp-4h]@1
16
17 memset(&v8, 0xCCu, 0x60u);
18 v9 = malloc(0x40u);
19 v15 = sub_401046((int)&v14, a2, a3);
20 sub_401050(&v13, &v12, &v11, &v10);
21 for ( i = 0; i < v15 >> 6; ++i )
22 {
23     sub_40100F(v9, (void *)((i << 6) + v14));
24     sub_401037((int)&v13, (int)&v12, (int)&v11, (int)&v10, (int)v9);
25 }
26 v3 = sub_401032(v10);
27 v4 = sub_401032(v11);
28 v5 = sub_401032(v12);
29 v6 = sub_401032(v13);
30 sprintf(a1, "%08x%08x%08x%08x", v6, v5, v4, v3);
31 sub_403A10((int)v9);
32 v9 = 0;
33 return sub_403A10(v14);
```


1. MD5算法

- ❑ 由输入的前4个字节计算得出的散列值与字符串“**fae8a9257e154175da4193dbf6552ef6**”进行比较，然后根据**strncmp()**的比较结果输出**Correct**或**Wrong**。由于参与计算散列值的字符串长度只有4个字节，因此可以通过暴力破解的方式得到正确的**flag**。



1. MD5算法

IDA - C:\Documents and Settings\Administrator\桌面\课内练习\第5章\第五章\随书代码-2\5-3\程序\MD5.exe

File Edit Jump Search View Debugger Options Windows Help

No debugger

Library function Data Regular function Unexplored Instruction External symbol

Functions window

Function name

sub_401005

sub_40100A

sub_40100F

sub_401014

sub_401019

sub_40101E

sub_401023

sub_401028

sub_40102D

sub_401032

sub_401037

main

sub_401041

sub_401046

sub_40104B

sub_401050

main_0

sub_401220

sub_401270

sub_4012B0

sub_4012F0

sub_401330

sub_401370

sub_401400

sub_401490

sub_401520

sub_4015B0

sub_401630

sub_401710

sub_401820

sub_401870

sub_4026F0

__system

__strncmp

__strcpy

__strlen

__scanf

__printf

__chkexp

__memcpy

__malloc

__malloc_dbg

__nh_malloc

__nh_malloc_dbg

__heap_alloc

Line 1 of 253

Output window

42DC80: using guessed type int dword_42DC80;

42DC84: using guessed type int dword_42DC84;

Python

AU: idle Up Disk: 35GB

开始 程序 IDA_Pro_v6.8 程序 IDA - C:\Do...

12:20

401014()函数将用户输入数据v3的前4个字节计算出HASH值，保存在v9

即计算4个字节的hash值是32个字符（128bit）的
“fae8a9257e154175da4193dbf6552ef6”

1. MD5算法

@练习:

❑ 逆向MD5.EXE, 提交flag。

