AES算法分析

主逻辑

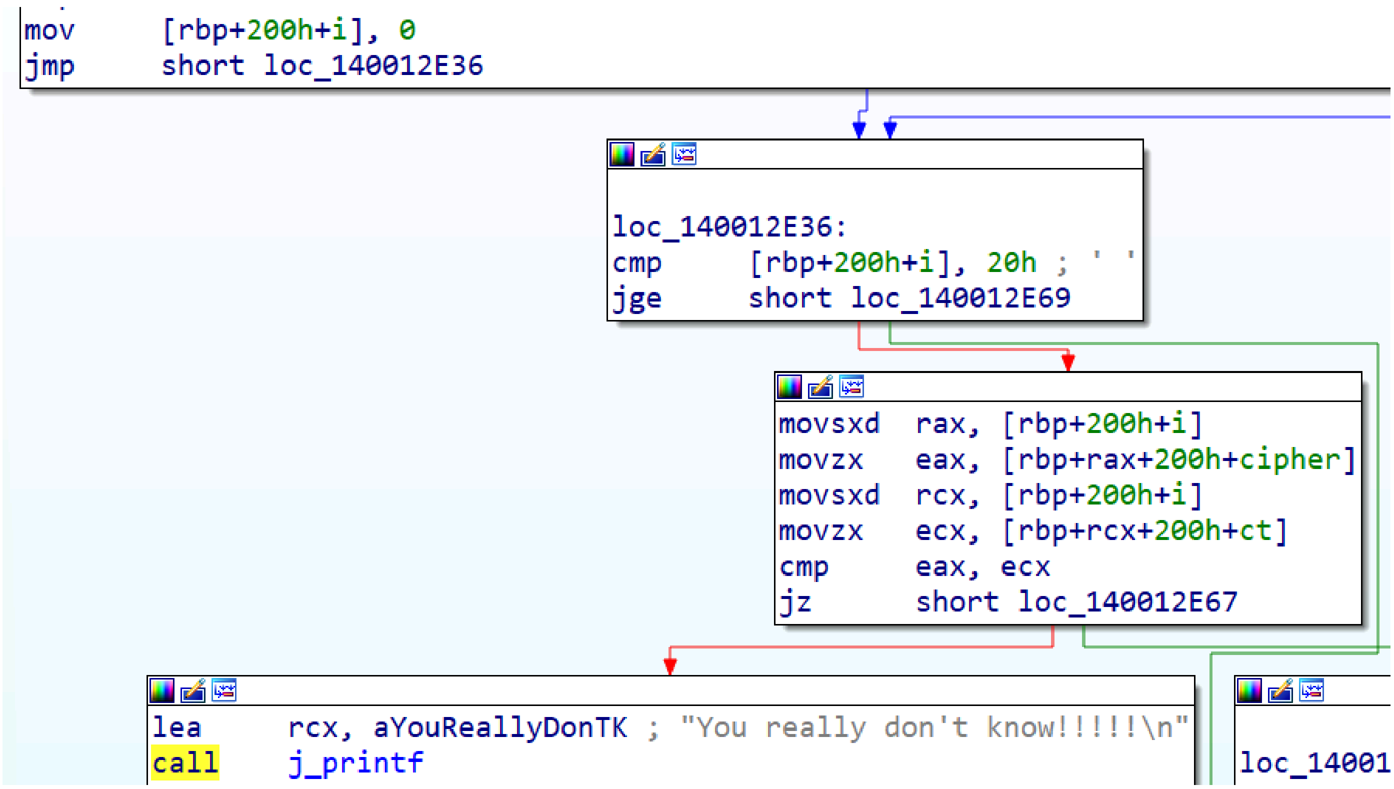
1. 首先定义 `data` 、 `key` 、 `cipher` 等一系列基础数据；

```
lea     rax, [rbp+200h+key]
lea     rcx, a12345678901234 ; "1234567890123456"
mov     rdi, rax
mov     rsi, rcx
mov     ecx, 11h
rep movsb
lea     rax, [rbp+200h+data]
lea     rcx, aAbcdefghijklmn ; "abcdefghijklmnopqrstuvwxyzabcdef"
mov     rdi, rax
mov     rsi, rcx
mov     ecx, 21h ; '!'
rep movsb
lea     rax, [rbp+200h+data+21h]
mov     rdi, rax
xor     eax, eax
mov     ecx, 7
rep stosb
mov     [rbp+200h+cipher], 0FCh
mov     [rbp+200h+cipher+1], 0ADh
mov     [rbp+200h+cipher+2], 71h ; 'q'
mov     [rbp+200h+cipher+3], 5Bh ; '['
mov     [rbp+200h+cipher+4], 0D7h
mov     [rbp+200h+cipher+5], 3Bh ; ';'
mov     [rbp+200h+cipher+6], 5Ch ; '\'
mov     [rbp+200h+cipher+7], 0B0h
mov     [rbp+200h+cipher+8], 48h ; 'H'
mov     [rbp+200h+cipher+9], 8Fh
mov     [rbp+200h+cipher+0Ah], 84h
mov     [rbp+200h+cipher+0Bh], 0Fh
mov     [rbp+200h+cipher+0Ch], 3Bh ; ';'
mov     [rbp+200h+cipher+0Dh], 0ADh
mov     [rbp+200h+cipher+0Eh], 78h ; 'x'
mov     [rbp+200h+cipher+0Fh], 89h
mov     [rbp+200h+cipher+10h], 0D0h
mov     [rbp+200h+cipher+11h], 0E7h
```

2. 调用 `aesEncrypt()` ，参数为 `len(32)` 、 `ct` 、 `data` 、 `16` 和 `key` ；

```
mov     [rsp+230h+len], 20h ; ' ' ; len
lea     r9, [rbp+200h+ct] ; ct
lea     r8, [rbp+200h+data] ; pt
mov     edx, 10h        ; keyLen
lea     rcx, [rbp+200h+key] ; key
call    j_?aesEncrypt@@YAHPEBEI0PEAEI@Z
```

3. 循环32轮遍历 `ct` 和 `cipher` 进行比较，相同则通关。



aesEncrypt() 分析

1. 初始化变量：

```
mov     rax, [rbp+430h+ct]
mov     [rbp+430h+pos], rax
lea     rax, [rbp+430h+aesKey]
mov     [rbp+430h+rk], rax
lea     rax, [rbp+430h+out]
mov     rdi, rax
xor     eax, eax
mov     ecx, 10h
rep stosb
lea     rax, [rbp+430h+actualKey]
mov     rdi, rax
xor     eax, eax
mov     ecx, 10h
rep stosb
lea     rax, [rbp+430h+state]
mov     rdi, rax
xor     eax, eax
mov     ecx, 10h
rep stosb
```

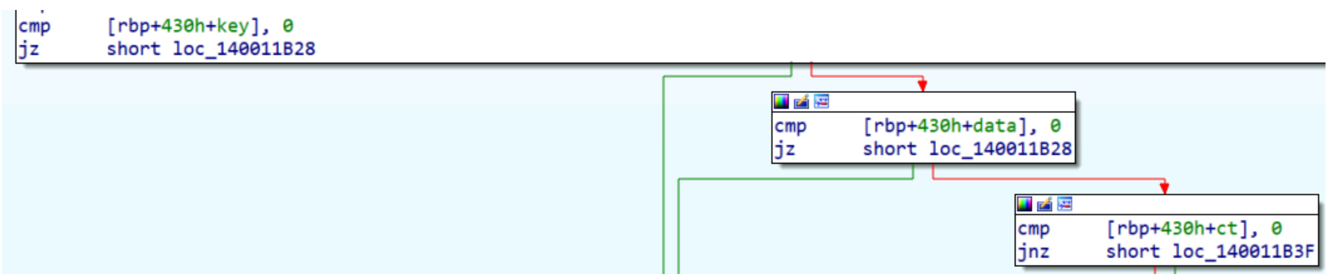
- ct （密文） 赋值给 pos ；
- aesKey 赋值给 rk ；

- `out`（详细解释下，后续类似的不细说）：
    - `lea rax, [rbp+430h+out]`：将之前定义到某个地址的 `out` 的地址存入 `rax`；
    - `mov rdi, rax`：再放入 `rax`；
    - `xor eax, eax`： `eax` 清零；
    - `mov ecx, 10h`： `ecx` 赋值为16；
    - `rep stosb`：重复将 `al` 的值（0）写入 `[rdi]` （`out` 的值），每次递增 `rdi`，重复ecx次（16次），用于将out缓冲区清零（共16字节）。

```
lea     rax, [rbp+430h+out]
mov     rdi, rax
xor     eax, eax
mov     ecx, 10h
rep stosb
```

- `actualKey`：16字节数组，存具体的加密密钥；
- `state`：16字节数组，存状态数组。

2. 一系列参数限制：
    - `key`、`data`、`ct` 不为空；



    - `keyLen` 即密钥长不大于16；

```
cmp     dword ptr [rbp+430h+keyLen], 10h
jbe     short loc_140011B5F
```

    - `len` 即明文长度必须是16的倍数。

```
xor     edx, edx
mov     eax, [rbp+430h+len]
mov     ecx, 10h
div     ecx
mov     eax, edx
test    eax, eax
jz      short loc_140011B8B
```

3. 密钥扩展过程，首先将 `key` 复制到 `actualKey` 里，然后将 `aesKey(out)`、`10h` 和 `actualKey` 作为参数，传入 `keyExpansion()` 完成密钥扩展；

```
loc_140011B8B:
mov     eax, dword ptr [rbp+430h+keyLen]
mov     r8d, eax        ; Size
mov     rdx, [rbp+430h+key] ; Src
lea     rcx, [rbp+430h+actualKey] ; void *
call    j_memcpy_0
nop
lea     r8, [rbp+430h+aesKey] ; aesKey
mov     edx, 10h        ; keyLen
lea     rcx, [rbp+430h+actualKey] ; key
call    j_?keyExpansion@@YAHPEBEIPEAUAesKey@@@Z ; keyExpansion(uchar const *,uint,AesKey *)
nop
```

4. 逐字节进行加密，即分组加密，每个分组大小为16字节，并进行初始化，使用 `loadStateArray()` 加载状态矩阵，并进行一次密钥加（ `addRoundKey()` ）；

```
loc_140011BD0:
mov     eax, [rbp+430h+len]
cmp     [rbp+430h+i], eax
jnb     loc_140011CEA
```

```
; "inLen is invalid.\n"
```

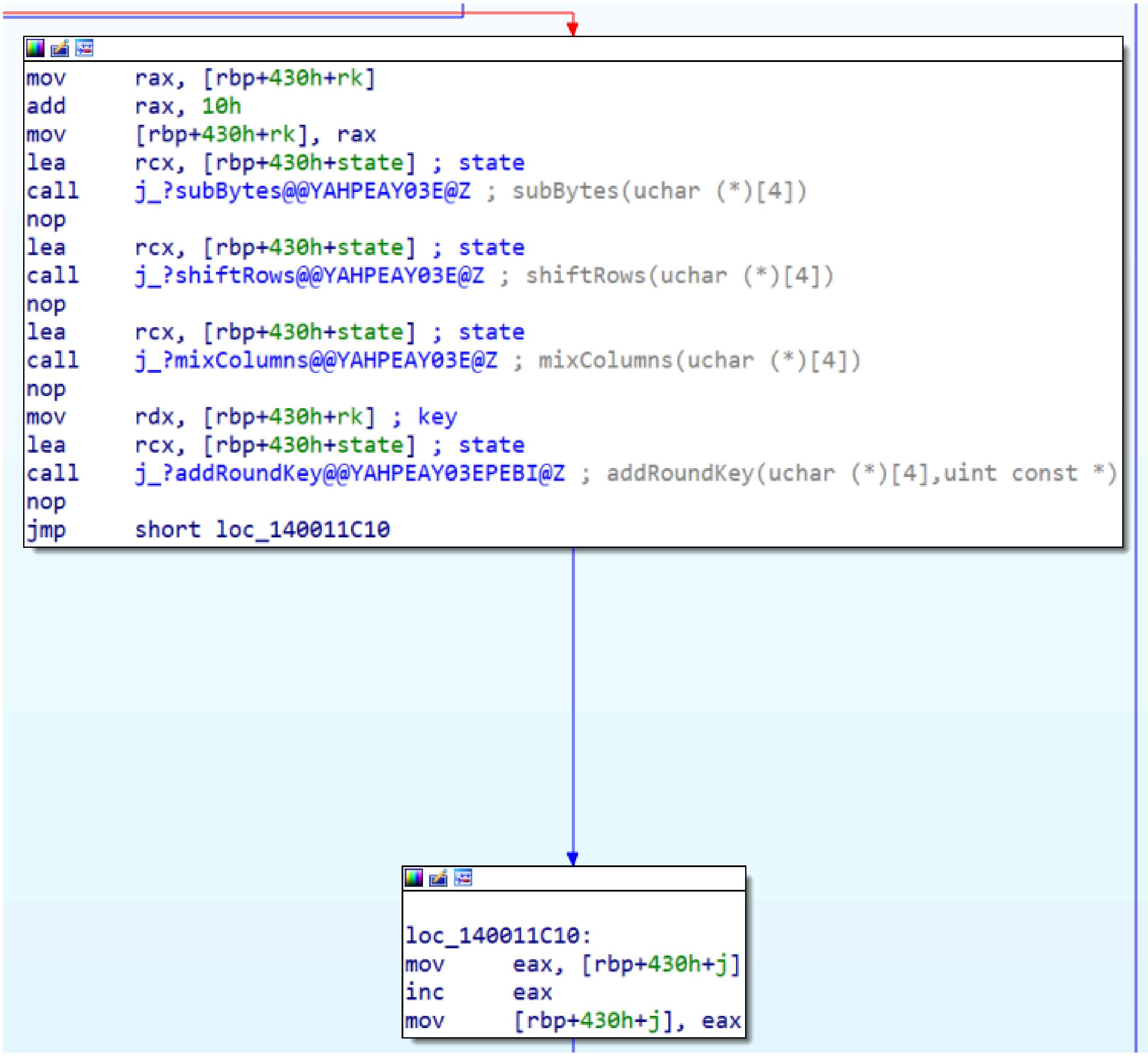
```
mov     rdx, [rbp+430h+pt] ; in
lea     rcx, [rbp+430h+state] ; state
call    j_?loadStateArray@@YAHPEAY03EPEBE@Z ; loadStateArray(uchar (*)[4],uchar const *)
nop
mov     rdx, [rbp+430h+rk] ; key
lea     rcx, [rbp+430h+state] ; state
call    j_?addRoundKey@@YAHPEAY03EPEBI@Z ; addRoundKey(uchar (*)[4],uint const *)
nop
mov     [rbp+430h+j], 1
jmp     short loc_140011C18
```

5. 每次加密循环10轮（ `j` 从1到10）：

```
mov     [rbp+430h+j], 1
jmp     short loc_140011C18
```

```
loc_140011C18:
cmp     [rbp+430h+j], 0Ah
jge     short loc_140011C6D
```

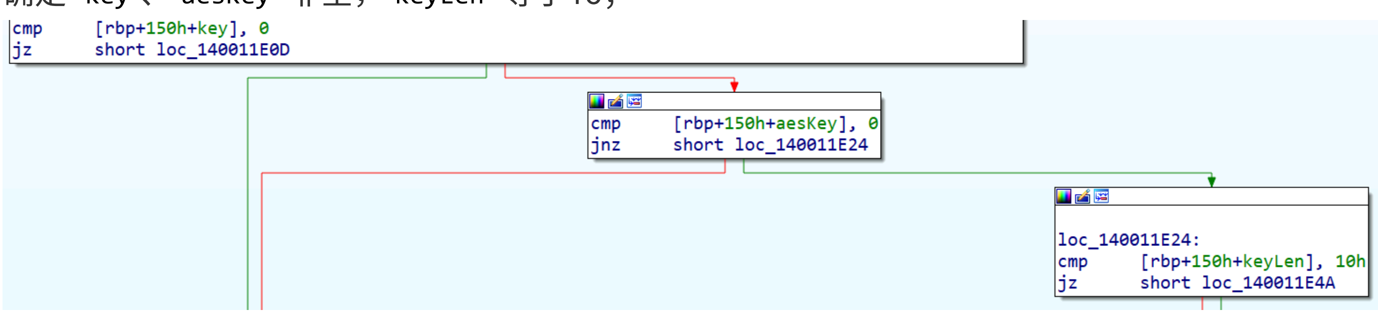
- 前9轮：首先取密钥到 `rk` ，然后依次进行 `subBytes()` 、 `shiftRows()` 、 `mixColumns()` 和 `addRoundKey()` 过程；



- 第10轮：不进行 `mixColumns()` ，最后使用 `storeStateArray()` 将状态矩阵存入 `pos(out)` 。

`keyExpansion()` 分析

1. 确定 `key` 、 `aesKey` 非空， `keyLen` 等于16；



2. 后续有 `aesKey` 加一个值表示某个变量，不知道实在干什么，所以找到 `aesKey` 最开始的定义，发现其数据类型为 `AesKey` 这样一个结构体，数据结构如下，通过加某一个值可以找到相应属性；

```
00000000 AesKey          struc ; (sizeof=0x164, align=0x4, copyof_485)
00000000                                         ; XREF: ?aesEncrypt@@YAHPEBEI0PEAEI@Z/r
00000000 eK              dd 44 dup(?)
000000B0 dK              dd 44 dup(?)
00000160 Nr              dd ?
00000164 AesKey          ends
```

3. 将 `ek` （加密密钥）存入 `w` ， `dk` （解密密钥）存入 `v` ；

```
loc_140011E4A:
mov     rax, [rbp+150h+aesKey]
mov     [rbp+150h+w], rax
mov     rax, [rbp+150h+aesKey]
add     rax, 0B0h
mov     [rbp+150h+v], rax
mov     [rbp+150h+i], 0
jmp     short loc_140011E77
```

4. `i` 从0到3循环4轮，实现小端序转大端序，具体流程为取出 `key[4*i]` ，先取低8位，再取低16-9位分别放到高8位和高9-16位，再将高16位字节交换放到低16位，最后放入 `w` 中，完成初始化读入；

```
loc_140011E81:
mov     eax, [rbp+150h+i]
shl     eax, 2
cdqe
mov     rcx, [rbp+150h+key]
add     rcx, rax
mov     rax, rcx
mov     ecx, 1
imul    rcx, 0
movzx   eax, byte ptr [rax+rcx]
and     eax, 0FFh
shl     eax, 18h
mov     ecx, [rbp+150h+i]
shl     ecx, 2
movsxd  rcx, ecx
mov     rdx, [rbp+150h+key]
add     rdx, rcx
mov     rcx, rdx
mov     edx, 1
imul    rdx, 1
movzx   ecx, byte ptr [rcx+rdx]
and     ecx, 0FFh
shl     ecx, 10h
or      eax, ecx
mov     ecx, [rbp+150h+i]
shl     ecx, 2
movsxd  rcx, ecx
mov     rdx, [rbp+150h+key]
add     rdx, rcx
mov     rcx, rdx
mov     edx, 1
imul    rdx, 2
movzx   ecx, byte ptr [rcx+rdx]
and     ecx, 0FFh
shl     ecx, 8
or      eax, ecx
mov     ecx, [rbp+150h+i]
shl     ecx, 2
movsxd  rcx, ecx
mov     rdx, [rbp+150h+key]
add     rdx, rcx
mov     rcx, rdx
mov     edx, 1
imul    rdx, 3
movzx   ecx, byte ptr [rcx+rdx]
and     ecx, 0FFh
or      eax, ecx
movsxd  rcx, [rbp+150h+i]
mov     rdx, [rbp+150h+w]
mov     [rdx+rcx*4], eax
xor     eax, eax
test    eax, eax
jnz     loc_140011E81
```

5. 十轮循环生成完整密钥（`ek`）：
    - 从 `w[3]` 开始取，每一个元素4个字节，所以计算了 `w+12`：

```
mov     eax, 4
imul    rax, 0
mov     ecx, 4
imul    rcx, 3
mov     rdx, [rbp+150h+w]
mov     ecx, [rdx+rcx]
```

    - 取出高位数第2个字节，查S盒（一维数组，直接用字节值作为索引查），再左移24位，低24位取0，即将原来的第二个字节查S盒的值作为生成的第一个字节；

```
shr     ecx, 10h
and     ecx, 0FFh
mov     ecx, ecx
lea     rdx, ?S@@3PAEA   ; uchar near * S
movzx   ecx, byte ptr [rdx+rcx]
shl     ecx, 18h
and     ecx, 0FF000000h
```

    - 后续类似上一步，将每个字节经过S盒放到合适的位置，即整体一个4字节的字（S盒处理后）按字节循环左移，结果存到 `ecx` 中；

```
mov     edx, 4
imul    rdx, 3
mov     r8, [rbp+150h+w]
mov     edx, [r8+rdx]
shr     edx, 8
and     edx, 0FFh
mov     edx, edx
lea     r8, ?S@@3PAEA    ; uchar near * S
movzx   edx, byte ptr [r8+rdx]
shl     edx, 10h
and     edx, 0FF0000h
xor     ecx, edx
mov     edx, 4
imul    rdx, 3
mov     r8, [rbp+150h+w]
mov     edx, [r8+rdx]
shr     edx, 0
and     edx, 0FFh
mov     edx, edx
lea     r8, ?S@@3PAEA    ; uchar near * S
movzx   edx, byte ptr [r8+rdx]
shl     edx, 8
and     edx, 0FF00h
xor     ecx, edx
mov     edx, 4
imul    rdx, 3
mov     r8, [rbp+150h+w]
mov     edx, [r8+rdx]
shr     edx, 18h
and     edx, 0FFh
mov     edx, edx
lea     r8, ?S@@3PAEA    ; uchar near * S
movzx   edx, byte ptr [r8+rdx]
and     edx, 0FFh
```

- 将上面处理的结果与 `w[0]` （ `rax` 值为1）、 `rcon[i*4]` 异或，存到 `eax` 中；

```
mov     rdx, [rbp+150h+w]
mov     eax, [rdx+rax]
xor     eax, ecx
movsxd  rcx, [rbp+150h+i]
lea     rdx, rcon
xor     eax, [rdx+rcx*4]
```

- 存回 `w[4]` ；

```
mov     ecx, 4
imul    rcx, 4
mov     rdx, [rbp+150h+w]
mov     [rdx+rcx], eax
```

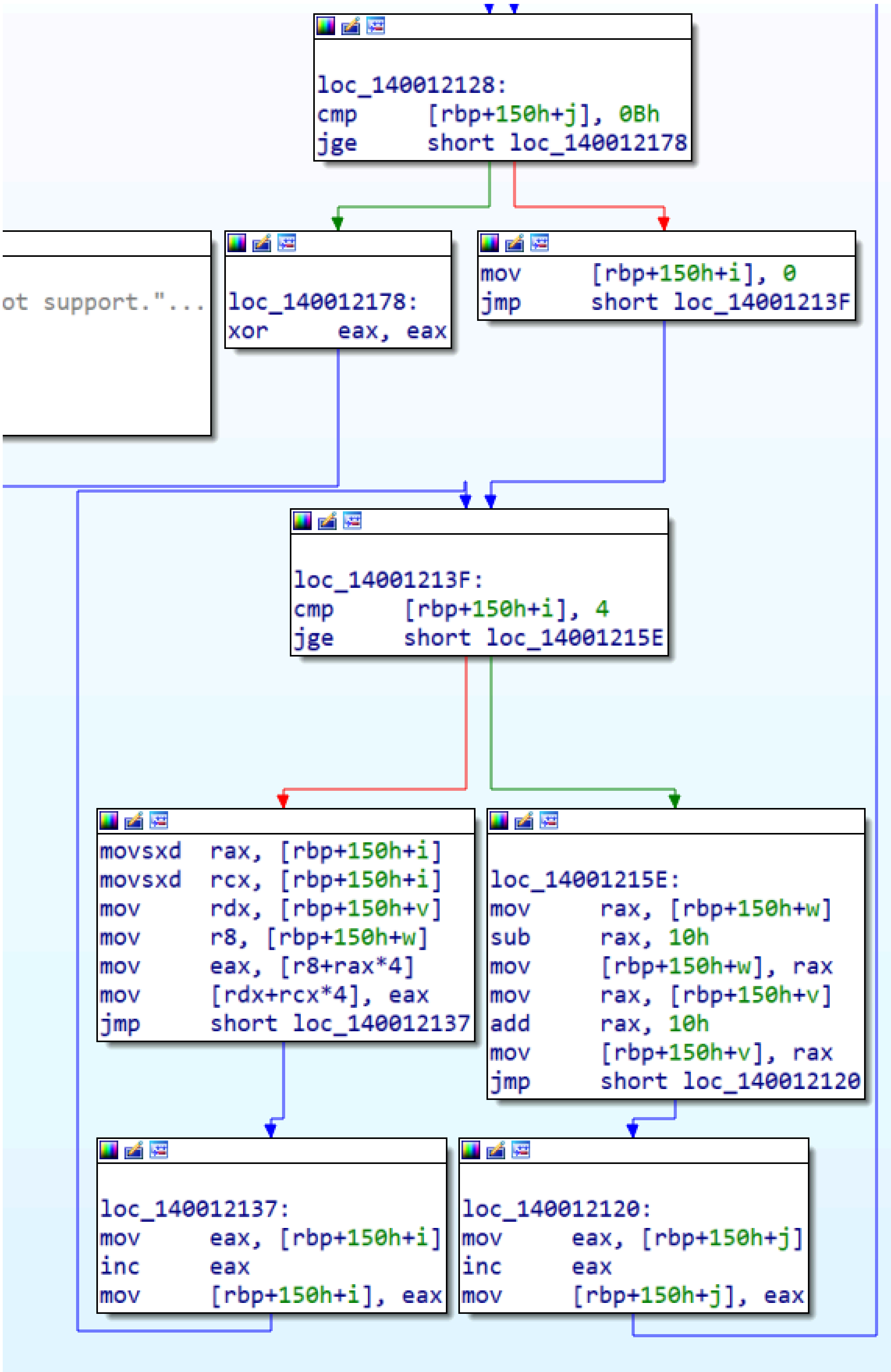
- 接下来计算 `w[5]` ，直接计算 `w[1] ^ w[4]` ，写入 `w[5]` ，后面的 `w[6]` 和 `w[7]` 同理；

```
mov     eax, 4
imul    rax, 1
mov     ecx, 4
imul    rcx, 4
mov     rdx, [rbp+150h+w]
mov     r8, [rbp+150h+w]
mov     ecx, [r8+rcx]
mov     eax, [rdx+rax]
xor     eax, ecx
mov     ecx, 4
imul    rcx, 5
mov     rdx, [rbp+150h+w]
mov     [rdx+rcx], eax
```

6. 将 `aesKey + 160` 也就是最后一个生成的密钥字（`w[11]`）写入 `w`，`j` 置0，进新循环（实际上是写入 `dk` 即解密密钥；

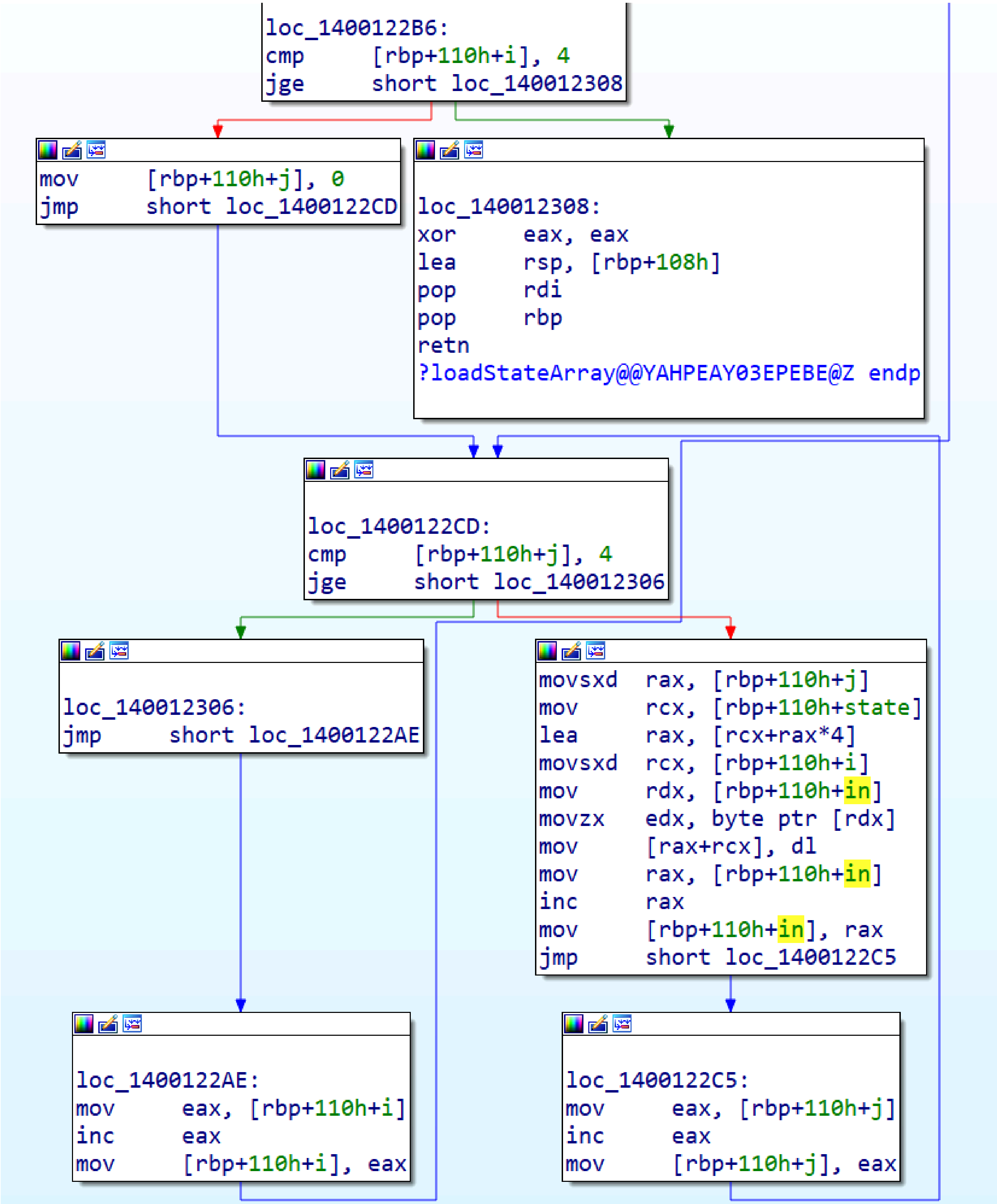
```
loc_140012106:
mov     rax, [rbp+150h+aesKey
add     rax, 0A0h
mov     [rbp+150h+w], rax
mov     [rbp+150h+j], 0
jmp     short loc_140012128
```

7. `j` 计数器循环11次，对应 `ek` 的11个密钥字，每个密钥字里面4个字节（以 `i` 计数），按字节处理，将 `w[i*4]` 放入 `v[i*4]`，完成复制后，`w` 减16，即选取 `ek` 中的上一个密钥字，`v` 加16，选取 `dk` 中的下一个密钥字存储空间，完成后退出当前函数。



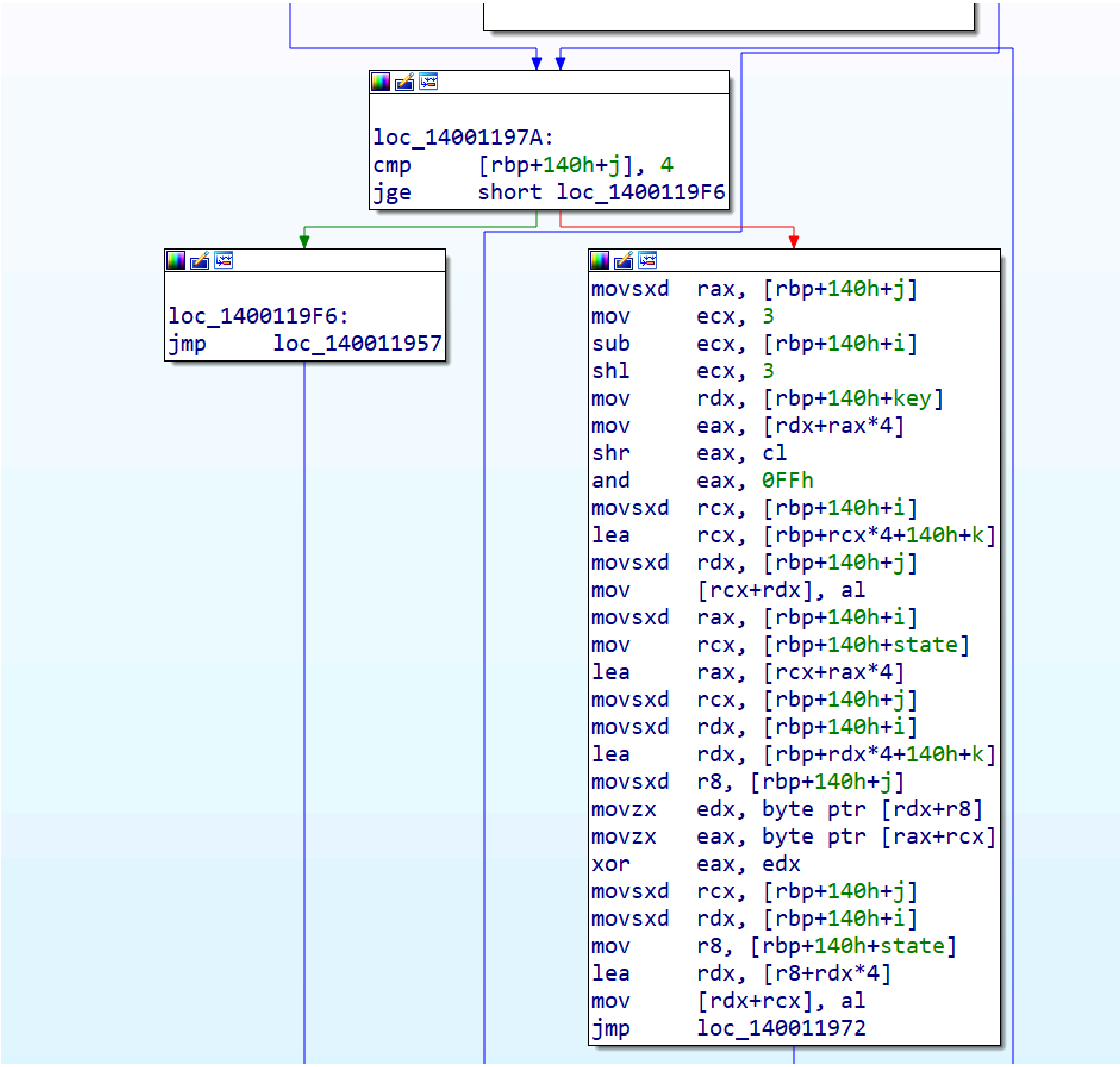
`loadStateArray()` 分析

每一列为一个字（4字节），加载状态矩阵，即 `state[j][i] = *in++`。


loc_1400122B6:
cmp [rbp+110h+i], 4
jge short loc_140012308
mov [rbp+110h+j], 0
jmp short loc_1400122CD
loc_140012308:
xor eax, eax
lea rsp, [rbp+108h]
pop rdi
pop rbp
retn
?loadStateArray@@YAHPEAY03EPEBE@Z endp
loc_1400122CD:
cmp [rbp+110h+j], 4
jge short loc_140012306
loc_140012306:
jmp short loc_1400122AE
movsxd rax, [rbp+110h+j]
mov rcx, [rbp+110h+state]
lea rax, [rcx+rax*4]
movsxd rcx, [rbp+110h+i]
mov rdx, [rbp+110h+in]
movzx edx, byte ptr [rdx]
mov [rax+rcx], dl
mov rax, [rbp+110h+in]
inc rax
mov [rbp+110h+in], rax
jmp short loc_1400122C5
loc_1400122AE:
mov eax, [rbp+110h+i]
inc eax
mov [rbp+110h+i], eax
loc_1400122C5:
mov eax, [rbp+110h+j]
inc eax
mov [rbp+110h+j], eax

addRoundKey() 分析

i 、 j 从0到3构成16轮循环，计算 key[j*4] 右移 8*(3-i) ，取低8位，然后存入 k[4*i+j] ，这是将一个4个32位字转换成16个字节的4*4二维数组 k 存储的过程（每一列是一个密钥字），然后将 state 和 k 对应位置异或存入 state 。



subBytes() 分析

i 、 j 从0到3，查 S[state[4*i+j]] 放入 state[4*i+j] ，实现字节代换。

```
movsxd  rax, [rbp+110h+i]
mov     rcx, [rbp+110h+state]
lea     rax, [rcx+rax*4]
movsxd  rcx, [rbp+110h+j]
movzx   eax, byte ptr [rax+rcx]
lea     rcx, ?S@@3PAEA  ; uchar near * S
movsxd  rdx, [rbp+110h+i]
mov     r8, [rbp+110h+state]
lea     rdx, [r8+rdx*4]
movsxd  r8, [rbp+110h+j]
movzx   eax, byte ptr [rcx+rax]
mov     [rdx+r8], al
jmp     short loc_140012A40
```

`shiftRows()` 分析

1. 大端序读取进 `block` （4个字节，所以索引是 `i*4` ）；

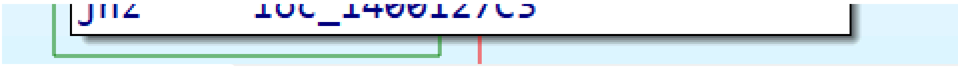
```
loc_1400126D5:
movsxd  rax, [rbp+130h+i]
mov     rcx, [rbp+130h+state]
lea     rax, [rcx+rax*4]
mov     ecx, 1
imul    rcx, 0
movzx   eax, byte ptr [rax+rcx]
and     eax, 0FFh
shl     eax, 18h
movsxd  rcx, [rbp+130h+i]
mov     rdx, [rbp+130h+state]
lea     rcx, [rdx+rcx*4]
mov     edx, 1
imul    rdx, 1
movzx   ecx, byte ptr [rcx+rdx]
and     ecx, 0FFh
shl     ecx, 10h
or      eax, ecx
movsxd  rcx, [rbp+130h+i]
mov     rdx, [rbp+130h+state]
lea     rcx, [rdx+rcx*4]
mov     edx, 1
imul    rdx, 2
movzx   ecx, byte ptr [rcx+rdx]
and     ecx, 0FFh
shl     ecx, 8
or      eax, ecx
movsxd  rcx, [rbp+130h+i]
mov     rdx, [rbp+130h+state]
lea     rcx, [rdx+rcx*4]
mov     edx, 1
imul    rdx, 3
movzx   ecx, byte ptr [rcx+rdx]
and     ecx, 0FFh
or      eax, ecx
movsxd  rcx, [rbp+130h+i]
mov     [rbp+rcx*4+130h+block], eax
xor     eax, eax
test    eax, eax
jnz     loc_1400126D5
```

2. 把 `block[i]` 左移 `i*8` 位和右移 `32-i*8` 位的结果做或运算，实现循环左移功能；（核心变换功能）

```
movsxd  rax, [rbp+130h+i]
mov     ecx, [rbp+130h+i]
shl     ecx, 3
mov     eax, [rbp+rax*4+130h+block]
shl     eax, cl
movsxd  rcx, [rbp+130h+i]
mov     [rbp+130h+var_28], rcx
mov     edx, [rbp+130h+i]
shl     edx, 3
mov     edi, 20h ; ' '
sub     edi, edx
mov     edx, edi
movzx   ecx, dl
mov     rdx, [rbp+130h+var_28]
mov     edx, [rbp+rdx*4+130h+block]
shr     edx, cl
mov     ecx, edx
or      eax, ecx
movsxd  rcx, [rbp+130h+i]
mov     [rbp+rcx*4+130h+block], eax
```

3. 把 `block` 读回 `state` 。

```
loc_1400127C3:
movsxd  rax, [rbp+130h+i]
mov     eax, [rbp+rax*4+130h+block]
shr     eax, 18h
and     eax, 0FFh
movsxd  rcx, [rbp+130h+i]
mov     rdx, [rbp+130h+state]
lea     rcx, [rdx+rcx*4]
mov     edx, 1
imul    rdx, 0
mov     [rcx+rdx], al
movsxd  rax, [rbp+130h+i]
mov     eax, [rbp+rax*4+130h+block]
shr     eax, 10h
and     eax, 0FFh
movsxd  rcx, [rbp+130h+i]
mov     rdx, [rbp+130h+state]
lea     rcx, [rdx+rcx*4]
mov     edx, 1
imul    rdx, 1
mov     [rcx+rdx], al
movsxd  rax, [rbp+130h+i]
mov     eax, [rbp+rax*4+130h+block]
shr     eax, 8
and     eax, 0FFh
movsxd  rcx, [rbp+130h+i]
mov     rdx, [rbp+130h+state]
lea     rcx, [rdx+rcx*4]
mov     edx, 1
imul    rdx, 2
mov     [rcx+rdx], al
movsxd  rax, [rbp+130h+i]
mov     eax, [rbp+rax*4+130h+block]
and     eax, 0FFh
movsxd  rcx, [rbp+130h+i]
mov     rdx, [rbp+130h+state]
lea     rcx, [rdx+rcx*4]
mov     edx, 1
imul    rdx, 3
mov     [rcx+rdx], al
xor     eax, eax
test    eax, eax
jnz     loc_1400127C3
```

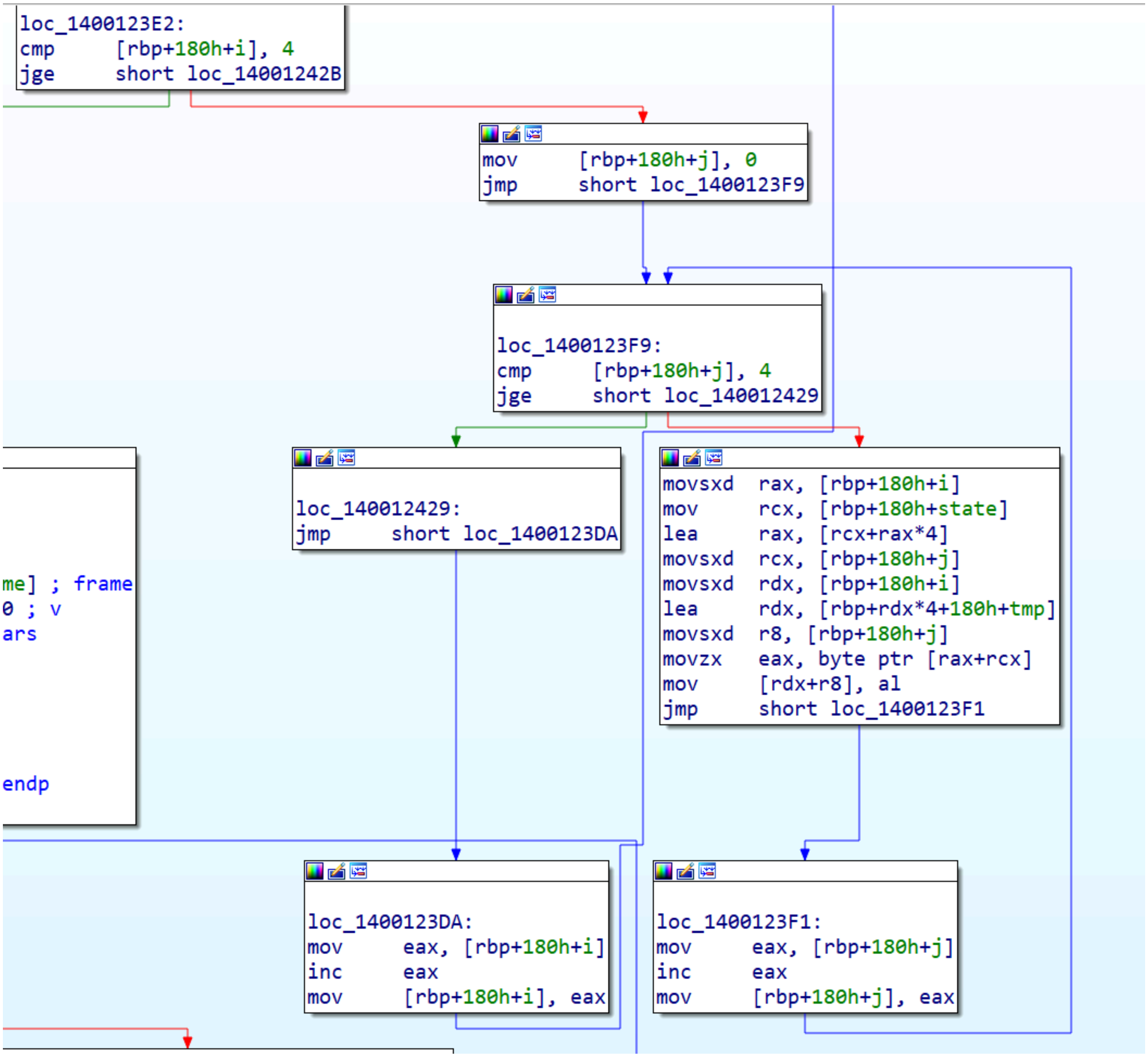


mixColumns() 分析

1. 初始化参数 `M`；

```
mov     [rbp+180h+M], 2
mov     [rbp+180h+M+1], 3
mov     [rbp+180h+M+2], 1
mov     [rbp+180h+M+3], 1
mov     [rbp+180h+M+4], 1
mov     [rbp+180h+M+5], 2
mov     [rbp+180h+M+6], 3
mov     [rbp+180h+M+7], 1
mov     [rbp+180h+M+8], 1
mov     [rbp+180h+M+9], 1
mov     [rbp+180h+M+0Ah], 2
mov     [rbp+180h+M+0Bh], 3
mov     [rbp+180h+M+0Ch], 3
mov     [rbp+180h+M+0Dh], 1
mov     [rbp+180h+M+0Eh], 1
mov     [rbp+180h+M+0Fh], 2
```

2. 把 `state` 读入 `tmp`；



3. 将 `tmp[0][j]` 和 `M[i][0]` 作为参数传入 `GMul()`，结果存入 `var_20`；

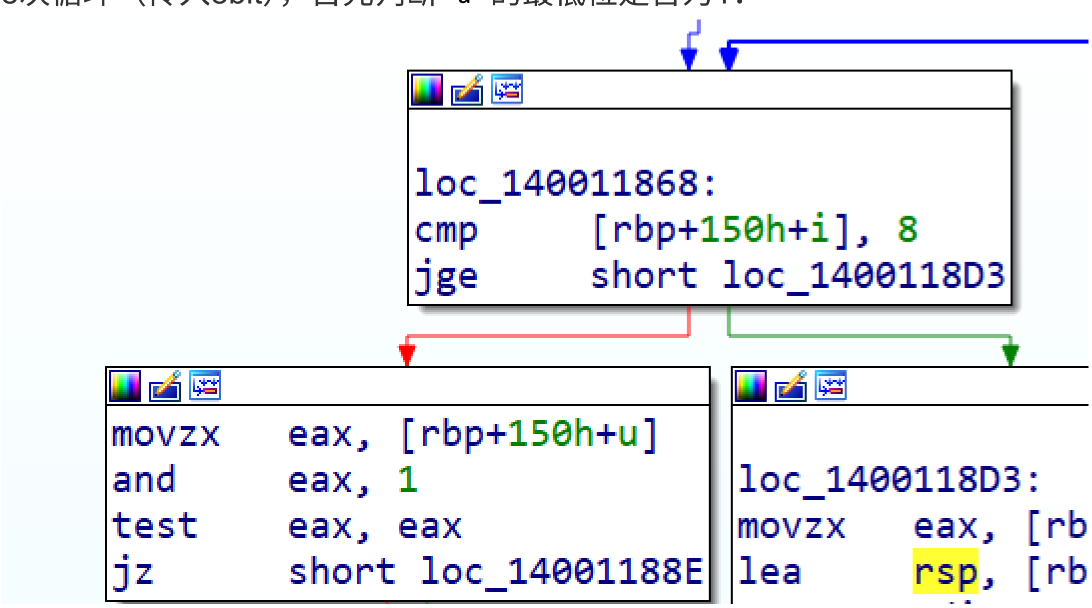
```
mov     eax, 4
imul    rax, 0
lea     rax, [rbp+rax+180h+tmp]
movsxd  rcx, [rbp+180h+j]
movsxd  rdx, [rbp+180h+i]
lea     rdx, [rbp+rdx*4+180h+M]
mov     [rbp+180h+var_28], rdx
mov     r8d, 1
imul    r8, 0
movzx   edx, byte ptr [rax+rcx] ; v
mov     rax, [rbp+180h+var_28]
movzx   ecx, byte ptr [rax+r8] ; u
call    j_?GMul@@YAEEE@Z ; GMul(uchar,uchar)
movzx   eax, al
mov     [rbp+180h+var_20], eax
```

4. 同理后续将 `tmp[1][j]` 和 `M[i][1]` 、 `tmp[2][j]` 和 `M[i][2]` 、 `tmp[3][j]` 和 `M[i][3]` 依次传入 `GMul()` ，并把最后的四个结果异或，存入 `state[i][j]` 。

```
mov     eax, ecx
movsxd  rcx, [rbp+180h+i]
mov     rdx, [rbp+180h+state]
lea     rcx, [rdx+rcx*4]
movsxd  rdx, [rbp+180h+j]
mov     [rcx+rdx], al
```

GMul() 分析

1. 8次循环（传入8bit），首先判断 `u` 的最低位是否为1：



- 若为1： `p = p ^ v` ，实际上是做加法；

```
movzx   eax, [rbp+150h+v]
movzx   ecx, [rbp+150h+p]
xor     ecx, eax
mov     eax, ecx
mov     [rbp+150h+p], al
```

- 若不为1：则跳过上述步骤；

2. 把 `u` 的最高位作为 `flag` ，然后 `v` 左移1位，若 `flag` 为1则做 `v ^ 0x1b` ，若为0则跳过，最后 `u` 右移一位进入下一次循环。

```
loc_14001188E:
movzx   eax, [rbp+150h+v]
and     eax, 80h
mov     [rbp+150h+flag], eax
movzx   eax, [rbp+150h+v]
shl     al, 1
mov     [rbp+150h+v], al
cmp     [rbp+150h+flag], 0
jz      short loc_1400118C2
```

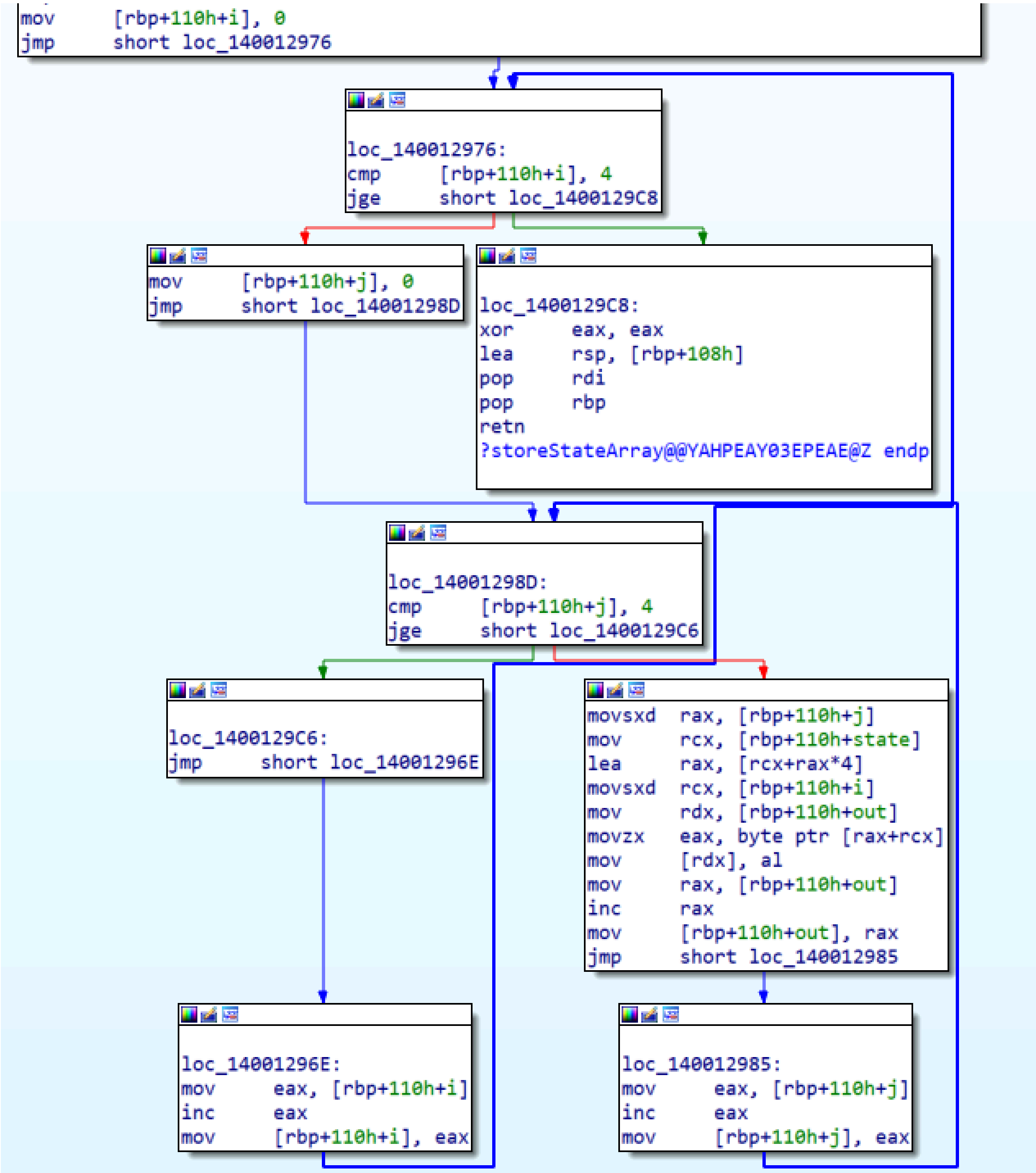
```
movzx   eax, [rbp+150h+v]
xor     eax, 1Bh
mov     [rbp+150h+v], al
```

```
loc_1400118C2:
movzx   eax, [rbp+150h+u]
shr     al, 1
mov     [rbp+150h+u], al
jmp     short loc_140011860
```

整体内容为在 $GF(2^8)$ 有限域上的乘法的快速实现。

`storeStateArray()` 分析

`state` 读入 `out` 即输出内容。


mov [rbp+110h+i], 0
jmp short loc_140012976
loc_140012976:
cmp [rbp+110h+i], 4
jge short loc_1400129C8
mov [rbp+110h+j], 0
jmp short loc_14001298D
loc_1400129C8:
xor eax, eax
lea rsp, [rbp+108h]
pop rdi
pop rbp
retn
?storeStateArray@@YAHPEAY03EPEAE@Z endp
loc_14001298D:
cmp [rbp+110h+j], 4
jge short loc_1400129C6
loc_1400129C6:
jmp short loc_14001296E
movsxd rax, [rbp+110h+j]
mov rcx, [rbp+110h+state]
lea rax, [rcx+rax*4]
movsxd rcx, [rbp+110h+i]
mov rdx, [rbp+110h+out]
movzx eax, byte ptr [rax+rcx]
mov [rdx], al
mov rax, [rbp+110h+out]
inc rax
mov [rbp+110h+out], rax
jmp short loc_140012985
loc_14001296E:
mov eax, [rbp+110h+i]
inc eax
mov [rbp+110h+i], eax
loc_140012985:
mov eax, [rbp+110h+j]
inc eax
mov [rbp+110h+j], eax