A Silicon Bug in Apple's A7 SoC

Proteas of Pangu Lab

2023-09

About me

- ID: Proteas
- Security researcher of Pangu Lab.
- Mainly focusing on security research related to Apple's products.
- @ProteasWang

Glossary

- SoC: System on a Chip
- AP: Application Processor
- SEP: Secure Enclave Processor
- AMCC: Apple's Memory Cache Controller
- TZ: Trust Zone
- KPP: Kernel Patch Protection
- RE: Reverse Engineering

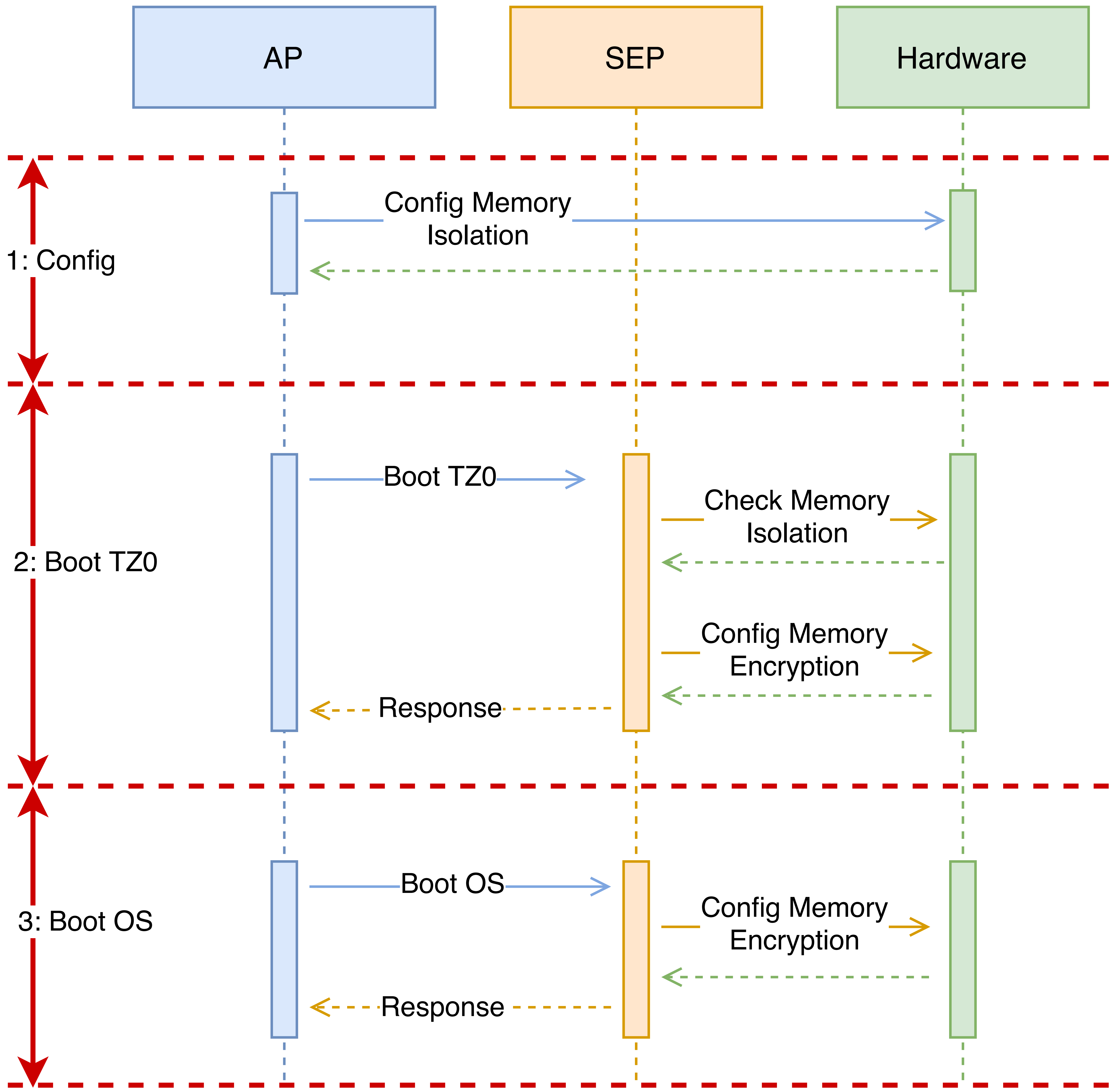
Agenda

- Overview of SEP Boot Process
- The Bug
- Data Actions of SEP Boot Process
- Conclusion
- References
- Appendix: Reverse Engineering SEP Boot Process

Overview of SEP Boot Process

Overview of SEP Boot Process

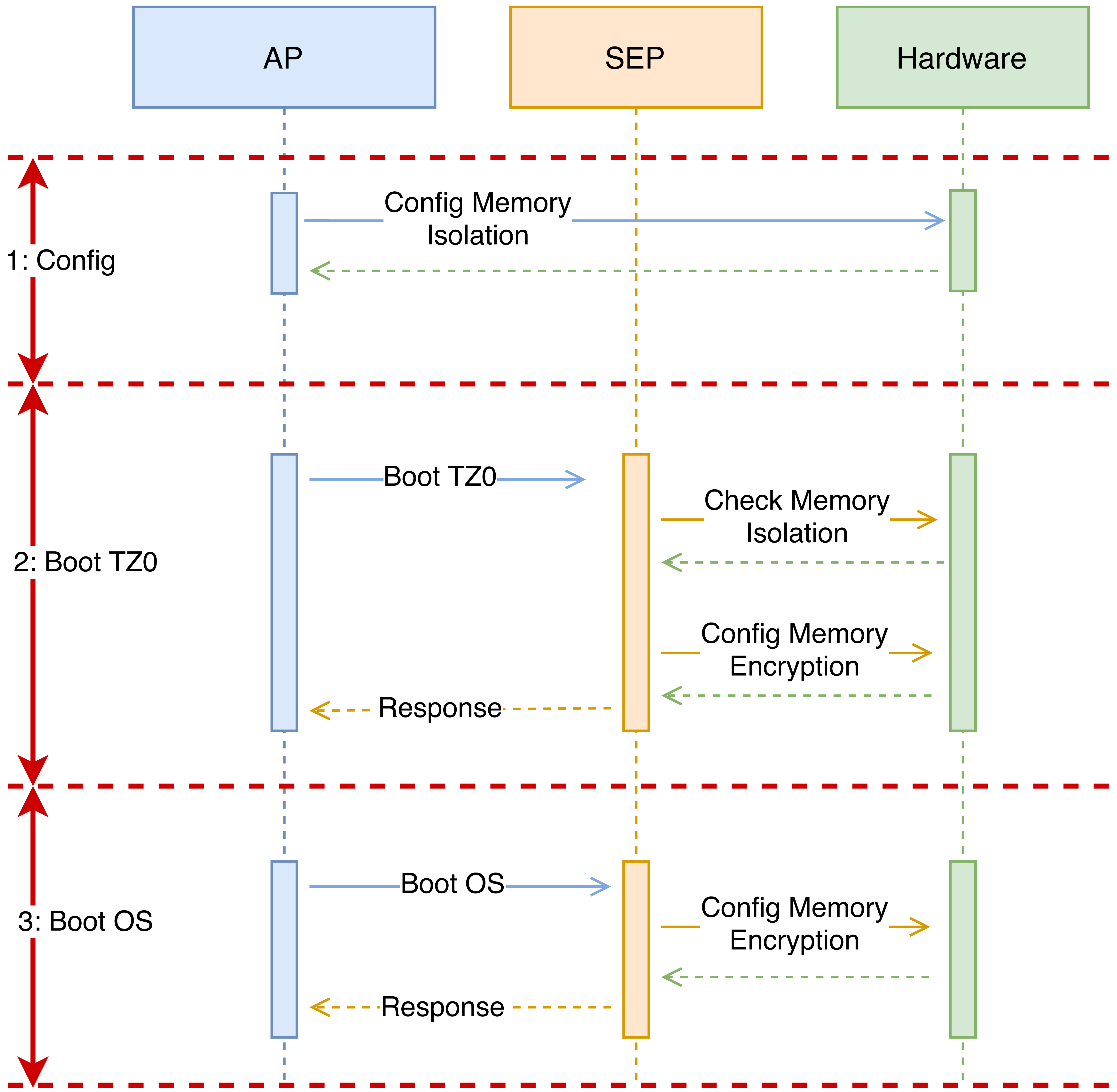
- We can simply divide the boot process of SEP into three stages.
- 1st stage: AP configs the memory isolation hardware.
- 2nd stage: AP sends the `BootTZ0` command to SEP.
- SEP checks the configuration of memory isolation, config the memory encryption hardware, and sends response to AP.



Overview of SEP Boot Process

- 3rd stage: AP loads SEP firmware into memory, sends the `BootOS` command to SEP, and provides SEP with the physical address of the firmware.

- SEP copies the firmware to its own space, checks it, and then boots it.

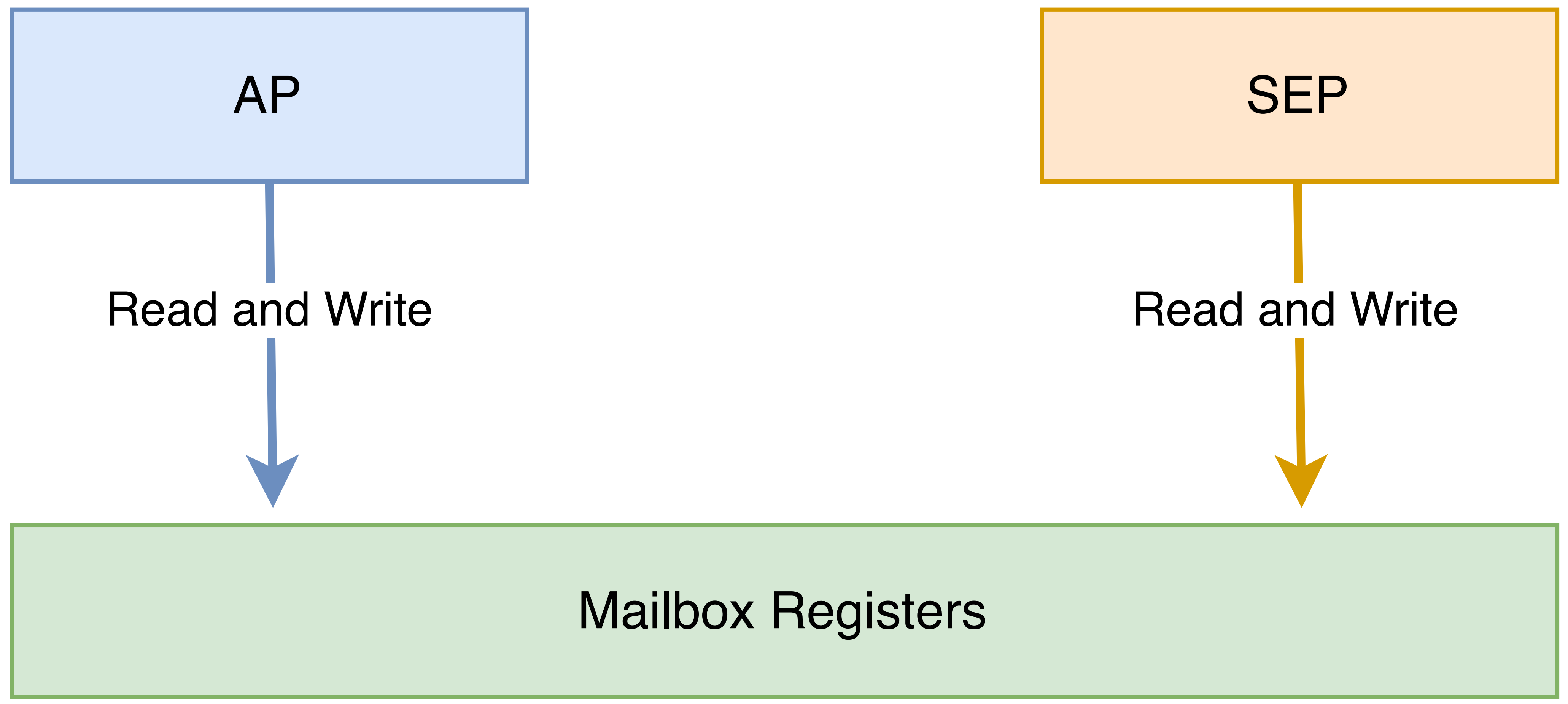


Overview of SEP Boot Process

- What is the communication mechanism between AP and SEP ?
- What are the behavioral characteristics of memory isolation hardware ?
- What are the behavioral characteristics of memory encryption hardware ?

Overview of SEP Boot Process: Communication

- The communication mechanism: Mailbox
- Mailbox registers:
  - Control registers
  - Status registers
  - Data registers



- The mailbox registers are mapped to both AP and SEP, and share the same physical address.
- How does it work actually ?

Overview of SEP Boot Process: Communication

- the definition of a message

```
struct sep_message {
    uint8_t        endpoint;
    uint8_t        tag;
    uint8_t        opcode;
    uint8_t        param;

    uint32_t    data;
} __attribute__((packed));
```

Overview of SEP Boot Process: Communication

- the definition of a message

```
enum {
    kOpCode_Ping = 1,
    kOpCode_Ping_Response = kOpCode_Ping + 100,

    kOpCode_Ping2 = 2,
    kOpCode_Ping2_Response = kOpCode_Ping2 + 100,

    kOpCode_GenerateNonce = 3,
    kOpCode_GenerateNonce_Response = kOpCode_GenerateNonce + 100,

    kOpCode_GetNonceWord = 4,
    kOpCode_GetNonceWord_Response = kOpCode_GetNonceWord + 100,

    kOpCode_BootTZ0 = 5,
    kOpCode_BootTZ0_Response = kOpCode_BootTZ0 + 100,

    kOpCode_BootSEPOS = 6,
    kOpCode_BootSEPOS_Response = kOpCode_BootSEPOS + 100,

    kOpCode_SendARTData = 7,
    kOpCode_SendARTData_Response = kOpCode_SendARTData + 100,
};
```

Overview of SEP Boot Process: Communication

- AP sends a command/message to SEP

1. Check Status Reg
2. Config Ctrl Reg
3. Write Data

Type	Virt	Phys
Status	0x20D001008	0x20D001008
Ctrl	0x20D001004	0x20D001004
Data	0x20D001010	0x20D001010
Data	0x20D001014	0x20D001014

Overview of SEP Boot Process: Communication

- AP sends a command/message to SEP

```
// iOS-v7.1-11D167-iPhone6,1, iBEC
// 0x830001A3C
int32_t __fastcall akf_send_mbox(const uint64_t msg, uint32_t wait_timeout)
{
    enter_critical_section();
    if ( (MEMORY32[0x20D001008] & 0x10000) != 0 ) { // status reg: send
        task_event = &akf_wrappers[10 * 2 + 6];
        while ( 1 ) {
            ret = -1;
            if ( !wait_timeout )
                break;
            MEMORY32[0x20D001004] = 1;// interrupt reg: send
            if ( wait_timeout == -1 ) {
                event_wait(task_event);
            }
            else {
                ret = 0xFFFFFFFE;
                if ( !event_wait_timeout(task_event, wait_timeout) )
                    break;
            }
            if ( (MEMORY32[0x20D001008] & 0x10000) == 0 )// status reg: send
                goto L_Panic;
        }
    }
    else {
L_Panic:
        if ( (MEMORY32[0x20D001008] & 0xC0000) != 0 )// status reg: send
            panic("akf_send_mbox", "ASSERT FAILED at (%s:%s:%d): %s\n", v7, v8, v9, v10);
        ret = 0;
        MEMORY64[0x20D001010] = msg;// data reg: send
    }
    exit_critical_section();
    return ret;
}
```

Overview of SEP Boot Process: Communication

- SEP receives a command/message from AP
- Phys = Virt - 0x30000000 + 0x200000000

Type	Virt	Phys
Status	0x3D000B88	0x20D000B88
Ctrl	0x3D000B80	0x20D000B80
Ctrl	0x3D000B84	0x20D000B84
Data	0x3D000B98	0x20D000B98
Data	0x3D000B9C	0x20D000B9C

Overview of SEP Boot Process: Communication

- SEP receives a command/message from AP

```
// AppleSEPROM-A7-B0
// 0x100077B8
void __fastcall ReceiveMessageFromAP(sep_message *msg)
{
  uint32_t v1; // r3

  while ( (MEMORY32[0x3D000B88] & 0x20000) != 0 )
  {
    MEMORY32[0x3D000B84] = 0x10;
    __dsb(0xFu);
    __isb(0xFu);
    __wfi();
    MEMORY32[0x3D000B80] = 0x10;
  }
  v1 = MEMORY32[0x3D000B9C];
  *(_DWORD *)&msg->ep = MEMORY32[0x3D000B98];
  msg->data = v1;
}
```

Overview of SEP Boot Process: Communication

- SEP sends a command/message to AP

Type	Virt	Phys
Status	0x3D000BA0	0x20D000BA0
Ctrl	0x3D000B80	0x20D000B80
Ctrl	0x3D000B84	0x20D000B84
Data	0x3D000BB0	0x20D000BB0
Data	0x3D000BB4	0x20D000BB4

Overview of SEP Boot Process: Communication

- SEP sends a command/message to AP

```
// AppleSEPROM-A7-B0
// 0x1000781C
void __fastcall SendMessageToAP(sep_message *msg)
{
  int v1; // r1
  uint32_t v2; // r0

  while ( (MEMORY[0x3D000BA0] & 0x20000) == 0 )
  {
    MEMORY[0x3D000B84] = 0x100;
    __dsb(0xFu);
    __isb(0xFu);
    __wfi();
    MEMORY[0x3D000B80] = 0x100;
  }
  v1 = *(_DWORD *)&msg->ep;
  v2 = msg->data;
  MEMORY[0x3D000BB0] = v1;
  MEMORY[0x3D000BB4] = v2;
}
```

Overview of SEP Boot Process: Communication

- AP receives a command/message from SEP

Type	Virt	Phys
Status	0x20D001020	0x20D001020
Ctrl	0x20D001004	0x20D001004
Data	0x20D001038	0x20D001038
Data	0x20D00103C	0x20D00103C

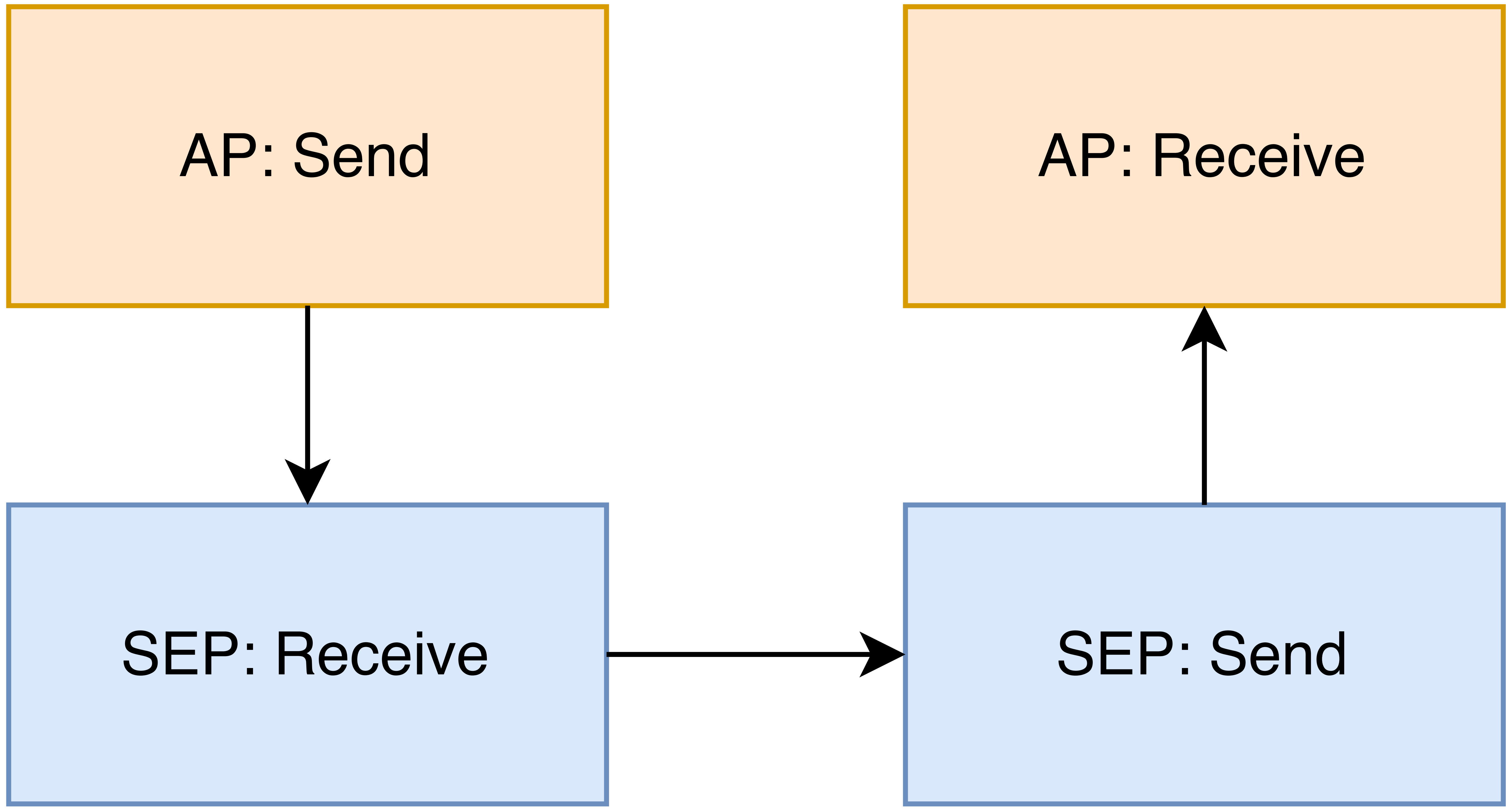
Overview of SEP Boot Process: Communication

- AP receives a command/message from SEP

```
// iOS-v7.1-11D167-iPhone6,1, iBEC
// 0x83000193C
int32_t __fastcall akf_recv_mbox(uint64_t *msg, uint32_t wait_timeout)
{
    enter_critical_section();
    if ( (MEMORY32[0x20D001020] & 0x20000) != 0 ) { // status reg: receive
        task_event = &akf_wrappers[10 * 2 + 3];
        while ( 1 ) {
            ret = -1;
            if ( !wait_timeout )
                break;
            MEMORY32[0x20D001004] = 0x1000;// interrupt reg: receive
            if ( wait_timeout == -1 ) {
                event_wait(task_event);
            }
            else {
                ret = -2;
                if ( !event_wait_timeout(task_event, wait_timeout) )
                    break;
            }
            if ( (MEMORY32[0x20D001020] & 0x20000) == 0 )// status reg: receive
                goto L_Panic;
        }
    }
    else {
L_Panic:
        if ( (MEMORY32[0x20D001020] & 0xC0000) != 0 )// status reg: receive
            panic("akf_recv_mbox", "ASSERT FAILED at (%s:%s:%d): %s\n", v7, v8, v9, v10);
        ret = 0;
        *msg = MEMORY64[0x20D001038];// data reg: receive
    }
    exit_critical_section();
    return ret;
}
```

Overview of SEP Boot Process: Communication

- The communication mechanism: Mailbox



Overview of SEP Boot Process: Memory Isolation

- Memory Isolation: AMCC allows for the one-time configuration of a memory region, and once the configuration is completed, AP can not access this area.
- Memory isolation is a feature of the AMCC hardware.
- Let's see how it works through examples on A7.

Overview of SEP Boot Process: Memory Isolation

- There are 2 regions which can be configured on A7, TZ0 and TZ1.
- TZ0 is used by SEP.
- TZ1 is used by KPP.
- 5 registers involved.

Overview of SEP Boot Process: Memory Isolation

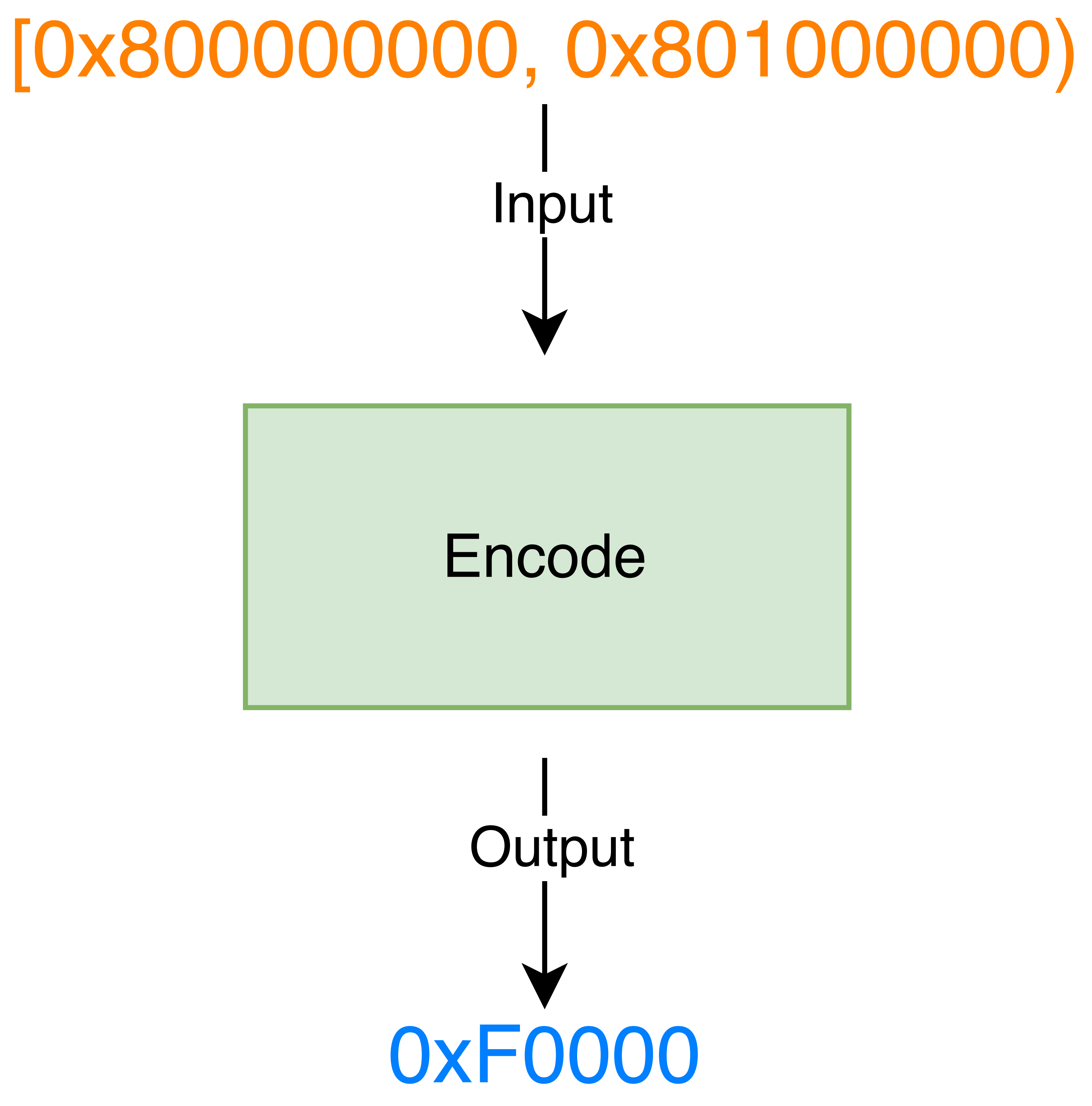
Name	Virt-AP	Phys	Virt-SEP
TZ0 Value	0x200000908	0x200000908	0x30000908
TZ1 Value	0x20000090C	0x20000090C	0x3000090C
TZ0 Lock	0x200000910	0x200000910	0x30000910
TZ1 Lock	0x200000914	0x200000914	0x30000914
Dev Mem Size	0x20000081C	0x20000081C	0x3000081C

Overview of SEP Boot Process: Memory Isolation

- Taking TZ0 as an example to illustrate the workflow.
- The physical address of iDevice starts from 0x8_0000_0000.
- TZ0 is used to indicate a region: [StartOffset, EndOffset).
- The offsets are 64-bits.
- TZ0 is a single 32-bits register.
- The value of TZ0 needs to be encoded from [StartOffset, EndOffset).

Overview of SEP Boot Process: Memory Isolation

- TZ0: [0x800000000, 0x801000000)
- Size of TZ0 region is 16M.
- The encoded value is 0xF0000.



Overview of SEP Boot Process: Memory Isolation

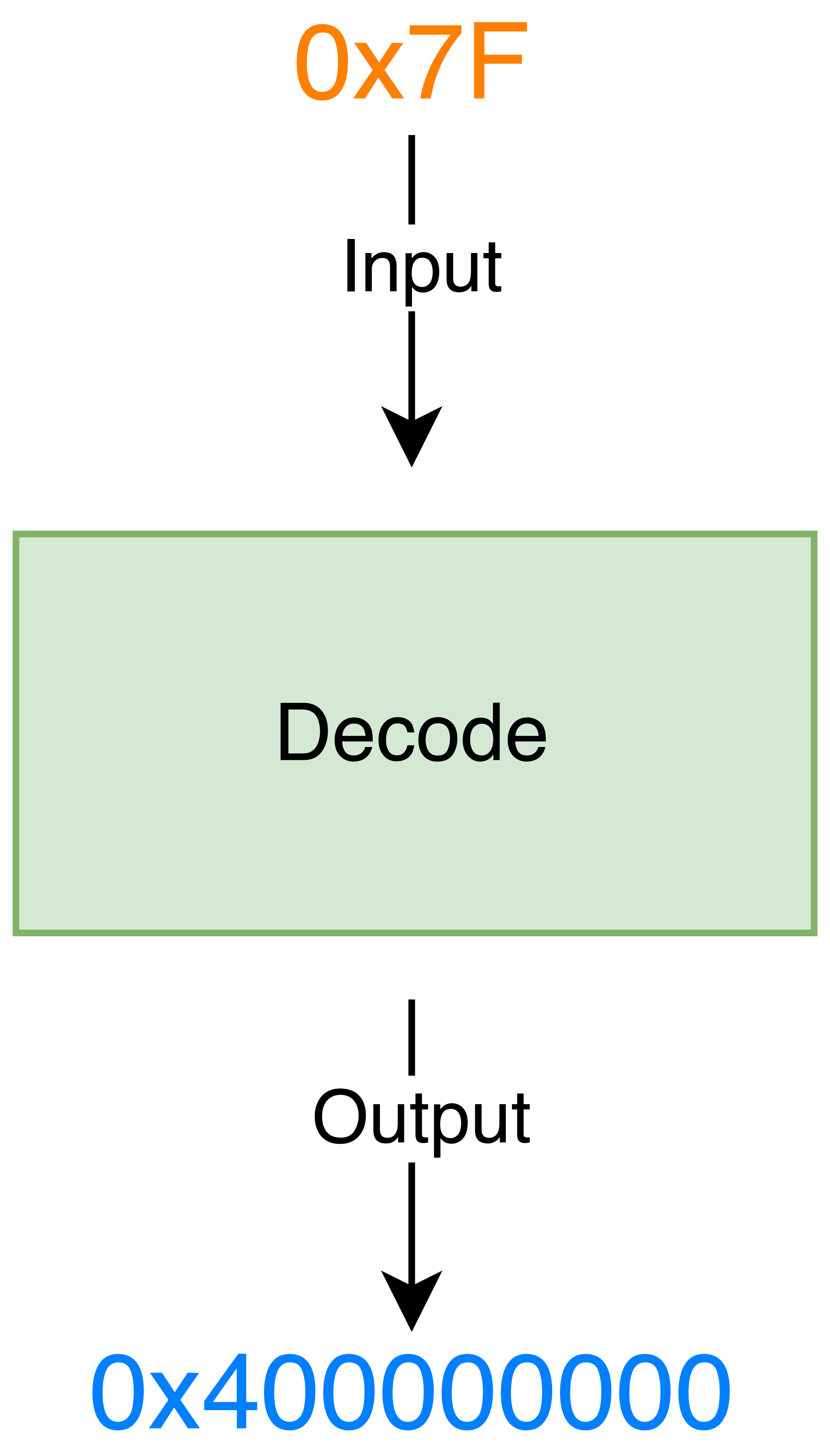
```c
uint32_t TZ_Encode(uint64_t startPA, uint64_t endPA)
{
    uint64_t startOffset = (startPA >> 20) & 0x3FFF;
    uint64_t endOffset = (((endPA >> 20) - 1) << 16) & 0x3FFF0000;

    uint32_t tzValue = endOffset | startOffset;
    return tzValue;
}


void TZ_Decode(uint32_t tzValue, uint64_t *startPA, uint64_t *endPA)
{
    uint64_t startOffset = ((uint64_t)tzValue & 0x3FFF) << 20;
    uint64_t endOffset = ((((uint64_t)tzValue & 0x3FFF0000) >> 16) + 1) << 20;

    if (startPA) {
        *startPA = startOffset;
    }
    if (endPA) {
        *endPA = endOffset;
    }
}
```

Overview of SEP Boot Process: Memory Isolation

- The device memory size register is used to do validation.
- The end of TZ0 and TZ1 should be less than the device memory size.
- The value of this register also needs to be encoded.
- SEP will decode the value of this register.

Overview of SEP Boot Process: Memory Isolation

- Max Input: 0x7F
- Output: 0x400000000 (16GB)



```
// AppleSEPROM-A7-B0
// 0x10008E2E
__PAIR64__((gTZ_DevSizeRegVal & 0x7F) >> 5, 0x8000000) + (gTZ_DevSizeRegVal << 27);
```

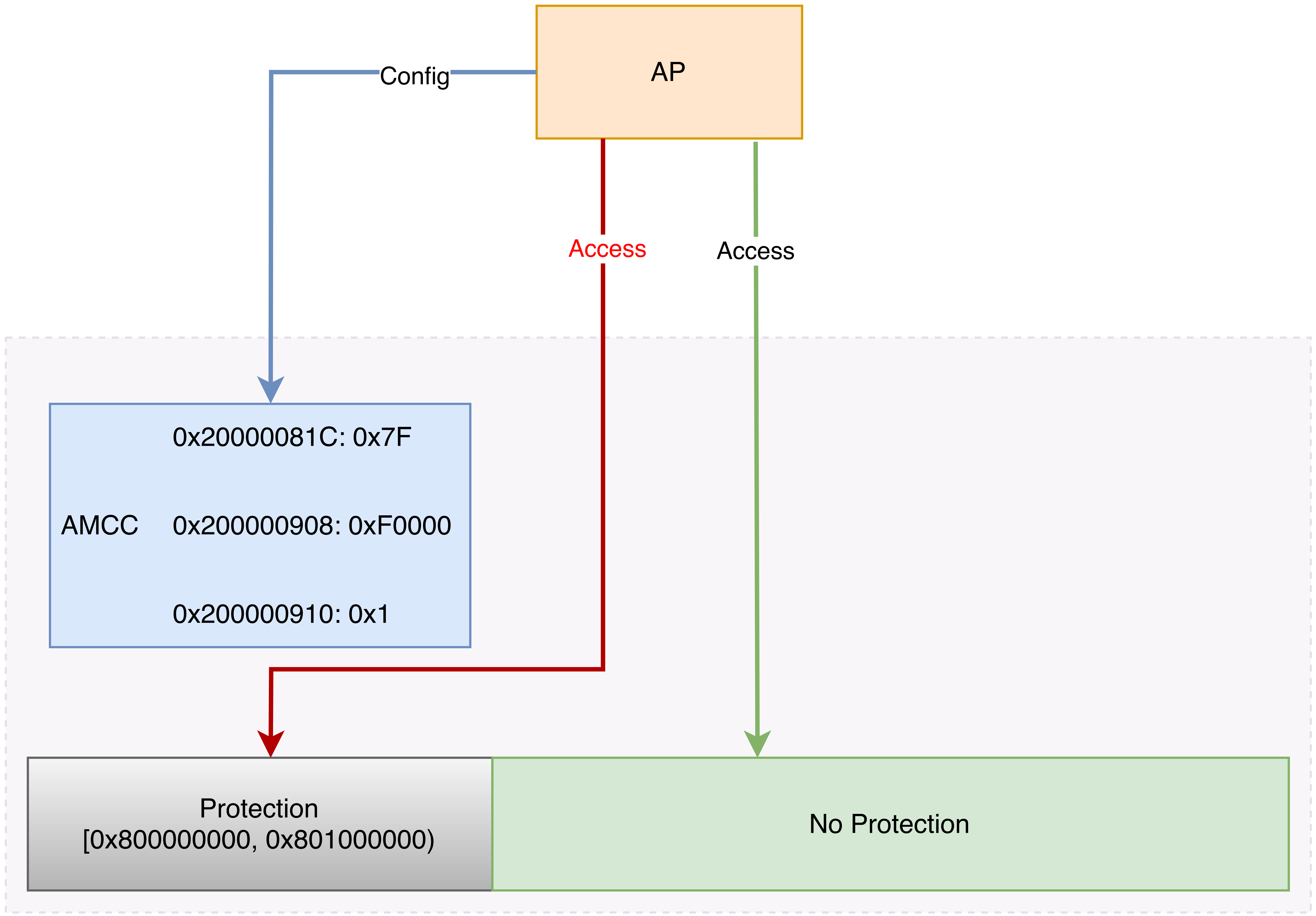
Overview of SEP Boot Process: Memory Isolation

1. AP configs the device memory size register:
   - MEMORY32[0x20000081C] = 0x7F
2. AP configs the TZ0 value register:
   - MEMORY32[0x200000908] = 0xF0000
   - [0x800000000, 0x801000000)
3. AP locks the TZ0 lock register:
   - MEMORY32[0x200000910] = 1

Overview of SEP Boot Process: Memory Isolation

- After the lock register be locked,
- When AP reads this region [0x800000000, 0x801000000),
- zeros are returned.

Overview of SEP Boot Process: Memory Isolation



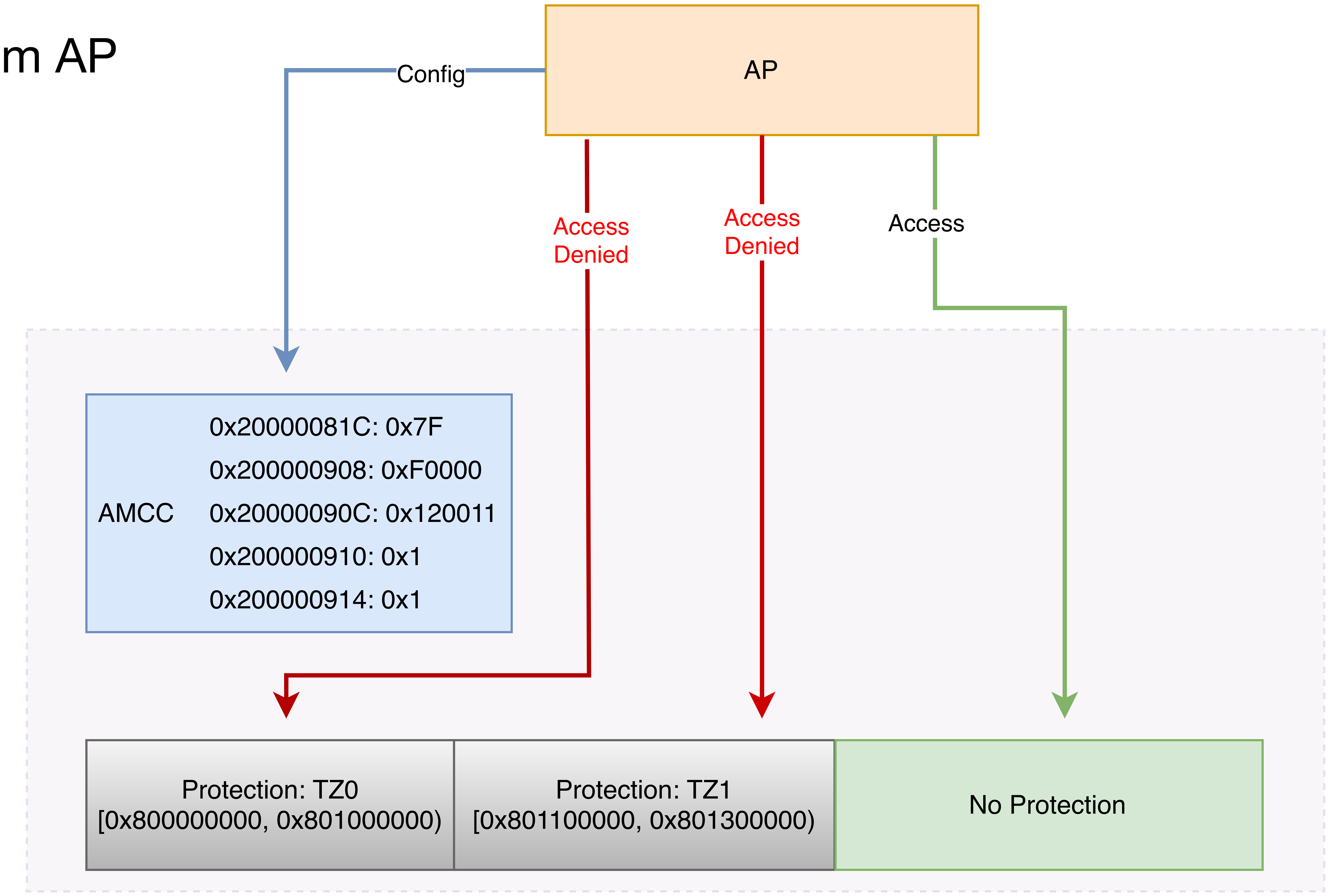
Overview of SEP Boot Process: Memory Isolation

- TZ0 Region: [0x800000000, 0x801000000)
- TZ0: 0xF0000
- TZ1 Region: [0x801100000, 0x801300000)
- TZ1: 0x120011

- TZ0-Lock: 0x1
- TZ1-Lock: 0x1

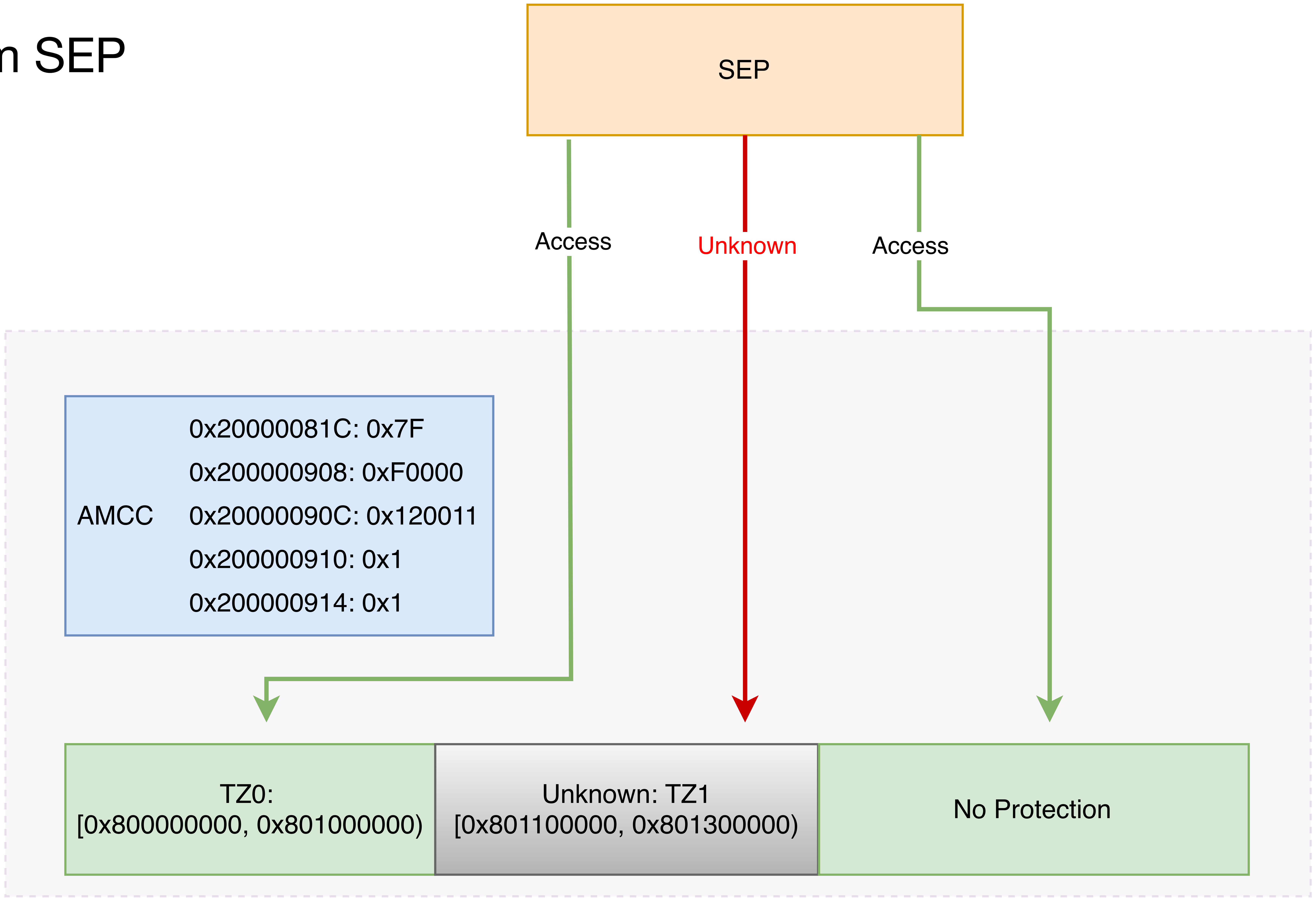
Overview of SEP Boot Process: Memory Isolation

- View from AP



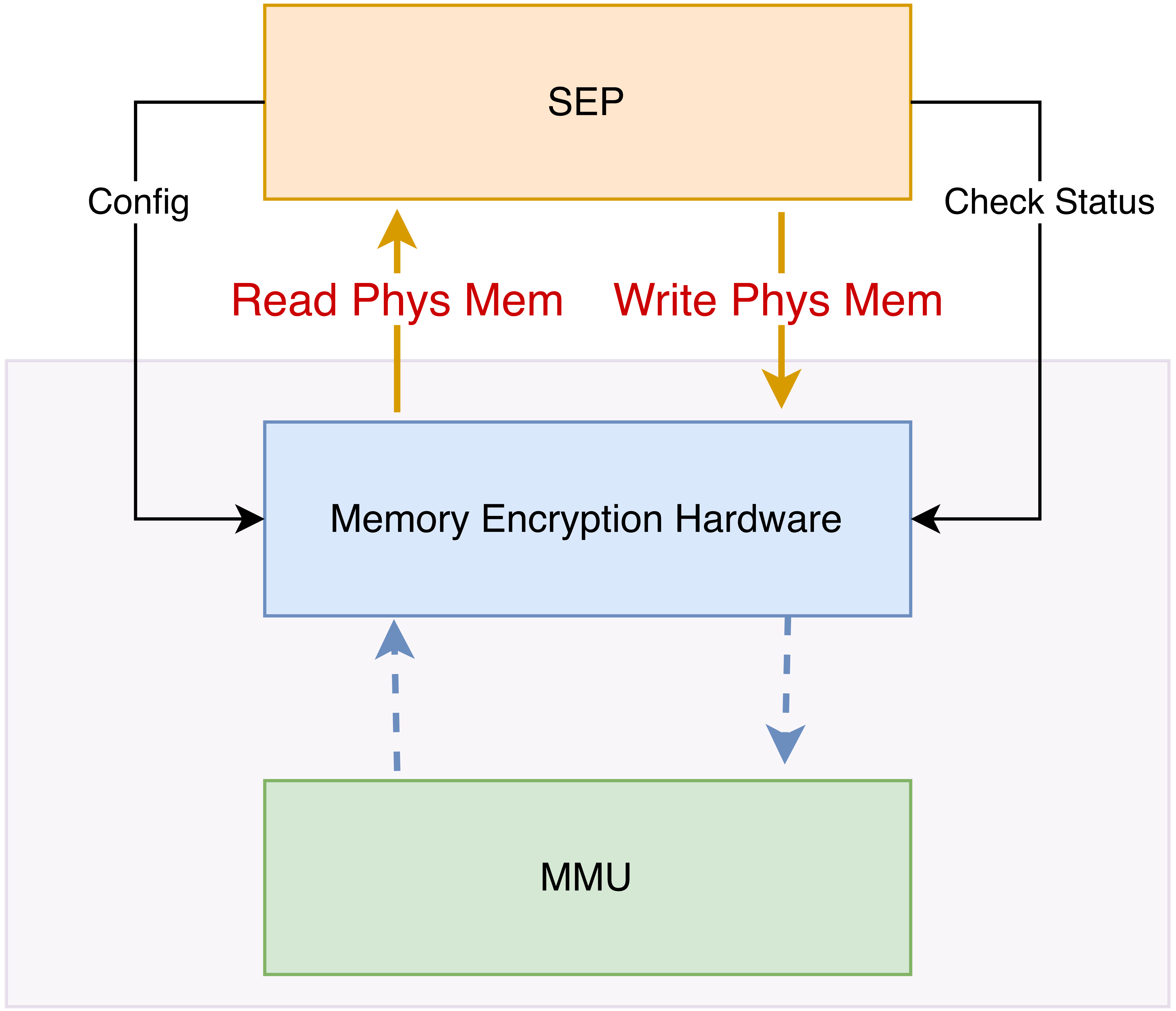
Overview of SEP Boot Process: Memory Isolation

- View from SEP



Overview of SEP Boot Process: Memory Encryption

- Memory Encryption Hardware
- Consider it as a memory proxy
- Transparent memory encryption
- Transparent memory decryption



Overview of SEP Boot Process: Memory Encryption

- Memory Encryption Hardware
  - Ctrl Registers
  - Status Registers
  - Data Registers
- Ctrl Registers can be used to enable encryption channels.
- Data Registers are mainly used to config the keys.

Overview of SEP Boot Process: Memory Encryption

- Ctrl Registers: enable encryption channel

```
// 0x10004A8C
void __fastcall EnableEncryptionChan(uint8_t chan)
{
  MEMORY32[0x3D100004] = ((chan & 1) << 10) | 0x220;
  MEMORY32[0x3D100080] = 0;
  MEMORY32[0x3D100084] = 0;
  MEMORY32[0x3D100088] = 0;
  MEMORY32[0x3D10008C] = 0;
  MEMORY32[0x3D100090] = 0;
  MEMORY32[0x3D100094] = 0;
  MEMORY32[0x3D100098] = 0;
  MEMORY32[0x3D10009C] = 0;
}
```

Overview of SEP Boot Process: Memory Encryption

- Status Registers: check status

```
// 0x10004A20
void __cdecl WaitingAESHardware()
{
  MEMORY32[0x3D300004] = 1;
  sub_10005570(1000000u);
  while ( (MEMORY32[0x3D30000C] & 3) == 0 )
  {
    Get_0x3D000890_Bit_31_CheckIsZero();
    __wfe();
  }
  if ( (MEMORY32[0x3D30000C] & 2) != 0 )
    Spin2();
}
```

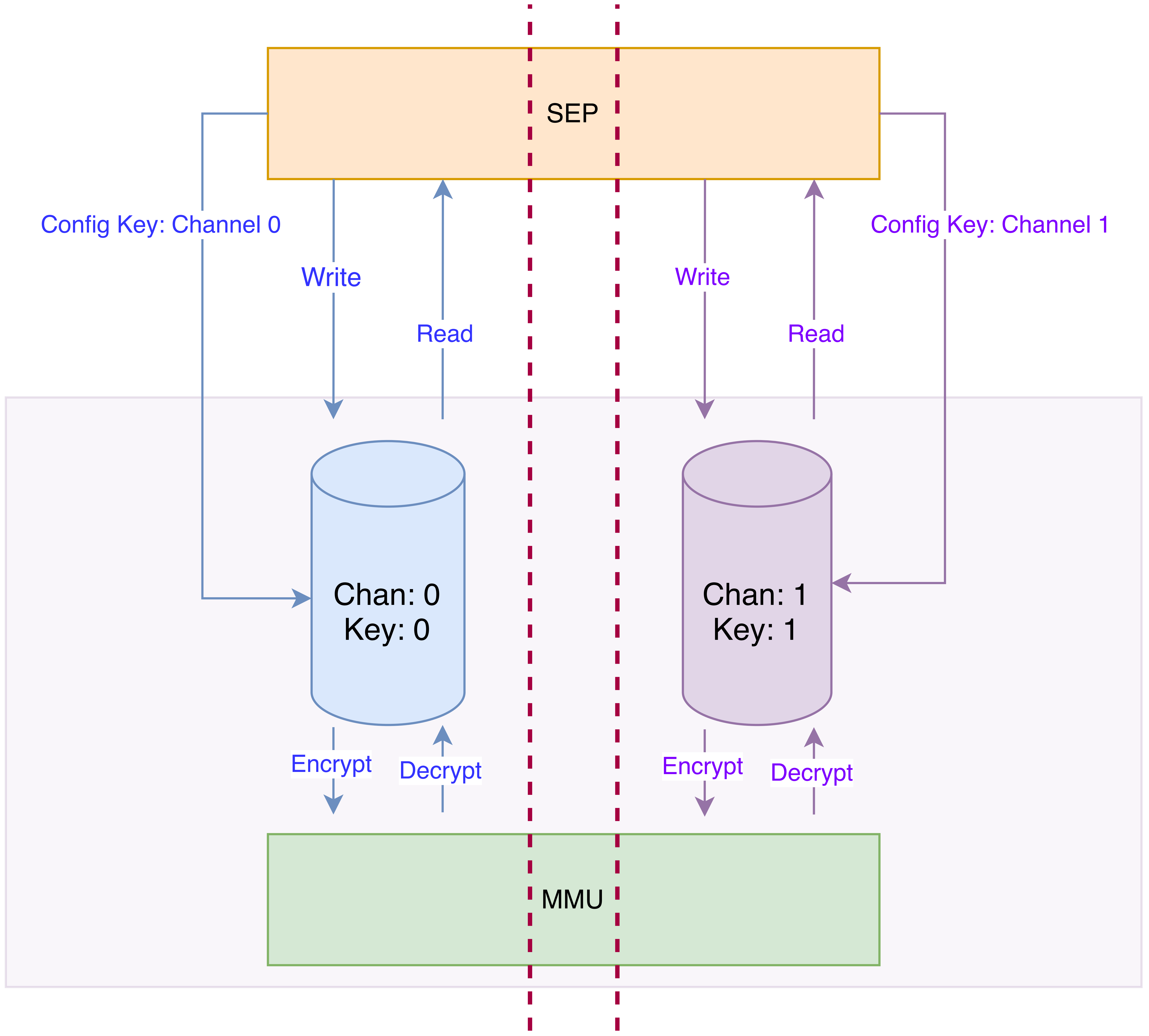
Overview of SEP Boot Process: Memory Encryption

- Data Registers: config keys

```
// 0x100049E8
void __fastcall AES_SetKey_256Bits(uint32_t *keyBuf, uint32_t magic)
{
  MEMORY32[0x3D300040] = magic;
  MEMORY32[0x3D300044] = 0;

  MEMORY32[0x3D300048] = keyBuf[0];
  MEMORY32[0x3D30004C] = keyBuf[1];
  MEMORY32[0x3D300050] = keyBuf[2];

  MEMORY32[0x3D300054] = keyBuf[3];
  MEMORY32[0x3D300058] = keyBuf[4];
  MEMORY32[0x3D30005C] = keyBuf[5];

  AES_Write_0x204_To_0x3D300008();
}
```

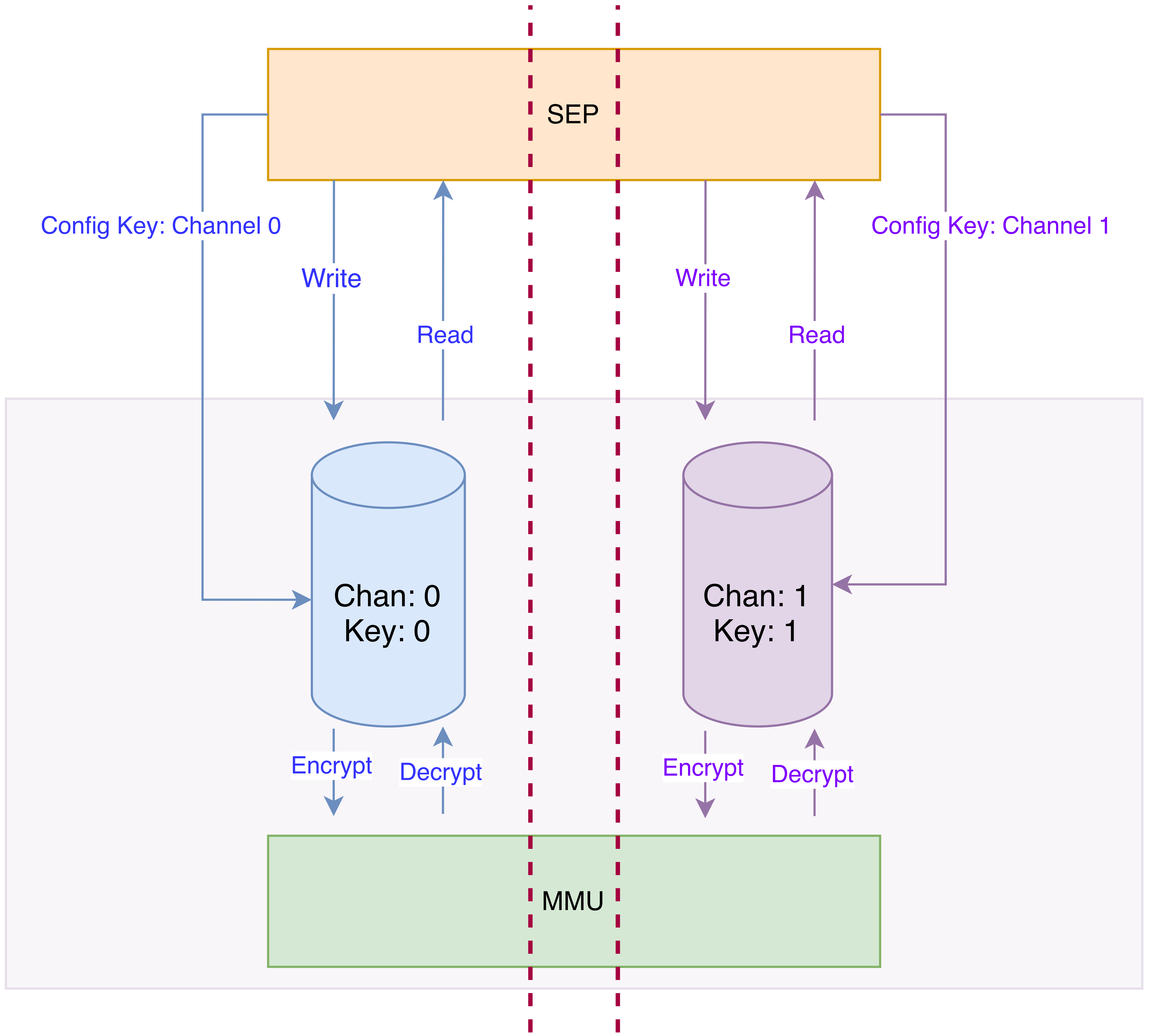
Overview of SEP Boot Process: Memory Encryption

- What are encryption channels ?
- An encryption channel has its own key.



Overview of SEP Boot Process: Memory Encryption

- When writing data to physical memory through channel 0,
- the data will be encrypted with key 0.
- When reading data from physical memory through channel 0,
- the data will be decrypted with key 0.



Overview of SEP Boot Process: Memory Encryption

- How to do channel selection ?
- By adding channel information at the beginning of the physical address,
- The memory encryption hardware will automatically select the right channel,
- When writing to physical address,
- Or reading from physical address.

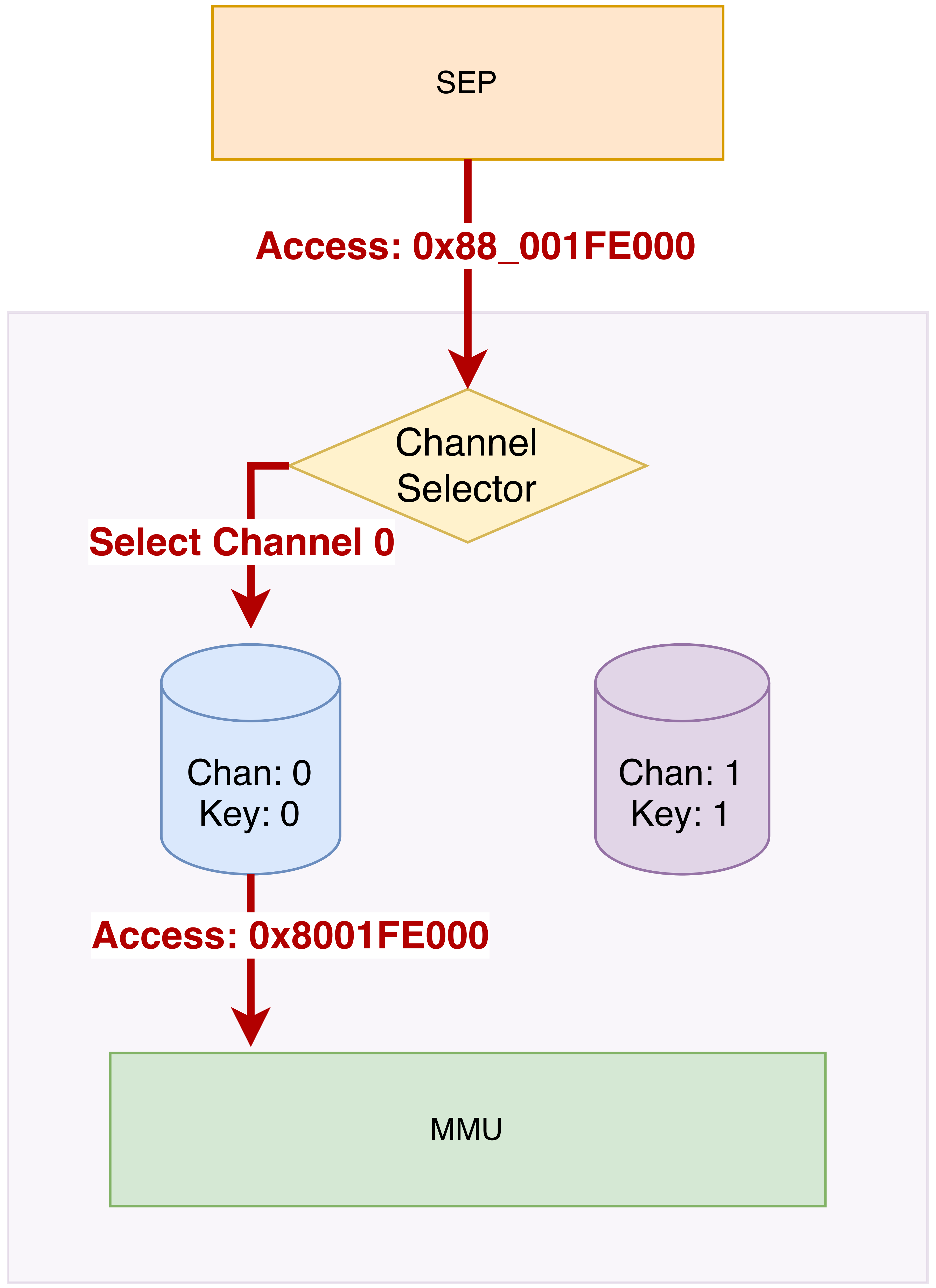
Overview of SEP Boot Process: Memory Encryption

- Adding channel information at the beginning of the physical address
-

Name	Phys Addr Prefix	Example Addr	Actual Phys Addr
Channel 1	0xC8	0x**C8**001FE000	0x8001FE000
Channel 0	0x88	0x**88**001FE000	0x8001FE000
Raw	0x08	0x**08**001FE000	0x8001FE000

Overview of SEP Boot Process: Memory Encryption

- Illustration of Channel Selection



Overview of SEP Boot Process: Memory Encryption

- There 2 encryption channels: 0xC8, 0x88.
- 0xC8 is used by the `BootTZ0` stage.
- 0x88 is used by the `BootOS` stage.

The Bug

The Bug: Channel Selection and Memory Addressing

- `ChanInfo + PA` is 40 bits, for example: 0xC8_001F_E000
- According to the specification and configuration, the PA should be 40 bits.
- SoC Design/Implementation Decision:
- Comply with the spec or make adjustments in implementation ?

Name	Phys Addr Prefix	Example Addr	Actual Phys Addr
Channel 1	0xC8	0x**C8001FE000**	0x8001FE000
Channel 0	0x88	0x**88001FE000**	0x8001FE000
Raw	0x08	0x**08001FE000**	0x8001FE000

The Bug: Channel Selection and Memory Addressing

- If making adjustments, there are several ways, one of which is as follows:
- The left most 2 bits have enough info to do channel selection,
- And the PA is 38 bits, which are directly used to address memory.
- 0xC8_001F_E000
- 0b1100_1000_0x001F_E000

Chan 1	0xC8	0b1100_1000
Chan 0	0x88	0b1000_1000
Raw	0x08	0b0000_1000

The Bug: Channel Selection and Memory Addressing

- If complying with the spec,
- all 8 bits can be used by the channel selector,
- or the lower 6 bits are fully ignored.
- In this case,
- the TZ offsets are 32 bits in hardware,
- the final PA will be constructed with a way.

Chan 1	0xC8	0b1100_1000
Chan 0	0x88	0b1000_1000
Raw	0x08	0b0000_1000

The Bug: Channel Selection and Memory Addressing

- What is the logic of the encryption channel selection ?
- This should be implemented using `Verilog`,
- Let’s implement the guessed logic in `C`.

The Bug: Channel Selection and Memory Addressing

- All 8 bits are used by the channel selector

Chan 1	0xC8	0b1100_1000
Chan 0	0x88	0b1000_1000
Raw	0x08	0b0000_1000

```
if (((PHYS_ADDR_WITH_CHAN_INFO >> 32) & 0xC8) == 0xC8) {
    // Select_Channel_0xC8();
}
else if (((PHYS_ADDR_WITH_CHAN_INFO >> 32) & 0x88) == 0x88) {
    // Select_Channel_0x88();
}
else if (((PHYS_ADDR_WITH_CHAN_INFO >> 32) & 0x08) == 0x08) {
    // Select_Channel_0x08();
}
```

The Bug: Channel Selection and Memory Addressing

- The lower 6 bits are fully ignored

Chan 1	0xC8	0b1100_1000
Chan 0	0x88	0b1000_1000
Raw	0x08	0b0000_1000

```
if ((PHYS_ADDR_WITH_CHAN_INFO >> 38) == 0b11) {
    // Select_Channel_0xC8();
}
else if ((PHYS_ADDR_WITH_CHAN_INFO >> 38) == 0b10) {
    // Select_Channel_0x88();
}
else if ((PHYS_ADDR_WITH_CHAN_INFO >> 38) == 0b00) {
    // Select_Channel_0x08();
}
```

The Bug: 1st Issue

- How does the channel selection unit respond to invalid input ?
- The TZ offsets in the ROM codes are 64 bits on A7.
- TZ Offset: 0x01_001F_E000
- **Chan Info: 0xC8 + 0x01 = 0xC9**

The Bug: 1st Issue

- It should halt, or report error.
- But the hardware accepts the invalid input, and continue to work.
- TZ Offset: 0x01_001F_E000
- PA Offset: 0x001F_E000
- PA Addr: 0x8_0000_0000 + 0x001F_E000 = 0x8_001F_E000
- This is the 1st issue: `hardbird`,
- An input validation issue in the channel selection hardware.

The Bug: 1st Issue

- The TZ offsets in the ROM codes are 64 bits on A7.
- The vendor have fixed this hardware issue by adding constraints to ROM codes,
- On A8 and later, the TZ offsets are 32 bits,
- So the TZ offsets will not affect the channel selection bits.

The Bug: 2nd Issue

- The TZ offsets are 32 bits in memory encryption hardware,
- The TZ offsets are 64 bits in AMCC.
- This is the 2nd issue: Offsets Inconsistency in Hardware
- `blackbird`: Offsets Inconsistency in Hardware and ROM Codes.
- The root cause of the 2 bugs is "Offsets Inconsistency”.
- Let’s use `blackbird` to reference the 2 bugs.

The Bug: PoC-A7

```
1. Mem Size Reg: MEMORY32[0x20000081C] = 0x7F
2.      TZ0 Reg: MEMORY32[0x200000908] = 0x100F1000
3.      TZ1 Reg: MEMORY32[0x20000090C] = 0x10121011
4. TZ0-Lock Reg: MEMORY32[0x200000910] = 0x1
5. TZ1-Lock Reg: MEMORY32[0x200000914] = 0x1
6.     Send Cmd: BootTZ0
7.  View Memory: [0x800000000, 0x801000000)
```

The Bug: PoC-A7

- MEMORY32[0x20000081C] = 0x7F
- AP tells SEP that the device memory size is 0x4_0000_0000 (16GB).

- MEMORY32[0x200000908] = 0x100F1000
- TZ0 Offset: [0x1_0000_0000, 0x1_0100_0000), 16MB
- If the device memory size is not set to 16GB,
- the TZ0 offset will not pass validation by SEP.

The Bug: PoC-A7

- MEMORY32[0x20000090C] = 0x10121011
- TZ1 Offset: [0x101100000, 0x101300000)

- MEMORY32[0x200000910] = 1
- MEMORY32[0x200000914] = 1
- Lock the locks.

The Bug: PoC-A7

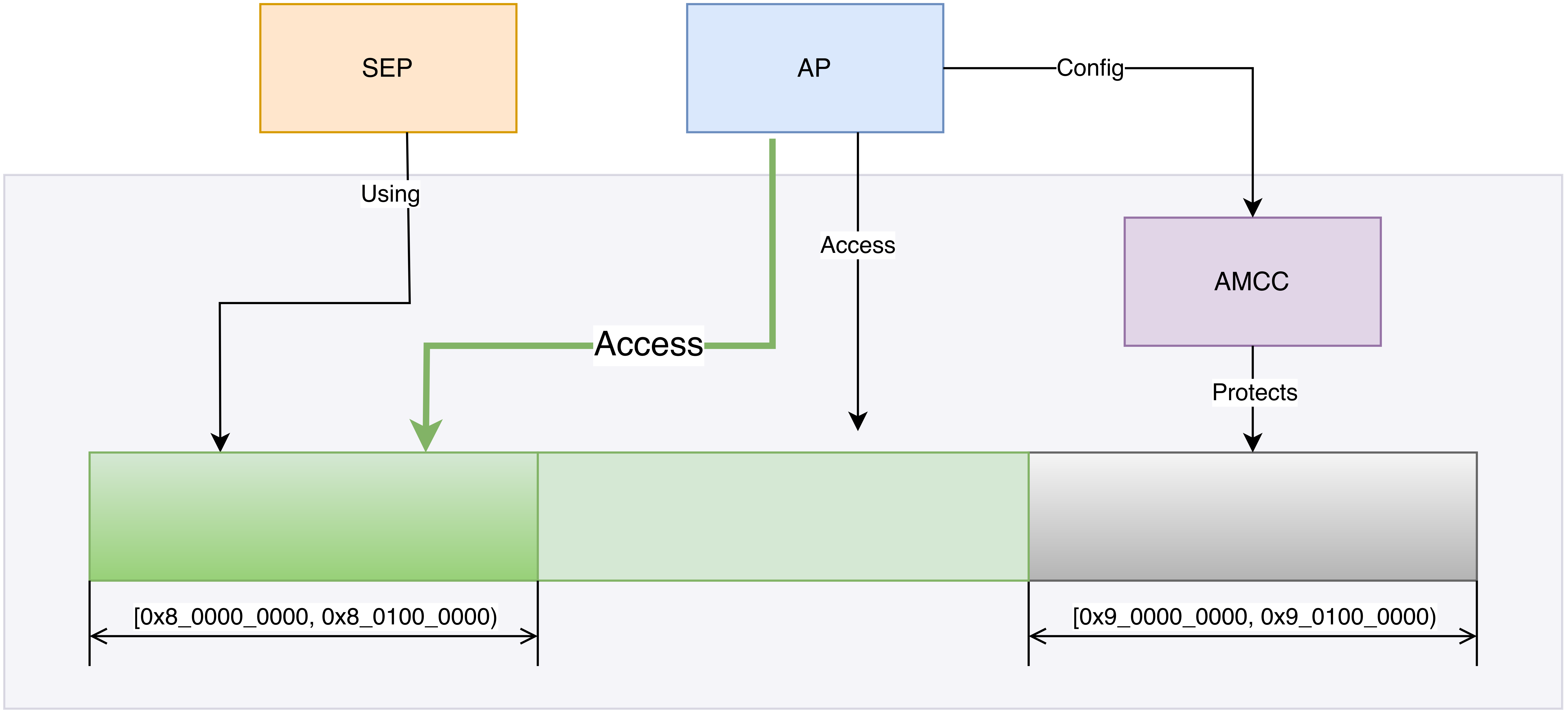
- TZ0 Offset: [0x1_0000_0000, 0x1_0100_0000), 16MB

- 0x8_0000_0000 + 0x1_0000_0000 = 0x9_0000_0000
- AMCC Protects: [0x9_0000_0000, 0x9_0100_0000)

- 0x8_0000_0000 + 0x0000_0000 = 0x8_0000_0000
- SEP Using: [0x8_0000_0000, 0x8_0100_0000)
- After `BootTZ0`, AP can access SEP's memory: [0x8_0000_0000, 0x8_0100_0000)

The Bug: PoC-A7

A7:
 0x20000081C: 0x7F
 0x200000908: 0x100F1000
 0x20000090C: 0x10121011
 0x200000910: 0x1
 0x200000914: 0x1



SEP
AP
Config
Using
Access
AMCC
Access
Protects
[0x8_0000_0000, 0x8_0100_0000)
[0x9_0000_0000, 0x9_0100_0000)

The Bug: PoC-A7, iPhone6,1

```
898c9b8847addcc7:252
[+] sep cmd: boot tz0
[+] start to mount bootfs
[+] success to mount bootfs
[*] create msg, endpoint: 0xff, opcode: Ping2(2), param: 0
[*] msg reader, endpoint: 0xff, opcode: <null>(102), param: 0, data: 0x00000001
[+] tz0: 0x100f1000, tz1: 0x10121011
[+] lock tz0
[+] lock tz1
[+] tz0: [0x900000000, 0x901000000]
[+] tz1: [0x901100000, 0x901300000]
[*] create msg, endpoint: 0xff, opcode: BootTZ0(5), param: 0
[*] msg reader, endpoint: 0xff, opcode: <null>(105), param: 0, data: 0x00000000
[+] boot tz0: done
```

The Bug: PoC-A7, iPhone6,1

```
898c9b8847addcc7:252
[+] sep cmd: dump tz0
[+] tz0: [0x900000000, 0x901000000]
[+] tz0: [0x800000000, 0x801000000]
TZ0+0xFFC000::
  0000: 17 3B 51 4E 66 52 93 6F 92 49 16 EC 40 B5 05 53 | .;QNfR.o.I..@..S
  0010: 3E EC B3 15 24 C3 16 6F 3B D8 28 68 8B EE 98 48 | >...$..o;.(h...H
  0020: 24 7F 8E 6C A8 D5 E3 81 FD 1E 91 C3 37 82 AA 0C | $..l........7...
  0030: 60 18 E8 F8 81 B8 C5 CF 41 27 9F 6C C0 A6 D5 2E | `.......A'.l....
  0040: 7A E3 62 72 91 7B C2 D1 E1 95 F1 21 F5 1E 24 08 | z.br.{.....!..$.
  0050: 13 2C 8E C3 69 70 59 6B 49 CB D7 02 23 F1 56 AE | .,..ipYkI...#.V.
  0060: 8F E1 30 A1 B4 C0 E2 89 96 A9 6F C4 16 BF 45 2A | ..0.......o...E*
  0070: F0 A3 2A BF F3 89 F3 6A A4 FB 19 00 73 81 31 E0 | ..*....j....s.1.
  0080: 4C E9 42 7F A6 5F 68 92 11 50 01 CE D6 35 3D 69 | L.B.._h..P...5=i
  0090: 5E 4E 42 40 08 F4 66 BF E2 0B 2F 23 70 11 3A 77 | ^NB@..f.../#p.:w
  00A0: 99 65 7B 48 98 4E 93 D4 84 FC 53 5A D0 22 7C C0 | .e{H.N....SZ."|.
  00B0: DD 38 A7 67 25 A6 E5 88 F8 1C 81 76 B2 BD 4C 4D | .8.g%......v..LM
  00C0: CE EB F2 C3 0D CE 96 A6 AB 18 6E B4 FA CB 50 F4 | ..........n...P.
  00D0: 20 89 7A B8 6E D4 B6 4A DE 32 C5 D7 B0 A8 F3 B6 |  .z.n..J.2......
  00E0: F2 DC 7B 68 CE A2 94 AB 65 65 67 BD 37 A3 E2 11 | ..{h....eeg.7...
  00F0: F3 D5 DA D6 91 E2 C3 89 4A 01 53 84 67 9C 97 CB | ........J.S.g...
```

The Bug: PoC-A7, iPhone6,1

- There is no MAC on A7,
- When AP edits the encrypted memory, SEP will get incorrect data.
- When reading memory which is not initialized by SEP, A7 will not halt.
- This is why there are fewer memory initializations on the SEP-ROM of A7.

Data Actions of SEP Boot Process

Data Actions of SEP Boot Process: BootTZ0

- Basic Configuration Info:
    - TZ0 Offset: [0x1_0000_0000, 0x1_0100_0000)
    - TZ0 Region: [0x8_0000_0000, 0x8_0100_0000)
- Memory info before `BootTZ0`:

VA	PA	Size	XN	PXN	AP	Attr Index	Attr
0x00004000	0x020DA04000	0x1000	Executable	Privileged Executable	Read-only	0	0x04, Device memory
0x10000000	0x020DA00000	0x100000	Executable	Privileged Executable	Read-only	3	0x4F, Normal memory
0x10180000	0x0080000000	0x3000	Executable-Never	Privileged Executable-Never	Read/write	4	0x04, Device memory
0x30000000	0x0200000000	0x10000000	Executable-Never	Privileged Executable-Never	Read/write	0	0x04, Device memory

Data Actions of SEP Boot Process: BootTZ0

- Code Info:

```
// 0x10004EA4
void __noreturn CheckAndConfig_AfterBootTZ0()
{
    // ……
    Map_0x10180000_0x1000_NotEncrypted();
    bzero_0x10180078_0xF88();
    GenRandom(gKey_Chan_0xC8_TZ0, 192u);
    SetEncryptionKey_Chan1(gKey_Chan_0xC8_TZ0);
    CopyPageTableAndConfig_TTBR0_AfterBootTZ0();
    AddPageTableItem_AfterBootTZ0();
    bzero_Addr_0x10120140_Size_0x148();
    copy_from_0x1000F000_to_0x10120000_size_0x140();
    bzero_10130000_0x2000();
    GenRandom((uint8_t *)&gStackCookie, 32u);
    SetStackValue_From_0x3D600F80_To_0x3D601000();
    // ……
}
```

Data Actions of SEP Boot Process: BootTZ0

- Map 0x18000000
- Type: Normal Memory.

Virt	Size	Chan+PA	PA
0x10180000	0x1000	0x08001FF000	0x8001FF000

```
// 0x10004EA4
void __noreturn CheckAndConfig_AfterBootTZ0()
{
    // ……
->  Map_0x10180000_0x1000_NotEncrypted();
    bzero_0x10180078_0xF88();
    GenRandom(gKey_Chan_0xC8_TZ0, 192u);
    SetEncryptionKey_Chan1(gKey_Chan_0xC8_TZ0);
    CopyPageTableAndConfig_TTBR0_AfterBootTZ0();
    AddPageTableItem_AfterBootTZ0();
    bzero_Addr_0x10120140_Size_0x148();
    copy_from_0x1000F000_to_0x10120000_size_0x140();
    bzero_10130000_0x2000();
    GenRandom((uint8_t *)&gStackCookie, 32u);
    SetStackValue_From_0x3D600F80_To_0x3D601000();
    // ……
}
```

Data Actions of SEP Boot Process: BootTZ0

- bzero(0x10180078, 0xF88)

Virt	Size	Chan+PA	PA
0x10180078	0xF88	0x08001FF078	0x8001FF078

```
// 0x10004EA4
void __noreturn CheckAndConfig_AfterBootTZ0()
{
    // ……
    Map_0x10180000_0x1000_NotEncrypted();
->  bzero_0x10180078_0xF88();
    GenRandom(gKey_Chan_0xC8_TZ0, 192u);
    SetEncryptionKey_Chan1(gKey_Chan_0xC8_TZ0);
    CopyPageTableAndConfig_TTBR0_AfterBootTZ0();
    AddPageTableItem_AfterBootTZ0();
    bzero_Addr_0x10120140_Size_0x148();
    copy_from_0x1000F000_to_0x10120000_size_0x140();
    bzero_10130000_0x2000();
    GenRandom((uint8_t *)&gStackCookie, 32u);
    SetStackValue_From_0x3D600F80_To_0x3D601000();
    // ……
}
```

Data Actions of SEP Boot Process: BootTZ0

- Generate random num: 0x10180018

Virt	Size	Chan+PA	PA
0x10180018	0x18	0x08001FF018	0x8001FF018

```
// 0x10004EA4
void __noreturn CheckAndConfig_AfterBootTZ0()
{
    // ……
    Map_0x10180000_0x1000_NotEncrypted();
    bzero_0x10180078_0xF88();
->  GenRandom(gKey_Chan_0xC8_TZ0, 192u);
    SetEncryptionKey_Chan1(gKey_Chan_0xC8_TZ0);
    CopyPageTableAndConfig_TTBR0_AfterBootTZ0();
    AddPageTableItem_AfterBootTZ0();
    bzero_Addr_0x10120140_Size_0x148();
    copy_from_0x1000F000_to_0x10120000_size_0x140();
    bzero_10130000_0x2000();
    GenRandom((uint8_t *)&gStackCookie, 32u);
    SetStackValue_From_0x3D600F80_To_0x3D601000();
    // ……
}
```

Data Actions of SEP Boot Process: BootTZ0

- Generate random num: 0x10180018

```
0000: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 | ................
0010: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 | ................
0020: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 | ................
0030: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 | ................
0040: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 | ................
0050: 00 00 00 00 00 00 00 00 5A 38 C7 16 F0 5C BC 8D | ........Z8...\..
0060: 39 44 DC 95 49 D0 61 3F D2 0D 7F 44 65 1A 06 DD | 9D..I.a?...De...
0070: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 | ................
```

- Actual Address: 0x18 + 0x40 = 0x58

- 0x40 should be the cache line size.

- Don’t know how it works internally yet.

Data Actions of SEP Boot Process: BootTZ0

- Copy page table from ROM to RAM
- Map 0x10180000
- Type: Normal Memory

Virt	Size	Chan+PA	PA
0x10180000	0x3000	0xC900FFA000	0x800FFA000

```
// 0x10004EA4
void __noreturn CheckAndConfig_AfterBootTZ0()
{
    // ……
    Map_0x10180000_0x1000_NotEncrypted();
    bzero_0x10180078_0xF88();
    GenRandom(gKey_Chan_0xC8_TZ0, 192u);
    SetEncryptionKey_Chan1(gKey_Chan_0xC8_TZ0);
->  CopyPageTableAndConfig_TTBR0_AfterBootTZ0();
    AddPageTableItem_AfterBootTZ0();
    bzero_Addr_0x10120140_Size_0x148();
    copy_from_0x1000F000_to_0x10120000_size_0x140();
    bzero_10130000_0x2000();
    GenRandom((uint8_t *)&gStackCookie, 32u);
    SetStackValue_From_0x3D600F80_To_0x3D601000();
    // ……
}
```

Data Actions of SEP Boot Process: BootTZ0

- AP: [0x800FFA000, 0x800FFD000)
- This memory region is used to save page table.
- When SEP edits this region,
- whether AP can see the changes is highly dependent on the hardware model.
- The precise timing of data synchronization is generally left to the discretion of the processor's implementation.

Data Actions of SEP Boot Process: BootTZ0

- Copy 3 pages,
- From 0x10001000 to 0x10180000

```
// 0x10004EA4
void __noreturn CheckAndConfig_AfterBootTZ0()
{
    // ……
    Map_0x10180000_0x1000_NotEncrypted();
    bzero_0x10180078_0xF88();
    GenRandom(gKey_Chan_0xC8_TZ0, 192u);
    SetEncryptionKey_Chan1(gKey_Chan_0xC8_TZ0);
->  CopyPageTableAndConfig_TTBR0_AfterBootTZ0();
    AddPageTableItem_AfterBootTZ0();
    bzero_Addr_0x10120140_Size_0x148();
    copy_from_0x1000F000_to_0x10120000_size_0x140();
    bzero_10130000_0x2000();
    GenRandom((uint8_t *)&gStackCookie, 32u);
    SetStackValue_From_0x3D600F80_To_0x3D601000();
    // ……
}
```

Virt	Size	Chan+PA	PA
0x10001000	0x3000		0x20DA01000

Virt	Size	Chan+PA	PA
0x10180000	0x3000	0xC900FFA000	0x800FFA000

Data Actions of SEP Boot Process: BootTZ0

- Change PTE,
- write 0 to 0x10180000
- 0x20DA02003 -> 0

Virt	Size	Chan+PA	PA
0x10180000	0x8	0xC900FFA000	0x800FFA000

```
// 0x10004EA4
void __noreturn CheckAndConfig_AfterBootTZ0()
{
    // ……
    Map_0x10180000_0x1000_NotEncrypted();
    bzero_0x10180078_0xF88();
    GenRandom(gKey_Chan_0xC8_TZ0, 192u);
    SetEncryptionKey_Chan1(gKey_Chan_0xC8_TZ0);
->  CopyPageTableAndConfig_TTBR0_AfterBootTZ0();
    AddPageTableItem_AfterBootTZ0();
    bzero_Addr_0x10120140_Size_0x148();
    copy_from_0x1000F000_to_0x10120000_size_0x140();
    bzero_10130000_0x2000();
    GenRandom((uint8_t *)&gStackCookie, 32u);
    SetStackValue_From_0x3D600F80_To_0x3D601000();
    // ……
}
```

Data Actions of SEP Boot Process: BootTZ0

- Change PTE,
- write 0 to 0x10181020
- 0x20DA04683 -> 0

Virt	Size	Chan+PA	PA
0x10181020	0x8	0xC900FFB020	0x800FFB020

```
// 0x10004EA4
void __noreturn CheckAndConfig_AfterBootTZ0()
{
    // ……
    Map_0x10180000_0x1000_NotEncrypted();
    bzero_0x10180078_0xF88();
    GenRandom(gKey_Chan_0xC8_TZ0, 192u);
    SetEncryptionKey_Chan1(gKey_Chan_0xC8_TZ0);
->  CopyPageTableAndConfig_TTBR0_AfterBootTZ0();
    AddPageTableItem_AfterBootTZ0();
    bzero_Addr_0x10120140_Size_0x148();
    copy_from_0x1000F000_to_0x10120000_size_0x140();
    bzero_10130000_0x2000();
    GenRandom((uint8_t *)&gStackCookie, 32u);
    SetStackValue_From_0x3D600F80_To_0x3D601000();
    // ……
}
```

Data Actions of SEP Boot Process: BootTZ0

- Change PTE,
- write PTE to 0x10180400
- 0x20DA03003 -> 0xC900FFC003

Virt	Size	Chan+PA	PA
0x10180400	0x8	0xC900FFA400	0x800FFA400

```
// 0x10004EA4
void __noreturn CheckAndConfig_AfterBootTZ0()
{
    // ……
    Map_0x10180000_0x1000_NotEncrypted();
    bzero_0x10180078_0xF88();
    GenRandom(gKey_Chan_0xC8_TZ0, 192u);
    SetEncryptionKey_Chan1(gKey_Chan_0xC8_TZ0);
->  CopyPageTableAndConfig_TTBR0_AfterBootTZ0();
    AddPageTableItem_AfterBootTZ0();
    bzero_Addr_0x10120140_Size_0x148();
    copy_from_0x1000F000_to_0x10120000_size_0x140();
    bzero_10130000_0x2000();
    GenRandom((uint8_t *)&gStackCookie, 32u);
    SetStackValue_From_0x3D600F80_To_0x3D601000();
    // ……
}
```

Data Actions of SEP Boot Process: BootTZ0

- Change PTE,
- write PTE to 0x10182F80
- 0x0 -> 0x6000C900FFA607
- 0x0 -> 0x6000C900FFB607
- 0x0 -> 0x6000C900FFC607

Virt	Size	Chan+PA	PA
0x10182F80	0x18	0xC900FFCF80	0x800FFCF80

```
// 0x10004EA4
void __noreturn CheckAndConfig_AfterBootTZ0()
{
    // ……
    Map_0x10180000_0x1000_NotEncrypted();
    bzero_0x10180078_0xF88();
    GenRandom(gKey_Chan_0xC8_TZ0, 192u);
    SetEncryptionKey_Chan1(gKey_Chan_0xC8_TZ0);
->  CopyPageTableAndConfig_TTBR0_AfterBootTZ0();
    AddPageTableItem_AfterBootTZ0();
    bzero_Addr_0x10120140_Size_0x148();
    copy_from_0x1000F000_to_0x10120000_size_0x140();
    bzero_10130000_0x2000();
    GenRandom((uint8_t *)&gStackCookie, 32u);
    SetStackValue_From_0x3D600F80_To_0x3D601000();
    // ……
}
```

Data Actions of SEP Boot Process: BootTZ0

- Change Memory Type
- Range: [0x10180000, 0x10183000)
- Type: Device Memory
- Load the new page table.

```
// 0x10004EA4
void __noreturn CheckAndConfig_AfterBootTZ0()
{
    // ……
    Map_0x10180000_0x1000_NotEncrypted();
    bzero_0x10180078_0xF88();
    GenRandom(gKey_Chan_0xC8_TZ0, 192u);
    SetEncryptionKey_Chan1(gKey_Chan_0xC8_TZ0);
->  CopyPageTableAndConfig_TTBR0_AfterBootTZ0();
    AddPageTableItem_AfterBootTZ0();
    bzero_Addr_0x10120140_Size_0x148();
    copy_from_0x1000F000_to_0x10120000_size_0x140();
    bzero_10130000_0x2000();
    GenRandom((uint8_t *)&gStackCookie, 32u);
    SetStackValue_From_0x3D600F80_To_0x3D601000();
    // ……
}
```

Data Actions of SEP Boot Process: BootTZ0

- Change PTE,
- write PTE to 0x101F2A00
- 0x0 -> 0x60000900FFF607

Virt	Size	Chan+PA	PA
0x101F2A00	0x8	0xC9001FCA00	0x8001FCA00

```
// 0x10004EA4
void __noreturn CheckAndConfig_AfterBootTZ0()
{
    // ……
    Map_0x10180000_0x1000_NotEncrypted();
    bzero_0x10180078_0xF88();
    GenRandom(gKey_Chan_0xC8_TZ0, 192u);
    SetEncryptionKey_Chan1(gKey_Chan_0xC8_TZ0);
    CopyPageTableAndConfig_TTBR0_AfterBootTZ0();
->  AddPageTableItem_AfterBootTZ0();
    bzero_Addr_0x10120140_Size_0x148();
    copy_from_0x1000F000_to_0x10120000_size_0x140();
    bzero_10130000_0x2000();
    GenRandom((uint8_t *)&gStackCookie, 32u);
    SetStackValue_From_0x3D600F80_To_0x3D601000();
    // ……
}
```

Data Actions of SEP Boot Process: BootTZ0

- Change PTE,
- write PTE to 0x101F2980
- 0x0 -> 0x6000C9001F8607
- 0x0 -> 0x6000C9001F9607

```
// 0x10004EA4
void __noreturn CheckAndConfig_AfterBootTZ0()
{
    // ……
    Map_0x10180000_0x1000_NotEncrypted();
    bzero_0x10180078_0xF88();
    GenRandom(gKey_Chan_0xC8_TZ0, 192u);
    SetEncryptionKey_Chan1(gKey_Chan_0xC8_TZ0);
    CopyPageTableAndConfig_TTBR0_AfterBootTZ0();
->  AddPageTableItem_AfterBootTZ0();
    bzero_Addr_0x10120140_Size_0x148();
    copy_from_0x1000F000_to_0x10120000_size_0x140();
    bzero_10130000_0x2000();
    GenRandom((uint8_t *)&gStackCookie, 32u);
    SetStackValue_From_0x3D600F80_To_0x3D601000();
    // ……
}
```

Virt	Size	Chan+PA	PA
0x101F2980	0x10	0xC9001FC980	0x8001FC980

Data Actions of SEP Boot Process: BootTZ0

- Change PTE,
- write PTE to 0x101F2900
- 0x0 -> 0x6000C9001FD607

Virt	Size	Chan+PA	PA
0x101F2900	0x8	0xC9001FC900	0x8001FC900

```
// 0x10004EA4
void __noreturn CheckAndConfig_AfterBootTZ0()
{
    // ……
    Map_0x10180000_0x1000_NotEncrypted();
    bzero_0x10180078_0xF88();
    GenRandom(gKey_Chan_0xC8_TZ0, 192u);
    SetEncryptionKey_Chan1(gKey_Chan_0xC8_TZ0);
    CopyPageTableAndConfig_TTBR0_AfterBootTZ0();
->  AddPageTableItem_AfterBootTZ0();
    bzero_Addr_0x10120140_Size_0x148();
    copy_from_0x1000F000_to_0x10120000_size_0x140();
    bzero_10130000_0x2000();
    GenRandom((uint8_t *)&gStackCookie, 32u);
    SetStackValue_From_0x3D600F80_To_0x3D601000();
    // ……
}
```

Data Actions of SEP Boot Process: BootTZ0

- Change PTE,
- write PTE to 0x101F2B00
- 0x0 -> 0x6000C9001FE607

Virt	Size	Chan+PA	PA
0x101F2B00	0x8	0xC9001FCB00	0x8001FCB00

```
// 0x10004EA4
void __noreturn CheckAndConfig_AfterBootTZ0()
{
    // ……
    Map_0x10180000_0x1000_NotEncrypted();
    bzero_0x10180078_0xF88();
    GenRandom(gKey_Chan_0xC8_TZ0, 192u);
    SetEncryptionKey_Chan1(gKey_Chan_0xC8_TZ0);
    CopyPageTableAndConfig_TTBR0_AfterBootTZ0();
->  AddPageTableItem_AfterBootTZ0();
    bzero_Addr_0x10120140_Size_0x148();
    copy_from_0x1000F000_to_0x10120000_size_0x140();
    bzero_10130000_0x2000();
    GenRandom((uint8_t *)&gStackCookie, 32u);
    SetStackValue_From_0x3D600F80_To_0x3D601000();
    // ……
}
```

Data Actions of SEP Boot Process: BootTZ0

- New Memory Info:

VA	PA	Size	XN	PXN	AP	Attr Index	Attr
0x10000000	0x020DA00000	0x100000	Executable	Privileged Executable	Read-only	3	0x4F, Normal memory
0x10120000	0xC9001FD000	0x1000	Executable-Never	Privileged Executable-Never	Read/write	1	0x4F, Normal memory
0x10130000	0xC9001F8000	0x2000	Executable-Never	Privileged Executable-Never	Read/write	1	0x4F, Normal memory
0x10140000	0x09001FF000	0x1000	Executable-Never	Privileged Executable-Never	Read/write	1	0x4F, Normal memory
0x10150000	------------	------	---------------	---------------------------	----------	-	-------------------
0x10160000	0xC9001FE000	0x1000	Executable-Never	Privileged Executable-Never	Read/write	1	0x4F, Normal memory
0x10180000	0x0080000000	0x3000	Executable-Never	Privileged Executable-Never	Read/write	4	0x04, Device memory
0x101F0000	0xC9001FA000	0x3000	Executable-Never	Privileged Executable-Never	Read/write	1	0x4F, Normal memory
0x30000000	0x0200000000	0x10000000	Executable-Never	Privileged Executable-Never	Read/write	0	0x04, Device memory

Data Actions of SEP Boot Process: BootTZ0

- bzero(0x10120140, 0x148)

Virt	Size	Chan+PA	PA
0x10120140	0x148	0xC9001FD140	0x8001FD140

```
// 0x10004EA4
void __noreturn CheckAndConfig_AfterBootTZ0()
{
    // ……
    Map_0x10180000_0x1000_NotEncrypted();
    bzero_0x10180078_0xF88();
    GenRandom(gKey_Chan_0xC8_TZ0, 192u);
    SetEncryptionKey_Chan1(gKey_Chan_0xC8_TZ0);
    CopyPageTableAndConfig_TTBR0_AfterBootTZ0();
    AddPageTableItem_AfterBootTZ0();
->  bzero_Addr_0x10120140_Size_0x148();
    copy_from_0x1000F000_to_0x10120000_size_0x140();
    bzero_10130000_0x2000();
    GenRandom((uint8_t *)&gStackCookie, 32u);
    SetStackValue_From_0x3D600F80_To_0x3D601000();
    // ……
}
```

Data Actions of SEP Boot Process: BootTZ0

- memcpy(0x10120000, 0x1000F000, 0x140)

```
// 0x10004EA4
void __noreturn CheckAndConfig_AfterBootTZ0()
{
    // ……
    Map_0x10180000_0x1000_NotEncrypted();
    bzero_0x10180078_0xF88();
    GenRandom(gKey_Chan_0xC8_TZ0, 192u);
    SetEncryptionKey_Chan1(gKey_Chan_0xC8_TZ0);
    CopyPageTableAndConfig_TTBR0_AfterBootTZ0();
    AddPageTableItem_AfterBootTZ0();
    bzero_Addr_0x10120140_Size_0x148();
->  copy_from_0x1000F000_to_0x10120000_size_0x140();
    bzero_10130000_0x2000();
    GenRandom((uint8_t *)&gStackCookie, 32u);
    SetStackValue_From_0x3D600F80_To_0x3D601000();
    // ……
}
```

Virt	Size	Chan+PA	PA
0x1000F000	0x140		0x20DA0F000

Virt	Size	Chan+PA	PA
0x10120000	0x140	0xC9001FD000	0x8001FD000

Data Actions of SEP Boot Process: BootTZ0

- bzero(0x10130000, 0x2000)

Virt	Size	Chan+PA	PA
0x10130000	0x2000	0xC9001F8000	0x8001F8000

```
// 0x10004EA4
void __noreturn CheckAndConfig_AfterBootTZ0()
{
    // ……
    Map_0x10180000_0x1000_NotEncrypted();
    bzero_0x10180078_0xF88();
    GenRandom(gKey_Chan_0xC8_TZ0, 192u);
    SetEncryptionKey_Chan1(gKey_Chan_0xC8_TZ0);
    CopyPageTableAndConfig_TTBR0_AfterBootTZ0();
    AddPageTableItem_AfterBootTZ0();
    bzero_Addr_0x10120140_Size_0x148();
    copy_from_0x1000F000_to_0x10120000_size_0x140();
->  bzero_10130000_0x2000();
    GenRandom((uint8_t *)&gStackCookie, 32u);
    SetStackValue_From_0x3D600F80_To_0x3D601000();
    // ……
}
```

Data Actions of SEP Boot Process: BootTZ0

- Generate stack cookie

Virt	Size	Chan+PA	PA
0x101200D4	0x4	0xC9001FD0D4	0x8001FD0D4

```
// 0x10004EA4
void __noreturn CheckAndConfig_AfterBootTZ0()
{
    // ……
    Map_0x10180000_0x1000_NotEncrypted();
    bzero_0x10180078_0xF88();
    GenRandom(gKey_Chan_0xC8_TZ0, 192u);
    SetEncryptionKey_Chan1(gKey_Chan_0xC8_TZ0);
    CopyPageTableAndConfig_TTBR0_AfterBootTZ0();
    AddPageTableItem_AfterBootTZ0();
    bzero_Addr_0x10120140_Size_0x148();
    copy_from_0x1000F000_to_0x10120000_size_0x140();
    bzero_10130000_0x2000();
->  GenRandom((uint8_t *)&gStackCookie, 32u);
    SetStackValue_From_0x3D600F80_To_0x3D601000();
    // ……
}
```

Data Actions of SEP Boot Process: BootTZ0

- set stack value

Virt	Size	Chan+PA	PA
0x3D600F80	0x80		0x20D600F80

```
// 0x10004EA4
void __noreturn CheckAndConfig_AfterBootTZ0()
{
    // ……
    Map_0x10180000_0x1000_NotEncrypted();
    bzero_0x10180078_0xF88();
    GenRandom(gKey_Chan_0xC8_TZ0, 192u);
    SetEncryptionKey_Chan1(gKey_Chan_0xC8_TZ0);
    CopyPageTableAndConfig_TTBR0_AfterBootTZ0();
    AddPageTableItem_AfterBootTZ0();
    bzero_Addr_0x10120140_Size_0x148();
    copy_from_0x1000F000_to_0x10120000_size_0x140();
    bzero_10130000_0x2000();
    GenRandom((uint8_t *)&gStackCookie, 32u);
->  SetStackValue_From_0x3D600F80_To_0x3D601000();
    // ……
}
```

Data Actions of SEP Boot Process: BootOS

```
// 0x10005D8C
void LoadAndBootSEPOS(uint32_t paLow, uint32_t paHigh, uint32_t bootType)
{
    // ……
    Map_0x20000000_ByEncChan(2);
    GetFwBufAddrAndSize(&fwBuf, &fwBufSize);
    CopyFirmwareFromAPToLocalBuf(paLow, paHigh, fwBuf, fwBufSize);
    // memset(fwBuf + img4Size, 0, fwBufSize - img4Size);
    Image4Load(&img4Info, bootType, fwBuf, fwBufSize);
    UnMap_0x20000000();
    // ……
    GenRandom_At_0x10140000(0);
    SetEncryptionKey_Addr_0x10140000(0);
    sizeFromIM4PEndToFwBufBegin = img4Info.im4pPayloadAddr + img4Info.im4pPayloadSize - fwBuf;
    // PageAlignedInplaceChangeEncChan_From_0xC8_To_0x88(sizeFromIM4PEndToFwBufBegin)
    // ……
    Map_0x20000000_ByEncChan(1);
    MoveImg4PayloadToTZ0Start(fwBuf, img4Info.im4pPayloadAddr, img4Info.im4pPayloadSize);
    memset(fwBuf + img4Info.im4pPayloadSize, 0, fwBufSize - img4Info.im4pPayloadSize);
    DecryptImg4Payload(fwBuf, fwBuf, img4Info.im4pPayloadSize, &v15, keySizeBits, &a6);
    UnMap_0x20000000();
    SendBootOSRespondToAP();
    SetBootArgAndJumpToOS(&img4Info);
    // ……
}
```

Data Actions of SEP Boot Process: BootOS

- Map 0x20000000
  - Map PA 0x0 to Chan+PA 0xC900000000
  - Map VA 0x20000000 to PA 0x0

Virt	Size	Chan+PA	PA
0x20000000	0xFF7000	0xC900000000	0x800000000

```
// 0x10005D8C
void LoadAndBootSEPOS(uint32_t paLow, uint32_t paHigh, uint32_t bootType)
{
    // ……
->  Map_0x20000000_ByEncChan(2);
    GetFwBufAddrAndSize(&fwBuf, &fwBufSize);
    CopyFirmwareFromAPToLocalBuf(paLow, paHigh, fwBuf, fwBufSize);
   // memset(fwBuf + img4Size, 0, fwBufSize - img4Size);
    Image4Load(&img4Info, bootType, fwBuf, fwBufSize);
    UnMap_0x20000000();
    // ……
}
```

Data Actions of SEP Boot Process: BootOS

- Map 0x20000000
    - Add PTEs
    - Depends on TZ0 size

Virt	Size	Chan+PA	PA
0x101F0800	0x40	0xC900FFA800	0x800FFA800

Virt	Size	Chan+PA	PA
0x101F1000	0xFB8	0xC900FFB000	0x800FFB000

```
// 0x10005D8C
void LoadAndBootSEPOS(uint32_t paLow, uint32_t paHigh, uint32_t bootType)
{
    // ……
->  Map_0x20000000_ByEncChan(2);
    GetFwBufAddrAndSize(&fwBuf, &fwBufSize);
    CopyFirmwareFromAPToLocalBuf(paLow, paHigh, fwBuf, fwBufSize);
   // memset(fwBuf + img4Size, 0, fwBufSize - img4Size);
    Image4Load(&img4Info, bootType, fwBuf, fwBufSize);
    UnMap_0x20000000();
    // ……
}
```

Data Actions of SEP Boot Process: BootOS

- Copy in the firmware
  - Map 0x10150000 to external PA
  - Copy from 0x10150000 to 0x20000000
  - Page by page

Virt	Size	Chan+PA	PA
0x20000000	IMG4Size	0xC900000000	0x800000000

```
// 0x10005D8C
void LoadAndBootSEPOS(uint32_t paLow, uint32_t paHigh, uint32_t bootType)
{
    // ……
    Map_0x20000000_ByEncChan(2);
    GetFwBufAddrAndSize(&fwBuf, &fwBufSize);
->  CopyFirmwareFromAPToLocalBuf(paLow, paHigh, fwBuf, fwBufSize);
   // memset(fwBuf + img4Size, 0, fwBufSize - img4Size);
    Image4Load(&img4Info, bootType, fwBuf, fwBufSize);
    UnMap_0x20000000();
    // ……
}
```

Data Actions of SEP Boot Process: BootOS

- clear memory

Virt	Size	Chan+PA	PA
0x20000000 + IMG4Size	0xFF7000 - IMG4Size	0xC900000000 + IMG4Size	0x800000000 + IMG4Size

```
// 0x10005D8C
void LoadAndBootSEPOS(uint32_t paLow, uint32_t paHigh, uint32_t bootType)
{
    // ……
    Map_0x20000000_ByEncChan(2);
    GetFwBufAddrAndSize(&fwBuf, &fwBufSize);
    CopyFirmwareFromAPToLocalBuf(paLow, paHigh, fwBuf, fwBufSize);
->  // memset(fwBuf + img4Size, 0, fwBufSize - img4Size);
    Image4Load(&img4Info, bootType, fwBuf, fwBufSize);
    UnMap_0x20000000();
    // ……
}
```

Data Actions of SEP Boot Process: BootOS

- Unmap 0x20000000

Virt	Size	Chan+PA	PA
0x20000000	IMG4Size	0xC900000000	0x800000000

```
// 0x10005D8C
void LoadAndBootSEPOS(uint32_t paLow, uint32_t paHigh, uint32_t bootType)
{
    // ……
    Map_0x20000000_ByEncChan(2);
    GetFwBufAddrAndSize(&fwBuf, &fwBufSize);
    CopyFirmwareFromAPToLocalBuf(paLow, paHigh, fwBuf, fwBufSize);
   // memset(fwBuf + img4Size, 0, fwBufSize - img4Size);
    Image4Load(&img4Info, bootType, fwBuf, fwBufSize);
->  UnMap_0x20000000();
    // ……
}
```

Data Actions of SEP Boot Process: BootOS

- Unmap 0x20000000
    - Delete PTEs

Virt	Size	Chan+PA	PA
0x101F0800	0x40	0xC900FFA800	0x800FFA800

```
// 0x10005D8C
void LoadAndBootSEPOS(uint32_t paLow, uint32_t paHigh, uint32_t bootType)
{
    // ……
    Map_0x20000000_ByEncChan(2);
    GetFwBufAddrAndSize(&fwBuf, &fwBufSize);
    CopyFirmwareFromAPToLocalBuf(paLow, paHigh, fwBuf, fwBufSize);
   // memset(fwBuf + img4Size, 0, fwBufSize - img4Size);
    Image4Load(&img4Info, bootType, fwBuf, fwBufSize);
->  UnMap_0x20000000();
    // ……
}
```

Data Actions of SEP Boot Process: BootOS

- Generate random num: 0x10140000

Virt	Size	Chan+PA	PA
0x10140000	0x18	0x08001FF000	0x8001FF000

```
// 0x10005D8C
void LoadAndBootSEPOS(uint32_t paLow, uint32_t paHigh, uint32_t bootType)
{
    // ......
->  GenRandom_At_0x10140000(0);
    SetEncryptionKey_Addr_0x10140000(0);
    // ......
}
```

Data Actions of SEP Boot Process: BootOS

- Change encryption channel.
  - Map 0x10150000 to Chan-0xC8
  - Copy from 0x10150000 to 0x10160000
  - Map 0x10150000 to Chan-0x88
  - Copy from 0x10160000 to 0x10150000

Virt	Size	Chan+PA	PA
0x10150000	0x1000	0xC900000000	0x800000000
0x10160000	0x1000	0xC900FFE000	0x800FFE000
0x10150000	0x1000	0x8900000000	0x800000000

```
// 0x10005D8C
void LoadAndBootSEPOS(uint32_t paLow, uint32_t paHigh, uint32_t bootType)
{
    // ……
    sizeFromIM4PEndToFwBufBegin = img4Info.im4pPayloadAddr + img4Info.im4pPayloadSize - fwBuf;
->  // PageAlignedInplaceChangeEncChan_From_0xC8_To_0x88(sizeFromIM4PEndToFwBufBegin)
    // ……
}
```

Data Actions of SEP Boot Process: BootOS

- Map 0x20000000
    - Add PTEs
- Channel Switching only applied to firmware

Virt	Size	Chan+PA	PA
0x20000000	0xFF7000	0x8900000000	0x800000000

PTE Virt	Size	Chan+PA	PA
0x101F0800	0x40	0xC900FFA800	0x800FFA800

```
// 0x10005D8C
void LoadAndBootSEPOS(uint32_t paLow, uint32_t paHigh, uint32_t bootType)
{
    // ……
->  Map_0x20000000_ByEncChan(1);
    MoveImg4PayloadToTZ0Start(fwBuf, img4Info.im4pPayloadAddr, img4Info.im4pPayloadSize);
    memset(fwBuf + img4Info.im4pPayloadSize, 0, fwBufSize - img4Info.im4pPayloadSize);
    DecryptImg4Payload(fwBuf, fwBuf, img4Info.im4pPayloadSize, &v15, keySizeBits, &a6);
    UnMap_0x20000000();
    SendBootOSRespondToAP();
    SetBootArgAndJumpToOS(&img4Info);
    // ……
}
```

Data Actions of SEP Boot Process: BootOS

- Move the firmware data to the start of the TZ0 buffer,
- Clear the memory after the firmware,
- And then decrypt the firmware in place.
- Unmap 0x20000000

Virt	Size	Chan+PA	PA
0x101F0800	0x40	0xC900FFA800	0x800FFA800

```
// 0x10005D8C
void LoadAndBootSEPOS(uint32_t paLow, uint32_t paHigh, uint32_t bootType)
{
    // ……
    Map_0x20000000_ByEncChan(1);
->  MoveImg4PayloadToTZ0Start(fwBuf, img4Info.im4pPayloadAddr, img4Info.im4pPayloadSize);
->  memset(fwBuf + img4Info.im4pPayloadSize, 0, fwBufSize - img4Info.im4pPayloadSize);
->  DecryptImg4Payload(fwBuf, fwBuf, img4Info.im4pPayloadSize, &v15, keySizeBits, &a6);
->  UnMap_0x20000000();
    SendBootOSRespondToAP();
    SetBootArgAndJumpToOS(&img4Info);
    // ……
}
```

Data Actions of SEP Boot Process: BootArg & Trampoline

- Map 0x0
  - Map PA 0x0 to PA 0xC900000000
  - Map VA 0x0 to PA 0x0

PTE Virt	Size	Chan+PA	PA
0x101F0000	0x40	0xC900FFA000	0x800FFA000

PTE Virt	Size	Chan+PA	PA
0x101F1000	0xFC0	0xC900FFB000	0x800FFB000

Virt	Size	Chan+PA	PA
0x0	0xFF8000	0x8900000000	0x800000000

Tramp Virt	Size	Chan+PA	PA
0xFF7000	0x1000	0x8900FF7000	0x800FF7000

Data Actions of SEP Boot Process: BootArg & Trampoline

- Place boot arguments to 0xFF7800 (0xFF7000 + 0x800)

Virt	Size	Chan+PA	PA
0xFF7800	*0x60*	0x8900FF7800	0x800FF7800

Data Actions of SEP Boot Process: BootArg & Trampoline

- Clear Memory:
  - DCCMVAC_Range(0x10140000, 0x78)
  - bzero(0x10120000, 0x1000)
  - bzero(0x10130000, 0x2000)
  - bzero(0x10160000, 0x1000)
  - SetStackValue_From_0x3D600F80_To_0x3D601000()

Data Actions of SEP Boot Process: BootArg & Trampoline

- Copy exception vector and boot trampoline
  - memcpy(0xFF7000, 0x10000000, 0x280)
  - memcpy(0xFF7020, 0xFF7040, 0x20)
  - DCCMVAC_Range(0xFF7000, 0x280)
  - ICIMVAU_Range(0xFF7000, 0x280)

Virt	Size	Chan+PA	PA
0xFF7000	0x1000	0x8900FF7000	0x800FF7000

Data Actions of SEP Boot Process: BootArg & Trampoline

- Run the boot trampoline located at VA 0xFF70C0.
- Then, jump to VA 0x0, which is the address of the firmware.

Data Actions of SEP Boot Process: BootArg & Trampoline

```
// 0x00FF70C0
void TrampStart() {
  ConfigAndJumpToAddrZero();
}

void ConfigAndJumpToAddrZero() {
  _WriteSystemReg(VBAR, VBAR_Tramp); __isb(0xFu);
  _WriteSystemReg(MAIR0, _ReadSystemReg(MAIR0) & 0xFFFFFF00 | 4);
  __dsb(0xFu); __isb(0xFu);
  for ( i = 0x10000000; i < 0x10000100; i += 0x1000 ) _WriteSystemReg(TLBIMVA, i);
  __dsb(0xFu); __isb(0xFu);
  MEMORY[0x3D200038] = 1;
  _WriteSystemReg(SCTLR, 0xC51078u);
  __dsb(0xFu); __isb(0xFu);
  InvalidateAllCachedCopiesOfTranslationTableEntries();
  DCCISW_AllAddrSpace();
  ICIALLU_AddrZero();
  MovR10ToR0_JumpToAddrZero();
}

void MovR10ToR0_JumpToAddrZero()
{
  JUMPOUT(0);
}
```

Conclusion

- The bug is an input validation issue in the channel selection hardware.
- AP is a 64-bit processor, while SEP is a 32-bit processor.
- This increases the complexity of SoC design.
- Variant Analysis ?
- We should not only learn from our own mistakes,
- But also learn from others' mistakes.

References

- https://www.blackhat.com/docs/us-16/materials/us-16-Mandt-Demystifying-The-Secure-Enclave-Processor.pdf
- https://github.com/windknown/presentations/blob/master/Attack_Secure_Boot_of_SEP.pdf
- https://github.com/axi0mX/ipwndfu
- https://securerom.fun/
- https://github.com/checkra1n/pongoOS
- https://support.apple.com/guide/security/
- https://developer.arm.com/

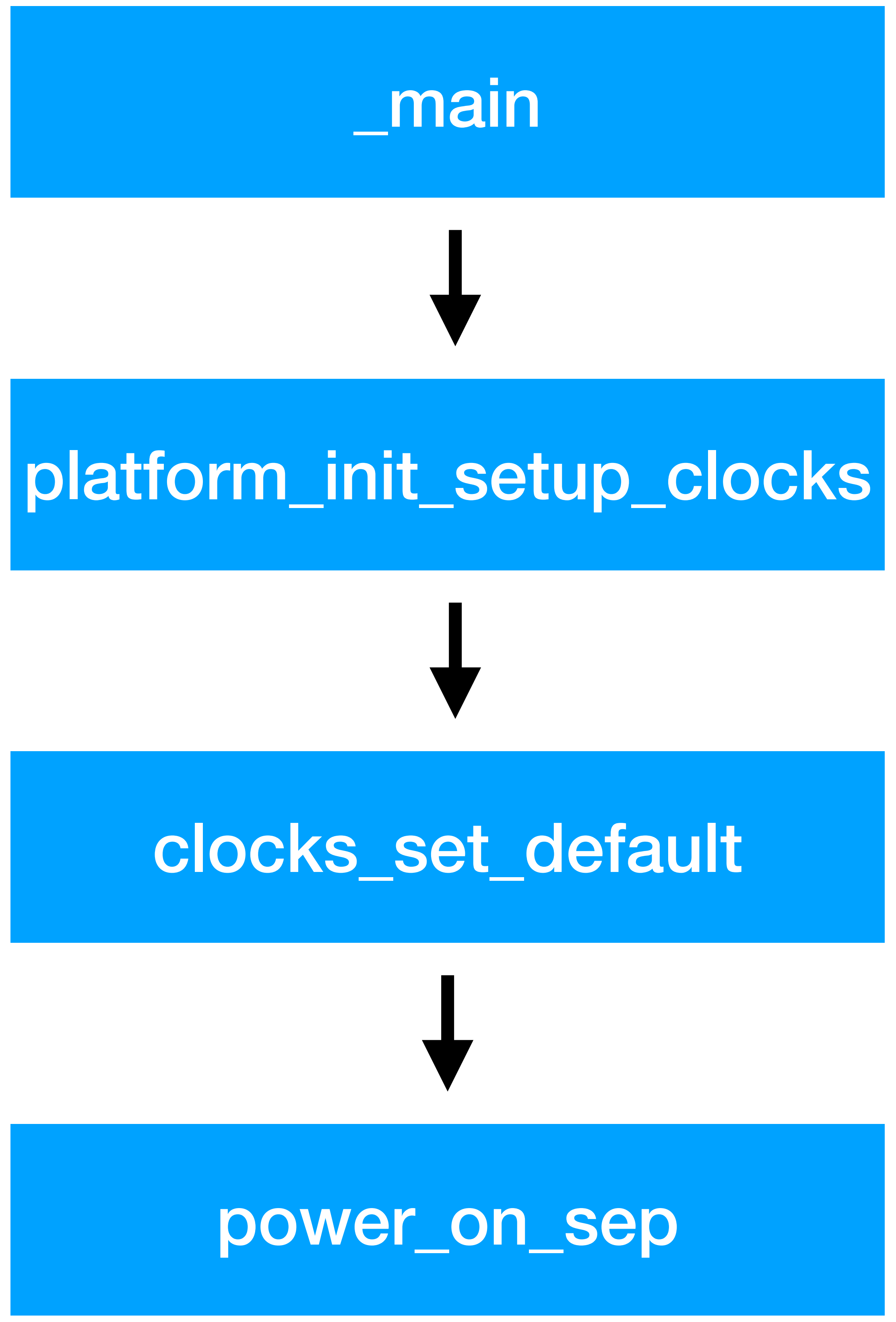
Thanks and Questions

- Questions?

Appendix: Reverse Engineering SEP Boot Process

RE SEP Boot Process: Power on

- SEP is powered on by SecureROM
- seems there is no way to only reboot SEP



```
// power_on_sep: A7
MEMORY[0x20E020268] &= ~0x10000000u;
while ( (MEMORY[0x20E020268] & 0xF0) != 0xF0 )
    ;
```

```
// power_on_sep: A8, A9, A10
MEMORY[0x20E080400] &= ~0xEFFFFBFF;
while ( (MEMORY[0x20E080400] & 0xF0) != 0xF0 )
    ;
```

RE SEP Boot Process: Init

```
SEPROM:00000000                 CODE32
SEPROM:00000000                 LDR             PC, =On_Reset
SEPROM:00000004 ; ---------------------------------------------------------------------------
SEPROM:00000004                 LDR             PC, =On_UndInst
SEPROM:00000008 ; ---------------------------------------------------------------------------
SEPROM:00000008                 LDR             PC, =On_TrapCall
SEPROM:0000000C ; ---------------------------------------------------------------------------
SEPROM:0000000C                 LDR             PC, =On_PrefetchAbort
SEPROM:00000010 ; ---------------------------------------------------------------------------
SEPROM:00000010                 LDR             PC, =On_DataAbort
SEPROM:00000014 ; ---------------------------------------------------------------------------
SEPROM:00000014                 LDR             PC, =On_NotUsed
SEPROM:00000018 ; ---------------------------------------------------------------------------
SEPROM:00000018                 LDR             PC, =On_IRQ
SEPROM:0000001C ; ---------------------------------------------------------------------------
SEPROM:0000001C                 LDR             PC, =On_FIQ
SEPROM:0000001C ; ---------------------------------------------------------------------------
```

RE SEP Boot Process: Init

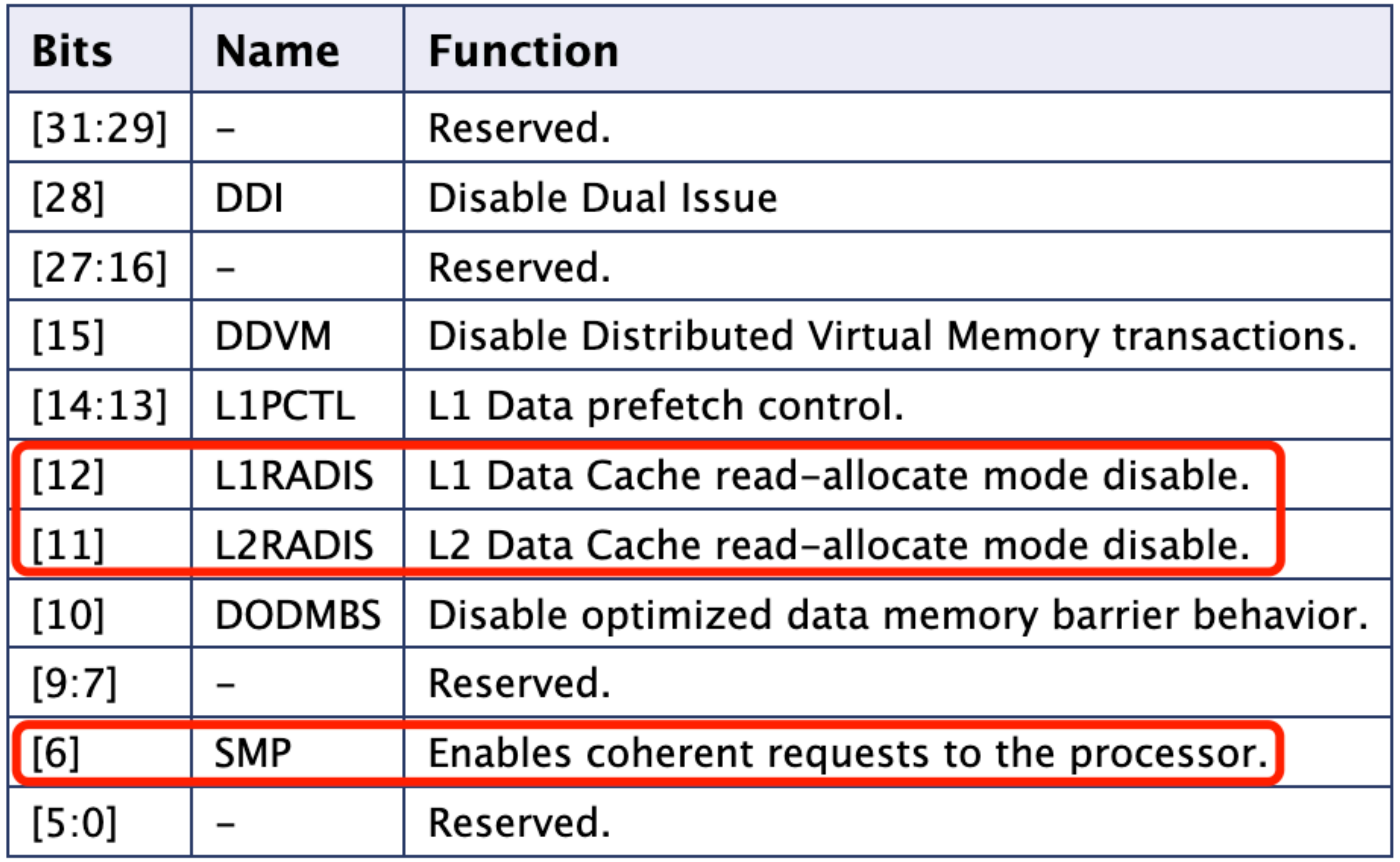
```
SEPROM:0000408C On_Reset                                        ; CODE XREF: SEPROM:00000000↑j
SEPROM:0000408C                                                 ; DATA XREF: SEPROM:00000000↑o ...
SEPROM:0000408C                 ISB             SY
SEPROM:00004090                 MRRC            CNTPCT, R8, SB
SEPROM:00004094                 ISB             SY
SEPROM:00004098                 MOVS            R0, #0
SEPROM:0000409C                 MCR             VBAR, R0
SEPROM:000040A0                 ISB             SY
SEPROM:000040A4                 MRC             ACTLR, R0
SEPROM:000040A8                 ORR             R0, R0, #0x1840
SEPROM:000040AC                 MCR             ACTLR, R0
SEPROM:000040B0                 ISB             SY
SEPROM:000040B4                 MOVW            R0, #0x4F04
SEPROM:000040B8                 MOVT            R0, #0x4FF4
SEPROM:000040BC                 MCR             MAIR0, R0
SEPROM:000040C0                 MOVW            R0, #0x404
SEPROM:000040C4                 MOVT            R0, #0x404
SEPROM:000040C8                 MCR             MAIR1, R0
SEPROM:000040CC                 MOVW            R0, #0x1000
SEPROM:000040D0                 MOVT            R0, #0xDA0
SEPROM:000040D4                 MOVS            R1, #2
SEPROM:000040D8                 MCRR            TTBR0, R0, R1
SEPROM:000040DC                 MOVW            R0, #0x2302
SEPROM:000040E0                 MOVT            R0, #0x8080
SEPROM:000040E4                 MCR             TTBCR, R0
SEPROM:000040E8                 ISB             SY
SEPROM:000040EC                 MOVW            R0, #0x187D
SEPROM:000040F0                 MOVT            R0, #0x30C5
SEPROM:000040F4                 MCR             SCTLR, R0
SEPROM:000040F8                 DSB             SY
SEPROM:000040FC                 ISB             SY
SEPROM:00004100                 MOVW            R12, #0x4289
SEPROM:00004104                 MOVT            R12, #0x1000
SEPROM:00004108                 BX              R12
SEPROM:00004108 ; End of function On_Reset
```

RE SEP Boot Process: Init

```
void On_Reset()
{
    __isb(0xFu);
    __asm { MRRC            CNTPCT, R8, SB }
    __isb(0xFu);
    _WriteSystemReg(VBAR, 0);
    __isb(0xFu);
    _WriteSystemReg(ACTLR, _ReadSystemReg(ACTLR) | 0x1840);
    __isb(0xFu);
    _WriteSystemReg(MAIR0, 0x4FF44F04u);
    _WriteSystemReg(MAIR1, 0x04040404u);
    __asm { MCRR            TTBR0, R0, R1; 0x20DA01000 }
    _WriteSystemReg(TTBCR, 0x80802302);
    __isb(0xFu);
    _WriteSystemReg(SCTLR, 0x30C5187Du);
    __dsb(0xFu);
    __isb(0xFu);
    MEMORY[0x10004288]();
}
```

RE SEP Boot Process: Init

- ACTLR is IMPLEMENTATION DEFINED.
- Don’t have the manual, use Cortex-A7’s manual as a reference
- ACTLR |= 0x1840
- 0b1_1000_0100_0000



Bits	Name	Function
[31:29]	–	Reserved.
[28]	DDI	Disable Dual Issue
[27:16]	–	Reserved.
[15]	DDVM	Disable Distributed Virtual Memory transactions.
[14:13]	L1PCTL	L1 Data prefetch control.
[12]	L1RADIS	L1 Data Cache read-allocate mode disable.
[11]	L2RADIS	L2 Data Cache read-allocate mode disable.
[10]	DODMBS	Disable optimized data memory barrier behavior.
[9:7]	–	Reserved.
[6]	SMP	Enables coherent requests to the processor.
[5:0]	–	Reserved.

RE SEP Boot Process: Init

- VBAR = 0
- This is a temporary exception vector, which will be changed later.
- MAIR0: 0x4FF44F04
- MAIR1: 0x04040404

Index	7	6	5	4	3	2	1	0
Value	04	04	04	04	4F	F4	4F	04

RE SEP Boot Process: Init

Index	7	6	5	**4**	3	2	1	0
Value	04	04	04	**04**	4F	F4	4F	04

- 0x04: Device memory, Device-nGnRE memory
- 0x4F: Normal memory, Outer Non-cacheable, Inner Write-Back Non-transient
- 0xF4: Normal memory, Outer Write-Back Non-transient

RE SEP Boot Process: Init

- TTBR0 = 0x20DA01000

VA	PA	Size	XN	PXN	AP	Attr Index	Attr
0x00004000	0x020DA04000	0x1000	Executable	Privileged Executable	Read-only	0	0x04, Device memory
0x10000000	0x020DA00000	0x100000	Executable	Privileged Executable	Read-only	3	0x4F, Normal memory
0x10180000	0x0080000000	0x3000	Executable-Never	Privileged Executable-Never	Read/write	4	0x04, Device memory
0x30000000	0x0200000000	0x10000000	Executable-Never	Privileged Executable-Never	Read/write	0	0x04, Device memory

- PTE for VA: 0x4000
- This is the page to which `MCRR TTBR0, R0, R1` belongs
- The purpose of this item is just for setting page table

RE SEP Boot Process: Init

VA	PA	Size	XN	PXN	AP	Attr Index	Attr
0x00004000	0x020DA04000	0x1000	Executable	Privileged Executable	Read-only	0	0x04, Device memory
0x10000000	0x020DA00000	0x100000	Executable	Privileged Executable	Read-only	3	0x4F, Normal memory
0x10180000	0x0080000000	0x3000	Executable-Never	Privileged Executable-Never	Read/write	**4**	**0x04**, Device memory
0x30000000	0x0200000000	0x10000000	Executable-Never	Privileged Executable-Never	Read/write	0	0x04, Device memory

- 0x10000000: ROM
- 0x10180000: not be used yet
- 0x30000000: MMIO & Stack

RE SEP Boot Process: Init

- TTBCR: 0x80802302
- 0b1000_0000_1000_0000_0010_0011_0000_0010
  - T0SZ, bits[2:0]: 2, The size offset of the memory region addressed by TTBR0
  - IRGN0, bits[9:8]: 3, Inner cacheability attribute for memory associated with translation table walks
  - SH0, bits[13:12], 2, Shareability attribute for memory associated with translation table walks using TTBR0

RE SEP Boot Process: Init

- TTBCR: 0x80802302
- 0b1000_0000_1000_0000_0010_0011_0000_0010
    - A1, bit[22]: 0, Selects whether TTBR0 or TTBR1 defines the ASID
    - EPD1, bit[23]: 1, Translation table walk disable for translations using TTBR1
    - EAE, bit[31]: 1, Extended Address Enable, Use the 40-bit translation system, with the Long-descriptor translation table format.

RE SEP Boot Process: Init

- SCTLR: 0x30C5187D
- 0b0011_0000_1100_0101_0001_1000_0111_1101
  - M, bit[0]: 1, MMU enable
  - A, bit[1]: 0, Alignment fault checking disabled
  - C, bit[2]: 1, Data and unified caches enabled
  - Bits[4:3]: 3, Reserved, RAO/SBOP
  - CP15BEN, bit[5]: 1, CP15 barrier operations enabled
  - Bit[6]: 1, Reserved, RAO/SBOP

RE SEP Boot Process: Init

- SCTLR: 0x30C5187D
- 0b0011_0000_1100_0101_0001_1000_0111_1101
    - Z, bit[11]: 1, Program flow prediction enabled
    - I, bit[12]: 1, Instruction caches enabled
    - Bit[16]: 1, Reserved, RAO/SBOP.
    - Bit[18]: 1, Reserved, RAO/SBOP.
    - FI, bit[21]: 0, All performance features enabled

RE SEP Boot Process: Init

- SCTLR: 0x30C5187D
- 0b0011_0000_1100_0101_0001_1000_0111_1101
  - U, bit[22]: 1, In ARMv7 this bit is RAO/SBOP, indicating use of the alignment model described in Alignment support.
  - Bit[23]: 1, Reserved, RAO/SBOP.
  - NMFI, bit[27]: 0, Software can mask FIQs by setting the CPSR.F bit to 1.
  - TRE, bit[28]: 1, TEX remap enabled
  - AFE, bit[29]: 1, In the translation table descriptors, AP[0] is the Access flag

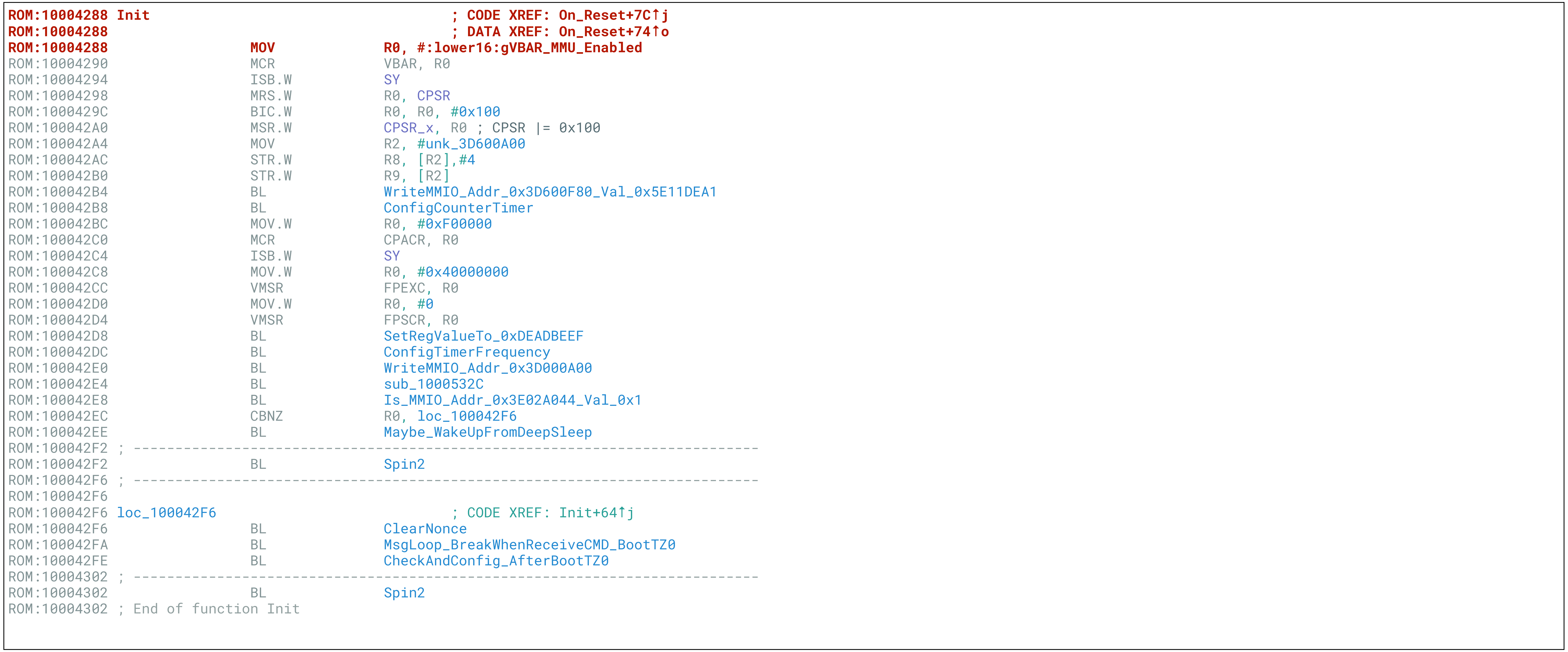
RE SEP Boot Process: Init

- Where are we ?

```
void On_Reset()
{
    __isb(0xFu);
    __asm { MRRC            CNTPCT, R8, SB }
    __isb(0xFu);
    _WriteSystemReg(VBAR, 0);
    __isb(0xFu);
    _WriteSystemReg(ACTLR, _ReadSystemReg(ACTLR) | 0x1840);
    __isb(0xFu);
    _WriteSystemReg(MAIR0, 0x4FF44F04u);
    _WriteSystemReg(MAIR1, 0x04040404u);
    __asm { MCRR            TTBR0, R0, R1; 0x20DA01000 }
    _WriteSystemReg(TTBCR, 0x80802302);
    __isb(0xFu);
    _WriteSystemReg(SCTLR, 0x30C5187Du);
    __dsb(0xFu);
    __isb(0xFu);
    MEMORY[0x10004288](); // <- We are here
}
```

RE SEP Boot Process: Init

- Now we are here



```
ROM:10004288 Init                                    ; CODE XREF: On_Reset+7C↑j
ROM:10004288                                         ; DATA XREF: On_Reset+74↑o
ROM:10004288                 MOV             R0, #:lower16:gVBAR_MMU_Enabled
ROM:10004290                 MCR             VBAR, R0
ROM:10004294                 ISB.W           SY
ROM:10004298                 MRS.W           R0, CPSR
ROM:1000429C                 BIC.W           R0, R0, #0x100
ROM:100042A0                 MSR.W           CPSR_x, R0 ; CPSR |= 0x100
ROM:100042A4                 MOV             R2, #unk_3D600A00
ROM:100042AC                 STR.W           R8, [R2],#4
ROM:100042B0                 STR.W           R9, [R2]
ROM:100042B4                 BL              WriteMMIO_Addr_0x3D600F80_Val_0x5E11DEA1
ROM:100042B8                 BL              ConfigCounterTimer
ROM:100042BC                 MOV.W           R0, #0xF00000
ROM:100042C0                 MCR             CPACR, R0
ROM:100042C4                 ISB.W           SY
ROM:100042C8                 MOV.W           R0, #0x40000000
ROM:100042CC                 VMSR            FPEXC, R0
ROM:100042D0                 MOV.W           R0, #0
ROM:100042D4                 VMSR            FPSCR, R0
ROM:100042D8                 BL              SetRegValueTo_0xDEADBEEF
ROM:100042DC                 BL              ConfigTimerFrequency
ROM:100042E0                 BL              WriteMMIO_Addr_0x3D000A00
ROM:100042E4                 BL              sub_1000532C
ROM:100042E8                 BL              Is_MMIO_Addr_0x3E02A044_Val_0x1
ROM:100042EC                 CBNZ            R0, loc_100042F6
ROM:100042EE                 BL              Maybe_WakeUpFromDeepSleep
ROM:100042F2 ; ---------------------------------------------------------------------------
ROM:100042F2                 BL              Spin2
ROM:100042F6 ; ---------------------------------------------------------------------------
ROM:100042F6
ROM:100042F6 loc_100042F6                            ; CODE XREF: Init+64↑j
ROM:100042F6                 BL              ClearNonce
ROM:100042FA                 BL              MsgLoop_BreakWhenReceiveCMD_BootTZ0
ROM:100042FE                 BL              CheckAndConfig_AfterBootTZ0
ROM:10004302 ; ---------------------------------------------------------------------------
ROM:10004302                 BL              Spin2
ROM:10004302 ; End of function Init
```

RE SEP Boot Process: Init

```
void Init() {
  _WriteSystemReg(VBAR, 0x10004000u); __isb(0xFu);
  _R0 = __get_CPSR() & 0xFFFFFEFF;
  __asm { MSR.W           CPSR_x, R0; CPSR |= 0x100 }
  unk_3D600A00 = v0;
  dword_3D600A04 = v1;
  WriteMMIO_Addr_0x3D600F80_Val_0x5E11DEA1();
  ConfigCounterTimer();
  _WriteSystemReg(CPACR, 0xF00000u); __isb(0xFu);
  _R0 = 0x40000000; __asm { VMSR            FPEXC, R0 }
  _R0 = 0; __asm { VMSR            FPSCR, R0 }
  SetRegValueTo_0xDEADBEEF();
  ConfigTimerFrequency();
  WriteMMIO_Addr_0x3D000A00();
  sub_1000532C();
  if ( !Is_MMIO_Addr_0x3E02A044_Val_0x1() )
    Maybe_WakeUpFromDeepSleep();
  ClearNonce();

  MsgLoop_BreakWhenReceiveCMD_BootTZ0();
  CheckAndConfig_AfterBootTZ0();
}
```

RE SEP Boot Process: Init

- CPSR |= 0x100
    - bit [8]: 1, SError interrupt mask bit, Exception masked.
- CPACR: 0xF00000
    - cp10, bits [21:20]: 0x3, This control permits full access to the floating-point and Advanced SIMD functionality from PL0 and PL1.
    - cp11, bits [23:22], 0x3
- FPEXC: 0x40000000
    - EN, bit[30]: 1, The Advanced SIMD and Floating-point Extensions are enabled and operate normally.
- FPSCR: 0

RE SEP Boot Process: Init

- Init() -> MsgLoop_BreakAfter_BootTZ0()

```
// Addr: 0x10004C44
void __cdecl MsgLoop_BreakWhenReceiveCMD_BootTZ0()
{
  sep_message_without_data msg; // r0

  InitMailbox();
  SendResponse(0xD200FF, 1);
  do
    msg = (sep_message_without_data)ReadMsg();
  while ( ProcessMsg(msg) );
}
```

- When `ProcessMsg` returns 0, the loop will break

RE SEP Boot Process: BootTZ0

- Init() -> MsgLoop_BreakAfter_BootTZ0() -> ProcessMsg()

```
enum OPCODE
{
  opInvalid = 0x0,
  opPING = 0x1,
  opPING2 = 0x2,
  opGenNonce = 0x3,
  opGetNonceWord = 0x4,
  opBootTZ0 = 0x5,
  opPanic = 0xA,
};

struct CMD_Table_Item
{
  OPCODE cmd;
  int (*handler)(void);
};
```

RE SEP Boot Process: BootTZ0

- Init() -> MsgLoop_BreakAfter_BootTZ0() -> ProcessMsg()

```
ROM:10004E6C ; CMD_Table_Item gCMD_Table
ROM:10004E6C gCMD_Table      CMD_Table_Item <opPING, CMDHandler_Ping+1>
ROM:10004E6C                                         ; DATA XREF: ProcessMsg+18↑o
ROM:10004E6C                 CMD_Table_Item <opPING2, CMDHandler_Ping1+1>
ROM:10004E6C                 CMD_Table_Item <opGenNonce, CMDHandler_GenNonce+1>
ROM:10004E6C                 CMD_Table_Item <opGetNonceWord, CMDHandler_GetNonce+1>
ROM:10004E6C                 CMD_Table_Item <opBootTZ0, CMDHandler_BootTZ0+1>
ROM:10004E6C                 CMD_Table_Item <opPanic, CMDHandler_Panic+1>
ROM:10004E6C                 CMD_Table_Item  <0>
```

RE SEP Boot Process: BootTZ0

- Init() -> MsgLoop_BreakAfter_BootTZ0() -> ProcessMsg()

```
// 0x10004D44
int __fastcall ProcessMsg(sep_message_without_data msg)
{
  CMD_Table_Item *CMD_TablePtr; // r3
  OPCODE cmd; // r4
  int (*handler)(void); // r5

  if ( msg.endpoint != 0xFF )
    return OnInvalidEndpoint_RetVal_0x1(*(unsigned __int16 *)&msg.endpoint);
  CMD_TablePtr = &gCMD_Table;
  while ( 1 )
  {
    cmd = CMD_TablePtr->cmd;
    handler = (int (*)(void))CMD_TablePtr->handler;
    ++CMD_TablePtr;
    if ( cmd == opInvalid )
      break;
    if ( msg.opcode == cmd )
      return handler();
  }
  return OnInvalidOpCode_RetVal_0x1(*(sep_message_without_data **)&msg);
}
```

RE SEP Boot Process: BootTZ0

- AP sends `Boot TZ0` to SEP
  - Write MMIO: TZ0 Size
  - Write MMIO: TZ1 Size
  - Write MMIO: Device Memory Size
  - Write MMIO: TZ0 Lock
  - Write MMIO: TZ1 Lock
  - Send `Boot TZ0` command

RE SEP Boot Process: BootTZ0

- AP sends `Boot TZ0` to SEP

```
enum {
    kOpCode_Ping = 1,
    kOpCode_Ping2 = 2,
    kOpCode_GenerateNonce = 3,
    kOpCode_GetNonceWord = 4,
    kOpCode_BootTZ0 = 5,
    kOpCode_BootSEPOS = 6,
};

struct sep_message {
    uint8_t        endpoint;
    uint8_t        tag;
    uint8_t        opcode;
    uint8_t        param;
    uint32_t    data;
} __attribute__((packed));
```

RE SEP Boot Process: BootTZ0

- AP sends `Boot TZ0` to SEP

```
typedef int32_t (*akf_start_t)(const KFWRAPPER_TYPE_T kfw_type,
                               const addr_t firmware_address,
                               const uint64_t firmware_size);
typedef int32_t (*akf_stop_t)(const KFWRAPPER_TYPE_T kfw_type);
typedef int32_t (*akf_start_sep_t)(void);

typedef int32_t (*akf_send_mbox_t)(const KFWRAPPER_TYPE_T kfw_type,
                                   uint64_t msg,
                                   uint32_t wait_timeout);
typedef int32_t (*akf_recv_mbox_t)(const KFWRAPPER_TYPE_T kfw_type,
                                   volatile uint64_t *msg,
                                   uint32_t wait_timeout);
typedef int32_t (*sep_client_reader_t)(const KFWRAPPER_TYPE_T kfw_type,
                                       volatile uint64_t *msg,
                                       uint32_t wait_timeout);
```

RE SEP Boot Process: BootTZ0

- AP sends `Boot TZ0` to SEP

```
// iOS-v11.0-15A372-iPhone6,1, iBEC
akf_start_t akf_start = (akf_start_t)0x0000000830003B68;
akf_stop_t akf_stop = (akf_stop_t)0x0000000830004040;
akf_send_mbox_t akf_send_mbox = (akf_send_mbox_t)0x0000000830003F7C;
akf_recv_mbox_t akf_recv_mbox = (akf_recv_mbox_t)0x0000000830003F4C;

// A7
volatile uint32_t *gTZ0ValuePtr = (volatile uint32_t *)0x200000908;
volatile uint32_t *gTZ1ValuePtr = (volatile uint32_t *)0x20000090C;
volatile uint32_t *gTZ0LockPtr = (volatile uint32_t *)0x200000910;
volatile uint32_t *gTZ1LockPtr = (volatile uint32_t *)0x200000914;
volatile uint32_t *gPhyMemSizePtr = (volatile uint32_t *)0x20000081C;

int32_t akf_send_mbox2(const KFWRAPPER_TYPE_T kfw_type,
                       uint64_t *msg,
                       uint32_t wait_timeout)
{
    uint64_t msg_val = *msg;
    return akf_send_mbox(kfw_type, msg_val, wait_timeout);
}
```

RE SEP Boot Process: BootTZ0

- AP sends `Boot TZ0` to SEP

```
void ConfigTZ0(void)
{
    uint32_t tz0Val = TZ_Encode(gTZ0StartAddr, gTZ0EndAddr);
    uint32_t tz1Val = TZ_Encode(gTZ1StartAddr, gTZ1EndAddr);

    *gTZ0ValuePtr = tz0Val;
    *gTZ1ValuePtr = tz1Val;

    *gTZ0LockPtr = 0x1;
    *gTZ1LockPtr = 0x1;

    // set device memory size
    *gPhyMemSizePtr = 0x7F;

    while (*gTZ0LockPtr != 1) {
        *gTZ1LockPtr = 1;
    }

    while (*gTZ1LockPtr != 1) {
        *gTZ1LockPtr = 1;
    }
}
```

RE SEP Boot Process: BootTZ0

- AP sends `Boot TZ0` to SEP

```
void BootTZ0(void)
{
    akf_start_sep();

    _sep_create_message(&_sep_send_msg, kMsg_BootTZ0, 0, 0);
    if (akf_send_mbox2(KFW_SEP, (uint64_t *)&_sep_send_msg, A7IOP_NO_WAIT)) {
        printf("[-] unable to send BootTZ0 to SEP mailbox\n");
        goto exit;
    }
    if (_sep_client_reader(BOOT_TZ0_TIMEOUT)) {
        printf("[-] SEP not responding to BootTZ0\n");
        goto exit;
    }

exit:
    akf_stop(KFW_SEP);
}
```

RE SEP Boot Process: BootTZ0

- AP sends `Boot TZ0` to SEP

```c
void sep_cmd_boot_tz0(void)
{
    printf("[+] sep cmd: boot tz0\n");

    Ping2();
    ConfigTZ0();
    BootTZ0();

    printf("[+] boot tz0: done\n");
}
```

RE SEP Boot Process: BootTZ0

- Init() -> MsgLoop_BreakAfter_BootTZ0() -> ProcessMsg() -> handler()

```
// 0x10004DB0
int __fastcall CMDHandler_Ping(unsigned __int16 a1)
{
  SendResponse(a1 | 0x650000, 0);
  return 1;
}


// 0x10004E3C
int __fastcall CMDHandler_BootTZ0(unsigned __int16 a1)
{
  SendResponse(a1 | 0x690000, 0);
  return 0;
}
```

- When received `Boot TZ0`, `MsgLoop_BreakAfter_BootTZ0` will return.

RE SEP Boot Process: BootTZ0

- Init() -> CheckAndConfig_AfterBootTZ0()

```
// 0x10004EA4
void __noreturn CheckAndConfig_AfterBootTZ0()
{
  if ( !Get_0x3E02A004_Bit_31() )
    Spin2();
  if ( Check_0x3E02A004_Bit_30() )
    Spin2();
  CheckLockStatus_SaveTZ0Values();
  Maybe_ConfigMemEncryptionHw();
  Map_0x10180000_0x1000_NotEncrypted();
  bzero_0x10180078_0xF88();
  GenRandom(gKey_Chan_0xC8_TZ0, 192u);
  SetEncryptionKey_Chan1(gKey_Chan_0xC8_TZ0);
  CopyPageTableAndConfig_TTBR0_AfterBootTZ0();
  AddPageTableItem_AfterBootTZ0();
  bzero_Addr_0x10120140_Size_0x148();
  copy_from_0x1000F000_to_0x10120000_size_0x140();
  bzero_10130000_0x2000();
  GenRandom((uint8_t *)&gStackCookie, 32u);
  SetStackValue_From_0x3D600F80_To_0x3D601000();
  Maybe_CallTestFunctionOnDebugFirmware();
  MsgLoopAfterBootTZ0();
  while ( 1 )
  {
    __wfi();
    __wfe();
  }
}
```

RE SEP Boot Process: BootTZ0

- Init() -> CheckAndConfig_AfterBootTZ0() -> CheckLockStatus_SaveTZ0Values()

```
// 0x100055F8
void __cdecl CheckLockStatus_SaveTZ0Values()
{
  unsigned __int64 tz0End; // r2
  bool isTZEnd_GE_TZStart; // cf
  unsigned int tz0Size; // r2

  if ( (gTZ0_LockReg & 1) == 0 )
    Spin2();
  if ( (gTZ1_LockReg & 1) == 0 )
    Spin2();
  LODWORD(gTZ0_Offset_Start) = gTZ0_Value << 20;
  HIDWORD(gTZ0_Offset_Start) = (unsigned __int16)(gTZ0_Value & 0x3FFF) >> 12;
  tz0End = (unsigned __int64)(((gTZ0_Value & 0x3FFF0000) >> 16) + 1) << 20;
  gTZ0_Offset_End = tz0End;
  isTZEnd_GE_TZStart = (unsigned int)tz0End >= gTZ0_Value << 20;
  tz0Size = tz0End - (gTZ0_Value << 20);
  if ( HIDWORD(tz0End) != ((unsigned __int16)(gTZ0_Value & 0x3FFF) >> 12) + !isTZEnd_GE_TZStart )
    Spin2();
  if ( !tz0Size )
    Spin2();
  if ( tz0Size > Size_256MB )
    Spin2();
  gTZ0Size = tz0Size;
}
```

RE SEP Boot Process: BootTZ0

- Init() -> CheckAndConfig_AfterBootTZ0() -> Map_0x10180000_0x1000_NotEncrypted()

```
// 0x10004BF8
void __cdecl Map_0x10180000_0x1000_NotEncrypted()
{
  __int64 physAddr; // r0

  physAddr = TZ0_End_Minus_0x1000();
  MapPA_0x80000000_ToPA(physAddr, 0x1000u);
}

// 0x10004B30
void __fastcall MapPA_0x80000000_ToPA(__int64 targetPA, unsigned int size)
{
  if ( (targetPA & 0xFFF) != 0 || __PAIR64__(size & 0xFFF, size) > 0x3000 )
    Spin2();
  Map_0x10180000_0x10183000_DeviceMem(targetPA, HIDWORD(targetPA), size, size & 0xFFF);
  MapPA_0x80000000(targetPA + 0x80000000, (unsigned __int64)(targetPA - 0x80000000i64) >> 32);
  ConfigPhysToPhysMap(0x80000000, 0);
  sub_10004B18(size + 0x7FFFF000, 0);
  SetMAIR_0x10180000_0x10183000_ToNormal();
}
```

RE SEP Boot Process: BootTZ0

- Init() -> CheckAndConfig_AfterBootTZ0() -> CopyPageTableAndConfig_TTBR0_AfterBootTZ0()

```
// 0x10005068
void __cdecl CopyPageTableAndConfig_TTBR0_AfterBootTZ0()
{
  __int64 *ptROM; // r0
  _QWORD *ptTZ0; // r1
  __int64 v2; // d0
  __int64 v3; // d1
  __int64 v4; // d2
  __int64 v5; // d3
  uint64_t srcValue; // r0
  __int64 *ptrPageTable_BootTZ0_L1; // r2
  int loopCount; // r3

  EncryptMap_0x10180000_To_0xFFA000_0x3000();
  ptROM = &gPageTable_ROM_L2;                   // 0x10001000
  ptTZ0 = &gPageTable_TZ0_0x10180000;
  do                                            // copy page table from rom to ram
  {
    v2 = *ptROM;
    v3 = ptROM[1];
    v4 = ptROM[2];
    v5 = ptROM[3];
    ptROM += 4;
    *ptTZ0 = v2;
    ptTZ0[1] = v3;
    ptTZ0[2] = v4;
    ptTZ0[3] = v5;
    ptTZ0 += 4;
  }
  while ( (unsigned int)ptROM < 0x10004000 );
  gPageTable_TZ0_0x10180000 = 0i64;
  qword_10181020 = 0i64;
  gPageTable_BootTZ0_L2_Item_To_L3_2 = gPageTable_ROM_L2_Item_To_L3 & 0xFFFFFF0000000FFFui64 | (GetEncryptChanAddr_C8_Offset_0xFF8000()
                                                                                                 + 0x4000);
  srcValue = (GetEncryptChanAddr_C8_Offset_0xFF8000() + 0x2000) | 0x60000000000607i64;
  ptrPageTable_BootTZ0_L1 = &gPageTable_BootTZ0_L1;
  loopCount = 3;
  do
  {
    *ptrPageTable_BootTZ0_L1++ = srcValue;
    srcValue += 0x1000i64;
    --loopCount;
  }
  while ( loopCount );
  Map_0x10180000_0x10183000_DeviceMem(srcValue, HIDWORD(srcValue), (uint32_t)ptrPageTable_BootTZ0_L1, 0);
  GetEncryptChanAddr_C8_Offset_0xFF8000();
  __dsb(0xFu);
  __asm { MCRR            TTBR0, R0, R1; }
  __isb(0xFu);
  UpdateTLB();
}
```

RE SEP Boot Process: BootTZ0

- Init() -> CheckAndConfig_AfterBootTZ0() -> AddPageTableItem_AfterBootTZ0()

```
// 0x10005158
void AddPageTableItem_AfterBootTZ0()
{
  uint64_t v0; // r0
  char v1; // cf
  uint64_t pteValue; // r0
  uint64_t *dstAddr; // r2
  int loopCount; // r12

  MEMORY[0x101F2A00] = TZ0_End_Minus_0x1000() | 0x600000000000607i64;
  v0 = GetEncryptChanAddr_C8_Offset_0xFF8000();
  LODWORD(pteValue) = v0 | 0x607;
  HIDWORD(pteValue) = (v1 + HIDWORD(v0)) | 0x600000;
  dstAddr = (uint64_t *)0x101F2980;
  loopCount = 2;
  do
  {
    *dstAddr++ = pteValue;
    pteValue += 0x1000i64;
    --loopCount;
  }
  while ( loopCount );
  MEMORY[0x101F2900] = (GetEncryptChanAddr_C8_Offset_0xFF8000() + 0x5000) | 0x600000000000607i64;
  MEMORY[0x101F2B00] = (GetEncryptChanAddr_C8_Offset_0xFF8000() + 0x6000) | 0x600000000000607i64;
  UpdateTLB();
}
```

RE SEP Boot Process: BootOS

- Init() -> CheckAndConfig_AfterBootTZ0() -> MsgLoopAfterBootTZ0()

```
// 0x10007924
void __cdecl MsgLoopAfterBootTZ0()
{
  int msgRet; // r2
  __int64 pa; // [sp+0h] [bp-10h] BYREF

  sub_10008B78(5);
  SaveTZ0InfoToLocal();
  sub_100062D4();
  InitMailbox2();
  SendMsg_OpCode_0xD2_Data_0x2();
  pa = 0i64;
  msgRet = MsgLoop_BootOS(&pa);
  if ( msgRet != 3 )
  {
    if ( (unsigned int)(msgRet - 1) <= 1 )
      LoadAndBootSEPOS(pa, HIDWORD(pa), msgRet);
    Spin3();
  }
  Maybe_OnResume();
}
```

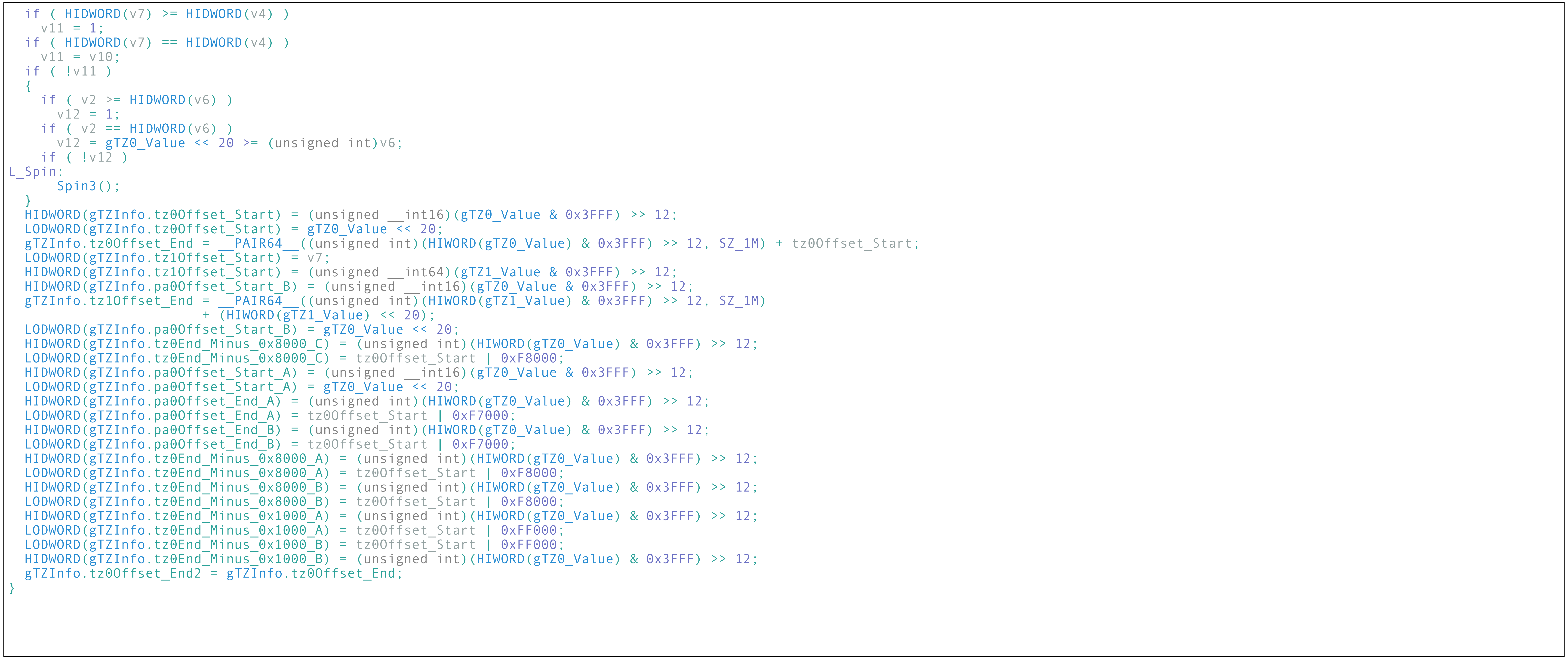
RE SEP Boot Process: BootOS

- Init() -> CheckAndConfig_AfterBootTZ0() -> SaveTZ0InfoToLocal()

```
// 0x10008DF8
void __cdecl SaveTZ0InfoToLocal()
{
  valid = 0;
  gDeviceMemSize = __PAIR64__((unsigned __int8)(gTZ_MemSizeRegVal & 0x7F) >> 5, 0x8000000) + (gTZ_MemSizeRegVal << 27);// 0x3000081C = 0x2000081C = 0x7
  v1 = 0;
  gTZInfo.unkVal0 = 0i64;
  gTZInfo.deviceMemSize = gDeviceMemSize;
  v2 = (unsigned __int16)(gTZ0_Value & 0x3FFF) >> 12;
  tz0Offset_Start = HIWORD(gTZ0_Value) << 20;
  v4 = __PAIR64__((unsigned int)(HIWORD(gTZ0_Value) & 0x3FFF) >> 12, SZ_1M) + tz0Offset_Start;
  tz0BasePA_Plus1M = tz0Offset_Start + SZ_1M;
  if ( (unsigned int)v4 > gTZ0_Value << 20 )
    valid = 1;
  isTZ0Valid = HIDWORD(v4) > v2;
  if ( HIDWORD(v4) == v2 )
    isTZ0Valid = valid;
  v6 = __PAIR64__((unsigned int)(HIWORD(gTZ1_Value) & 0x3FFF) >> 12, SZ_1M) + (HIWORD(gTZ1_Value) << 20);
  if ( !isTZ0Valid )
    goto L_Spin;
  v7 = (unsigned __int64)(gTZ1_Value & 0x3FFF) << 20;
  if ( HIDWORD(v6) > HIDWORD(v7) )
    v1 = 1;
  if ( HIDWORD(v6) == HIDWORD(v7) )
    v1 = (unsigned int)v6 > (unsigned __int16)gTZ1_Value << 20;
  if ( !v1 )
    goto L_Spin;
  v8 = 0;
  v9 = HIDWORD(v4) <= HIDWORD(gDeviceMemSize);
  if ( HIDWORD(v4) == HIDWORD(gDeviceMemSize) )
    v9 = tz0BasePA_Plus1M <= (unsigned int)gDeviceMemSize;
  if ( !v9 )
    goto L_Spin;
  if ( HIDWORD(v6) <= HIDWORD(gDeviceMemSize) )
    v8 = 1;
  if ( HIDWORD(v6) == HIDWORD(gDeviceMemSize) )
    v8 = (unsigned int)v6 <= (unsigned int)gDeviceMemSize;
  if ( !v8 )
    goto L_Spin;
  v10 = 0;
  v11 = 0;
  v12 = 0;
  if ( (unsigned int)v7 >= tz0BasePA_Plus1M )
    v10 = 1;
  if ( HIDWORD(v7) >= HIDWORD(v4) )
```

RE SEP Boot Process: BootOS

- Init() -> CheckAndConfig_AfterBootTZ0() -> SaveTZ0InfoToLocal()



```
  if ( HIDWORD(v7) >= HIDWORD(v4) )
    v11 = 1;
  if ( HIDWORD(v7) == HIDWORD(v4) )
    v11 = v10;
  if ( !v11 )
  {
    if ( v2 >= HIDWORD(v6) )
      v12 = 1;
    if ( v2 == HIDWORD(v6) )
      v12 = gTZ0_Value << 20 >= (unsigned int)v6;
    if ( !v12 )
L_Spin:
      Spin3();
  }
  HIDWORD(gTZInfo.tz0Offset_Start) = (unsigned __int16)(gTZ0_Value & 0x3FFF) >> 12;
  LODWORD(gTZInfo.tz0Offset_Start) = gTZ0_Value << 20;
  gTZInfo.tz0Offset_End = __PAIR64__((unsigned int)(HIWORD(gTZ0_Value) & 0x3FFF) >> 12, SZ_1M) + tz0Offset_Start;
  LODWORD(gTZInfo.tz1Offset_Start) = v7;
  HIDWORD(gTZInfo.tz1Offset_Start) = (unsigned __int64)(gTZ1_Value & 0x3FFF) >> 12;
  HIDWORD(gTZInfo.pa0Offset_Start_B) = (unsigned __int16)(gTZ0_Value & 0x3FFF) >> 12;
  gTZInfo.tz1Offset_End = __PAIR64__((unsigned int)(HIWORD(gTZ1_Value) & 0x3FFF) >> 12, SZ_1M)
                        + (HIWORD(gTZ1_Value) << 20);
  LODWORD(gTZInfo.pa0Offset_Start_B) = gTZ0_Value << 20;
  HIDWORD(gTZInfo.tz0End_Minus_0x8000_C) = (unsigned int)(HIWORD(gTZ0_Value) & 0x3FFF) >> 12;
  LODWORD(gTZInfo.tz0End_Minus_0x8000_C) = tz0Offset_Start | 0xF8000;
  HIDWORD(gTZInfo.pa0Offset_Start_A) = (unsigned __int16)(gTZ0_Value & 0x3FFF) >> 12;
  LODWORD(gTZInfo.pa0Offset_Start_A) = gTZ0_Value << 20;
  HIDWORD(gTZInfo.pa0Offset_End_A) = (unsigned int)(HIWORD(gTZ0_Value) & 0x3FFF) >> 12;
  LODWORD(gTZInfo.pa0Offset_End_A) = tz0Offset_Start | 0xF7000;
  HIDWORD(gTZInfo.pa0Offset_End_B) = (unsigned int)(HIWORD(gTZ0_Value) & 0x3FFF) >> 12;
  LODWORD(gTZInfo.pa0Offset_End_B) = tz0Offset_Start | 0xF7000;
  HIDWORD(gTZInfo.tz0End_Minus_0x8000_A) = (unsigned int)(HIWORD(gTZ0_Value) & 0x3FFF) >> 12;
  LODWORD(gTZInfo.tz0End_Minus_0x8000_A) = tz0Offset_Start | 0xF8000;
  HIDWORD(gTZInfo.tz0End_Minus_0x8000_B) = (unsigned int)(HIWORD(gTZ0_Value) & 0x3FFF) >> 12;
  LODWORD(gTZInfo.tz0End_Minus_0x8000_B) = tz0Offset_Start | 0xF8000;
  HIDWORD(gTZInfo.tz0End_Minus_0x1000_A) = (unsigned int)(HIWORD(gTZ0_Value) & 0x3FFF) >> 12;
  LODWORD(gTZInfo.tz0End_Minus_0x1000_A) = tz0Offset_Start | 0xFF000;
  LODWORD(gTZInfo.tz0End_Minus_0x1000_B) = tz0Offset_Start | 0xFF000;
  HIDWORD(gTZInfo.tz0End_Minus_0x1000_B) = (unsigned int)(HIWORD(gTZ0_Value) & 0x3FFF) >> 12;
  gTZInfo.tz0Offset_End2 = gTZInfo.tz0Offset_End;
}
```

RE SEP Boot Process: BootOS

- Init() -> CheckAndConfig_AfterBootTZ0() -> SaveTZ0InfoToLocal()

```
struct TZ0Info_A7
{
  uint64_t unkVal0;
  uint64_t deviceMemSize;
  uint64_t tz0Offset_Start;
  uint64_t tz0Offset_End;
  uint64_t tz1Offset_Start;
  uint64_t tz1Offset_End;
  uint64_t pa0Offset_Start_B;
  uint64_t tz0End_Minus_0x8000_C;
  uint64_t pa0Offset_Start_A;
  uint64_t pa0Offset_End_A;
  uint64_t pa0Offset_End_B;
  uint64_t tz0End_Minus_0x8000_A;
  uint64_t tz0End_Minus_0x8000_B;
  uint64_t tz0End_Minus_0x1000_A;
  uint64_t tz0End_Minus_0x1000_B;
  uint64_t tz0Offset_End2;
};
```

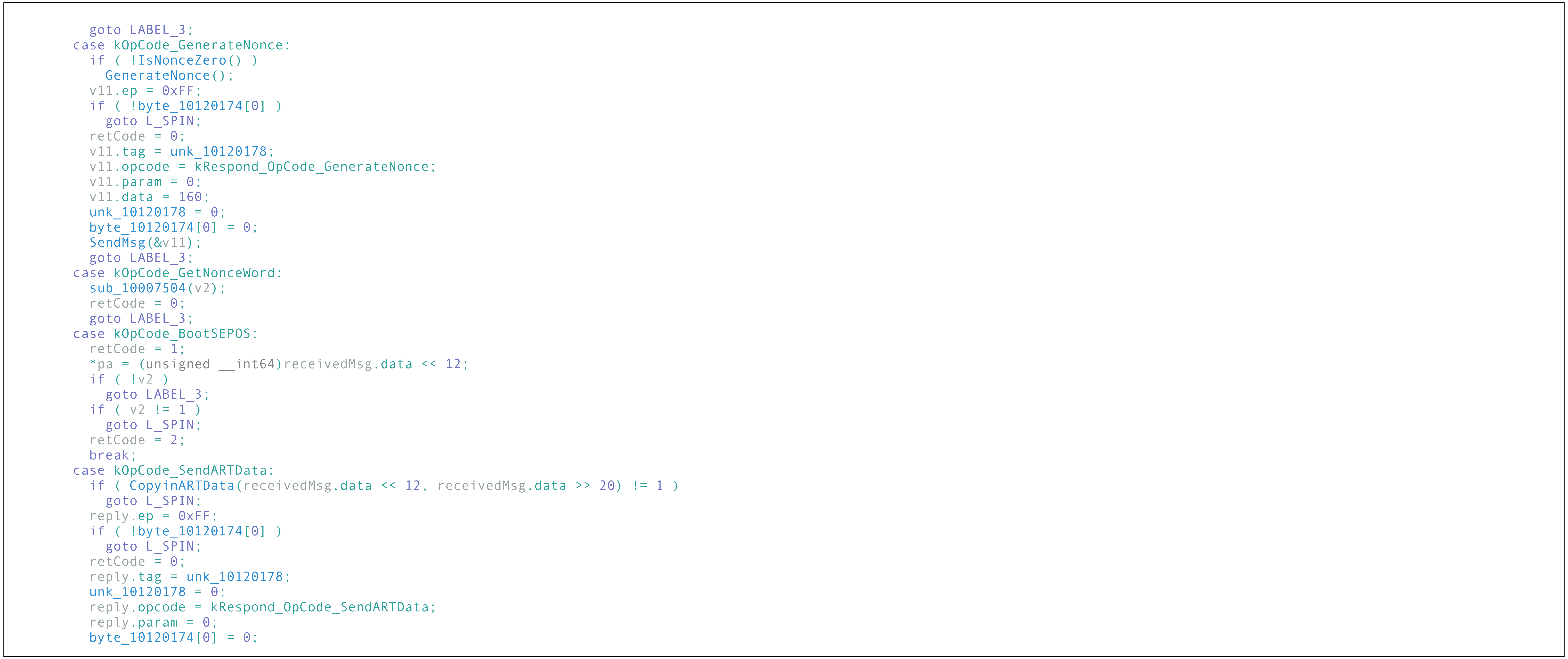
RE SEP Boot Process: BootOS

- MsgLoopAfterBootTZ0() -> MsgLoop_BootOS()

```
// 0x10007568
int __fastcall MsgLoop_BootOS(_QWORD *pa)
{
  retCode = 0;
  v16 = gStackCookie;
LABEL_3:
  while ( !retCode )
  {
    WaitMessage(&receivedMsg);
    if ( byte_10120174[0] == 1 )
L_SPIN:
      Spin3();
    byte_10120174[0] = 1;
    unk_10120178 = receivedMsg.tag;
    if ( receivedMsg.ep == 255 )
    {
      v2 = HIBYTE(*(_DWORD *)&receivedMsg.ep);
      retCode = 3;
      switch ( receivedMsg.opcode )
      {
        case kOpCode_Ping:
          v9.ep = 0xFF;
          v9.tag = receivedMsg.tag;
          retCode = 0;
          v9.opcode = kRespond_OpCode_Ping;
          v9.param = 0;
          unk_10120178 = 0;
          byte_10120174[0] = 0;
          v9.data = 0;
          SendMsg(&v9);
          goto LABEL_3;
        case kOpCode_Ping2:
          retCode = 0;
          v10.ep = 0xFF;
          v10.tag = receivedMsg.tag;
          v10.opcode = kRespond_OpCode_Ping2;
          v10.param = 0;
          v10.data = 2;
          unk_10120178 = 0;
          byte_10120174[0] = 0;
          SendMsg(&v10);
          goto LABEL_3;
```

RE SEP Boot Process: BootOS

- MsgLoopAfterBootTZ0() -> MsgLoop_BootOS()



```
      goto LABEL_3;
    case kOpCode_GenerateNonce:
      if ( !IsNonceZero() )
        GenerateNonce();
      v11.ep = 0xFF;
      if ( !byte_10120174[0] )
        goto L_SPIN;
      retCode = 0;
      v11.tag = unk_10120178;
      v11.opcode = kRespond_OpCode_GenerateNonce;
      v11.param = 0;
      v11.data = 160;
      unk_10120178 = 0;
      byte_10120174[0] = 0;
      SendMsg(&v11);
      goto LABEL_3;
    case kOpCode_GetNonceWord:
      sub_10007504(v2);
      retCode = 0;
      goto LABEL_3;
    case kOpCode_BootSEPOS:
      retCode = 1;
      *pa = (unsigned __int64)receivedMsg.data << 12;
      if ( !v2 )
        goto LABEL_3;
      if ( v2 != 1 )
        goto L_SPIN;
      retCode = 2;
      break;
    case kOpCode_SendARTData:
      if ( CopyinARTData(receivedMsg.data << 12, receivedMsg.data >> 20) != 1 )
        goto L_SPIN;
      reply.ep = 0xFF;
      if ( !byte_10120174[0] )
        goto L_SPIN;
      retCode = 0;
      reply.tag = unk_10120178;
      unk_10120178 = 0;
      reply.opcode = kRespond_OpCode_SendARTData;
      reply.param = 0;
      byte_10120174[0] = 0;
```

RE SEP Boot Process: BootOS

- MsgLoopAfterBootTZ0() -> MsgLoop_BootOS()

```
      byte_10120174[0] = 0;
      reply.data = 0;
      SendMsg(&reply);
      goto LABEL_3;
    case kOpCode_Resume:
      goto LABEL_3;
    case kOpCode_SetFlag:
      v3 = receivedMsg.data;
      v4 = receivedMsg.data;
      if ( receivedMsg.data )
        v4 = 1;
      SetMessageFlag(v4);
      v13.ep = 0xFF;
      if ( !byte_10120174[0] )
        goto L_SPIN;
      retCode = 0;
      v13.tag = unk_10120178;
      unk_10120178 = 0;
      v13.opcode = kRespond_OpCode_SetFlag;
      v13.param = 0;
      byte_10120174[0] = 0;
      v13.data = v3;
      SendMsg(&v13);
      goto LABEL_3;
    case kOpCode_PanicImmediatly:
      v14.ep = -1;
      v14.tag = receivedMsg.tag;
      v14.opcode = kRespond_OpCode_PanicImmediatly;
      v14.param = 0;
      unk_10120178 = 0;
      byte_10120174[0] = 0;
      v14.data = 0;
      SendMsg(&v14);
      goto L_SPIN;
    default:
      retCode = 0;
      v15.ep = 0xFF;
      v15.tag = receivedMsg.tag;
      v15.opcode = 0xC7;
      v15.param = 0;
      v15.data = receivedMsg.opcode;
```

RE SEP Boot Process: BootOS

- MsgLoopAfterBootTZ0() -> MsgLoop_BootOS()

```
          v15.data = receivedMsg.opcode;
          unk_10120178 = 0;
          byte_10120174[0] = 0;
          SendMsg(&v15);
          goto LABEL_3;
      }
    }
    else
    {
      msg.ep = receivedMsg.ep;
      retCode = 0;
      msg.tag = receivedMsg.tag;
      msg.opcode = 0;
      msg.param = 0;
      unk_10120178 = 0;
      byte_10120174[0] = 0;
      msg.data = 0;
      SendMsg(&msg);
    }
  }
  if ( gStackCookie != v16 )
    PRTS_StackCookieError();
  return retCode;
}
```

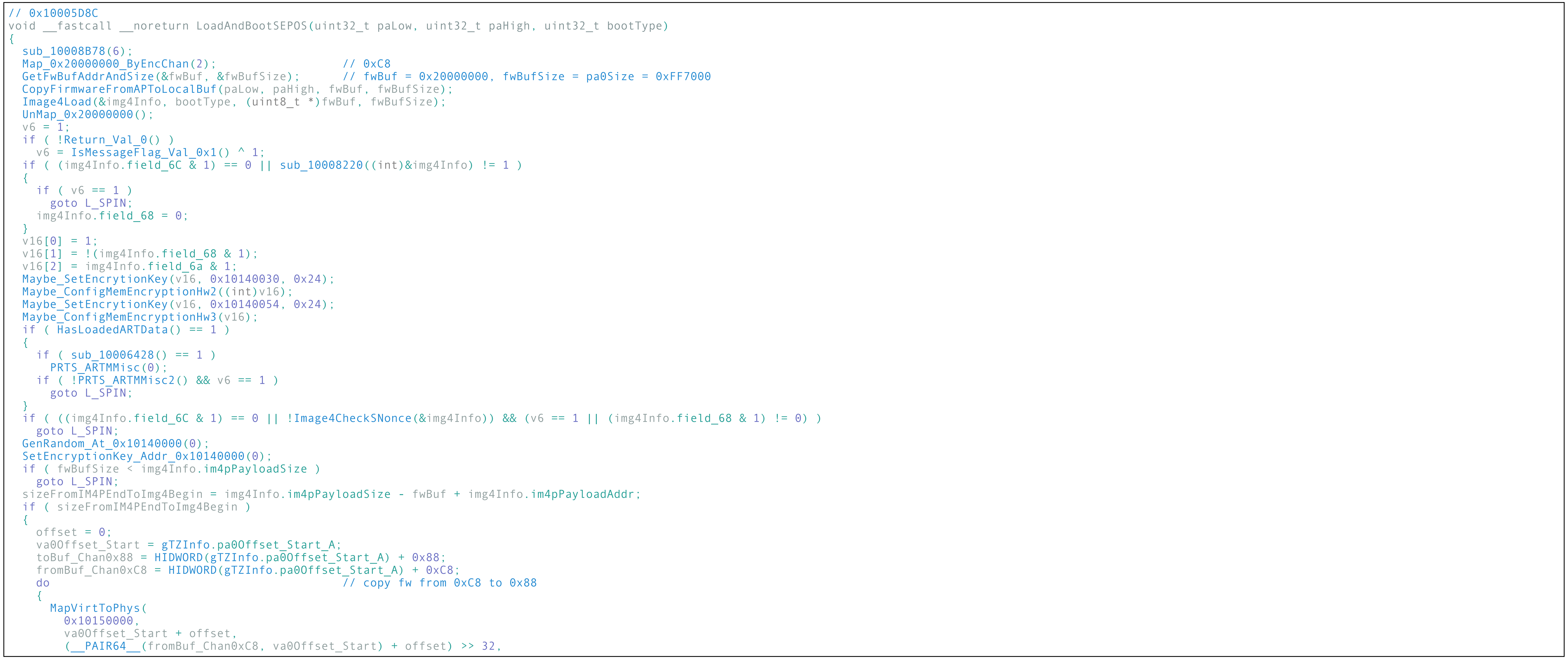
RE SEP Boot Process: BootOS

- AP sends BootOS command to SEP

```
void BootSEPOS(void)
{
    akf_start_sep();
    uint32_t data = (uint32_t)(gSEPFwFinalAddr >> 12);
    _sep_create_message2(&_sep_send_msg, kMsg_BootSEPOS, 0, 0, data);

    uint32_t ret = akf_send_mbox2(KFW_SEP, (uint64_t *)&_sep_send_msg, A7IOP_NO_WAIT);
    if (ret) {
        printf("[-] unable to send BootSEPOS to SEP mailbox, abandoning...\n");
        goto exit;
    }

    ret = _sep_client_reader(BOOT_SEPOS_TIMEOUT);
    if (ret) {
        printf("[-] SEP not respoding to BootSEPOS, abandoning...\n");
        goto exit;
    }

exit:
    akf_stop(KFW_SEP);
}
```

RE SEP Boot Process: BootOS

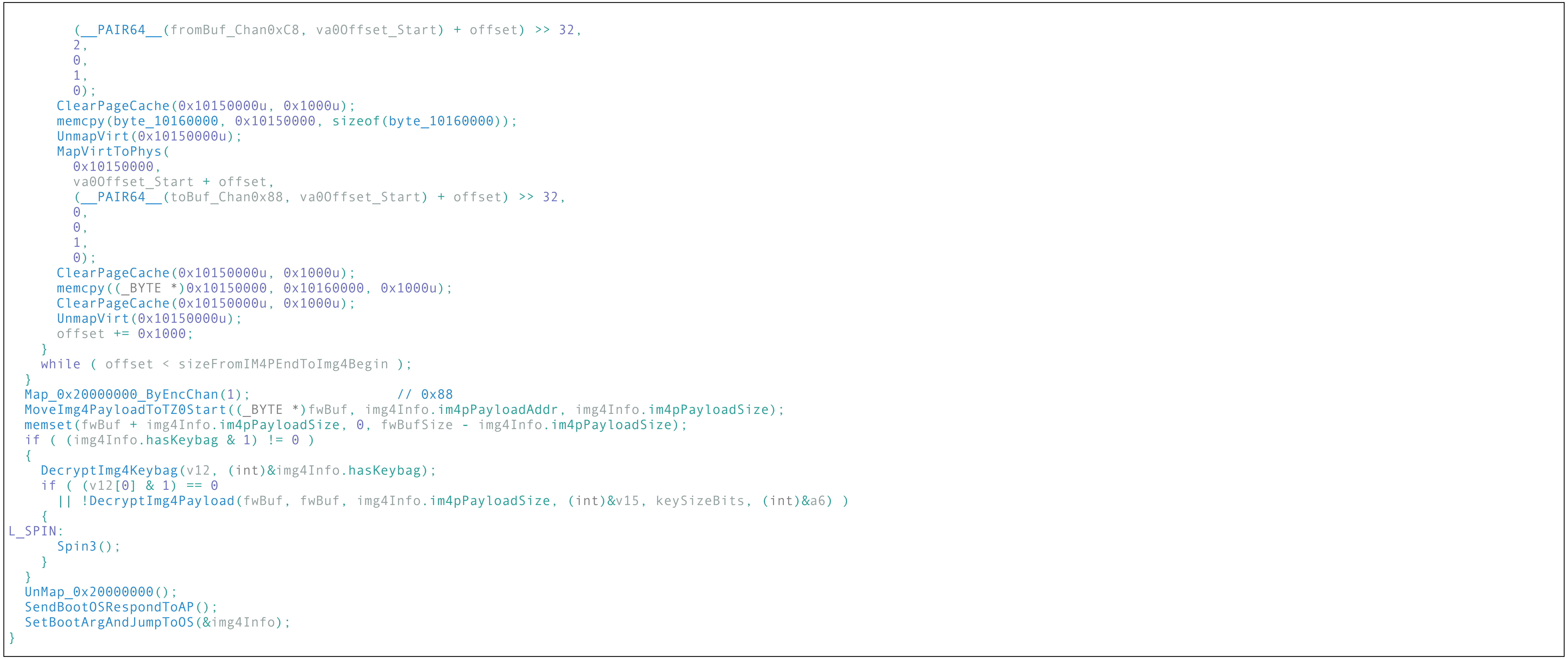
- MsgLoopAfterBootTZ0() -> LoadAndBootSEPOS()



```
// 0x10005D8C
void __fastcall __noreturn LoadAndBootSEPOS(uint32_t paLow, uint32_t paHigh, uint32_t bootType)
{
  sub_10008B78(6);
  Map_0x20000000_ByEncChan(2);                  // 0xC8
  GetFwBufAddrAndSize(&fwBuf, &fwBufSize);      // fwBuf = 0x20000000, fwBufSize = pa0Size = 0xFF7000
  CopyFirmwareFromAPToLocalBuf(paLow, paHigh, fwBuf, fwBufSize);
  Image4Load(&img4Info, bootType, (uint8_t *)fwBuf, fwBufSize);
  UnMap_0x20000000();
  v6 = 1;
  if ( !Return_Val_0() )
    v6 = IsMessageFlag_Val_0x1() ^ 1;
  if ( (img4Info.field_6C & 1) == 0 || sub_10008220((int)&img4Info) != 1 )
  {
    if ( v6 == 1 )
      goto L_SPIN;
    img4Info.field_68 = 0;
  }
  v16[0] = 1;
  v16[1] = !(img4Info.field_68 & 1);
  v16[2] = img4Info.field_6a & 1;
  Maybe_SetEncrytionKey(v16, 0x10140030, 0x24);
  Maybe_ConfigMemEncryptionHw2((int)v16);
  Maybe_SetEncrytionKey(v16, 0x10140054, 0x24);
  Maybe_ConfigMemEncryptionHw3(v16);
  if ( HasLoadedARTData() == 1 )
  {
    if ( sub_10006428() == 1 )
      PRTS_ARTMMisc(0);
    if ( !PRTS_ARTMMisc2() && v6 == 1 )
      goto L_SPIN;
  }
  if ( ((img4Info.field_6C & 1) == 0 || !Image4CheckSNonce(&img4Info)) && (v6 == 1 || (img4Info.field_68 & 1) != 0) )
    goto L_SPIN;
  GenRandom_At_0x10140000(0);
  SetEncryptionKey_Addr_0x10140000(0);
  if ( fwBufSize < img4Info.im4pPayloadSize )
    goto L_SPIN;
  sizeFromIM4PEndToImg4Begin = img4Info.im4pPayloadSize - fwBuf + img4Info.im4pPayloadAddr;
  if ( sizeFromIM4PEndToImg4Begin )
  {
    offset = 0;
    va0Offset_Start = gTZInfo.pa0Offset_Start_A;
    toBuf_Chan0x88 = HIDWORD(gTZInfo.pa0Offset_Start_A) + 0x88;
    fromBuf_Chan0xC8 = HIDWORD(gTZInfo.pa0Offset_Start_A) + 0xC8;
    do                                          // copy fw from 0xC8 to 0x88
    {
      MapVirtToPhys(
        0x10150000,
        va0Offset_Start + offset,
        (__PAIR64__(fromBuf_Chan0xC8, va0Offset_Start) + offset) >> 32,
```

RE SEP Boot Process: BootOS

- MsgLoopAfterBootTZ0() -> LoadAndBootSEPOS()



```
        (__PAIR64__(fromBuf_Chan0xC8, va0Offset_Start) + offset) >> 32,
        2,
        0,
        1,
        0);
      ClearPageCache(0x10150000u, 0x1000u);
      memcpy(byte_10160000, 0x10150000, sizeof(byte_10160000));
      UnmapVirt(0x10150000u);
      MapVirtToPhys(
        0x10150000,
        va0Offset_Start + offset,
        (__PAIR64__(toBuf_Chan0x88, va0Offset_Start) + offset) >> 32,
        0,
        0,
        1,
        0);
      ClearPageCache(0x10150000u, 0x1000u);
      memcpy((_BYTE *)0x10150000, 0x10160000, 0x1000u);
      ClearPageCache(0x10150000u, 0x1000u);
      UnmapVirt(0x10150000u);
      offset += 0x1000;
    }
    while ( offset < sizeFromIM4PEndToImg4Begin );
  }
  Map_0x20000000_ByEncChan(1);                  // 0x88
  MoveImg4PayloadToTZ0Start((_BYTE *)fwBuf, img4Info.im4pPayloadAddr, img4Info.im4pPayloadSize);
  memset(fwBuf + img4Info.im4pPayloadSize, 0, fwBufSize - img4Info.im4pPayloadSize);
  if ( (img4Info.hasKeybag & 1) != 0 )
  {
    DecryptImg4Keybag(v12, (int)&img4Info.hasKeybag);
    if ( (v12[0] & 1) == 0
      || !DecryptImg4Payload(fwBuf, fwBuf, img4Info.im4pPayloadSize, (int)&v15, keySizeBits, (int)&a6) )
    {
L_SPIN:
      Spin3();
    }
  }
  UnMap_0x20000000();
  SendBootOSRespondToAP();
  SetBootArgAndJumpToOS(&img4Info);
}
```

RE SEP Boot Process: BootOS

- MsgLoopAfterBootTZ0() -> LoadAndBootSEPOS() -> Map_0x20000000_ByEncChan()

```
// 0x100072FC
void __fastcall Map_0x20000000_ByEncChan(int channel)
{
  uint32_t va0Size; // r5
  int chanPrefix; // r0

  va0Size = LODWORD(gTZInfo.pa0Offset_End_A) - LODWORD(gTZInfo.pa0Offset_Start_A);
  if ( channel )
  {
    if ( channel == 2 )
    {
      chanPrefix = 0xC8;
    }
    else
    {
      if ( channel != 1 )
        Spin3();
      chanPrefix = 0x88;
    }
  }
  else
  {
    chanPrefix = 8;
  }
  Config_PA_To_PA_Map(
    0i64,
    gTZInfo.pa0Offset_End_A - gTZInfo.pa0Offset_Start_A,
    __PAIR64__(chanPrefix + HIDWORD(gTZInfo.pa0Offset_Start_A), gTZInfo.pa0Offset_Start_A));
  MapPA_Enable();
  __dsb(0xFu);
  __isb(0xFu);
  MapVA_0x20000000(va0Size);
}
```

RE SEP Boot Process: BootOS

- MsgLoopAfterBootTZ0() -> LoadAndBootSEPOS() -> GetFwBufAddrAndSize()

```
// 0x1000729C
_DWORD *__fastcall GetFwBufAddrAndSize(_DWORD *outAddr, unsigned int *outSize)
{
  uint64_t pa0Size; // kr00_8
  BOOL isSizeInvalid; // r3

  pa0Size = gTZInfo.pa0Offset_End_A - gTZInfo.pa0Offset_Start_A;
  isSizeInvalid = LODWORD(gTZInfo.pa0Offset_End_A) -
LODWORD(gTZInfo.pa0Offset_Start_A) <= (unsigned int)SZ_256MB;
  if ( (gTZInfo.pa0Offset_End_A - gTZInfo.pa0Offset_Start_A) >> 32 )
    isSizeInvalid = 0;
  if ( !isSizeInvalid )
    Spin3();
  *outAddr = 0x20000000;
  *outSize = pa0Size;
  return outAddr;
}
```

RE SEP Boot Process: BootOS

- MsgLoopAfterBootTZ0() -> LoadAndBootSEPOS() -> CopyFirmwareFromAPToLocalBuf()

```
// 0x10007420
void __fastcall CopyFirmwareFromAPToLocalBuf(unsigned int paLow, unsigned int paHigh, int fwBuf, unsigned int fwBufSize)
{
  unsigned int img4Size2; // r1
  unsigned int curOffset; // r6
  unsigned int img4Size; // [sp+0h] [bp-18h] BYREF

  if ( (fwBufSize & 0xFFF) != 0 )
    goto L_Spin;
  if ( fwBufSize > SZ_256MB )
    goto L_Spin;
  CopyOnePage_FromExternalPhysToLocalBuf(paLow, paHigh, (_BYTE *)fwBuf);
  if ( !GetImage4Size(fwBuf, 0x1000u, &img4Size) )
    goto L_Spin;
  img4Size2 = img4Size;
  if ( img4Size > fwBufSize )
L_Spin:
    Spin3();
  if ( img4Size > 0x1000 )
  {
    for ( curOffset = 0x1000; curOffset < img4Size; curOffset += 0x1000 )
    {
      CopyOnePage_FromExternalPhysToLocalBuf(
        curOffset + paLow,
        (__PAIR64__(paHigh, curOffset) + paLow) >> 32,
        (_BYTE *)(fwBuf + curOffset));
      img4Size2 = img4Size;
    }
  }
  memset(fwBuf + img4Size2, 0, fwBufSize - img4Size2);
}
```

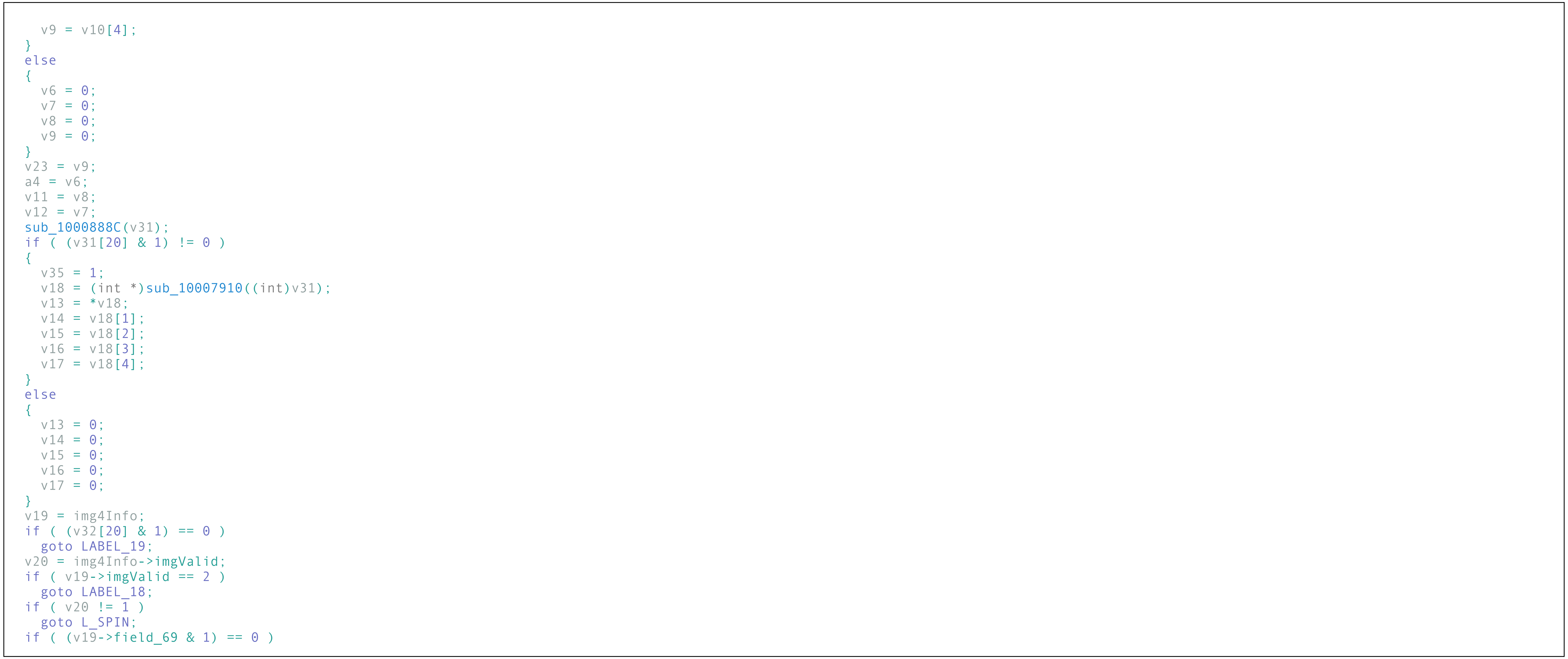
RE SEP Boot Process: BootOS

- MsgLoopAfterBootTZ0() -> LoadAndBootSEPOS() -> SetBootArgAndJumpToOS()

```
// 0x10008B9C
void __fastcall SetBootArgAndJumpToOS(Image4Info *img4Info)
{
  pa0Size2 = LODWORD(gTZInfo.pa0Offset_End_B) - LODWORD(gTZInfo.pa0Offset_Start_A);
  pa0Size = LODWORD(gTZInfo.pa0Offset_End_B) - LODWORD(gTZInfo.pa0Offset_Start_A);
  Map_FromAddrZero_To_TZ0End_Minus_0x8000_Chan_0x88(LODWORD(gTZInfo.pa0Offset_End_B) - LODWORD(gTZInfo.pa0Offset_Start_A));
  bootArg = (SEPBootArg *)(pa0Size2 + 0x800);
  im4pTag = img4Info->im4pTag;
  if ( !im4pTag )
    goto L_SPIN;
  tz0Offset_End = gTZInfo.tz0Offset_End;
  tz0Offset_Start = gTZInfo.tz0Offset_Start;
  isTZSizeValid = 0;
  tzSize3 = LODWORD(gTZInfo.tz0Offset_End) - LODWORD(gTZInfo.tz0Offset_Start);
  if ( LODWORD(gTZInfo.tz0Offset_End) - LODWORD(gTZInfo.tz0Offset_Start) != -1 )
    isTZSizeValid = 1;
  if ( (gTZInfo.tz0Offset_End - gTZInfo.tz0Offset_Start) >> 32 )
    isTZSizeValid = 0;
  if ( !isTZSizeValid )
L_SPIN:
    Spin3();
  v5 = 0;
  v36 = 0;
  v35 = 0;
  v34 = 0;
  v33 = 0;
  memcpy(v32, (int)&img4Info->field_20 + 3, 0x15u);
  tz0Addr_High32_0x88 = HIDWORD(gTZInfo.tz0Offset_Start) + 0x88;
  reservedTZSpace_StartAddr = LODWORD(gTZInfo.tz0End_Minus_0x8000_C) - LODWORD(gTZInfo.pa0Offset_Start_B);
  if ( (v32[20] & 1) != 0 )
  {
    v36 = 1;
    v10 = (int *)sub_10007910((int)v32);
    v5 = *v10;
    v6 = v10[1];
    v7 = v10[2];
    v8 = v10[3];
    v9 = v10[4];
```

RE SEP Boot Process: BootOS

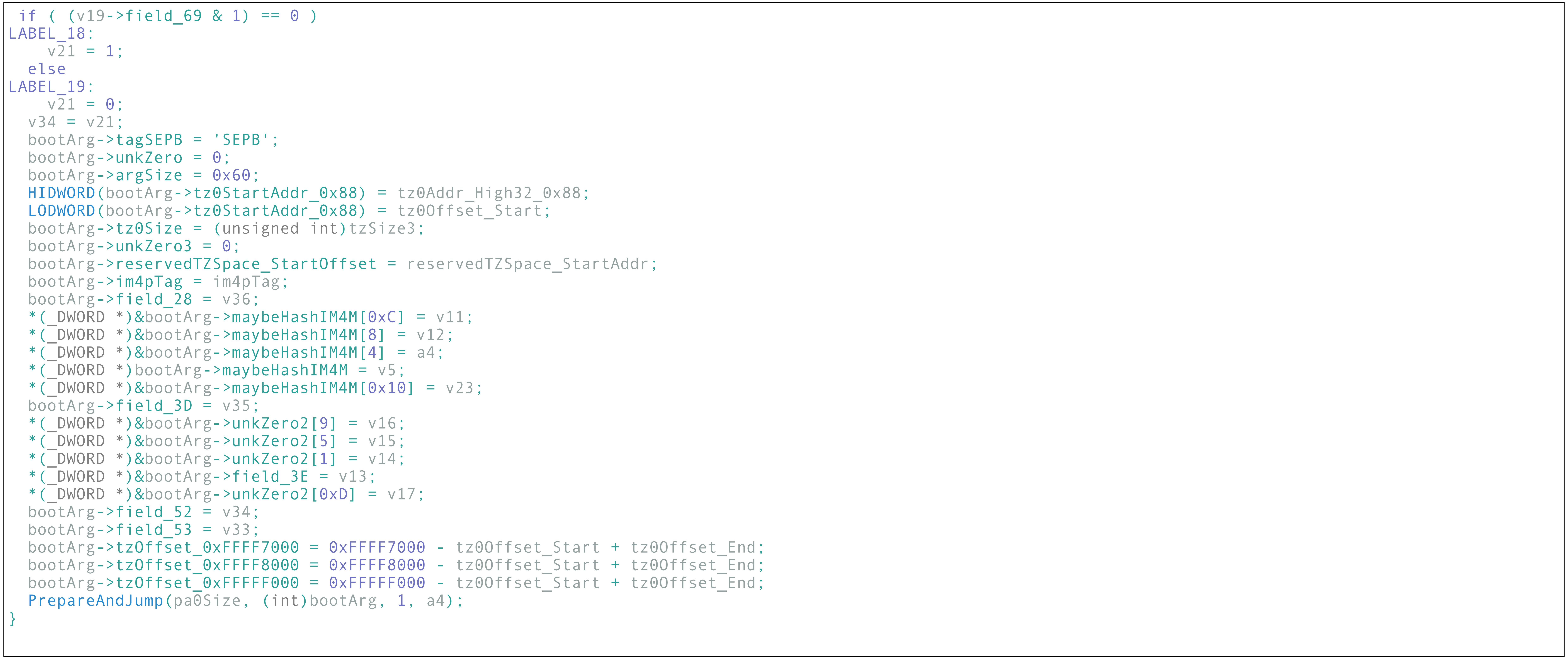
- MsgLoopAfterBootTZ0() -> LoadAndBootSEPOS() -> SetBootArgAndJumpToOS()



```
  v9 = v10[4];
}
else
{
  v6 = 0;
  v7 = 0;
  v8 = 0;
  v9 = 0;
}
v23 = v9;
a4 = v6;
v11 = v8;
v12 = v7;
sub_1000888C(v31);
if ( (v31[20] & 1) != 0 )
{
  v35 = 1;
  v18 = (int *)sub_10007910((int)v31);
  v13 = *v18;
  v14 = v18[1];
  v15 = v18[2];
  v16 = v18[3];
  v17 = v18[4];
}
else
{
  v13 = 0;
  v14 = 0;
  v15 = 0;
  v16 = 0;
  v17 = 0;
}
v19 = img4Info;
if ( (v32[20] & 1) == 0 )
  goto LABEL_19;
v20 = img4Info->imgValid;
if ( v19->imgValid == 2 )
  goto LABEL_18;
if ( v20 != 1 )
  goto L_SPIN;
if ( (v19->field_69 & 1) == 0 )
```

RE SEP Boot Process: BootOS

- MsgLoopAfterBootTZ0() -> LoadAndBootSEPOS() -> SetBootArgAndJumpToOS()



```
 if ( (v19->field_69 & 1) == 0 )
LABEL_18:
    v21 = 1;
  else
LABEL_19:
    v21 = 0;
  v34 = v21;
  bootArg->tagSEPB = 'SEPB';
  bootArg->unkZero = 0;
  bootArg->argSize = 0x60;
  HIDWORD(bootArg->tz0StartAddr_0x88) = tz0Addr_High32_0x88;
  LODWORD(bootArg->tz0StartAddr_0x88) = tz0Offset_Start;
  bootArg->tz0Size = (unsigned int)tzSize3;
  bootArg->unkZero3 = 0;
  bootArg->reservedTZSpace_StartOffset = reservedTZSpace_StartAddr;
  bootArg->im4pTag = im4pTag;
  bootArg->field_28 = v36;
  *(_DWORD *)&bootArg->maybeHashIM4M[0xC] = v11;
  *(_DWORD *)&bootArg->maybeHashIM4M[8] = v12;
  *(_DWORD *)&bootArg->maybeHashIM4M[4] = a4;
  *(_DWORD *)bootArg->maybeHashIM4M = v5;
  *(_DWORD *)&bootArg->maybeHashIM4M[0x10] = v23;
  bootArg->field_3D = v35;
  *(_DWORD *)&bootArg->unkZero2[9] = v16;
  *(_DWORD *)&bootArg->unkZero2[5] = v15;
  *(_DWORD *)&bootArg->unkZero2[1] = v14;
  *(_DWORD *)&bootArg->field_3E = v13;
  *(_DWORD *)&bootArg->unkZero2[0xD] = v17;
  bootArg->field_52 = v34;
  bootArg->field_53 = v33;
  bootArg->tzOffset_0xFFFF7000 = 0xFFFF7000 - tz0Offset_Start + tz0Offset_End;
  bootArg->tzOffset_0xFFFF8000 = 0xFFFF8000 - tz0Offset_Start + tz0Offset_End;
  bootArg->tzOffset_0xFFFFF000 = 0xFFFFF000 - tz0Offset_Start + tz0Offset_End;
  PrepareAndJump(pa0Size, (int)bootArg, 1, a4);
}
```

RE SEP Boot Process: BootOS

- MsgLoopAfterBootTZ0() -> LoadAndBootSEPOS() -> SetBootArgAndJumpToOS() -> PrepareAndJump()

```
// 0x10004198
void __fastcall PrepareAndJump(int pa0Size, int bootArg, int clearMem, int a4)
{
  ReadCounterTimer(7u, bootArg, clearMem, a4);
  v7 = Get_TZ0Size();
  if ( clearMem )
  {
    DCCMVAC_Range(0x10140000, 0x78);
    bzero2(qword_10120000, 0x1000);
    bzero2(byte_10130000, 0x2000);
    bzero2(byte_10160000, 0x1000);
    SetStackValue_From_0x3D600F80_To_0x3D601000();
  }
  toAddr = (_QWORD *)pa0Size;
  fromAddr = (_QWORD *)start;
  do
  {
    v10 = *fromAddr; v11 = fromAddr[1]; v12 = fromAddr[2]; v13 = fromAddr[3];
    fromAddr += 4;
    *toAddr = v10; toAddr[1] = v11; toAddr[2] = v12; toAddr[3] = v13;
    toAddr += 4;
  }
  while ( (unsigned int)fromAddr < 0x10000280 );
  dstAddr = (int *)(pa0Size + 0x20);
  pa0Size_Plus_0x40 = pa0Size + 0x40;
  loopCount = 8;
  do
  {
    *dstAddr++ = pa0Size_Plus_0x40;
    pa0Size_Plus_0x40 += 4;
    --loopCount;
  }
  while ( loopCount );
  DCCMVAC_Range(pa0Size, 0x280);
  ICIMVAU_Range(pa0Size, 0x280);
  ((void (__fastcall *)(int, int))(pa0Size + 0xC0))(bootArg, v7);// TrampStart
}
```

RE SEP Boot Process: BootOS Trampoline

- PrepareAndJump() -> TrampStart()

```
void __noreturn TrampStart()
{
  ConfigAndJumpToAddrZero();
}

void __noreturn ConfigAndJumpToAddrZero()
{
  unsigned int i; // r0

  _WriteSystemReg(VBAR, (unsigned int)VBAR_Tramp);
  __isb(0xFu);
  _WriteSystemReg(MAIR0, _ReadSystemReg(MAIR0) & 0xFFFFFF00 | 4);
  __dsb(0xFu);
  __isb(0xFu);
  for ( i = 0x10000000; i < 0x10000100; i += 0x1000 )
    _WriteSystemReg(TLBIMVA, i);
  __dsb(0xFu);
  __isb(0xFu);
  MEMORY[0x3D200038] = 1;
  _WriteSystemReg(SCTLR, 0xC51078u);
  __dsb(0xFu);
  __isb(0xFu);
  InvalidateAllCachedCopiesOfTranslationTableEntries();
  DCCISW_AllAddrSpace();
  ICIALLU_AddrZero();
  MovR10ToR0_JumpToAddrZero();
}
```

RE SEP Boot Process: BootOS Trampoline

- PrepareAndJump() -> TrampStart() -> MovR10ToR0_JumpToAddrZero()

```
void __noreturn MovR10ToR0_JumpToAddrZero()
{
  JUMPOUT(0);
}
```

- The sep firmware is at address 0.