S32G On-demand SMR Verification Usage

by John Li (nxa08200)

This application doc explains the application method of S32G HSE On_demand SMR verification. The example application demonstrated in this doc is:

- Secure Bootloader verification of Linux Bootloader fip.bin

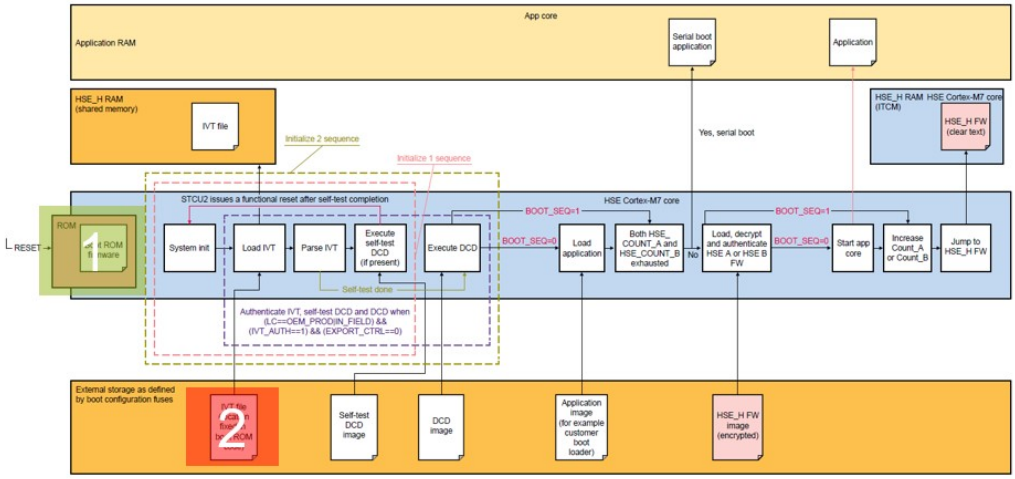
History	Comments	Author
V1	● Create the doc	● John.Li
V2	● Translate to Eng	● John.Li

Contents

1 Background Description and Reference Materials

1.1 Background Description

The flowchart of HSE Secure Boot is as follows:



The whole Secure Boot process is as follows:

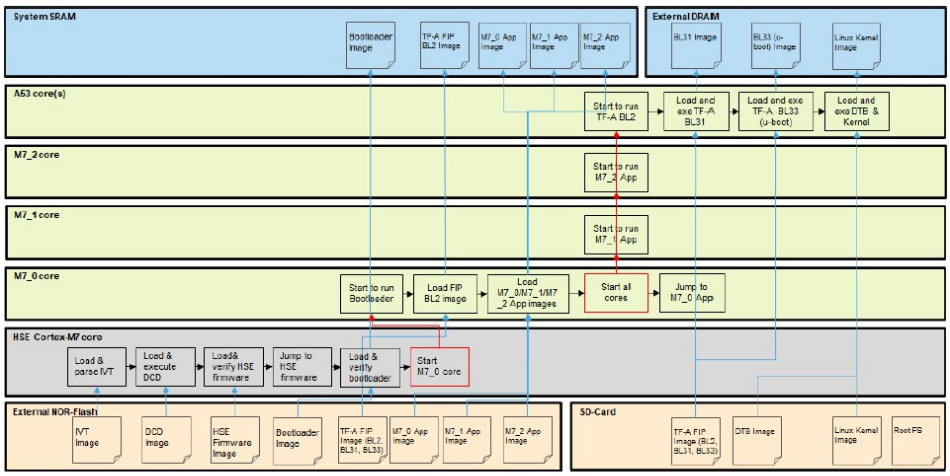


Figure 3. **Secure boot flow**

So the current anti tamper verification chain are:

- HSE ROM code verify IVT。
- HSE FW verify bootloader。

Still lack of:

- Bootloader calls HSE to verify fip.bin (ATF+Uboot): Bootloader is running on M7. In order to improve efficiency, it is recommended to use HSE verification.
- Uboot verifies the kernel, (uboot call HSE, Please refer Linux User manual).
- Kernel verification of file system (using relevant kernel public technology like DM-verity…).

The focus of this doc is to explain and solve the problem that Bootloader calls HSE to verify fip.bin (ATF+Uboot) and start ATF+Uboot.

Note:

- **OnDemand SMR verification can be called by Bootloader or Autosar secure app.**
- **OnDemand SMR verification is not the only way to do fip.bin verification. Because the Bootloader project can call the crypto MCAL driver, it is also possible to call the driver to program and verify.**
- **In general, a normal boot needs to start a Bootloader to configure resources, so the practicability of postSMR that has a parallel relationship with preSMR is not as flexible as OnDemand SMR.**
- **In order to avoid HSE/M7 competing for QSPI NOR access, instead of using the method of HSE directly loading SMR from QSPI NOR, it is verified by HSE from SRAM after M7 loaded fip.bin into SRAM, and the Bootloader is solely responsible for QSPI NOR access after bootloader is started.**
- **All anti tamper trust chains only consider anti tampering, not encryption, anti denial, to avoid extending the startup speed too much, but this doc does not test the impact of OnDemand SMR on the startup speed.**
- **This doc uses fip.bin as an example to illustrate that the OnDemand SMR verification, M-core images is the same way.**

1.2 Reference Materials

Take S32G2 RDB2 as sample:

Num.	Materials	Comments	How to get it
1	Platform_Software_Integration _S32G2_2022_06.exe	Bootloader project, can refer its doc <<Bootloader_UserManual.pdf>> Secure Boot chapter	www.nxp.com/s32g
2	AN13750: Enabling Multicore Application on S32G2 using S32G2 Platform Software Integration – Application note	Bootloader project user guide, include the Secure Boot configuration (XueweiWang)	www.nxp.com/s32g
3	S32G_Bootloader_V*.pdf	Bootloader application doc, include Non-Secure Boot part(Johnli)	https://community.nxp.com

			/t5/NXP-Designs-Knowledge -Base/ S32G-Bootloader-Customzition/ta-p/ 1519838
4	S32G_Secure_boot_*.pdf	Secure boot application doc and project, this project base on it and re-development (Johnli)	https://community.nxp.com/ t5/NXP-Designs-Knowledge-Base/ S32G-Secure-boot-chinese-version/ta-p/ 1718083
5	HSE_H/M Firmware Reference Manual	HSE FW Reference Manual	www.nxp.com/s32g (Need secure file access right)
6	Secure Boot with HSE V0 - Draft B	Secure Boot user doc	www.nxp.com/s32g (Need secure file access right)
7	HSE_DEMOAPP_S32G2_0_1_0_5.exe	HSE Demo App(S32DS version)	www.nxp.com/s32g
8	HSE_FW_S32G2_0_1_0_5.exe	HSE FW install package, can refer its doc<<HSE_FW_H_S32G2XX_0.1.0.5 _HSE_Service_API_ Reference_Manual.pdf>>	www.nxp.com/s32g
9	S32G2_LinuxBSP_36.0_User_Manual.pdf	Linux BSP user manual	www.nxp.com/s32g
10	S32G_ATF_BSP32_V*.pdf	ATF customization doc(Johnli, Chinese version)	https://community.nxp.com/ t5/NXP-Designs-Knowledge-Base/ S32G-ATF-customization-application-doc /ta-p/1450561

2 S32G On-demand SMR Verification

2.1 SMR Verify

Refer to the document <<HSE_H/M Firmware Reference Manual>> for details of SMR verification.

2.2 On-demand SMR Verify

On demand SMR verify is as follows:

When the SMR entry is not configured to be linked to the CR table, the verification will not run automatically, and the calling service hseSmrVerifySrv_t needs to be called to active verification (or trigger automatically when the checkPeriod is not equal to 0).

The service has two parameters:

- entryIndex: specifies the index of the SMR to be verified.
- Options: SMR verification options:

◆ HSE_ SMR_ VERIFICATION_ OPTION_ NONE: the default verification method: if the SMR has been loaded from external flash memory, it is only verified in SRAM; If SMR has not been loaded from external flash memory, it is loaded into SRAM memory and verified.

◆ HSE_ SMR_ VERIFICATION_ OPTION_ NO_ LOAD: SMR verifies from external flash memory (using pSmrSrc address), even if pSmrDest is specified and loaded into SRAM memory.

◆ HSE_ SMR_ VERIFICATION_ OPTION_ RELOAD - SMR is loaded from external flash and verified, even if it has already been loaded.

This service can also be used to validate any SMR at runtime, regardless of the relationship between SMR and CR table. However, when the SMR to be verified is in the external Flash, it must be ensured that there is mainly no concurrent access to the external Flash. Therefore, it is recommended that the Bootloader copy the image to the SRAM and trigger the verification service. In addition, CR does not need to be configured when OnDemand SMR is installed.

3 Build the Development Environment

Firstly, set up a secure boot environment according to the requirements of the document <<S32G_Secure_boot_ *. Pdf>>, and turn off the following debugging switchers:

C:\NXP\Integration_Reference_Examples_S32G2_2022_06\code\framework\realtime\swc\bootloader\generic\src\Bootloader.c

```
// while (1); //johnli stop bootloader //stop continue booting, for debug
```

Keep the first debug switcher:

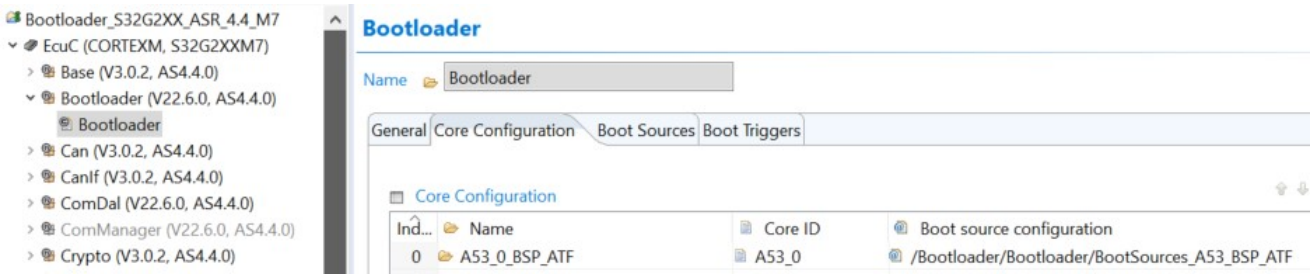
```
volatile int debug;

int main(void)

{…
debug =1;
while(debug); //Add debugging switcher to facilitate the use of lauterbach to track debugging.
```

Then modify the EB configuration as follows:

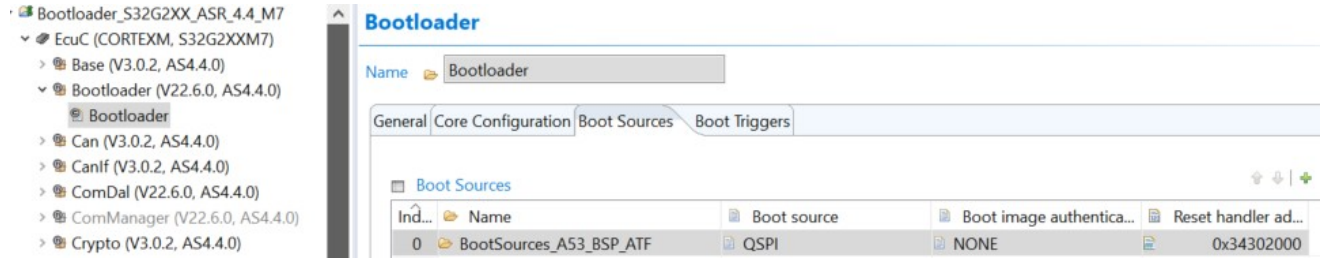
3.1 EB Configuration

Since only verification of A-core Linux fip.bin is considered, only A-core Boot Source is configured in the Bootloader project:

1. Open Bootloader(…)->Bootloader->Core Configuration: delete others except: A53_0_BSP_ATF:



2. Open Bootloader(…)->Bootloader->Boot Sources: delete others except : BootSources_A53_BSP_ATF, as the follow reset address:



Enter->BootSources_A53_BSP_ATF->Boot image fragment->ImageFragments_0: Need to set the loading address and size of the entire fip.bin. We can view the compiled log of ATF as follows:

Added BL2 and DTB to /opt/samba/nxa08200/S32G/BSP36/standalone/arm-trusted-firmware/build/s32g274ardb2/release/fip.bin successfully

Trusted Boot Firmware BL2: offset=0x100, size=0x4BDC0, cmdline="--tb-fw"

EL3 Runtime Firmware BL31: offset=0x4BEC0, size=0x18239, cmdline="--soc-fw"

Non-Trusted Firmware BL33: offset=0x64100, size=0xAF530, cmdline="--nt-fw"

SOC_FW_CONFIG: offset=0x113640, size=0x6D84, cmdline="--soc-fw-config"

CREATE /opt/samba/nxa08200/S32G/BSP36/standalone/arm-trusted-firmware/build/s32g274ardb2/release/bl2_w_dtb_size

CREATE /opt/samba/nxa08200/S32G/BSP36/standalone/arm-trusted-firmware/build/s32g274ardb2/release/fip.cfgout

MKIMAGE /opt/samba/nxa08200/S32G/BSP36/standalone/arm-trusted-firmware/build/s32g274ardb2/release/fip.s32

Image Layout

DCD: Offset: 0x200 Size: 0x1c

IVT: Offset: 0x1000 Size: 0x100

AppBootCode Header: Offset: 0x1200 Size: 0x40

Application: Offset: 0x1240 Size: 0x4c000

Boot Core: A53_0

IVT Location: SD/eMMC

Load address: 0x342fb140

Entry point: 0x34302000

ll build/s32g274ardb2/release/fip.*

-rw-r--r-- 1 nxa08200 nxp 1156096 Sep 14 15:46 build/s32g274ardb2/release/fip.bin

-rw-r--r-- 1 nxa08200 nxp 1160768 Sep 14 15:46 build/s32g274ardb2/release/fip.s32

So in BootSources_A53_BSP_ATF->General->Reset handler address, keep 0x34302000 unchanged.

In BootSources_A53_BSP_ATF->Boot image fragments->ImageFragments_0:

Load image at address (RAM)= 0x342fb140 //load address。

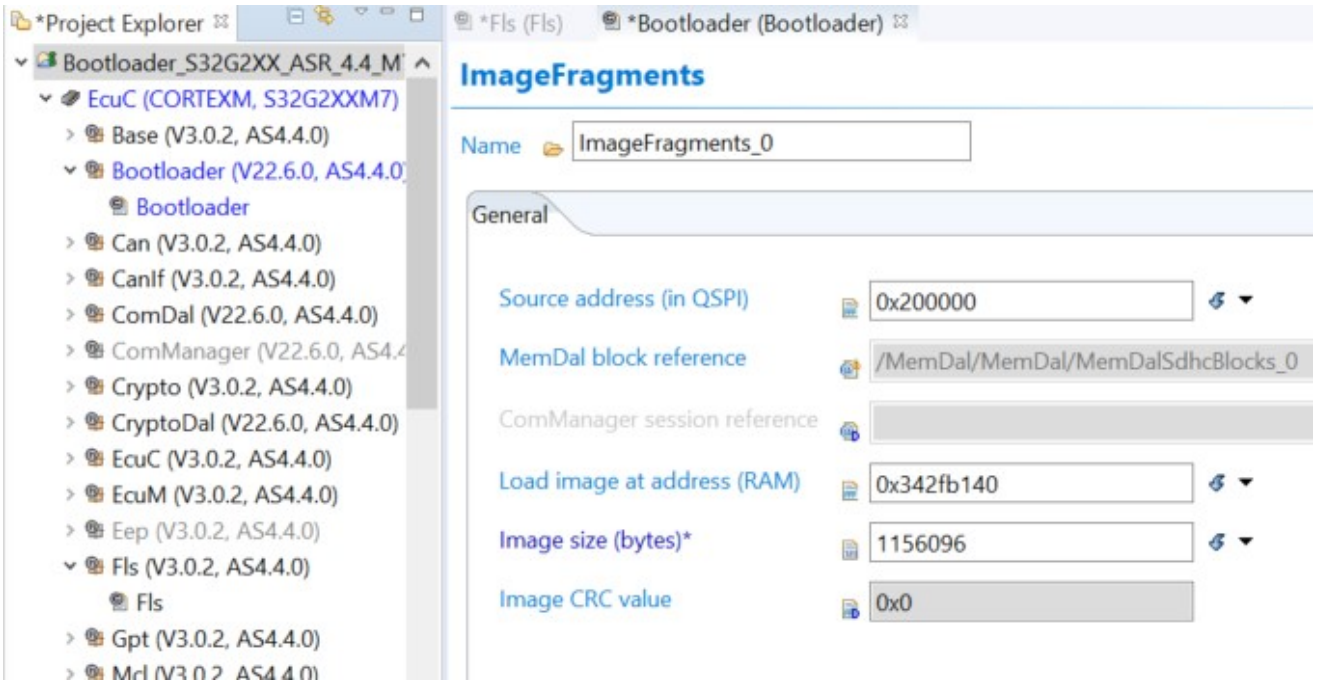
Attention the Notes: **“The address to load the image into RAM.**

NOTE !: The start address must be multiple of 8 if you choose CRC32 authentication method, otherwise must be multiple of 64.”

Therefore, the load image address requires 64 Byte alignment when compiling ATF.

Image size (bytes)= 1156096 >(0x113640+0x6D84)= 1156036, And 1156096 can be divided by 64.

In addition, Source address (in QSPI_NOR) is modified to 0x200000 to avoid overlapping with SYS-IMG published in secure boot:



3.2 ATF Compiling

Since the whole fip.bin (including ATF+Uboot) is verified as a whole, it is considered to put the whole fip.bin in the qspi nor instead of the SDcard/eMMC version compiled by the Bootloader project. According to the requirements of the document <<S32G2_LinuxBSP_36.0_User_Manual.pdf>>, it is compiled into a non self loading mode. The document description is as follows:

- Configure TF-A to read the FIP image from a defined memory address instead of reading it from the boot source storage (QSPI or MMC).

  The memory location of the FIP image is set using the **FIP_MEMORY_OFFSET** compilation parameter and can be used in two cases:

  1. Loading the FIP image from a predefined memory location instead of a storage device
  2. BL2 in-place execution from FIP image

  In the first case, TF-A uses the parameter to locate the image header and load the following stages from the given location, and in the second, the parameter contributes to in-place execution of the BL2 stage. In both cases, it is assumed the FIP image is copied to the specified memory address from outside TF-A. For example, the FIP image can be copied to specified SRAM address from the M7 bootloader configured as boot target, before starting BL2. This is how **FIP_MEMORY_OFFSET=<memory address>** is used for the first use case:

```
make CROSS_COMPILE=/path/to/your/toolchain/dir/bin/aarch64-none-linux-gnu- \
    ARCH=aarch64 PLAT=<plat> BL33=<path-to-u-boot-nodtb.bin> \
    FIP_MEMORY_OFFSET=0x34520000
```

  In this case, BL2 is not executed in-place as part of the FIP image and must be stored at a location that does not overlap with the FIP image, which is expected to be loaded at **FIP_MEMORY_OFFSET**.

  This feature can be paired with `BL2_BASE` to prepare the resulting image and BL2 stage for in-place execution. The in-place execution of the BL2 requires a three-step compilation:

  1. Compile the TF-A using a custom base address for the BL2 stage if the default value is not suitable. For this purpose, the `BL2_BASE` must be appended to the compilation command.

```
make CROSS_COMPILE=/path/to/your/toolchain/dir/bin/aarch64-none-linux-gnu- \
    ARCH=aarch64 PLAT=<plat> BL33=<path-to-u-boot-nodtb.bin> \
    BL2_BASE=0x34100000
...
Boot Core:      A53_0
IVT Location:   SD/eMMC
Load address:   0x340f8bc0
Entry point:    0x34100000
...
```

  2. Obtain the load address of the FIP image. In the above example, it is 0x340f8bc0 (see `Load address`)
  3. Recompile the TF-A using `FIP_MEMORY_OFFSET` set to identified load address.

```
make CROSS_COMPILE=/path/to/your/toolchain/dir/bin/aarch64-none-linux-gnu- \
    ARCH=aarch64 PLAT=<plat> BL33=<path-to-u-boot-nodtb.bin> \
    BL2_BASE=0x34100000 \
    FIP_MEMORY_OFFSET=0x340f8bc0
```

The actual compilation command is as follows. Refer to the previous section for compiling log.

```
make PLAT=s32g274ardb2 BL33=../u-boot/u-boot-nodtb.bin BL2_BASE=0x34302000
FIP_MEMORY_OFFSET=0x342fb140 LDFLAGS=""
```

3.3 Burn Image

According to the requirements of the document <<S32G_Secure_boot_ *. Pdf>>, create an IVT image, and then burn the image according to this document. Pay attention to the following points:

- Click Erase memory range... and select 0x0-0x500000. (It is emphasized that you need to erase first, so that all the redundant parts at the end of the fib.bin round to 64 BYTE are 0xFF, which can prevent errors).

Then burn the image as follows:

- Use flash tools to burn the bootloader image to QSPI:

Click Upload file to device... and burn "secureboot_odsmr.bin" to the address 0x0.

- Use flash tools to write A53 fip.bin into QSPI:

Click Upload file to device..., and burn "fip. bin" to the address 0x200000, the burning address reference, is the QSPI source address configured in the Bootloader MCAL, please note that the fip.bin file is burned, not the fip.s32 which included IVT head.

- The A53 Linux part uses a Linux PC to burn the Linux image to the SDcard.

4 Bootloader Codes Development

4.1 OnDemand SMR install

1. Get the fip.bin related information:

Function call:

C:\NXP\Integration_Reference_Examples_S32G2_2022_06\code\framework\realtime\swc\bootloader\platforms\S32G2XX\src\BootloaderSpecific.c：

```
StatusType Bl_ConfigureSecureBoot(void)
{…
    //johnli add for on-demand smr
   uint32_t u32FipSize;
   uint32_t u32FipRamAddr;
   uint32_t u32FipFlashAddr;
   uint8_t Bl_FipTag[32];
    uint32_t u32FipTagSize = 32;
    //end
  if (E_NOT_OK == Bl_IsSecureBootActive())
   {…
```

```
  Bl_GetHSEFwParams(&u32HseFwImageFlashAddr, &u32HseBackupFwImageFlashAddr);
```

实现：

```
static void Bl_GetFipParams(uint32_t *pFlashAddr, uint32_t *pRamAddr,
                            uint32_t *pSize)
{ // BootApplications is the boot source configured by EB. Since we only configured one A53 fip.bin bootapp and one image fragments, the serial number is 0
  *pFlashAddr =bootApplications[0].pImageFragments[0].u32FlashAddress;
  *pRamAddr = bootApplications[0].pImageFragments[0].u32DestinationAddress;
  *pSize = bootApplications[0].pImageFragments[0].u32Size;
}
```

2. Install OnDemand SMR(since OnDemand SMR does not require the boot verification, So do not need configure the CR):

Function call:

C:\NXP\Integration_Reference_Examples_S32G2_2022_06\code\framework\realtime\swc\bootloader\platforms\S32G2XX\src\BootloaderSpecific.c：

```
StatusType Bl_ConfigureSecureBoot(void)
{…
   if (E_OK == status)
     { // CMAC is also generated for the image. Note that the same key as the bootloader is used here, and different keys can also be used. You need to configure them by yourself
       status = CryptoDal_GenerateCmac(
       (uint8_t *) u32FipFlashAddr, &Bl_FipTag[0],
       BL_SEC_BOOT_KEY_INDEX, u32FipSize, &u32FipTagSize);
        //configure OnDemand SMR
       Bl_ConfigureFipSMR(&SMR_CR_Config, u32FipFlashAddr,
                          u32FipRamAddr, u32FipSize,
                          &Bl_FipTag[0], &u32FipTagSize);
        //Install OnDemand SMR, do not need install CR
       status = CryptoDal_SMR_Install(&SMR_CR_Config);
     }
```

Final implement function:

```
static void Bl_ConfigureFipSMR(
  CryptoDal_SMR_CR_EntryType *pSMR_CR_Config, uint32_t u32AppFlashAddress,
  int32_t u32AppRamAddr, uint32_t u32AppSize, uint8_t *pTag,
```

```
    uint32_t *pTagSize)
{
    pSMR_CR_Config->u32FlashStorageAddr = u32AppFlashAddress;
    pSMR_CR_Config->u32RamDestAddr = u32AppRamAddr;
    pSMR_CR_Config->u32AppSize = u32AppSize;
    pSMR_CR_Config->pTag = pTag;
    pSMR_CR_Config->pTagSize = pTagSize;
    pSMR_CR_Config->u8KeyIndex = BL_SEC_BOOT_KEY_INDEX;
    pSMR_CR_Config->u8EntryIndex = 2; // Note that the index of SMR needs to be modified
    pSMR_CR_Config->u8CoreID = HSE_APP_CORE3; // |  3   | A53_0  But in fact, this parameter is related to the configuration of CR, so do not use it
}
```

Implement the CryptoDal_SMR_Install function can refer the CryptoDal_SMR_CR_Install functionto delete the CR install part.

In addition, in order to prevent HSE and M7 access QSPI NOR at the same time, we use the Bootloader to copy fip.bin to SRAM first, and then trigger the verification service, instead of the HSE active the copying. Therefore, modify the SMR installation items accordingly (see the red words):

C:\NXP\Integration_Reference_Examples_S32G2_2022_06\code\framework\realtime\bsw\dal\cryptodal\generic\src\CryptoDal.c

```
FUNC(Std_ReturnType, CRYPTODAL_APP_CODE) CryptoDal_SMR_Install(P2VAR(CryptoDal_SMR_CR_EntryType, AUTOMATIC, CRYPTODAL_CONST) pSMR_CR_Config)
{
    VAR(Std_ReturnType, AUTOMATIC) eRetVal = E_NOT_OK;
    if (NULL_PTR != CryptoDal_pGlobalConfigPtr)
    {
#if CRYPTODAL_USE_CRYPTO
        eRetVal = CryptoDal_Crypto_SMR_Install(pSMR_CR_Config);
#endif /* CRYPTODAL_USE_CRYPTO */
    }
    return eRetVal;
}
```

C:\NXP\Integration_Reference_Examples_S32G2_2022_06\code\framework\realtime\bsw\dal\cryptodal\generic\include\CryptoDal.h

```
FUNC(Std_ReturnType, CRYPTODAL_APP_CODE) CryptoDal_SMR_Install(P2VAR(CryptoDal_SMR_CR_EntryType, AUTOMATIC, CRYPTODAL_CONST) pSMR_CR_Config);
```

C:\NXP\Integration_Reference_Examples_S32G2_2022_06\code\framework\realtime\bsw\dal\cryptodal\platforms\S32G2XX\src\CryptoDal_Crypto.c

```
FUNC(Std_ReturnType, CRYPTODAL_APP_CODE) CryptoDal_Crypto_SMR_Install(P2VAR(CryptoDal_SMR_CR_EntryType, AUTOMATIC, CRYPTODAL_CONST) pSMR_CR_Config)
{
  VAR(hseSmrEntry_t, AUTOMATIC) SmrEntry;
  VAR(hseCrEntry_t, AUTOMATIC) CoreResetEntry = {0};
  VAR(hseSrvDescriptor_t, AUTOMATIC) HseSrvDescriptor;
  VAR(hseSmrEntryInstallSrv_t, AUTOMATIC) SmrInstallRequest;
  VAR(Std_ReturnType, AUTOMATIC) eRetVal = E_OK;
  VAR(uint8, AUTOMATIC) u8CryptoKeyIndex;
  /* Fetch internal decryption engine key reference */
  u8CryptoKeyIndex = CryptoDal_Crypto_pGlobalConfigPtr[pSMR_CR_Config->u8KeyIndex].u8EncryptionKeyAlternateRef;
#if 0 // Since the previous call has imported the KEY, delete it here
  /* Register key in using the alternate slot. This descriptor shall reside in NVM for verification usage */
  eRetVal = Crypto_KeyElementSet(u8CryptoKeyIndex, CRYPTO_KEY_MATERIAL_U32, CryptoDal_Crypto_pGlobalConfigPtr[pSMR_CR_Config->u8KeyIndex].u8CryptoKey,
                CryptoDal_Crypto_pGlobalConfigPtr[pSMR_CR_Config->u8KeyIndex].u8EncryptionKeyLen);
#endif
  MemLib_MemSet(&SmrEntry, 0, sizeof(SmrEntry));
  /* Configure SMR entry */
  SmrEntry.pSmrSrc                = (HOST_ADDR)pSMR_CR_Config->u32RamDestAddr;// The source address of the SMR installation table entry is set to SRAM address
  SmrEntry.pSmrDest               = (HOST_ADDR)NULL; //dest sram address=null, means no flash to memory copy , Have no dest SRAM address, so no copy action
  SmrEntry.smrSize                = pSMR_CR_Config->u32AppSize;
  SmrEntry.checkPeriod            = 0;
  SmrEntry.configFlags            = HSE_SMR_CFG_FLAG_QSPI_FLASH;
  SmrEntry.authKeyHandle          = CRYPTODAL_SMR_CMAC_KEY_HANDLE(u8CryptoKeyIndex);
  SmrEntry.authScheme.macScheme.macAlgo          = HSE_MAC_ALGO_CMAC;
  SmrEntry.authScheme.macScheme.sch.cmac.cipherAlgo  = HSE_CIPHER_ALGO_AES;
  SmrEntry.pInstAuthTag[0]        = 0;
```

```
    SmrEntry.pInstAuthTag[1]                    = 0;

    /* Create a SMR install request */
    SmrInstallRequest.accessMode                = HSE_ACCESS_MODE_ONE_PASS;
    SmrInstallRequest.entryIndex                = pSMR_CR_Config->u8EntryIndex;
    SmrInstallRequest.pSmrEntry                 = (HOST_ADDR)&SmrEntry;
    SmrInstallRequest.pSmrData                  = (HOST_ADDR)pSMR_CR_Config->u32FlashStorageAddr; // It is read from flash during installation. Obtained from SRAM during verification
    SmrInstallRequest.smrDataLength             = pSMR_CR_Config->u32AppSize;
    SmrInstallRequest.pAuthTag[0]               = (HOST_ADDR)pSMR_CR_Config->pTag;
    SmrInstallRequest.authTagLength[0]          = *((uint32_t*)pSMR_CR_Config->pTagSize);

    MemLib_MemSet(&HseSrvDescriptor, 0, sizeof(HseSrvDescriptor));
    HseSrvDescriptor.srvId                  = HSE_SRV_ID_SMR_ENTRY_INSTALL;
    HseSrvDescriptor.hseSrv.smrEntryInstallReq = SmrInstallRequest;

    if (E_OK == eRetVal)
    {
        if(HSE_SRV_RSP_OK != CryptoDal_Crypto_HseSrv_Request(&HseSrvDescriptor))
        {
            eRetVal = E_NOT_OK;
        }
    }
    return eRetVal;
}
```

C:\NXP\Integration_Reference_Examples_S32G2_2022_06\code\framework\realtime\bsw\dal\cryptodal\platforms\S32G2XX\include\CryptoDal_Crypto.h

```
FUNC(Std_ReturnType, CRYPTODAL_APP_CODE) CryptoDal_Crypto_SMR_Install(P2VAR(CryptoDal_SMR_CR_EntryType, AUTOMATIC, CRYPTODAL_CONST) pSMR_CR_Config);
```

4.2 OnDemand SMR verify

As mentioned above, copy fip.bin to SRAM firstly, and then trigger OnDemand SMR verification.

Function call:

C:\NXP\Integration_Reference_Examples_S32G2_2022_06\code\framework\realtime\swc\bootloader\generic\src\Bootloader.c：

```
StatusType Bl_BootApplications(void)
{…
  Bl_LoadApplication(u8Index); //firstly load the fip.bin to SRAM
|-> Bl_FetchApplication
|   |-> Bl_LoadAndAuthFromQspi
|   |   |-> Bl_TransferImageFromQspi(u8ApplicationId, u8FragmentIdx);
// As follows, this function just calls the QSPI NOR DMA driver to copy the fip.bin from the flash address to the sram address:
static StatusType Bl_TransferImageFromQspi(uint8 u8ApplicationId,
                                           uint8 u8FragmentIdx)
{
  StatusType Status = E_NOT_OK;
  uint32 u32StorageAddress = bootApplications[u8ApplicationId]
                             .pImageFragments[u8FragmentIdx]
                             .u32FlashAddress;
  uint32 u32DestinationAddress = bootApplications[u8ApplicationId]
                                 .pImageFragments[u8FragmentIdx]
                                 .u32DestinationAddress;
  uint32 u32ChunkSize = BL_ALIGN_64B(bootApplications[u8ApplicationId]
                                     .pImageFragments[u8FragmentIdx]
                                     .u32Size);
  /* Start the DMA transfer with the 64-byte transfer size configuration. */
  Bl_StartDmaTransfer(u32StorageAddress, u32DestinationAddress,
                      BL_DMA_SIZE_64B, u32ChunkSize);
  /* Wait for DMA engine to transfer the application image */
  if (E_OK == Bl_WaitApplicationFetch())
  {
    Status = E_OK;
  }
  return Status;
}

  …
```

```
    if (E_OK == Status)
    {
#if (BL_SYNCHRONIZED_BOOT == STD_ON)
#if 1
// If OnDemandSMR verification fails, stop it, otherwise start it normally
if(E_NOT_OK == CryptoDal_OnDemandSMR_Verify(2))
{
    while(1);
}
else
{
    Bl_StartAllApplications();
}

#else
     Bl_StartAllApplications();
#endif
```

Final implement function:

C:\NXP\Integration_Reference_Examples_S32G2_2022_06\code\framework\realtime\bsw\dal\cryptodal\generic\src\CryptoDal.c

```
FUNC(Std_ReturnType, CRYPTODAL_APP_CODE) CryptoDal_OnDemandSMR_Verify(VAR(uint8, AUTOMATIC) entryIndex)
{
    VAR(Std_ReturnType, AUTOMATIC) eRetVal = E_NOT_OK;
    if (NULL_PTR != CryptoDal_pGlobalConfigPtr)
    {
#if CRYPTODAL_USE_CRYPTO
      eRetVal = CryptoDal_Crypto_OnDemandSMR_Verify(entryIndex);
#endif /* CRYPTODAL_USE_CRYPTO */
    }
    return eRetVal;
}
```

C:\NXP\Integration_Reference_Examples_S32G2_2022_06\code\framework\realtime\bsw\dal\cryptodal\generic\include\CryptoDal.h

```
FUNC(Std_ReturnType, CRYPTODAL_APP_CODE) CryptoDal_OnDemandSMR_Verify(VAR(uint8, AUTOMATIC) entryIndex);
```

C:\NXP\Integration_Reference_Examples_S32G2_2022_06\code\framework\realtime\bsw\dal\cryptodal\platforms\S32G2XX\src\CryptoDal_Crypto.c

```
FUNC(Std_ReturnType, CRYPTODAL_APP_CODE) CryptoDal_Crypto_OnDemandSMR_Verify(VAR(uint32, AUTOMATIC)entryIndex)
{
        VAR(hseSrvDescriptor_t, CRYPTODAL_VAR) HseSrvDescriptor;
        VAR(Std_ReturnType, AUTOMATIC) eRetVal = E_OK;
        /* Clear previous request */
        MemLib_MemSet(&HseSrvDescriptor, 0, sizeof(HseSrvDescriptor));
        /* Fill the service descriptor */
         HseSrvDescriptor.srvId                                    = HSE_SRV_ID_SMR_VERIFY;
         HseSrvDescriptor.hseSrv.smrVerifyReq.entryIndex = entryIndex;
        if (E_OK == eRetVal)
        {
                if (HSE_SRV_RSP_OK != CryptoDal_Crypto_HseSrv_Request(&HseSrvDescriptor))
                {
                        eRetVal = E_NOT_OK;
                }
        }
        return eRetVal;
    }
```

C:\NXP\Integration_Reference_Examples_S32G2_2022_06\code\framework\realtime\bsw\dal\cryptodal\platforms\S32G2XX\include\CryptoDal_Crypto.h

```
FUNC(Std_ReturnType, CRYPTODAL_APP_CODE) CryptoDal_Crypto_OnDemandSMR_Verify(VAR(uint32, AUTOMATIC)entryIndex);
```

5 Testing

Set the RDB2 board to QSPI_NOR normal boot, Put the Linux image fsl imagefsl-image-auto-s32g274ardb2.sdcard into the SDcard, insert it into the SDcard slot, connect the serial port and the JTag interface, and power on and start.

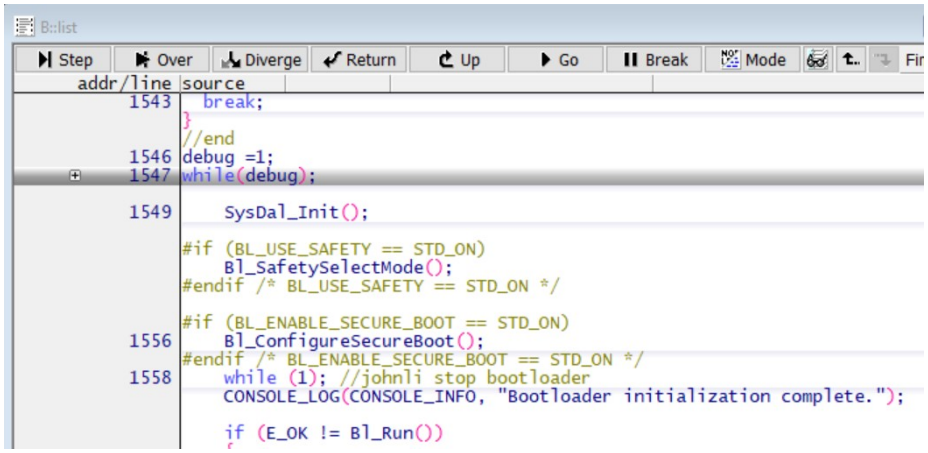
5.1 Lauterbach Tracking

Start and run the follow Lauterbach script:

C:\NXP\Integration_Reference_Examples_S32G2_2022_06\code\framework\realtime\swc\bootloader\platforms\S32G2XX\build\cmm\connect_s32gxx_m7.cmm to connect the board, and will stop at:

while(debug);

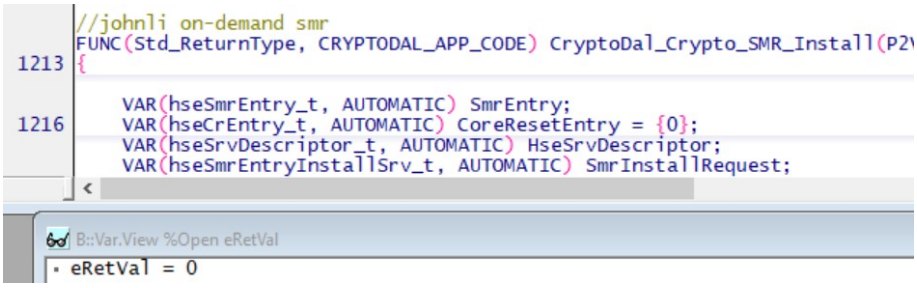
Then switch the code mode to source code mode, and you can use Lauterbach to trace:



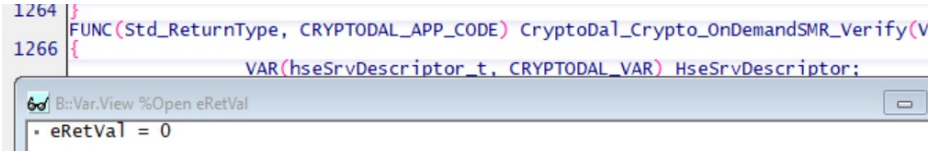
Double click “debug”, change “debug”=1 to “debug”=0, and you can continue running.

Firstly, make sure that Whether the bootApplications related parameters obtained by Bl_GetFipParams function are correct:

Secondly, confirm whether the installation of OnDemand SMR is successful:



After the Bl_ConfigureSecureBoot is completed, click go to run, it will restart, close and reopen Lauterbach to connect to the RDB2 board, modify the debug switcher and continue running, and confirm whether the OnDemand SMR verification is successful:



Finally, confirm whether the Linux Boot is successful:

Click “go” directly on the Trace32 IDE of Lauterbach to run. Check the serial port message to see if ATF/Uboot/Linux is started normally:

NOTICE: Reset status: Power-On Reset

NOTICE: BL2: v2.5(release):bsp36.0-2.5-dirty

NOTICE: BL2: Built : 07:47:27, Sep 14 2023

NOTICE: BL2: Booting BL31

U-Boot 2020.04 (Jul 27 2023 - 10:10:14 +0800)

CPU: NXP S32G274A rev. 2.0

Model: NXP S32G274A-RDB2

DRAM: 3.5 GiB

MMC: FSL_SDHC: 0

Loading Environment from MMC... OK

Configuring PCIe0 as RootComplex

PCIe0: Failed to get link up

PCI: Failed autoconfig bar 20

PCI: Failed autoconfig bar 24

In: serial@401c8000

Out: serial@401c8000

Err: serial@401c8000

Board revision: RDB2/GLDBOX Revision C

…

  fixup: pfe2 set to 00:01:be:be:ef:33

Starting kernel ...

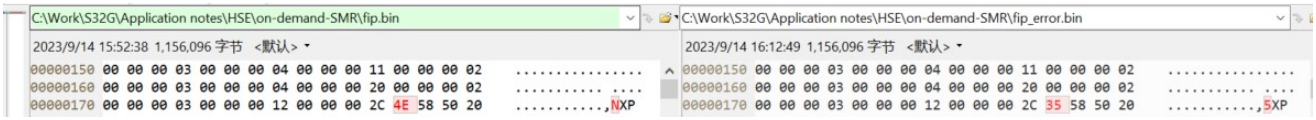
[0.000000] Booting Linux on physical CPU 0x0000000000 [0x410fd034]

[0.000000] Linux version 5.15.96-rt61-dirty (nxa08200@lsv11049.swis.cn-sha01.nxp.com) (aarch64-fsl-linux-gcc (GCC) 10.2.0, GNU ld (GNU Binutils) 2.35.1) #2 SMP PREEMPT Tue Aug 1 10:08:31 CST 2023

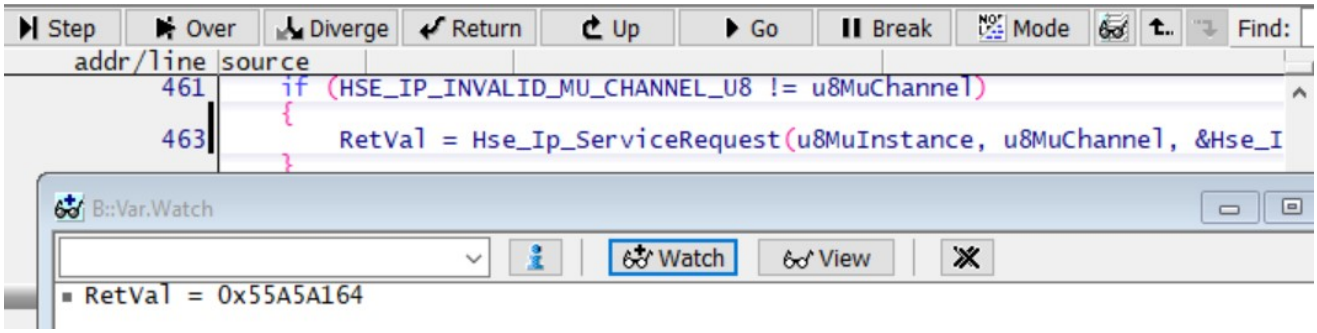
…

5.2 Fip.bin Broken Test

Modify a BYTE in the fip.bin:



Then re programed into QSPI_NOR. After restarting, confirm whether the OnDemand SMR verification is successful:

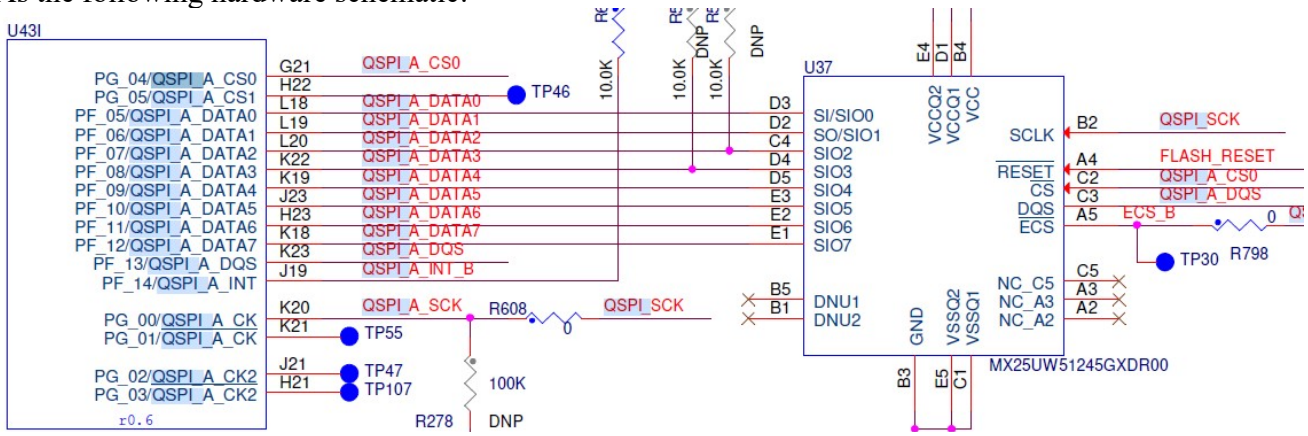


#define HSE_SRV_RSP_VERIFY_FAILED ((hseSrvResponse_t)0x55A5A164UL) /**< @brief HSE signals that a verification request fails (e.g. MAC and Signature verification). */

5.3 Probe the Hardware

In order to avoid competition between HSE and M7 for access to QSPI NOR, our configuration should ensure that during normal secure boot, HSE does not access QSPI NOR during verification. During installation, HSE can access QSPI NOR because it only runs once and theoretically the CPU will be waiting for installation.

As the following hardware schematic:



量测电阻R608(在RDB2板背面，靠近S32G2，在S32G2和QSPI NOR之间)，可以发现在执行函数：Probe the resistance R608 (on the back of the RDB2 board, near the S32G2, between the S32G2 and QSPI_NOR). It can be found that when execute the function:

CryptoDal_SMR_Install(&SMR_CR_Config);

Have the signals output.

When execute the function:

CryptoDal_OnDemandSMR_Verify(2)

Have no signals output, Meet the design requirements.