Rev. 1,9/2023

S32G Secure Boot Customization

by John Li (nxa08200)

本文说明S32G bootloader的Secure Boot功能，考虑以下定制：

- 增加HSE FW更新功能。
- 增加OTP Attribute设置功能。
- 增加IVT签名功能。

以完善Secure Boot功能。

历史	说明	作者
V1	● 创建本文	● John.Li

目录

1 参考资料

以 S32G2 RDB2 为例：

序号	资料	说明	如何获取
1	Platform_Software_Integration _S32G2_2022_06.exe	包含 bootloader 工程，安装后可参考其文档《Bootloader_UserManual.pdf》Secure Boot 一章	www.nxp.com/s32g
2	AN13750: Enabling Multicore Application on S32G2 using S32G2 Platform Software Integration – Application note	Bootloader 工程说明，包括 Secure Boot 配置说明(XueweiWang)。	www.nxp.com/s32g
3	S32G_Bootloader_V*.pdf	Bootloader 应用手册，包换非 Secure Boot 部分(Johnli)	https://community.nxp.com /t5/NXP-Designs-Knowledge -Base/ S32G-Bootloader-Customzition/ta-p/ 1519838
4	HSE_H/M Firmware Reference Manual	HSE FW 用户手册	www.nxp.com/s32g (需要 secure 访问权限)
5	Secure Boot with HSE V0 - Draft B	Secure Boot 说明文档	www.nxp.com/s32g (需要 secure 访问权限)
6	HSE_DEMOAPP_S32G2_0_1_0_5.exe	HSE Demo App(S32DS 版本)	www.nxp.com/s32g
7	HSE_FW_S32G2_0_1_0_5.exe	HSE FW 安装包，安装后可以查看文档《HSE_FW_H_S32G2XX_0.1.0.5 _HSE_Service_API_ Reference_Manual.pdf》	www.nxp.com/s32g

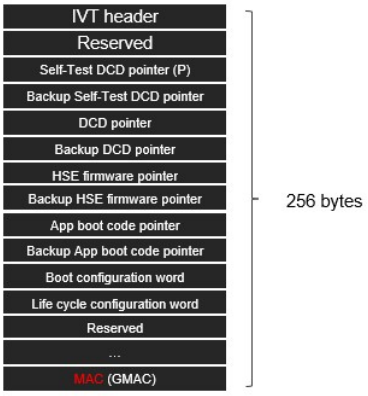
2 S32G Secure Boot 说明

Secure Boot 分为 Secure Boot 配置和 Secure Boot 两部分，配置完成后，HSE M7 核运行的 ROM 代码及 HSE FW 负责完成 Secure Boot 的相关认证，解密工作。

2.1 IVT 头格式与 Secure Boot 相关

SECURE BOOT CONFIGURATION IN THE IVT

- IVT: Boot configuration word
  - BOOT_SEQ
    - 0b - Non-secure boot. BootROM executes application image without authentication
    - 1b - Secure boot. HSE_H firmware executes application image after authentication
- IVT: Life cycle configuration word
  - IN_FIELD
  - OEM_PROD

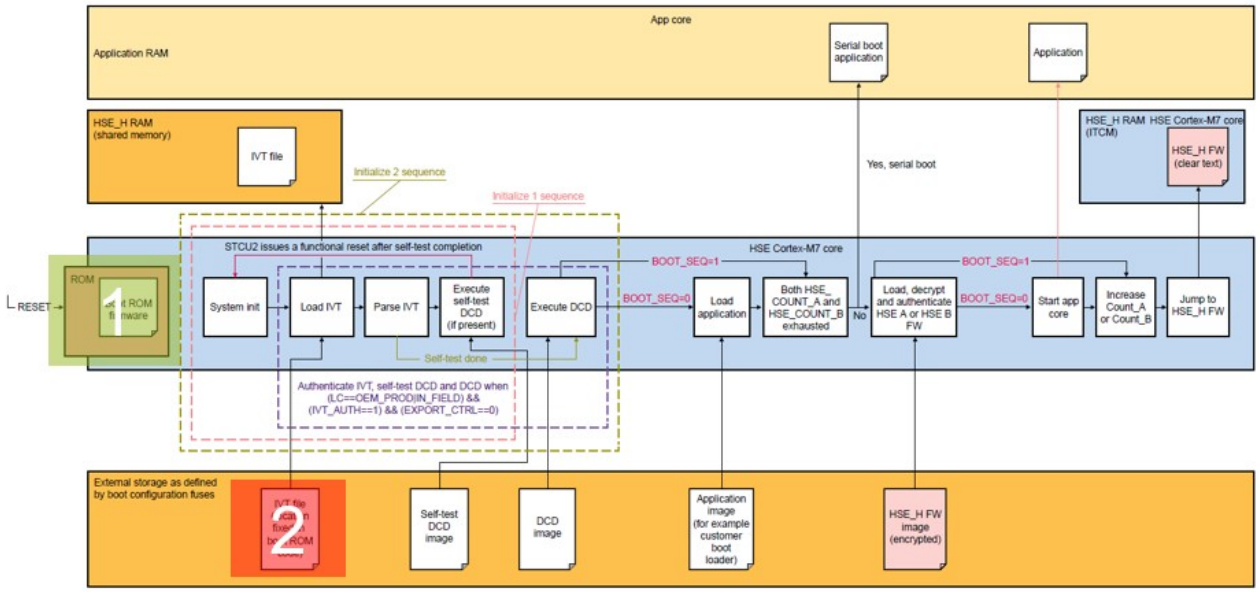


注意：

在制作 Bootloader 的 IVT 镜像时，要保证 BOOT_SEQ bit 没有置位，所以第一次启动为 Non-secure 启动。而因为 BL_ENABLE_SECURE_BOOT 编译宏在 Bootloader 工程配置为打开，所以程序判断为第一次启动，会进入到 Secure Boot configure 函数，此函数发现 BOOT_SEQ 没有置位，则会完成 Secure Boot 配置工作，然后将 IVT BOOT_SEQ bit 置位回 QSPI NOR 中，然后重启，之后就会一直使用 Secure Boot。

2.2 Secure Boot 流程

HSE Secure Boot 的流程图如下：



整个 Secure Boot 过程如下：

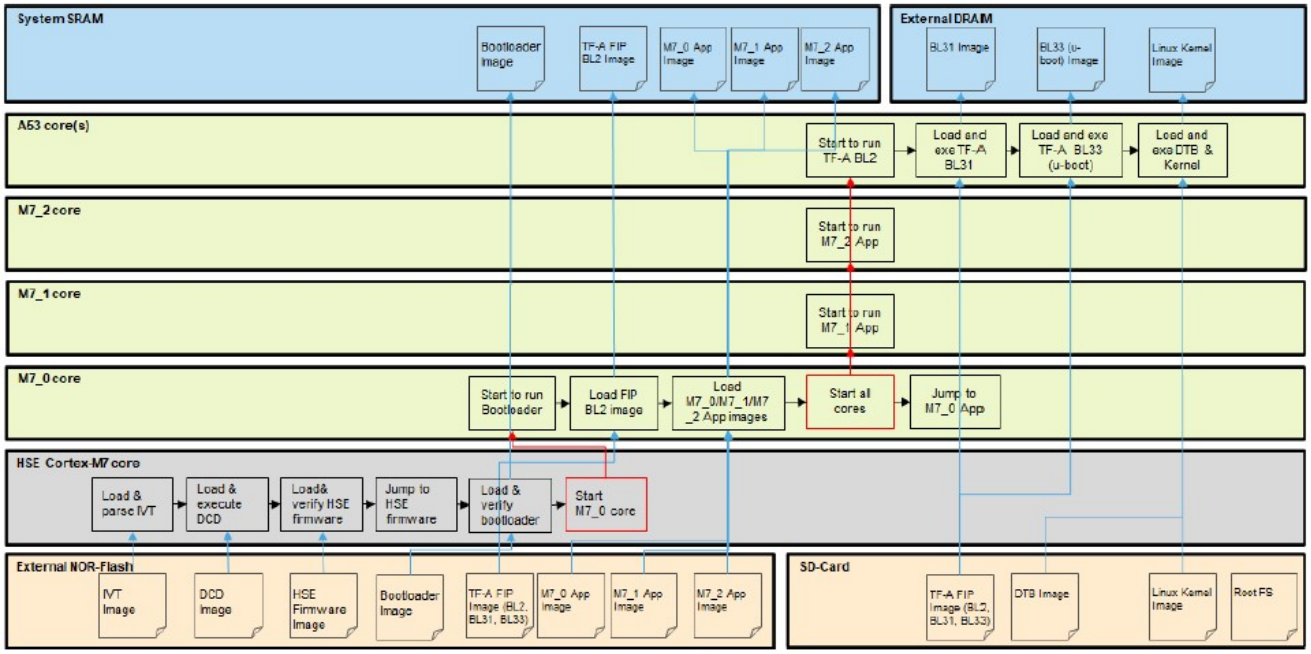


Figure 3. **Secure boot flow**

2.3 Secure Boot 配置

Secure Boot 整个流程主要由 HSE ROM code/HSE FW/Bootloader 完成，所以重点和难点其实在于 Secure Boot 的配置，一般来讲客户有以下几种配置方法：

- Non Secure Boot 后，运行 Autosar Crypto APP 后，调用 MCAL 接口完成 Secure Boot 的配置，然后重启。
- Non Secure Boot 启动后，在 Bootloader 里完成 Secure Boot 的配置，然后 Bootloader 自动重启，之后就全是 Secure Boot，Bootloader 工程就是如此设计的。
- 混合模式：就是部分配置由 Bootloader 自动配置，自动重启。部分配置则用运行起来的 APP 来配置。

一些配置适合使用 Bootloader 自动配置，比如说 Bootloader 镜像的签名，HSE FW 的更新，IVT 头的签名等，但是又有一些配置适合使用 APP 来配置，这些一般涉及到和厂线烧录服务器的交互，比如 KEY 的注入，OTP Attribute 的配置等。但是配置的方法没有一定之规，每个客户的做法不同。

先比较一下文档要求，Bootloader Secure Boot 工程和 HSE Demo Secure Boot 工程的配置过程如下：

步骤		Bootloader Secure Boot 工程	HSE Demo Secure Boot 工程
1	格式化 HSE key catalogs	格式化 HSE key catalogs	相同
2	注入密钥	注入用于生成 bootloader 镜像 CMAC 的 RAM 密钥及相同的 NVM 密钥(用于验证)	注入用于生成/验证 bootloader 镜像 CMAC 的 NVM 密钥
3			生成 SYS_IMG
4			保存 SYS_IMG 到 QSPI NOR
5	通过 SMR 配置 Secure Boot	为 bootloader 镜像生成 CMAC TAG	相似，功能更强
6		配置 bootloader 镜像的 SMR，此处要使用 NVM 密钥	相似，功能更强
7		安装 SMR	相似，功能更强
8		安装 CR	相似，功能更强
9	Publish SYS_IMG 到内存中	Publish SYS_IMG 到内存中	相同
10	将 SYS_IMG 写入 QSPI Nor	将 SYS_IMG 写入 QSPI Nor	相同
11			完成 HSE pink to blue update
12			完成 DCD/sefltest 签名
13	完成 IVT 签名		完成 IVT 签名
14	更新 IVT 头使能 Secure Boot	更新 IVT 头使能 Secure Boot	相同
15	配置其它 OTP Attribute		支持
16	配置 IVT AuTH OTP Attribute		支持

	使能 IVT 保护		
17	演进 LC 到 OEM_PROD		支持
18	重启	重启	相同

所以如果只考虑使用 Bootloader 工程的开发定制，要做到基本产品级别的 Secure Boot 保护的话，考虑增加以下定制功能，(当然，也可以自行开发 APP 来实现)。

- HSE FW 的 pink to blue 更新，可以提高启动速度，也增加了防止抄版的保证。
- 相关 OTP Attribute 的烧写 API，不测试，仅测试 ADKP 的烧写。
- IVT 头的签名功能，不测试启动，仅测试签名。

本文没有考虑以下：

- Bootloader 镜像加密，会影响到启动速度。
- SMR 仅配置一个启动镜像 Bootloader，大部分的应用需要用一个 Bootloader 来启动 M/A 核并且解决相关资源冲突，如时钟。
- DCD/selftest 的签名，需要时可以自行参考 HSE Demo APP 加入。

本文的定制部分主要参考 HSE Demo APP 的流程，所以 HSE Demo APP 为 HSE 的功能比较全的参考平台。

2.4 Secure Boot 涉及到的 HSE 内容

包括以下：

- 密钥目录格式化
- 密钥注入
- CMAC 生成
- SYS_IMG 发布
- HSE FW_IMG 更新
- SMR 配置
- CR 配置
- IVT AUTH
- OTP Attribute 设置及读取，包括 life cycle 演进等。
- …

参考文档《HSE_H/M Firmware Reference manual》和《Secure Boot with HSE V*》了解相关内容。

3 环境搭建

3.1 搭建编译环境

参考文档《AN13750》和《S32G_Bootloader_V*.pdf》，结合起来就可以搭建编译环境，此处不再详述，注意：

- 《AN13750》默认使用 Secure Boot，可以参考此文档，包括 HSE FW 的目录指定。
- 《S32G_Bootloader_V*.pdf》默认使用用 Non Secure Boot，但是如何去掉 SDHC 支持，以及如何解决编译错误可以参考此文档。
- 当只关注 bootloader 的 Secure Boot 部分时，我们重点关注 secure configure 以及 boot 部分，所以 bootloader 启动 M/A 部分本文不做描述。在代码：

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

```
volatile int debug;
int main(void)
{…
debug =1;
while(debug); //增加调试开关接口，方便使用lauterbach来跟踪调试。
Bl_ConfigureSecureBoot();
while (1); //johnli stop bootloader //停止继续 boot，方便调试。
if (E_OK != Bl_Run())
…
```

以上调试开关在正式代码中可以去掉。

另外，本工程也没有去修改 bootloader 项的镜像配置。

3.2 IVT 镜像制造

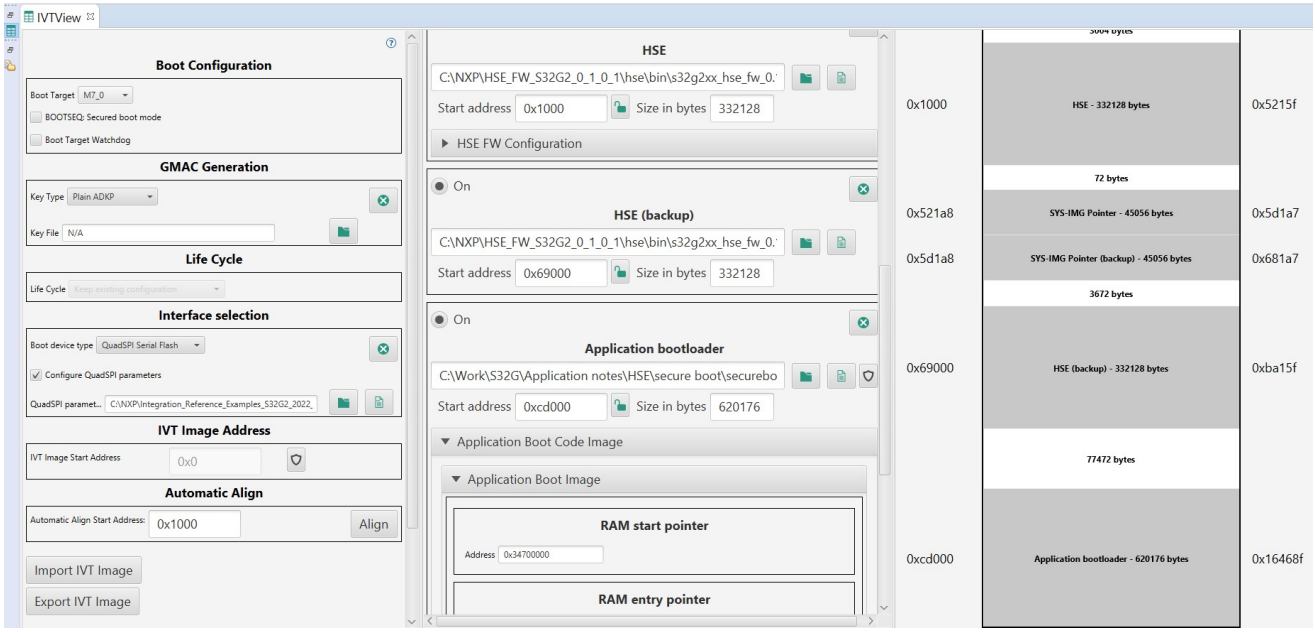
参考文档《AN13750》和《S32G_Bootloader_V*.pdf》，关于如何制造 Bootloader 的 IVT 镜像，**注意：**

- 因为之后我们考虑开发 FW update 的功能，所以在制造 IVT 镜像时，我们准备 FW main 镜像和 FW backup 镜像，这样方便做 FW 的备份。
- 另外注意因为 Fls 驱动的 sector 大小定义为：#define QSPI_SECTOR_SIZE 0x1000=4096，所以我们在工具里做 automatic align Start Address 时使用 0x1000。

- 对于需要更新的镜像，我们设置为以 0x1000 为 align，比如说：

1. HSE FW Start address=0x1000
2. HSE FW Backup Start address=0x69000
3. App bootloader=0xcd000
4. 这样的话，published 的 SYS-IMG 地址我们建议设置为 0x165000

如下图：



可见镜像最大地址为 0x165000，之后解释。

3.3 镜像烧写

参考文档《AN13750》和《S32G_Bootloader_V*.pdf》，关于如何烧写 Bootloader 的 IVT 镜像，**注意：**

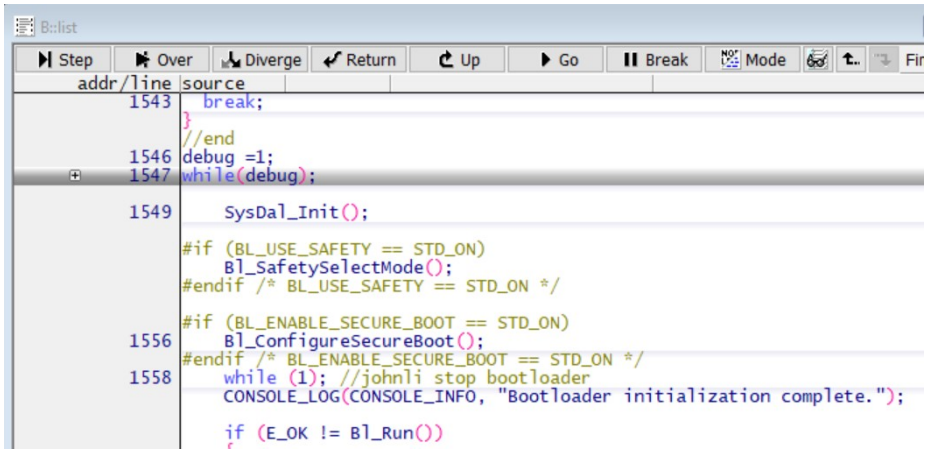
- 因为本工程不含有 M/A 核其它镜像，仅有一个 bootloader，所以我们只烧写 bootloader_blob.bin。

3.4 Bootloader Secure Boot 测试

S32G RDB2 板设置为 QSPI NOR 正常启动，启动后使用 Lauterbach 运行脚本：C:\NXP\Integration_Reference_Examples_S32G2_2022_06\code\framework\realtime\swc\bootloader\platforms\S32G2XX\build\cmm\connect_s32gxx_m7.cmm 连接上板子，板子会停在：

```
while(debug);
```

处，然后切换代码模式为源代码模式，就可以使用 Lauterbach 跟踪了：



双击 debug，将 debug=1 修改为 debug=0，就可以继续往下运行了。

4 Bootloader Secure Boot 代码与功能说明

4.1 EB 配置说明：

打开 Bootloader 工程：关于 KEY 安装信息的配置如下：

Bootloader_S32G2XX_ASR_4.4._M7->Bootloader(…)->Bootloader->General：

- Enable Secure Boot=checked //This flag indicates whether the Secure Boot sequence shall be activated or not
- Secure Boot Key Descriptor =/CryptoDal/CryptoDal/CryptoDalBswConfig/CryptoDescriptors_0 //从安全启动密钥描述符中选择一个加密描述符。 此密钥描述符包含密钥 镜像标签生成和验证所需的配置。 有关详细信息，请参阅下一段 描述符
- HSE System Image Address (1024 -> 4194304) =786432 //0xc0000 System Image Storage Address. At this flash address Bootloader will save the HSE system image 将 HSE 系统镜像地址设置为 QSPI 闪存中的给定地址。 确保 HSE 系统镜像地址不与 HSE 固件、引导加载程序镜像或应用镜像的任何已配置内存区域重叠。 系统镜像的保留内存区域将为 0x4000 字节。 如果这个地区 与上述任何镜像重叠，安全引导流程或来自 QSPI 的应用程序加载将失败。**因为如之前我们 IVT 工具导出的镜像最大地址=0x165000，所以我们这儿修改为 1462272=0x165000**。

注意：

HSE System Image Address 和随后的 0x4000 字节必须定义在 QSPI Flash 中, 与扇区地址对齐。这是必需的，因为引导加载程序要使用 QSPI Flash 驱动程序来更新 HSE 系统 图像。

基于以上原因，我们需要增加 FLS 驱动的 sector 映射，包括：

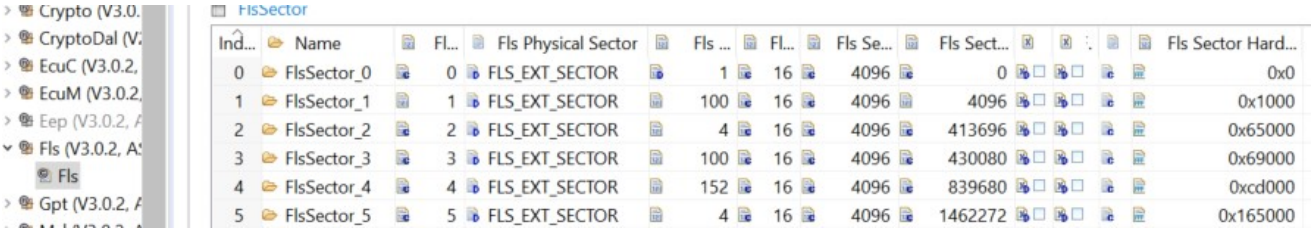
1. HSE FW Start address=0x1000 大小：#define HSE_FW_IMAGE_SIZE 0x64000 //johnli for 400KB =64X0x1000
2. HSE FW Backup Start address=0x69000
3. 这样的话，published 的 SYS-IMG 地址我们建议设置为 0x153000 // #define HSE_SYS_IMAGE_SIZE 0x4000 = 4X0x1000

Bootloader_S32G2XX_ASR_4.4._M7->Fls(…)->Fls->FlsSector:

本来 SYS-IMG 要写入的地址是在 0xc000=786432，连接四个 4096 sector= #define HSE_SYS_IMAGE_SIZE 0x4000：

0	FlsSector_0	0	FLS_EXT_SECTOR	1	16	4096	0	
1	FlsSector_1	1	FLS_EXT_SECTOR	1	16	782336	4096	
2	FlsSector_2	2	FLS_EXT_SECTOR	1	16	4096	786432	
3	FlsSector_3	3	FLS_EXT_SECTOR	1	16	4096	790528	
4	FlsSector_4	4	FLS_EXT_SECTOR	1	16	4096	794624	
5	FlsSector_5	5	FLS_EXT_SECTOR	1	16	4096	798720	
6	FlsSector_6	6	FLS_EXT_SECTOR	1	16	4096	802816	
7	FlsSector_7	7	FLS_EXT_SECTOR	1	16	4096	806912	

现在需要为所有要擦除和写入的 sector 配置 sector 映射，我们将映射修改如下：



Ind...	Name	Fl...	Fls Physical Sector	Fls ...	Fl...	Fls Se...	Fls Sect...	Fls Sector Hard...
0	FlsSector_0	0	FLS_EXT_SECTOR	1	16	4096	0	0x0
1	FlsSector_1	1	FLS_EXT_SECTOR	100	16	4096	4096	0x1000
2	FlsSector_2	2	FLS_EXT_SECTOR	4	16	4096	413696	0x65000
3	FlsSector_3	3	FLS_EXT_SECTOR	100	16	4096	430080	0x69000
4	FlsSector_4	4	FLS_EXT_SECTOR	152	16	4096	839680	0xcd000
5	FlsSector_5	5	FLS_EXT_SECTOR	4	16	4096	1462272	0x165000

其中 FlsSector_1 为 HSE FW 地址，FlsSector_3 为 HSE FW backup 地址，FlsSector_5 为 SYS-IMG publish 的地址。(FlsSector_0/2/4 不需要操作，但是逻辑 sector 需要连续地址，为了与物理地址配置一致，也加入)。

注意：

由于 S32DS IVT tools 和 Fls 驱动配置的逻辑 sector 大小都是 4096B，所以源代码中要相应修改，以防止 align 错误导致的 flash 操作失败的问题，如下：

C:\NXP\Integration_Reference_Examples_S32G2_2022_06\code\framework\realtime\swc\bootloader\generic\include\Bootloader.h

```
#define BL_ALIGN_4096B(x) BL_ALIGN_IMAGE_B(x, 12) //johnli add
```

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

```
static void Bl_SaveConfiguration(…)
{…
```

uint32_t u32SysImageSizeAligned = BL_ALIGN_4096B(u32SysImageSize); //从 BL_ALIGN_10246B 修改而来

这样 Fls 驱动就可以操作我们需要写入的目标地址了。

Bootloader_S32G2XX_ASR_4.4._M7->CryptoDal(…)->CryptoDal->Crypto Driver Abstraction Layer->CryptoDescriptors_0:

- Encryption Engine = CRYPTO // Encryption Engine used for encryption
- Encryption Algorithm= AES_256_CMAC // Encryption Algorithm for which the key is set
- Crypto Key = 0x85, 0xe3, 0xe6, 0x39, 0x1b, 0x13, 0xc2, 0xa3, 0x23, 0x69, 0xb2, 0x36, 0x80, 0x50, 0x4c, 0xbf, 0x1c, 0x12, 0x7b, 0x10, 0xd2, 0x36, 0x7f, 0xf6, 0x8c, 0x0c, 0x35, 0x6b, 0xa8, 0x86, 0x99, 0x0c //32X8=256bit. Encryption key value. Needs to be 16/32 coma separated values, for each bite.
- Crypto Key engine reference = /Crypto/Crypto/CryptoKeys/AES256CMAC_RAM_KEY // Encryption key engine internal reference, containing info about key storage. 用于生成标签

建议将用于标签生成的密钥存放在 RAM 密钥目录中，因为上电复位后将不再需要也无法访问，此密钥仅在安全启动生效之前使用

- Crypto Key alternate reference = /Crypto/Crypto/CryptoKeys/AES256CMAC_NVM_KEY // Encryption key engine alternate reference, used for key mirroring. 用于标签验证，备用密钥引用必须存储在 NVM 密钥目录中，因为此密钥槽将用于检查应用程序映像在之后发出的任何上电复位后是否未被修改

Bootloader_S32G2XX_ASR_4.4._M7->Crypto(…)->Crypto>CryptoKey->AES256CMAC_RAM_KEY:

- CryptoKeyId (0 -> 4294967295) =0 // Identifier of the Crypto Driver key.
- CryptoKeyTypeRef = /Crypto/Crypto/CryptoKeyTypes/Crypto_KT_AES256_CMAC // Refers to a pointer in the CRYPTO to a CryptoKeyType. The CryptoKeyType provides the information about which key elements are contained in a CryptoKey.

->AES256CMAC_NVM_KEY:

- CryptoKeyId (0 -> 4294967295) =1
- CryptoKeyTypeRef =/Crypto/Crypto/CryptoKeyTypes/CryptoKeyType_0

Bootloader_S32G2XX_ASR_4.4._M7->Crypto(…)->Crypto>CryptoKeyType-> Crypto_KT_AES256_CMAC:

- CryptoKeyElementRef= /Crypto/Crypto/CryptoKeyElements/Crypto_KE_AES256_CMAC_RAM_GENERATE //Refers to a Crypto Key Element, which holds the data of the Crypto Key Element.

-> CryptoKeyType_0:

- CryptoKeyElementRef= /Crypto/Crypto/CryptoKeyElements/ /Crypto/Crypto/CryptoKeyElements/Crypto_KE_AES256_CMAC_NVM_VERIFY

Bootloader_S32G2XX_ASR_4.4._M7->Crypto(…)->Crypto>CryptoKeyElement-> Crypto_KE_AES256_CMAC_RAM_GENERATE->General:

- CryptoKeyElementFormat = CRYPTO_KE_FORMAT_BIN_OCTET // Defines the format for the key element. This is the format used to provide or extract the key data from the driver.
- CryptoKeyElementReadAccess= CRYPTO_RA_ALLOWED // Define the reading access rights of the key element.
- CryptoKeyElementSize (1 -> 4294967295)= 32 //Maximum size of the Crypto Key Element value, in bytes. Will be used by Crypto driver to reserve internal memory for those Crypto Key Elements that do not use a HSE key
- CryptoKeyElementWriteAccess = CRYPTO_WA_ALLOWED //Defines the writing access rights of the key element
- Use HSE Key = checked // Vendor specific: Enables or disables the usage of a HSE key.
- HSE Key Catalog Group Ref = /Crypto/Crypto/RamKeyGroup_AES // Vendor specific: The 'HSE Key Catalog Group Ref' identifies the key group where the key is located in the NVM or RAM key catalog.
- HSE Key Slot (0 -> 255) =0 // Vendor specific: Slot of the key inside the key group selected above in the 'HSE Key Catalog Group Ref'.
- HSE Key Counter (0 -> 268435455) = 0 Vendor specific: 28 bits counter used to prevent the rollback attacks on key. When updating a key value and attributes the new counter value must be greater than the current counter value. At counter saturation(0xFFFFFFF)the key cannot be updated anymore.
- HSE SMR Flags (0x0 -> 0xffffffff) = 0x0 // Vendor specific: A map of bits that define which Secure Memory Region (SMR), indexed from 0 to 31, should be verified before the key can be used. Set to zero means not used.

->HseKeyFlags:

- HseKeyFlag_2=USAGE_SIGN //Vendor specific: The key flag specifies the operations or restrictions that can be applied to a key.

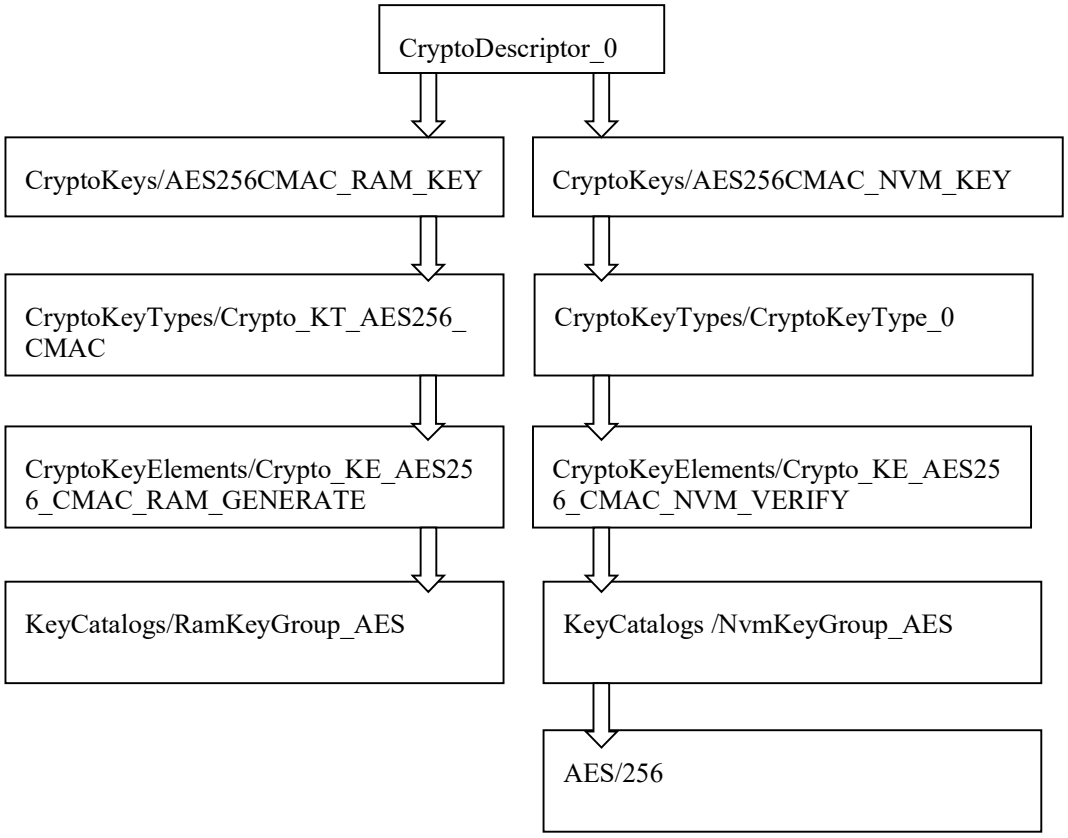
同理分析 Crypto_KE_AES256_CMAC_NVM_VERIFY，与上不同主要为：

- HSE Key Catalog Group Ref= /Crypto/Crypto/NvmKeyGroup_AES
- HseKeyFlag_0/2/3= USAGE_ENCRYPT/VERIFY/ENCRYPT

Bootloader_S32G2XX_ASR_4.4._M7->Crypto(…)->Crypto>KeyCatalogs->NvmKeyGroup_AES

- Key Type = AES //Vendor specific: Specifies the key type. It provides information about the interpretation of key data
- Number of key slots (1 -> 256) = 10 // Vendor specific: The number of key slots in the current key group.
- Max key length in bits (1 -> 65535) = 256 //Vendor specific: The maximum length of the key (in bits). All stored keys can have keyBitLen lower or equal to MaxKeyBitLen
- Key Owner= OWNER_CUST // Vendor specific: Specifies the key group owner.

总结：



生成配置：

C:\EB\tresos\workspace\Bootloader_S32G2XX_ASR_4.4_M7\output\src\CryptoDal_PbCfg.c

```
VAR(uint8, AUTOMATIC) CryptoDal_CryptoKey_0[32] = {0x85, 0xe3, 0xe6, 0x39, 0x1b, 0x13, 0xc2, 0xa3, 0x23, 0x69, 0xb2, 0x36, 0x80, 0x50, 0x4c, 0xbf, 0x1c, 0x12, 0x7b, 0x10, 0xd2, 0x36, 0x7f, 0xf6, 0x8c, 0x0c, 0x35, 0x6b, 0xa8, 0x86, 0x99, 0x0c};

const CryptoDal_ConfigType CryptoDal_Config[CRYPTODAL_MAX_CIPHERS] = {
  {
    32,    /* Key Length (bytes) */
    CRYPTODAL_AES_256_CMAC, /* Encryption Algorithm */
    CRYPTODAL_CRYPTO, /* Encryption Engine */
    CryptoDal_CryptoKey_0, /* Secret Key */
    0, /* Encryption engine internal reference */
    1 /* Encryption engine internal alternate reference */
  },
```

其它：

Bootloader_S32G2XX_ASR_4.4._M7->CryptoDal(…)->CryptoDal->General:

- UseCrypto=checked // This flag indicates whether CryptoDal is enabled or not.
- EnableHashingService=checked // This flag indicates whether hash support is enabled or not.
- EnableSMRSupport =checked // This flag indicates whether Secure Memory region support is enabled or not
- EnableCMACGeneration=checked // This flag indicates Message Authentication generation is enabled or not
- EnableKeyCatalogsFormat=checked // This flag enables the key catalog formatting service.

Bootloader_S32G2XX_ASR_4.4._M7->Crypto(…)->Crypto> CryptoDriverObject-> CryptoDriverObject_0

->General:

- CryptoQueueSize (0 -> 4294967295) =6 // Size of the queue in the Crypto Driver. Defines the maximum number of jobs in the Crypto Driver Object queue. If it is set to 0, queueing is disabled in the Crypto Driver Object. Note: The node value will be used as the element number when declaring an array variable for the QUEUE feature. So the maximum value depends on the memory space of each platform.

- MU Instance = MU_0 // Vendor specific: Selects one of the MU (Messaging Units) instances available on the platform to use for communication with HSE.
- Algorithms Type = CRYPTO_SYMMETRIC_ALGORITHMS // Vendor specific: Determines if the crypto algorithms (primitives) associated with the Crypto Driver Object are symmetric or asymmetric.

->CryptoPrimitiveRef->:

0:/Crypto/Crypto/CryptoPrimitives/CryptoPrimitive_0

1:/Crypto/Crypto/CryptoPrimitives/CryptoPrimitive_1

Bootloader_S32G2XX_ASR_4.4._M7->Crypto(…)->Crypto-> CryptoPrimitives-> CryptoPrimitives-> CryptoPrimitive_0

- CryptoPrimitiveAlgorithmFamily = CRYPTO_ALGOFAM_AES
- CryptoPrimitiveAlgorithmMode = CRYPTO_ALGOMODE_NOT_SET
- CryptoPrimitiveAlgorithmSecondaryFamily = CRYPTO_ALGOFAM_NOT_SET
- CryptoPrimitiveService = MAC_VERIFY

-> CryptoPrimitive_1

- CryptoPrimitiveAlgorithmFamily = CRYPTO_ALGOFAM_AES
- CryptoPrimitiveAlgorithmMode = CRYPTO_ALGOMODE_NOT_SET
- CryptoPrimitiveAlgorithmSecondaryFamily = CRYPTO_ALGOFAM_NOT_SET
- CryptoPrimitiveService = MAC_GENERATE

4.2 EB 生成代码说明：

Bootloader 代码启动流程中与 Secure Boot 相关部分如下：

Startup.s:
\Integration_Reference_Examples_S32G2_2022_06\code\framework\realtime\swc\bootloader\platforms\S32G2XX\src\m7

|-> SystemInit // Initialize the system (MPU, interrupts)

|->main:
\Integration_Reference_Examples_S32G2_2022_06\code\framework\realtime\swc\bootloader\generic\src\bootloader.c

| |->SysDal_Init

| | |->SysDal_StartUpInit

| | | |->SysDal_McuPlatformInitSeq

| | | | |->Mcu_Init，Mcu_SetMode，Mcu_InitClock(McuClockSettingConfig_0);，Mcu_DistributePllClock();// clock initial

| | | |->SysDal_WakeUpInit

| | | | |->InitBlockOneCallout

| | | | | |->//外设驱动初始化

…

| | | | | |-> CryptoDal_Init(&CryptoDal_Config[0]);

| | | | | | |-> CryptoDal_Crypto_Init

| | | | | | | |-> Crypto_Init(NULL_PTR); // /* Initialize hardware module */

| | | | | | | | |->Crypto_Ipw_Init //#define Crypto_Ipw_Init(partitionId) \

(Crypto_Hse_Init(partitionId))

| | | | | | | | | |->Hse_Ip_Init

| | | | | | | | |->Crypto_HandleNvramInfo

| | | | | | | | |->Crypto_Util_InitJobQueues

| | | | | | | |->Crypto_Exts_FormatKeyCatalogs /* Format key catalogs */

| |->Bl_ConfigureSecureBoot

| | |-> if (E_NOT_OK == Bl_IsSecureBootActive())//Check whether Secure Boot was configured previously.

/* Secure Boot is not active */ //所以如果之前的 IVT 中 Secure Boot mode 没有设置，但是工程却 enable 了 Secure Boot，则认为是第一次启动，所以执行之后的签名，安装 SMR，导出 SYS Image 和修改 IVT Secure Boot mode configuration 的动作，之后的启动就都是 Secure Boot 了。

Status = E_NOT_OK;

| | |->Bl_GetBootloaderParams(&u32BootAppFlashAddr, &u32BootAppRamAddr,

&u32BootAppSize); //从 IVT 头中读出系统启动镜像的相关信息

| | |->CryptoDal_GenerateCmac(

(uint8_t *) u32BootAppFlashAddr, &Bl_BootAppTag[0],

BL_SEC_BOOT_KEY_INDEX, u32BootAppSize, &u32TagSize);//为此镜像生成 CMAC 签名

其中 BL_SEC_BOOT_KEY_INDEX=0, 所以是使用的 Crypto_KE_AES256_CMAC_RAM_GENERATE。

| | | |->CryptoDal_pGlobalConfigPtr[u8KeyIndex].eEncryptionEngine;

| | | |->CryptoDal_Crypto_GenerateCmac(pInputData, pTag, u8KeyIndex, u32InputDataSize, pTagSize);

//以下调用首先注入 Key

| | | | |->Crypto_KeyElementSet(u8CryptoKeyIndex, CRYPTO_KEY_MATERIAL_U32, CryptoDal_Crypto_pGlobalConfigPtr[u8KeyIndex].u8CryptoKey,

CryptoDal_Crypto_pGlobalConfigPtr[u8KeyIndex].u8EncryptionKeyLen))

| | | | | |->Crypto_Ipw_ImportKey=Crypto_Hse_ImportKey

| | | | | | |->Crypto_Hse_EccLoadPlainPairKey

pHseSrvDescriptor->srvId = HSE_SRV_ID_IMPORT_KEY;

```
        pImportKeyReq->pKeyInfo       =
HSE_PTR_TO_HOST_ADDR(&Crypto_Hse_apMuState[u8MuInstance]->HseKeyInfo);
        pImportKeyReq->targetKeyHandle =
Crypto_aKeyElementList[u32KeyMaterialKeyElemIdx].u32HseKeyHandle;

        /* Only for encrypted ECC is needed*/
        pImportKeyReq->cipher.cipherKeyHandle     = HSE_INVALID_KEY_HANDLE;
        pImportKeyReq->keyContainer.authKeyHandle = HSE_INVALID_KEY_HANDLE;

        RetVal = Crypto_Hse_SendMsg(u8MuInstance, u8MuChannel, pHseSrvDescriptor, NULL_PTR);
|  |  |  |   |->Crypto_KeySetValid(u8CryptoKeyIndex))
if (CRYPTODAL_AES_256_CMAC == eAlgorithm)
            {
                CryptoDal_Crypto_CmacJobPrimitiveInfo.cryIfKeyId              = u8CryptoKeyIndex;
                CryptoDal_Crypto_CmacJob.jobPrimitiveInputOutput.inputPtr     = pInputData;
                CryptoDal_Crypto_CmacJob.jobPrimitiveInputOutput.inputLength  = u32InputDataSize;

                CryptoDal_Crypto_CmacJob.jobPrimitiveInputOutput.outputPtr     = pTag;
                CryptoDal_Crypto_CmacJob.jobPrimitiveInputOutput.outputLengthPtr = pTagSize;
                eRetVal = Crypto_ProcessJob(CRYPTODAL_CRYPTO_DRIVER_ID, &CryptoDal_Crypto_CmacJob);
            }///* Crypto driver object used */
#define CRYPTODAL_CRYPTO_DRIVER_ID          (0U)
/* --- Structure of the job to be passed to Crypto driver, requesting CMAC with specific keys
---------------------------------------------------------------------------- */
static VAR(Crypto_JobType, AUTOMATIC) CryptoDal_Crypto_CmacJob =
{
  1U,                                   /* jobId                - Identifier for the job structure */
  CRYPTO_JOBSTATE_IDLE,                         /* jobState             - Determines the current job state */
  {
    NULL_PTR,                             /* inputPtr             - Pointer to the input data. */
    0,                                  /* inputLength          - Contains the input length in bytes. */
    NULL_PTR,                             /* secondaryInputPtr    - Pointer to the secondary input data (for MacVerify,
SignatureVerify). */
    0U,                                 /* secondaryInputLength    - Contains the secondary input length in bytes. */
    NULL_PTR,                             /* tertiaryInputPtr     - Pointer to the tertiary input data (for MacVerify,
SignatureVerify). */
```

```
    0U,                                 /* tertiaryInputLength      - Contains the tertiary input length in bytes. */
    NULL_PTR,                           /* outputPtr                - Pointer to the output data. */
    0U,                                 /* outputLengthPtr          - Holds a pointer to a memory location containing the
output length in bytes. */
    NULL_PTR,                           /* secondaryOutputPtr       - Pointer to the secondary output data. */
    NULL_PTR,                           /* secondaryOutputLengthPtr - Holds a pointer to a memory location
containing the secondary output length in bytes. */
    0U,                                 /* input64                  - Versatile input parameter */
    NULL_PTR,                           /* verifyPtr                - Output pointer to a memory location holding a
Crypto_VerifyResultType */
    NULL_PTR,                           /* output64Ptr              - Output pointer to a memory location holding an
uint64. */
    CRYPTO_OPERATIONMODE_SINGLECALL     /* mode                     - Indicator of the
mode(s)/operation(s) to be performed */
  },
  &CryptoDal_Crypto_CmacJobPrimitiveInfo,   /* jobPrimitiveInfo       - Pointer to a structure containing
further information,
                                              depends on the job and the crypto primitive */
  &CryptoDal_Crypto_CmacInfoType,           /* jobInfo                - Pointer to a structure containing further
information,
                                              depends on the job and the crypto primitive */
  NULL_PTR                          /* jobRedirectionInfoRef      - Pointer to a structure containing further
information
                                              on the usage of keys as input and output for jobs. */
};
//以下安装 SMR/CR
| | | |->Bl_ConfigureBootloaderSMR(&SMR_CR_Config, u32BootAppFlashAddr,
                    u32BootAppRamAddr, u32BootAppSize,
                    &Bl_BootAppTag[0], &u32TagSize);
{
  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; //
/* Key index in the cryptodal descriptors list */
```

```
#define BL_SEC_BOOT_KEY_INDEX 0
   pSMR_CR_Config->u8EntryIndex = 1;
   pSMR_CR_Config->u8CoreID = HSE_APP_CORE0; //#define HSE_APP_CORE0       ((hseAppCore_t)0U)    /**< @brief Core0 */
}
|  |  |  |->CryptoDal_SMR_CR_Install(&SMR_CR_Config);
|  |  |  |   |->CryptoDal_Crypto_SMR_CR_Install
/* Fetch internal decryption engine key reference */
|  |  |  |  |     |->u8CryptoKeyIndex = CryptoDal_Crypto_pGlobalConfigPtr[pSMR_CR_Config->u8KeyIndex].u8EncryptionKeyAlternateRef;
/* Register key in using the alternate slot. This descriptor shall reside in NVM for verification usage */
|  |  |  |  |     |->Crypto_KeyElementSet(u8CryptoKeyIndex, CRYPTO_KEY_MATERIAL_U32, CryptoDal_Crypto_pGlobalConfigPtr[pSMR_CR_Config->u8KeyIndex].u8CryptoKey,
                  CryptoDal_Crypto_pGlobalConfigPtr[pSMR_CR_Config->u8KeyIndex].u8EncryptionKeyLen);
|  |  |  |  |     |->MemLib_MemSet(&SmrEntry, 0, sizeof(SmrEntry));
/* Configure SMR entry */
   SmrEntry.pSmrSrc                          = (HOST_ADDR)pSMR_CR_Config->u32FlashStorageAddr;
   SmrEntry.pSmrDest                         = (HOST_ADDR)pSMR_CR_Config->u32RamDestAddr;
   SmrEntry.smrSize                          = pSMR_CR_Config->u32AppSize;
   SmrEntry.checkPeriod                      = 0;
   SmrEntry.configFlags                      = HSE_SMR_CFG_FLAG_QSPI_FLASH; //#define HSE_SMR_CFG_FLAG_QSPI_FLASH       ((hseSmrConfig_t)0x0U)
   SmrEntry.authKeyHandle                    = CRYPTODAL_SMR_CMAC_KEY_HANDLE(u8CryptoKeyIndex);// #define HSE_KEY_CATALOG_ID_NVM       ((hseKeyCatalogId_t)1U)    /**< @brief NVM key catalog */
   SmrEntry.authScheme.macScheme.macAlgo     = HSE_MAC_ALGO_CMAC;   #define HSE_MAC_ALGO_CMAC          ((hseMacAlgo_t)0x11U)
   SmrEntry.authScheme.macScheme.sch.cmac.cipherAlgo  = HSE_CIPHER_ALGO_AES;// #define HSE_CIPHER_ALGO_AES        ((hseCipherAlgo_t)0x10U)
   SmrEntry.pInstAuthTag[0]                  = 0;
   SmrEntry.pInstAuthTag[1]                  = 0;

   /* Create a SMR install request */ //server request
   SmrInstallRequest.accessMode              = HSE_ACCESS_MODE_ONE_PASS;//HSE 访问模式
   SmrInstallRequest.entryIndex              = pSMR_CR_Config->u8EntryIndex;
   SmrInstallRequest.pSmrEntry               = (HOST_ADDR)&SmrEntry;
   SmrInstallRequest.pSmrData                = (HOST_ADDR)pSMR_CR_Config->u32FlashStorageAddr;
```

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; // #define HSE_SRV_ID_SMR_ENTRY_INSTALL ((hseSrvId_t)(HSE_SRV_VER_0 | 0x00000501UL)) //SMR 安装服务 ID

HseSrvDescriptor.hseSrv.smrEntryInstallReq = SmrInstallRequest;

| | | | | |->CryptoDal_Crypto_HseSrv_Request(&HseSrvDescriptor))

/* Configure a Core Reset Entry for Advanced Secure Boot */

CoreResetEntry.coreId = pSMR_CR_Config->u8CoreID;

CoreResetEntry.preBootSmrMap = (1UL << pSMR_CR_Config->u8EntryIndex);

CoreResetEntry.pPassReset = pSMR_CR_Config->u32RamDestAddr;

CoreResetEntry.crSanction = HSE_CR_SANCTION_KEEP_CORE_IN_RESET; //#define HSE_CR_SANCTION_KEEP_CORE_IN_RESET ((hseCrSanction_t)0x7455U) //CR 制裁方式为 core in reset

CoreResetEntry.startOption = HSE_CR_AUTO_START; #define HSE_CR_AUTO_START ((hseCrStartOption_t)0x35A5U)

/* Create the request to HSE using service descriptor */

HseSrvDescriptor.srvId = HSE_SRV_ID_CORE_RESET_ENTRY_INSTALL; #define HSE_SRV_ID_CORE_RESET_ENTRY_INSTALL ((hseSrvId_t)(HSE_SRV_VER_0 | 0x00000503UL))//CR 安装服务 ID

HseSrvDescriptor.hseSrv.crEntryInstallReq.crEntryIndex = 0;

HseSrvDescriptor.hseSrv.crEntryInstallReq.pCrEntry = HSE_PTR_TO_HOST_ADDR(&CoreResetEntry);

//以下将新的 sys-img publish

| | | | |->CryptoDal_GetSysImage(

&Bl_HseSysImage[0], &u32SysImageOffSet, &u32SysImageSize)

@details Shall be used to retrieve the System Image from the HSE subsystem. Whenever a new operation has been registered in the

HSE (Non volatile key, OTFAD configuration, SMR/CR configuration), a system configuration is being kept in HSE internal RAM.

Upon this request, the HSE will publish the system image using the device encryption key.

| | | | | |->CryptoDal_Crypto_GetSysImage(pImage, pSysImageOffset, pHseImageSize);

/* Clear previous request */

MemLib_MemSet(&HseSrvDescriptor, 0, sizeof(HseSrvDescriptor));

```
  HseSrvDescriptor.srvId                              = HSE_SRV_ID_GET_SYS_IMAGE_SIZE;
  HseSrvDescriptor.hseSrv.getSysImageSizeReq.pSysImageSize = (HOST_ADDR)&u32HSEImageSize;
  if (E_OK == eRetVal)
  {
    | | | | | | |->CryptoDal_Crypto_HseSrv_Request(&HseSrvDescriptor);
/* Clear previous request */
  MemLib_MemSet(&HseSrvDescriptor, 0, sizeof(HseSrvDescriptor));

  HseSrvDescriptor.srvId                              = HSE_SRV_ID_PUBLISH_SYS_IMAGE;
  HseSrvDescriptor.hseSrv.publishSysImageReq.publishOptions = HSE_PUBLISH_ALL_DATA_SETS;
  HseSrvDescriptor.hseSrv.publishSysImageReq.pPublishOffset = (HOST_ADDR)(pSysImageOffset);
  HseSrvDescriptor.hseSrv.publishSysImageReq.pBuffLength    = (HOST_ADDR)(pHseImageSize);
  HseSrvDescriptor.hseSrv.publishSysImageReq.pBuff          = (HOST_ADDR)(pImage);

  if (E_OK == eRetVal)
  {
      | | | | | | |->if (HSE_SRV_RSP_OK != CryptoDal_Crypto_HseSrv_Request(&HseSrvDescriptor))
| | | | |->Bl_SaveConfiguration(&Bl_HseSysImage[0], BL_SYS_IMAGE_STORAGE_ADDR,
                    u32SysImageOffSet, u32SysImageSize); ///* HSE system image storage address */
#define BL_SYS_IMAGE_STORAGE_ADDR 786432   EB configure it.
| | | | |   |->Fls_Write(u32SysImageStorageAddr + u32SysImageOffset,
          (const uint8 *) pSysImage, u32SysImageSizeAligned);
/* Update IVT configuration */
  MemLib_MemCpy(&Bl_FlashMirror[0], IVT_ADDR, QSPI_SECTOR_SIZE);
  pIvtUpdated = (Bl_ImageVectorTableType *) &Bl_FlashMirror[0];
  pIvtUpdated->u32BootCfgWord = IVT_BCW_SEC_BOOT_CONFIG_MASK; //此处重新设置 IVT 的 Secure Boot
enable
  pIvtUpdated->u32SysImageAddr = u32SysImageStorageAddr;
  pIvtUpdated->u32ExtFlashType = IVT_ADDR;
  pIvtUpdated->u32FlashPageSize = QSPI_SECTOR_SIZE;
| | | | |   |->Fls_Erase(IVT_ADDR, QSPI_SECTOR_SIZE);
| | | | |   |->Fls_Write(IVT_ADDR, (const uint8 *) &Bl_FlashMirror[0], QSPI_SECTOR_SIZE);
| | | | |->Mcu_PerformReset();
```

通过以上 Secure Boot 代码分析，安全启动执行流程包括以下步骤：

1. 第一次冷启动后（镜像与 HSE 固件、DCD、QSPI 头配置数据和 IVT 头一起写入 QSPI flash 中），引导加载程序在非安全环境运行时执行以下步骤：
   - 解析 IVT 以获取 Bootloader 现有配置（存储地址、起始地址和 镜像大小）。
   - 通过生成身份验证标签，使用在 CryptoDal 中配置的密钥对引导映像进行身份验证 使用 AES-256-CMAC 算法。
   - 上述密钥也存储在 HSE 非易失性密钥目录中，以便在断电后使用 复位 (POR)。
   - 安全内存区域 SMR 在 HSE 系统映像中注册，包含镜像参数（密钥、存储 地址、镜像大小、起始地址、目标启动内核和先前计算的身份验证标签）。
   - 包含非易失性密钥目录和安全内存区域配置的 HSE 系统映像 Sys Image 被导出到 RAM 内存，以便应用程序将其保存到闪存中。
   - 系统镜像 Sys Image 保存到 EB 配置地址的闪存中。这确保在下一次重启时先前注册的 SMR 和密钥将由 HSE 加载和执行。
   - IVT 使用新的配置参数进行更新（安全引导已启用且系统映像 地址被更新），如下图所示，并发出上电复位（POR）。 此外，用户还可以 在下面的数据转储中观察驻留在闪存中的 IVT 标头已被修改（BOOTSEQ 位 位于 0x28 处的现在已开启， HSE 系统映像地址已更新）。
2. 如果前面的操作成功，在下一次重置时，HSE 将检查 Bootloader 镜像的真实性并启动相应配置的内核。 如果修改了任何应用程序映像字节，内核将保持复位和启动状态，过程失败。

5 定制 1：HSE FW update

本章说明如何增加 HSE FW update 功能。

5.1 代码开发

1. 实现从 IVT 头读出 HSE FW 相关信息的函数：

   调用：

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

```
        uint32_t u32FwBlueImageSize = HSE_FW_IMAGE_SIZE;
        uint32_t u32FwPinkImageSize = HSE_FW_IMAGE_SIZE;
      Bl_GetHSEFwParams(&u32HseFwImageFlashAddr, &u32HseBackupFwImageFlashAddr);
      static void Bl_GetHSEFwParams(uint32_t *pHseFwAddr, uint32_t *pHseBackupFwAddr)
{
        *pHseFwAddr=pIVTConfig->u32HSEAddr;
```

```
        *pHseBackupFwAddr=pIVTConfig->u32HSEBckAddr;
}
```

其中，static Bl_ImageVectorTableType *pIVTConfig = IVT_ADDR； // /* Image Vector Table configuration address */#define IVT_ADDR 0x0 IVT 头地块为 QSPI NOR AHB 地址头，0x0。

增加 HSE_FW_IMAGE_SIZE 大小的宏，如下信息。

15.2.3 Image sizes

The below table specifies the maximum image sizes for the HSE firmware.

Table 135: Maximum image sizes

HSE image	HSE_H image size	HSE_M i
FW-IMG (as provided by NXP)	357KB **Note**: It is recommended to allocate 400KB (or more) in flash for each primary and back-up image	261KB **Note**: It i (or more back-up
SYS-IMG	44KB / 48KB on S32ZE **Note:** It is recommended to allocate 48KB in flash.	44KB **Note:** It i in flash

```
#define HSE_FW_IMAGE_SIZE 0x64000 //johnli for 400KB
```

2. 实现 HSE FW update 功能

```
/* HSE FW  Image */
static uint8_t Bl_HseFwImage[HSE_FW_IMAGE_SIZE]; //johnli add 准备一个内存 buffer 来储存 FW.
```

调用：

```
status = CryptoDal_GetFwImage(&Bl_HseFwImage[0], &u32HseFwImageFlashAddr ， &u32FwPinkImageSize, &u32FwBlueImageSize);
```

Dal函数体：

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_GetFwImage(P2VAR(uint8, AUTOMATIC, CRYPTODAL_CONST) pImage,
                                VAR(uint32, AUTOMATIC) u32FwPinkImageFlashAddress,
                                P2VAR(uint32, AUTOMATIC, CRYPTODAL_CONST) pFwPinkImageSize,
                                P2VAR(uint32, AUTOMATIC, CRYPTODAL_CONST) pFwBlueImageSize)
{
   VAR(Std_ReturnType, AUTOMATIC) eRetVal = E_NOT_OK;
   if (NULL_PTR != CryptoDal_pGlobalConfigPtr)
   {
#if CRYPTODAL_USE_CRYPTO
      eRetVal = CryptoDal_Crypto_GetFwImage(pImage, u32FwPinkImageFlashAddress, pFwPinkImageSize, pFwBlueImageSize);
#endif /* CRYPTODAL_USE_CRYPTO */
   }
   return eRetVal;
}
```

具体实现：
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_GetFwImage(P2VAR(uint8, AUTOMATIC, CRYPTODAL_CONST) pImage,
                                VAR(uint32, AUTOMATIC) u32FwPinkImageFlashAddress,
                                P2VAR(uint32, AUTOMATIC, CRYPTODAL_CONST) pFwPinkImageSize,
                                P2VAR(uint32, AUTOMATIC, CRYPTODAL_CONST) pFwBlueImageSize)
{
  VAR(hseSrvDescriptor_t, CRYPTODAL_VAR) HseSrvDescriptor;
  VAR(Std_ReturnType, AUTOMATIC) eRetVal = E_OK;
  VAR(uint32, AUTOMATIC) u32FwBlueImageSize = (*pFwBlueImageSize);
  VAR(uint32, AUTOMATIC) u32FwPinkImageSize = (*pFwPinkImageSize);
  VAR(uint32, AUTOMATIC) u32FwPinkImageFlashAddress = (*pFwPinkImageFlashAddress);

  //获得FW的尺寸。注意此函数虽然在代码中有实现，但是事实在转换正在运行的FW，只需要传递尺寸参数为0就可以了，所以此函数仅只在转换非运行状态的FW时有用。
  // u32FwPinkImageSize=GetHseFwSize((uintptr_t)u32FwPinkImageFlashAddress);
        //update FW from pink to blue发送FW updated服务，将pink FW做为输入，输出blue FW到内存中。其中pink FW的地址是： QSPI NOR AHB memory的地址。
       //update FW from pink to blue
         //get the pink FW size
        /* Clear previous request */
  MemLib_MemSet(&HseSrvDescriptor, 0, sizeof(HseSrvDescriptor));

        /* Fill the service descriptor */
  HseSrvDescriptor.srvId                          = HSE_SRV_ID_FIRMWARE_UPDATE;
  HseSrvDescriptor.hseSrv.firmwareUpdateReq.pInFwFile =
HSE_PTR_TO_HOST_ADDR(&u32FwPinkImageFlashAddress);
#if 1
  HseSrvDescriptor.hseSrv.firmwareUpdateReq.inFwFileLength = 0x0;
#else
  HseSrvDescriptor.hseSrv.firmwareUpdateReq.inFwFileLength = u32FwPinkImageSize;
#endif
  HseSrvDescriptor.hseSrv.firmwareUpdateReq.pOutFwBuffer   = HSE_PTR_TO_HOST_ADDR(pImage);
  HseSrvDescriptor.hseSrv.firmwareUpdateReq.pFwBufferLength        =
HSE_PTR_TO_HOST_ADDR(pFwBlueImageSize);
  if (E_OK == eRetVal)
  {
    if (HSE_SRV_RSP_OK != CryptoDal_Crypto_HseSrv_Request(&HseSrvDescriptor))
                {
                        eRetVal = E_NOT_OK;
                }
  }

  return eRetVal;
}
```

3. 实现更新HSE FW镜像的功能。

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

```
//write blue image to hse main fw
Bl_SaveFwConfiguration(&Bl_HseFwImage[0],u32HseFwImageFlashAddr,u32FwBlueImageSize);
```

具体实现：

```
static void Bl_SaveFwConfiguration(uint8_t *pFwImage,
                           uint32_t u32FwImageStorageAddr,
                           uint32_t u32FwImageSize)
{
   uint32_t u32FwImageSizeAligned = BL_ALIGN_4096B(u32FwImageSize);
       Fls_Erase(u32FwImageStorageAddr,
           u32FwImageSizeAligned);
   while (MEMIF_IDLE != Fls_GetStatus())
   {
      Fls_MainFunction();
   }
   Fls_Write(u32FwImageStorageAddr,
         (const uint8 *) pFwImage, u32FwImageSizeAligned);
   while (MEMIF_IDLE != Fls_GetStatus())
   {
      Fls_MainFunction();
   }
  …
}
```

头文件定义：
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_GetFwImage(P2VAR(uint8, AUTOMATIC, CRYPTODAL_CONST) pImage,
                           VAR(uint32, AUTOMATIC) u32FwPinkImageFlashAddress,
                           P2VAR(uint32, AUTOMATIC, CRYPTODAL_CONST) pFwPinkImageSize,
                           P2VAR(uint32, AUTOMATIC, CRYPTODAL_CONST) pFwBlueImageSize);
```

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

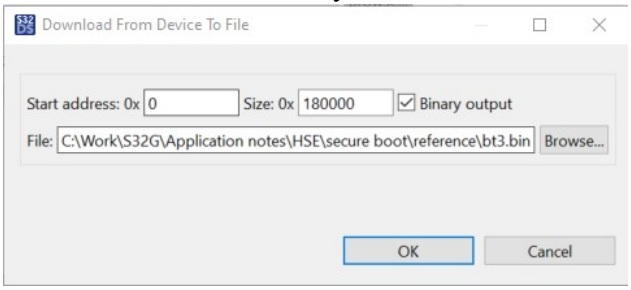
```
FUNC(Std_ReturnType, CRYPTODAL_APP_CODE) CryptoDal_Crypto_GetFwImage(P2VAR(uint8, AUTOMATIC, CRYPTODAL_CONST) pImage,
                           VAR(uint32, AUTOMATIC) u32FwPinkImageFlashAddress,
                           P2VAR(uint32, AUTOMATIC, CRYPTODAL_CONST) pFwPinkImageSize,
                           P2VAR(uint32, AUTOMATIC, CRYPTODAL_CONST) pFwBlueImageSize);
```

注意：
建议在将pink image更新为blue image后，将原有的pink image备份到HSE FW backup image flash address，客户可以自行调试。

5.2 测试

Secure Boot成功之后，将RDB2板重新设置为下载模式，使用flash tools读出镜像：

点击：Upload target and algorithm to hardware...，
点击：Download from device to file...导出Binary：



然后与原始镜像对比，查看地址0x1000处，可以看到原来的pink image是以D4开头的，而新的blue image是以D6开头的，而且可以正常Secure Boot。

6 定制 2：HSE OTP Attribute 设置

本章考虑烧写以下 OTP fuse：

- ADKP
- Authorization mode，from default pass-word to challenge-response(客户很少用到)
- Lifecycle
- IvtAuth

由于以上 OTP 操作是一次性的，所以只会编译代码，不做实际测试，只能 Lifecyle 做一下模拟测试，ADKP 在制作 IVT 签名前需要实际烧写。**特别小心此点!!!**

6.1 代码开发

1. 实现 OTP Attribute 设置/读取 API

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_SetAttribute(VAR(hseAttrId_t, AUTOMATIC) attrId,
                    VAR(uint32_t, AUTOMATIC) attrLen,
                     P2VAR(uint8, AUTOMATIC, CRYPTODAL_CONST) pAttr
                                                )
{
  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_SET_ATTR;
    HseSrvDescriptor.hseSrv.setAttrReq.attrId           = attrId;
    HseSrvDescriptor.hseSrv.setAttrReq.attrLen  = attrLen;
    HseSrvDescriptor.hseSrv.setAttrReq.pAttr            = HSE_PTR_TO_HOST_ADDR (pAttr);
   if (E_OK == eRetVal)
  {
     if (HSE_SRV_RSP_OK != CryptoDal_Crypto_HseSrv_Request(&HseSrvDescriptor))
            {
                    eRetVal = E_NOT_OK;
            }
  }
}

FUNC(Std_ReturnType, CRYPTODAL_APP_CODE) CryptoDal_Crypto_GetAttribute(VAR(hseAttrId_t,
AUTOMATIC) attrId,
                VAR(uint32_t, AUTOMATIC) attrLen,
                P2VAR(uint8, AUTOMATIC, CRYPTODAL_CONST) pAttr
                                                )
{
  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_GET_ATTR;
    HseSrvDescriptor.hseSrv.setAttrReq.attrId           = attrId;
    HseSrvDescriptor.hseSrv.setAttrReq.attrLen  = attrLen;
    HseSrvDescriptor.hseSrv.setAttrReq.pAttr            = HSE_PTR_TO_HOST_ADDR (pAttr);
   if (E_OK == eRetVal)
  {
     if (HSE_SRV_RSP_OK != CryptoDal_Crypto_HseSrv_Request(&HseSrvDescriptor))
            {
```

```
                    eRetVal = E_NOT_OK;
                }
    }
}
```

2. 实现烧录 ADKP 的函数：

调用：

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

```
/*
   The application debug key/password (ADK/P) to be programmed - 128-bits
   Used by `HSE_ProgramAdkp` function to program ADK/P in fuses
*/
static uint8_t applicationDebugKeyPassword[16] =
{
   0x00, 0x11, 0x22, 0x33, 0x44, 0x55, 0x66, 0x77, 0x88, 0x99, 0xAA, 0xBB, 0xCC, 0xDD, 0xEE, 0xFF
};
CryptoDal_ProgAdkp(&applicationDebugKeyPassword[0]);
```

Dal函数体：

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_ProgAdkp(P2VAR(uint8, AUTOMATIC, CRYPTODAL_CONST) pAdkp)
{
   VAR(Std_ReturnType, AUTOMATIC) eRetVal = E_NOT_OK;
   if (NULL_PTR != CryptoDal_pGlobalConfigPtr)
   {
#if CRYPTODAL_USE_CRYPTO
      eRetVal = CryptoDal_Crypto_ProgAdkp(pAdkp);
#endif /* CRYPTODAL_USE_CRYPTO */
   }
   return eRetVal;
}
```

具体实现：

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_ProgAdkp(P2VAR(uint8,
AUTOMATIC, CRYPTODAL_CONST) pAdkp)
{
  return
CryptoDal_Crypto_SetAttribute(HSE_APP_DEBUG_KEY_ATTR_ID,sizeof(hseAttrApplDebugKey_t),pAdkp);
}
```

3. 实现配置 Auth 模块为 CR:

调用：

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

Dal函数体：

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_SetAuthModeToCR()
{
  VAR(Std_ReturnType, AUTOMATIC) eRetVal = E_NOT_OK;
  if (NULL_PTR != CryptoDal_pGlobalConfigPtr)
  {
#if CRYPTODAL_USE_CRYPTO
    eRetVal = CryptoDal_Crypto_SetAuthModeToCR();
#endif /* CRYPTODAL_USE_CRYPTO */
  }
  return eRetVal;
}
```

具体实现：

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_SetAuthModeToCR(void)
{
  hseAttrDebugAuthMode_t debugAuthMode;
  debugAuthMode=HSE_DEBUG_AUTH_MODE_CR;
    return
CryptoDal_Crypto_SetAttribute(HSE_DEBUG_AUTH_MODE_CR,sizeof(hseAttrDebugAuthMode_t),&debugAuthMode);
}
```

4. 实现 lifecycle 演进:

调用：

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

```
CryptoDal_AdvanceLifecycle(HSE_LC_OEM_PROD);
CryptoDal_AdvanceLifecycle(HSE_LC_IN_FIELD);
```

Dal函数体：

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_AdvanceLifecycle(hseAttrSecureLifecycle_t targetLifeCycle)
{
  VAR(Std_ReturnType, AUTOMATIC) eRetVal = E_NOT_OK;
  if (NULL_PTR != CryptoDal_pGlobalConfigPtr)
  {
#if CRYPTODAL_USE_CRYPTO
    eRetVal = CryptoDal_Crypto_AdvanceLifecycle(targetLifeCycle);
#endif /* CRYPTODAL_USE_CRYPTO */
  }
  return eRetVal;
}
```

具体实现：

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_AdvanceLifecycle(VAR(hseAttrSecureLifecycle_t, AUTOMATIC) u32targetLifecycle)
{
    return
CryptoDal_Crypto_SetAttribute(HSE_SECURE_LIFECYCLE_ATTR_ID,sizeof(hseAttrSecureLifecycle_t),&u32targetLifecycle);
}
```

5. 实现 IVT Auth:

调用：

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

```
CryptoDal_SetIvtAuth();
```

Dal函数体：

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_SetIvtAuth()
{
  VAR(Std_ReturnType, AUTOMATIC) eRetVal = E_NOT_OK;
  if (NULL_PTR != CryptoDal_pGlobalConfigPtr)
  {
#if CRYPTODAL_USE_CRYPTO
    eRetVal = CryptoDal_Crypto_SetIvtAuth();
#endif /* CRYPTODAL_USE_CRYPTO */
  }
  return eRetVal;
```

```
}
```

具体实现：

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

```
/* Enable IVT/DCD/ST-DCD authentication */
FUNC(Std_ReturnType, CRYPTODAL_APP_CODE) CryptoDal_Crypto_SetIvtAuth(void)
{
    hseAttrConfigBootAuth_t ivtAuth;
     /* Enable IVT/DCD/ST-DCD signature verification by BootROM. */
   ivtAuth = HSE_IVT_AUTH;
    return CryptoDal_Crypto_SetAttribute(HSE_ENABLE_BOOT_AUTH_ATTR_ID,sizeof(hseAttrSecureLifecycle_t),&ivtAuth);
}
```

6. 整个函数调用：

```
#define HSE_PROG_OTP_ATTR 1
#if defined(HSE_PROG_OTP_ATTR)
volatile uint8_t gProgramAdkp = 0U;
volatile uint8_t gSetDebugAuthModeToChallengeResponse = 0U;
volatile uint8_t gAdvanceLifecycleToOemProd = 0U;
volatile uint8_t gAdvanceLifecycleToInField = 0U;
volatile uint8_t gSetIvtAuth = 0U;
/*
   The application debug key/password (ADK/P) to be programmed - 128-bits
   Used by `HSE_ProgramAdkp` function to program ADK/P in fuses
*/
static uint8_t applicationDebugKeyPassword[16] =
{
   0x00, 0x11, 0x22, 0x33, 0x44, 0x55, 0x66, 0x77, 0x88, 0x99, 0xAA, 0xBB, 0xCC, 0xDD, 0xEE, 0xFF
};
#endif
StatusType Bl_ConfigureSecureBoot(void)
{…
    if (E_OK == status)
    {
```

```
            #if defined(HSE_PROG_OTP_ATTR)
                                //program the OTP attribute
                                 /* Program Application debug key/password example. Current Lifecycle must be Customer
Delivery */
   if(0xDA == gProgramAdkp)
   {
      CryptoDal_ProgAdkp(&applicationDebugKeyPassword[0]);
   }

   /* Set Debug Authorization mode to Challenge-Response (Default mode is password-based) */
   if(0xDA == gSetDebugAuthModeToChallengeResponse)
   {
      CryptoDal_SetAuthModeToCR();
   }

   /* Advances Lifecycle to OEM Production Lifecycle */
   if(0xDA == gAdvanceLifecycleToOemProd)
   {
      CryptoDal_AdvanceLifecycle(HSE_LC_OEM_PROD);
   }

   /* Advances Lifecycle to Infield Lifecycle */
   if(0xDA == gAdvanceLifecycleToInField)
   {
      CryptoDal_AdvanceLifecycle(HSE_LC_IN_FIELD);
   }

            /* Enable IVT/DCD/ST-DCD authentication */
            if(0xDA == gSetIvtAuth)
            {
                        CryptoDal_SetIvtAuth();
            }
            #endif
            //end
…
```

头文件定义：

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

```
#include "hse_srv_attr.h" //johnli for compiling
…
FUNC(Std_ReturnType, CRYPTODAL_APP_CODE) CryptoDal_ProgAdkp(P2VAR(uint8, AUTOMATIC, CRYPTODAL_CONST)pAdkp);
FUNC(Std_ReturnType, CRYPTODAL_APP_CODE) CryptoDal_SetAuthModeToCR(void);
FUNC(Std_ReturnType, CRYPTODAL_APP_CODE) CryptoDal_AdvanceLifecycle(hseAttrSecureLifecycle_t targetLifeCycle);
FUNC(Std_ReturnType, CRYPTODAL_APP_CODE) CryptoDal_CheckLifecycle(P2VAR(hseAttrSecureLifecycle_t, AUTOMATIC, CRYPTODAL_CONST) ptargetLifeCycle);
FUNC(Std_ReturnType, CRYPTODAL_APP_CODE) CryptoDal_SetIvtAuth(void);
```

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_ProgAdkp(P2VAR(uint8, AUTOMATIC, CRYPTODAL_CONST) pAdkp);
FUNC(Std_ReturnType, CRYPTODAL_APP_CODE) CryptoDal_Crypto_SetAuthModeToCR(void);
FUNC(Std_ReturnType, CRYPTODAL_APP_CODE) CryptoDal_Crypto_AdvanceLifecycle(VAR(hseAttrSecureLifecycle_t, AUTOMATIC) u32targetLifecycle);
FUNC(Std_ReturnType, CRYPTODAL_APP_CODE) CryptoDal_Crypto_CheckLifecycle(P2VAR(uint32, AUTOMATIC, CRYPTODAL_CONST) ptargetLifecycle);
FUNC(Std_ReturnType, CRYPTODAL_APP_CODE) CryptoDal_Crypto_SetIvtAuth(void);
```

6.2 模拟测试

因为 OTP 是一次性的，所以以上代码仅编译，并未测试。

以下模拟一个 lifecycle 演进的模拟测试，开发以下测试代码：

调用：

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

```
#define HSE_PROG_OTP_ATTR_TEST 1
#if defined(HSE_PROG_OTP_ATTR_TEST)
volatile uint8_t gAdvanceLifecycleToSimulOemProd = 0xDA;
volatile uint8_t gAdvanceLifecycleToSimulInField = 0xDA;
hseAttrSecureLifecycle_t gLifecycle=HSE_LC_CUST_DEL
#endif
…
StatusType Bl_ConfigureSecureBoot(void)
```

```
{…
   if (E_NOT_OK == Bl_IsSecureBootActive())
   {…}
   else
    {
#if defined(HSE_PROG_OTP_ATTR_TEST)
                /* Advances Lifecycle to simulated OEM Production Lifecycle, do not prog fuse, reset recovery */
   if(0xDA == gAdvanceLifecycleToSimulOemProd)
   {
      CryptoDal_AdvanceLifecycle(HSE_LC_SIMULATED_OEM_PROD);
   }
      CryptoDal_CheckLifecycle(&gLifecycle);
   /* Advances Lifecycle to simulated Infield Lifecycle  do not prog fuse, reset recovery*/
   if(0xDA == gAdvanceLifecycleToSimulInField)
   {
      CryptoDal_AdvanceLifecycle(HSE_LC_SIMULATED_IN_FIELD);
   }
   CryptoDal_CheckLifecycle(&gLifecycle);
#endif
```

其中CryptoDal_CheckLifecycle函数：
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_CheckLifecycle(P2VAR(hseAttrSecureLifecycle_t, AUTOMATIC, CRYPTODAL_CONST)
ptargetLifeCycle)
{
   VAR(Std_ReturnType, AUTOMATIC) eRetVal = E_NOT_OK;
   if (NULL_PTR != CryptoDal_pGlobalConfigPtr)
   {
#if CRYPTODAL_USE_CRYPTO
      eRetVal = CryptoDal_Crypto_CheckLifecycle(ptargetLifeCycle);
#endif /* CRYPTODAL_USE_CRYPTO */
   }
   return eRetVal;
}
```

具体实现：
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_CheckLifecycle(P2VAR(uint32, AUTOMATIC, CRYPTODAL_CONST) ptargetLifecycle)
{
    return CryptoDal_Crypto_GetAttribute(HSE_SECURE_LIFECYCLE_ATTR_ID,sizeof(hseAttrSecureLifecycle_t),ptargetLifecycle);
}
```

用 lauterbach 跟踪，在设置完 lifecycle 为 HSE_LC_SIMULATED_OEM_PROD 和 HSE_LC_SIMULATED_IN_FIELD 后，读出的 lifecycle 为 HSE_LC_OEM_PROD 和 HSE_LC_IN_FIELD。

注意以上函数在非 secure configure 时运行，设置后直接读有效，reset 后如果不再配置其进入模拟模式，则无效，因为设置为模拟模式不会烧写 fuse。

建议：

对于 OTP attribute 的配置工作，建议每次烧写 fuse 之前都事先读以确认是否已经有过写操作，以防止过度烧写的情况，所以在写操作前建议增加读操作接口调用，下一章有 ADKP 的示例，建议其它 attribute 也相似操作。

7 定制 3：IVT 签名

将 IVT 头签名以防止篡改。

注意：

- IVT 签名时需要预先安装 ADKP，所以前章的 ADKP 烧写操作我们打开。
- IVT 启动认证需要满足两个条件：1：IVT_AUTH=1，2：LC 演进到 OEM_PROD/IN_FIELD，由于我们不打算演进芯片，所以只测试配置函数。
- 注意对整个 IVT 部分，包括 DCD 和 self-test DCD(如果有)，都需要增加 TAG 头，本节只说明 IVT 部分的代码从 HSE Demo porting 到 bootloader 的说明，对于 DCD 和 self-test DCD 部分，可以参考此办法 porting。
- 注意烧写 ADKP 时，需要应用程序拉 GPIO 为 VDD_EFUSE 供电，可以参考 HSE Demo 中 App_VddEfusePower 函数，本文没有说明此函数的 porting，MCAL 中可以使用 DIO 驱动来实现。
- 开机实现 VDD_EFUSE GPIO 拉动，可以采用两种方法，方法 1 是使用 IVT 中的 efuse marker，这种方法是由 HSE FW 来操作的，所以其中的 delay 要求文档是要求大于 1ms，并且需要实际测试，和寄存器检查，如下：

Table 128: VDD_EFUSE configuration word

Bit #	Description
31	GPIO Polarity: - 0: the VDD_EFUSE is powered when driving the GPIO low - 1: the VDD_EFUSE is powered when driving the GPIO high
16:30	GPIO MSCR number as mentioned in the IO Mux sheet for the GPIO (refer to [REF02]); based on device type, it can be in the ranges from below table.
0:15	Delay provided in microseconds. Must be greater than 1 millisecond. **Note**: The delay period must be measured from the moment the GPIO switch occurs and the NCSPD_STAT4 bit is set (refer to [REF02]). Note that the measurement may differ between the boards if different external components are used. A large enough delay must be programmed to ensure that the VDD_EFUSE supply is powered.

Table 129: MSCR number

Device Type	SUIL2_0_MSCR	SUIL2_1_MSCR	SUIL2_4_MSCR	SUIL2_5_MSCR
S32G2/G3	0 - 101	112 - 190	N/A	N/A
S32R45	0 - 101	102 - 133	N/A	N/A

方法 2: 使用 Self-DCD 或都 DDR Init DCD,将拉动 GPIO 的 DCD 代码放在 DCD 段最前面，这样在执行过 DCD 后，到 HSE FW 运行时间远远大于 1ms。

7.1 代码开发

1. 去掉之前的 IVT 保存代码

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

```
static void Bl_SaveConfiguration(…)
{
#if 0
  /* Update IVT configuration */
…
  #endif
}

static void Bl_SaveFwConfiguration(…)
{…
#if 0
  /* Update IVT configuration */
…
  #endif
}
```

2. 在调用 IVT GMAC 生成函数前，烧写 ADKP 做为密钥

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

```
volatile uint8_t gProgramAdkp = 0xDA;
StatusType Bl_ConfigureSecureBoot(void)
{…
hseAttrApplDebugKey_t adkpHash;
…
if(0xDA == gProgramAdkp)
  {
    if(HSE_SRV_RSP_OK != CryptoDal_GetAdkp(&adkpHash[0U])) //先判断 ADKP 是否已经烧写，如果没有再烧写。
    {
    CryptoDal_ProgAdkp(&applicationDebugKeyPassword[0]);
    }
  }
```

3. 实现 ADKP 检查函数。

Dal 函数体：

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_GetAdkp(P2VAR(uint8, AUTOMATIC, CRYPTODAL_CONST) pAdkp)
{
  VAR(Std_ReturnType, AUTOMATIC) eRetVal = E_NOT_OK;
  if (NULL_PTR != CryptoDal_pGlobalConfigPtr)
  {
#if CRYPTODAL_USE_CRYPTO
    eRetVal = CryptoDal_Crypto_GetAdkp(pAdkp);
#endif /* CRYPTODAL_USE_CRYPTO */
  }
  return eRetVal;
}
```

具体实现：

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_GetAdkp(P2VAR(uint8, AUTOMATIC,
CRYPTODAL_CONST)pAdkp);
```

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_GetAdkp(P2VAR(uint8, AUTOMATIC,
CRYPTODAL_CONST) pAdkp)
{
  return
CryptoDal_Crypto_GetAttribute(HSE_APP_DEBUG_KEY_ATTR_ID,sizeof(hseAttrApplDebugKey_t),pAdkp);
}
```

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_GetAdkp(P2VAR(uint8, AUTOMATIC,
CRYPTODAL_CONST) pAdkp);
```

4. 实现 IVT 保存函数。

调用：

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

```
//johnli add for signed IVT
                    Bl_SaveSignedIvtConfiguration(BL_SYS_IMAGE_STORAGE_ADDR,
                                u32HseBackupFwImageFlashAddr,
                                u32HseFwImageFlashAddr);
```

具体实现：

```
static void Bl_SaveSignedIvtConfiguration(uint32_t u32SysImageStorageAddr,
                                uint32_t u32FwBackupImageStorageAddr,
                                uint32_t u32FwImageStorageAddr)
{
  Bl_ImageVectorTableType *pIvtUpdated;
   hseAttrApplDebugKey_t adkpHash;
    /* Update IVT configuration */
  MemLib_MemCpy(&Bl_FlashMirror[0], IVT_ADDR, QSPI_SECTOR_SIZE);
  pIvtUpdated = (Bl_ImageVectorTableType *) &Bl_FlashMirror[0];

  pIvtUpdated->u32BootCfgWord = IVT_BCW_SEC_BOOT_CONFIG_MASK;
    pIvtUpdated->u32SysImageAddr = u32SysImageStorageAddr;//此函数也保存 sys_img publish 相关参数
  pIvtUpdated->u32HSEAddr = u32FwImageStorageAddr; //此函数也保存 fw_img update 相关参数
```

```
    pIvtUpdated->u32HSEBckAddr=u32FwBackupImageStorageAddr;
  pIvtUpdated->u32ExtFlashType = IVT_ADDR;
  pIvtUpdated->u32FlashPageSize = QSPI_SECTOR_SIZE;
 CryptoDal_SignIvt((uint8_t *)pIvtUpdated,0x10,(uint8_t *)pIvtUpdated->u32GMAC[0]);// //此函数生成 IVT GMAC tag


  Fls_Erase(IVT_ADDR, QSPI_SECTOR_SIZE);
  while (MEMIF_IDLE != Fls_GetStatus())
  {
    Fls_MainFunction();
  }
  Fls_Write(IVT_ADDR, (const uint8 *) &Bl_FlashMirror[0], QSPI_SECTOR_SIZE);
  while (MEMIF_IDLE != Fls_GetStatus())
  {
    Fls_MainFunction();
  }
}
```

5. 实现 IVT GMAC Tag 生成函数。

   Dal 函数体：

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_SignIvt(P2VAR(uint8, AUTOMATIC, CRYPTODAL_CONST) pImage,
                                VAR(uint32, AUTOMATIC) inTaglenth,
                                  P2VAR(uint8, AUTOMATIC, CRYPTODAL_CONST) pOutTagAddr);
```

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_SignIvt(P2VAR(uint8, AUTOMATIC, CRYPTODAL_CONST) pImage,
                                VAR(uint32, AUTOMATIC) inTaglenth,
                                 P2VAR(uint8, AUTOMATIC, CRYPTODAL_CONST) pOutTagAddr)
{
  VAR(Std_ReturnType, AUTOMATIC) eRetVal = E_NOT_OK;
  if (NULL_PTR != CryptoDal_pGlobalConfigPtr)
```

```
    {
#if CRYPTODAL_USE_CRYPTO
        eRetVal = CryptoDal_Crypto_SignIvt(pImage, inTaglenth, pOutTagAddr);
#endif /* CRYPTODAL_USE_CRYPTO */
    }
   return eRetVal;
}
```

具体实现：

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_SignIvt(P2VAR(uint8, AUTOMATIC, CRYPTODAL_CONST)pImage,
                                VAR(uint32, AUTOMATIC)inTaglenth,
                                P2VAR(uint8, AUTOMATIC, CRYPTODAL_CONST)pOutTagAddr);
```

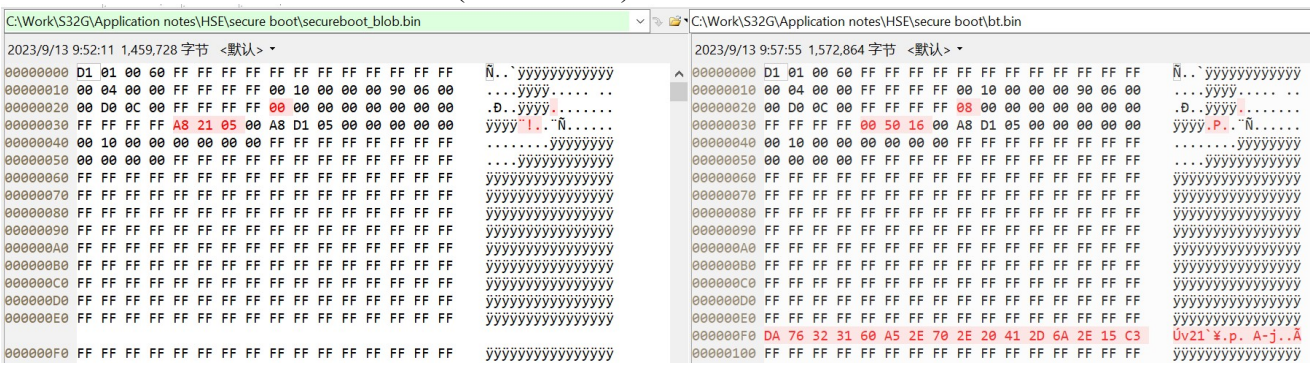
C:\NXP\Integration_Reference_Examples_S32G2_2022_06\code\framework\realtime\bsw\dal\cryptodal\platforms\S32G2XX\src\ CryptoDal_Crypto.c

```
FUNC(Std_ReturnType, CRYPTODAL_APP_CODE)CryptoDal_Crypto_SignIvt(P2VAR(uint8, AUTOMATIC, CRYPTODAL_CONST)pInImage,
                                VAR(uint32, AUTOMATIC)inTagLength,
                                P2VAR(uint8, AUTOMATIC, CRYPTODAL_CONST)pOutTagAddr)
{
   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_BOOT_DATA_IMAGE_SIGN;
           HseSrvDescriptor.hseSrv.bootDataImageSignReq.pInImage = (HOST_ADDR)(pInImage);
           HseSrvDescriptor.hseSrv.bootDataImageSignReq.inTagLength = inTagLength;
           HseSrvDescriptor.hseSrv.bootDataImageSignReq.pOutTagAddr = (HOST_ADDR)(pOutTagAddr);


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

7.2 模拟测试

本文并不实际测试IVT AUTH的启动认证效果(涉及到life cycle演进问题)，所以如下验证IVT GMAC Tag：

使用flash tools读出IVT头镜像(如之前说明)：



以及S32DS IVT工具导出的原生IVT镜像比较：Export IVT Image->C format

```
/* IVT binary */
const uint8_t ivt_binary[256] = {

        /* HEADER */
        0xd1, 0x1, 0x0, 0x60,

        /* reserved_1 */
        0xff, 0xff, 0xff, 0xff,
        /* Self-Test DCD */
        0xff, 0xff, 0xff, 0xff,
        /* Self-Test DCD (backup) */
        0xff, 0xff, 0xff, 0xff,
        /* DCD */
        0x0, 0x4, 0x0, 0x0,
        /* DCD (backup) */
        0xff, 0xff, 0xff, 0xff,
        /* HSE */
        0x0, 0x10, 0x0, 0x0,
        /* HSE (backup) */
        0x0, 0x90, 0x6, 0x0,
        /* Application bootloader */
        0x0, 0xd0, 0xc, 0x0,
        /* Application bootloader (backup) */
        0xff, 0xff, 0xff, 0xff,
        /* boot_config */
        0x0, 0x0, 0x0, 0x0,
        /* life_cycle_config */
        0x0, 0x0, 0x0, 0x0,
        /* reserved_2 */
        0xff, 0xff, 0xff, 0xff,
        /* hse_fw_config */
```

```
        0xa8, 0x21, 0x5, 0x0, 0xa8, 0xd1, 0x5, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x10, 0x0, 0x0,
        0x0, 0x0, 0x0, 0x0, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0x0, 0x0, 0x0, 0x0,
        0xff, 0xff, 0xff, 0xff,
        /* reserved_3 */
        0xff,…0xff,
        /* gmac */
        0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff,
};
```

可以看到BOOT_SEQ bit已经设置，SYS-IMG的地址已经更新，IVT头的GMAC值已经产生。