S32G HSE Mcal Crypto and Customization

by John Li (nxa08200)

本文说明S32G HSE MCAL的Crypto驱动，及其扩展功能开发，考虑以下定制：

- 增加Get Attribute的更多示例。

历史	说明	作者
V1	● 创建本文	● John.Li

目录

1 参考资料

1.1 参考资料

以 S32G2 RDB2 为例：

序号	资料	说明	如何获取
1	HSE 0.1.0.1 发布包： ● HSE_DEMOAPP_S32G2_0.1.0.1_SCR.txt ● HSE_DEMOAPP_S32G2_0_1_0_1.exe ● HSE_FW_S32G2_0.1.0.1_ReleaseNotes.pdf ● HSE_FW_S32G2_0.1.0.1_SCR.txt ● HSE_FW_S32G2_0_1_0_1.exe ● HSE_FW_S32G2_0_1_0_1.txt	安装两个安装包 exe，检查安装目前中文件与资源	www.nxp.com/s32g 安装后， HSE FW API 文档在：C:\NXP\HSE_FW_S32G2_0_1_0_1\docs\ HSE_FW_H_S32G2XX_0.1.0.1 _HSE_Service_API_Reference_Manual.pdf
2	● SW32_RTD_4.4_3.0.2_D2203_ReleaseNotes.pdf ● SW32G_RTD_4.4_3.0.2_D2203.exe	RTD MCAL	www.nxp.com/s32g 安装后，C:\NXP\SW32G_RTD_4.4_3.0.2 \eclipse\plugins\ Crypto_TS_T40D11M30I2R0\doc\ RTD_CRYPTO_IM.pdf RTD_CRYPTO_UM.pdf 为使用文档和集成文档
3	S32G2RM.pdf	芯片手册	www.nxp.com/s32g 查看 Message Unit 一章
4	RM649908-HSE_HM Firmware Reference Manual(0.8).pdf	HSE FW 芯片手册	www.nxp.com/s32g 需要 secure 访问权限
5	AN13750.pdf：Enabling Multicore Application on S32G2 using S32G2 Platform Software Integration	Bootloader 说明文档	www.nxp.com/s32g
6	S32G_Bootloader_V*.pdf	Bootloader 说明文档	https://community.nxp.com/ t5/NXP-Designs-Knowledge-Base/ S32G-Bootloader-Customzition/ta-p/1519838
7	AUTOSAR_SWS_CryptoDriver.pdf	Autosar 文	www.autosar.org

		件	

1.2 版本匹配说明

参考文档: C:\NXP\SW32G_RTD_4.4_3.0.2\eclipse\plugins\Crypto_TS_T40D11M30I2R0\doc\RTD_CRYPTO_IM.pdf：

The HSE files above should be retrieved from the release hse_fw_s32g2xx_0.1.0.1 or hse_fw_s32g2xx_1.1.0.1 for S32G2 platform, from the release hse_fw_s32g3xx_0.0.17.0 for S32G3 platform。The Crypto driver in this RTD release was tested using the firmware image and HSE interface files from the release hse_fw_s32g2xx_1.1.0.1 on S32G2 platform, from the release hse_fw_s32g3xx_0.0.17.0 on S32G3 platform。

以及：

C:\NXP\SW32G_RTD_4.4_3.0.2\eclipse\plugins\Crypto_TS_T40D11M30I2R0\examples\EBT\Hse_Ip_AesEncAsyncIrq_S32G274A_M7\ project_parameters.mk：

```
#The paths to the additional directories to be included at build phase
ADDITIONAL_INCLUDE_DIRS = C:/NXP/HSE_FW_S32G274_0_1_0_1/interface \
                          C:/NXP/HSE_FW_S32G274_0_1_0_1/interface/config \
                          C:/NXP/HSE_FW_S32G274_0_1_0_1/interface/inc_common \
                          C:/NXP/HSE_FW_S32G274_0_1_0_1/interface/inc_custom \
                          C:/NXP/HSE_FW_S32G274_0_1_0_1/interface/inc_services
```

所以默认 RTD3.0.2 测试使用的 HSE FW 版本为 0.1.0.1，实际上 HSW FW 会保持一定的 API 兼容性，所以可以使用其它版本的 HSW FW，本文是基于 0.1.0.1。

2 HSE FW 服务

HSE FW 的编程接口是基于服务的，本章说明其原理，可以选择阅读。

2.1 服务描述符

HSE 服务由主机通过消息单元 (MU) 通过服务通道触发。要求对存储在 RAM 中的服务描述符(通常情况下)中进行格式化。服务描述符是一种 ISO C99 数据结构,包含服务的不同参数, 由 HSE 固件执行。服务描述符由 C 类型 hseSrvDescriptor_t 定义为：

```
typedef struct hseSrvDescriptor_tag
{
/** @brief The service ID of the service descriptor */
hseSrvId_t srvId;
/** @brief The meta data related to the service */
```

hseSrvMetaData_t srvMetaData;

union

{

hse**XXX**Srv_t **XXX**;

hse**YYY**Srv_t **YYY**;

} hseSrv;

} hseSrvDescriptor_t;

服务(例如加密、签名验证等)由参数 srvID 中的服务 ID 标识。 服务描述符中所有后续参数的数量和类型取决于该服务 ID 和分组。 恩智浦 HSE 原生支持的每个服务描述符的完整详细描述参考 HSE API 文档：

struct hseSrvDescriptor_t

union hseSrvDescriptor_t.hseSrv

2.2 服务编号

服务 ID 由版本、类型、类和子类组成。它由 C 类型 hseSrvId_t 定义。 版本字段用于突出显示每个服务的当前版本。更改版本时，当前服务不再与以前的服务兼容：

Table 16: Service ID format

Bit field	31 ~ 24	23 ~ 16	15 ~ 8	7 ~ 0
Description	Service Version	Service Type	Service class	Service subclass

表：服务类型

服务类型	描述
0x00	服务请求由 HSE 以 FIFO 方式排队和执行； 这类服务如果仍在作业队列中，则可以取消服务
0xA5	服务请求由 HSE 直接执行； 这种类型的服务不能取消

下表列出了主机可以实例化的不同服务类。

表：服务类

服务类	描述	服务子类
0x00	HSE 固件管理服务	- 设置/读取系统属性 - 固件升级 - 取消服务 - 功能自检

		- 引导数据签名/验证 - 系统权限授权（用户/ 超级用户） - 导入/导出流上下文 - 系统映像发布/ OTFAD 服务
0x01	加密密钥管理	- 格式化密钥目录 - 密钥导入/导出 - 获取密钥信息/擦除密钥 - 密钥生成/派生 - 密钥协议
0x02	加密功能	- 加密/解密 - 签名/验证 - MAC 生成/验证 - 哈希
0x03	随机数生成	- 获得随机数
0x04	单调计数器管理	- 设置单调计数器值 - 增量计数器 - 读数计数器
0x05	安全内存区域 (SMR) 管理	- SMR 定义 - SMR 安装 - SMR 按需验证
0x06	IPSec	- 初始化 IPSEC 安全关联 (SA) - 处理 IPSEC 帧 - 获取期望的序号 作为一个 - 设置 SA 的预期顺序 数字
0x07 ~ 0x8F	RFU	
0x90	扩展服务	定制服务
0xA1	SHE	- 加载密钥，加载普通密钥 - 导出 RAM 密钥

		- 获取 UID - 启动正常，启动失败

服务 ID 由 C 宏 HSE_SRV_ID_XXX 定义，其中 XXX 表示服务类型。服务 ID 以 0x00A5XXXX 开头的请求无法取消。服务 ID 的完整列表 可以在 HSE API 文档中找到。

2.3 服务请求和响应

要触发服务请求，

主机：

- 通过所需的服务描述符在应用程序 RAM 中实例化服务类。
- 在其中一个 MU 实例中选择一个空闲的服务通道
- 向所选通道提供实例化服务描述符的地址。

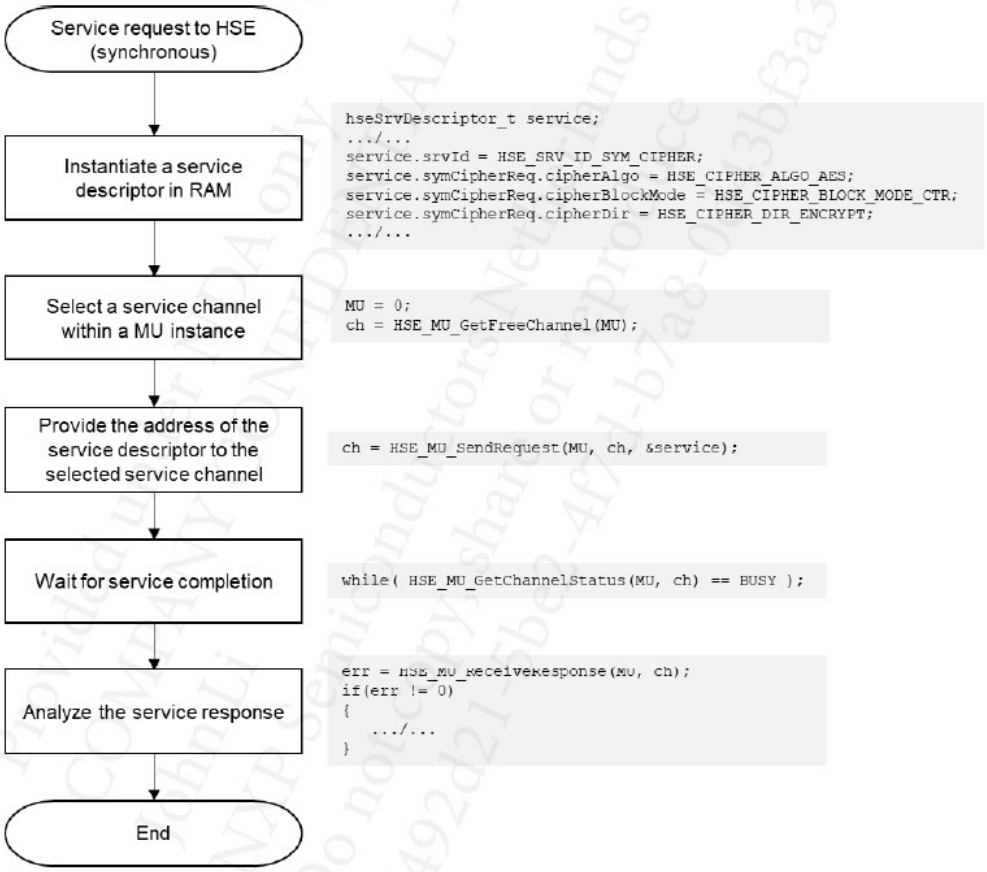
要检查服务请求的结果，

主机：

- 检查所选服务道的操作是否结束。
- 或者等到所选服务通道上的中断被触发。
- 从选定的通道中检索服务响应，并检查潜在的错误。

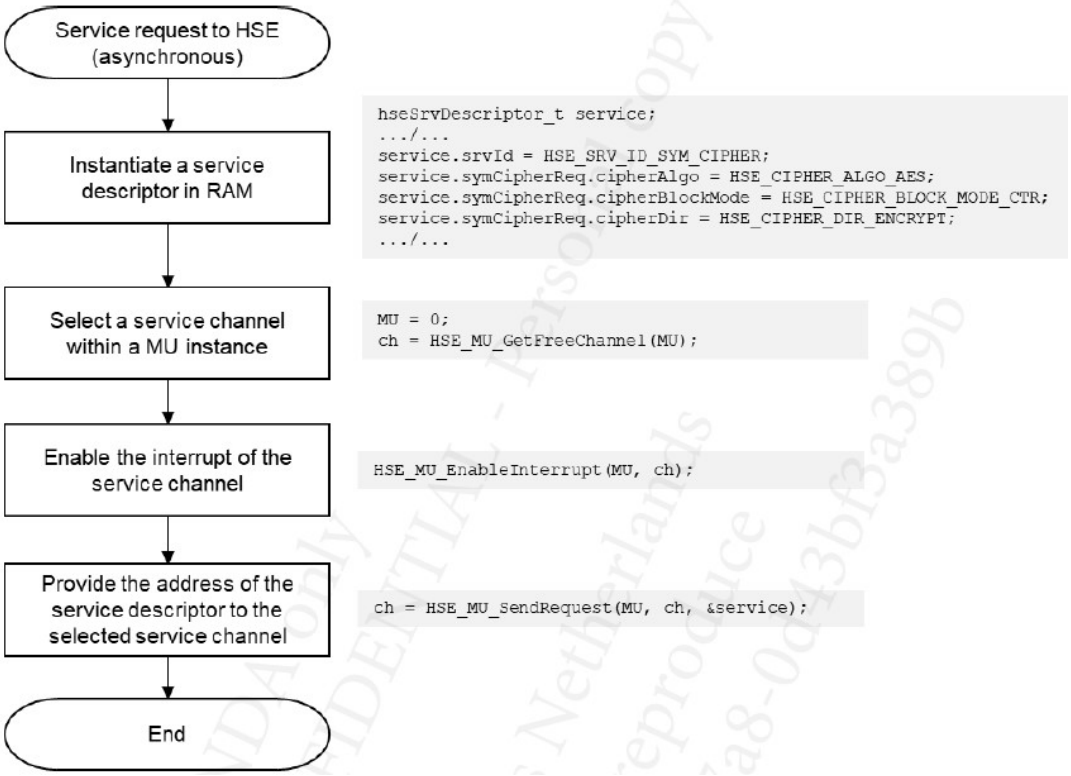
下面的流程说明了主机如何触发服务请求并处理服务响应的同步方式。该流程使用相应的代完成。

Figure 17: Service request flow (synchronous)



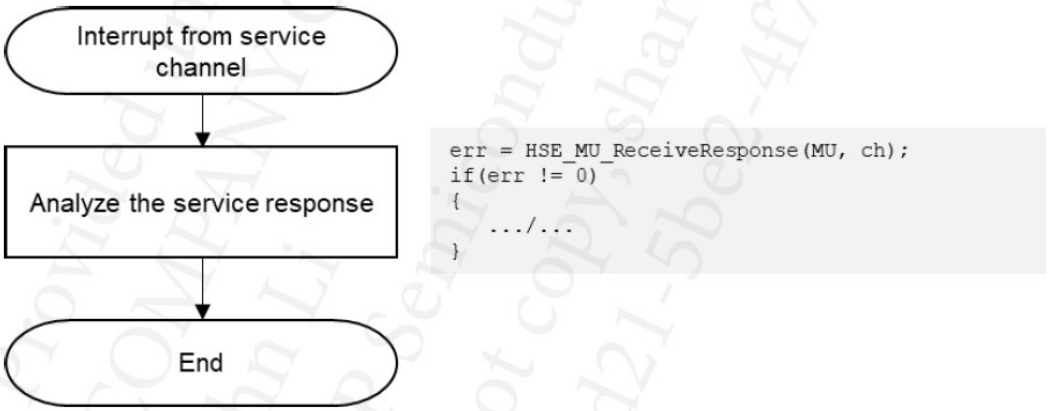
下面的流程说明了主机如何触发服务请求，检查服务响应的异步方式。 使用相应的代码完成流程：

Figure 18: Service request flow (asynchronous)



下面的流程说明了以异步方式检查服务响应的中断例程：

Figure 19: Service response flow (asynchronous)



HSE 始终通过服务通道之一提供服务响应，以指示服务请求结束。它指示请求的服务是否成功结束，值为 HSE_SRV_RSP_OK 或 因具有不同值的错误而终止。

服务响应由 C 类型 hseSrvResponse_t 定义。 服务响应的完整列表可以在 HSE API 文档中找到。重要：服务响应必须由主机读取以释放关联的服务通道。

2.4 服务执行

2.4.1 服务执行顺序

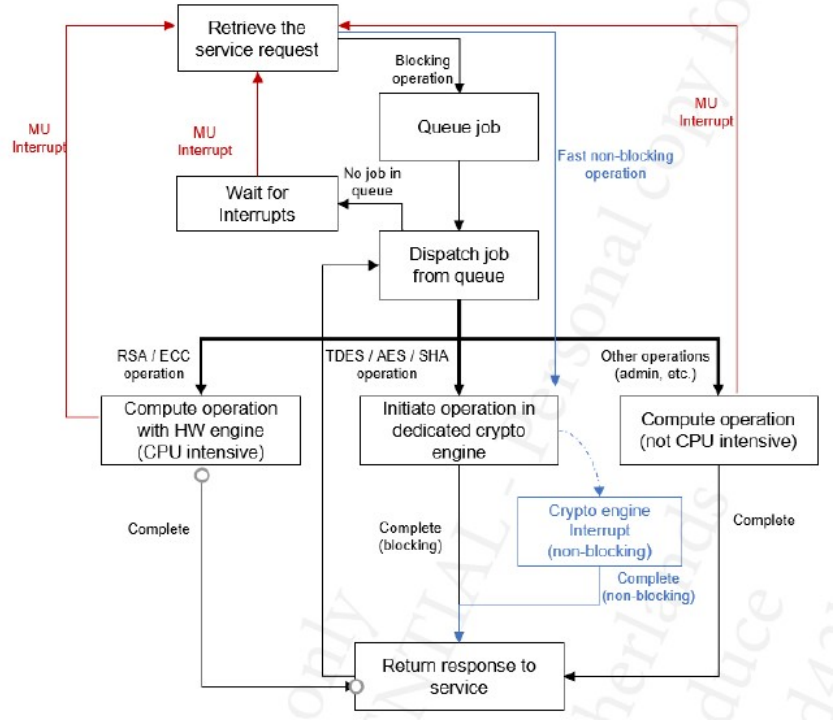
当多个 master 在不同的 MU 实例上触发多个服务请求时，HSE 实现以下服务执行顺序：

- HSE 按顺序解析每个 MU 实例，从接收到的 MU 实例开始 第一个服务请求，以循环方式（当到达最后一个 MU 实例时，HSE 回到 MU0)，直到所有服务请求都得到服务。
- 在一个 MU 实例中，HSE 搜索下一个要处理的服务请求，从 以循环方式服务到最后一个通道（当到达最后一个服务通道时，HSE 回到服务频道 #0）。
- 如果找到服务请求，则将其推送到作业队列； 然后 HSE 移动到下一个 MU 实例。
- 如果没有发现服务请求，HSE 移动到下一个 MU 实例。
- AES、MAC 或 SHA1/SHA2 快速操作会中断 ECC 或 RSA 这样的繁重操作。 HSE 为 ECC 分配 200 微秒的执行周期（在 400MHz HSE 核心频率下）。或对于 RSA，AES、MAC 和 SHA1/SHA2 操作分配为 1000 微秒。 这样，快速 操作不会完全饿死繁重的操作。

注意：非硬件加速加密操作被认为是繁重的 操作，并且可以被快速操作打断。

执行方案确保所有 MU 实例和所有服务通道得到平等服务，避免 一个 MU 实例优先于其他所有其他 MU 实例。 下图说明了 HSE 如何处理服务和工作。

Figure 20: Illustrating service execution by the HSE



基于加密密钥的加密操作（例如 AES）或哈希操作由专用加密引擎加速。 一旦使用要执行的操作进行初始化，它们就独立于 HSE 固件运行。完成后，将提供中断以将服务响应返回给主机。

CPU 密集型 RSA / ECC 操作（尽管使用硬件加速）可以被其他中断 服务请求。 这意味着其他加密操作（例如 AES）可以与那些 耗时的操作交织执行。

2.4.2 执行权（超级用户与用户）

2.4.2.1 定义

某些 HSE 服务的执行以授予主机的执行权为条件。 可以授予主机两个级别的执行权限：

- 超级用户 (SU) 权限
- 用户权限

下表列出了所选 HSE 服务中执行权限的差异。

表：密钥管理中的执行权限和各自的限制

服务	超级用户 (SU) 权限	用户权限
导入新的 NVM 密钥（即在空密钥槽中）加密/认证	可选	强制

NVM 密钥生成（即在空密钥槽中）	可能	不可能
NVM 密钥删除	可能	不可能
将部分 RAM 密钥复制到 NVM 密钥槽	可能	不可能
加载用户定义的 ECC 曲线	可能	不可能

表：HSE 配置中的执行权限和相应限制

服务	超级用户 (SU) 权限	用户权限
设置 HSE 系统属性	可能	不可能（除了 SETONCE- ATTR 属性类型）
验证主机系统映像（IVT、CFG）	可能	不可能
OTFAD 安装	可能	不可能
完成 SMR 条目更新（包括密钥句柄）	可能	不可能
更新核心重置条目 CR	可能	不可能
单调计数器配置	可能	不可能

此外，根据生命周期状态和授权条件，主机被赋予一个身份（HID） 执行权，如下表所述。

表：确定主机身份

执行权	授予条件	主机身份
SU	Reset 后	取决于 LC 值
User	Reset 后	取决于 LC 值。之后可以无条件授予 运行时 HSE 授权请求（具有用户权限）
SU	认证后	取决于授权密钥句柄的密钥组所有者

表：主机身份与 LC

LC	**Host identity (HID)**
CUST_DEL	System integrator (CUST)
OEM_PROD	OEM
IN_FIELD	Not identified (ANY)

FA	Not identified (ANY)

表：主机身份与密钥组所有者

Key group owner	**Host identity (HID)**
HSE_KEY_OWNER_CUST	System integrator (CUST)
HSE_KEY_OWNER_OEM	OEM
HSE_KEY_OWNER_ANY	Not identified (ANY)

主机可以使用 HSE 状态标志检查当前的执行权限和主机身份：HSE_STATUS_CUST_SUPER_USER 和 HSE_STATUS_OEM_SUPER_USER。

表：检查执行权限和主机身份

HSE_STATUS_ CUST_SUPER_USER	HSE_STATUS _OEM_SUPER_USER	执行权	**Host identity (HID)**
1	0	Su	CUST
0	1	SU	OEM
0	0	User	ANY
1	1	SU	ANY (key catalogs are not formatted, no key is installed)

2.4.2.2 复位后的执行权限

HSE 固件初始化和配置与 ECU 制造步骤保持一致，并且正在采取 在生命周期状态为 CUST_DEL 或 OEM_PROD 时放置。

表：LC 状态与 ECU 制造步骤

LC 状态	制造步骤	配置所有者
CUST_DEL	ECU 组装和初始配置	系统集成商（NXP 的客户）
OEM_PROD	ECU 车辆集成和最终配置	OEM

在这些 LC 状态下，主机在重置后被授予超级用户 (SU) 权限，这提供了配置 拥有者高执行权限且对服务请求没有限制。 在其他 LC 状态下，主机被授予重置后具有用户权限，这提供了最受限制的执行权限并限制了某些服务。 LC 状态 CUST_DEL 和 OEM_PROD 中复位后的执行权限可以强制为用户权限 在 hseAttrExtendedCustSecurityPolicy_t 中配置“Start As User”选项和 hseAttrExtendedCustSecurityPolicy 属性

表：复位后的执行权限

LC 状态		HSE 系统属性	Reset 后的主机执行权	**Host identity (HID)**
CUST_DEL		CUST_START_AS_USER == 0	SU	系统集成商（NXP 的客户）
CUST_DEL		CUST_START_AS_USER == 1	User	系统集成商（NXP 的客户）
OEM_DEL		CUST_START_AS_USER == 0	SU	OEM
OEM_DEL		CUST_START_AS_USER == 1	User	OEM
IN_FIELD		N/A	User	Not identified (ANY)
FA		N/A	User	Not identified (ANY)

2.4.2.3 申请 SU 权限

主机可以通过特定的管理服务临时请求 HSE 获得超级用户 (SU) 权限。 这 请求必须按如下方式进行身份验证：

- 主机触发请求，HSE 返回随机挑战
- 主机然后使用选定的挑战计算加密响应 身份验证密钥并将其提供给 HSE
- 如果提供的响应是预期的，则 HSE 授予主机 SU 权限

SU 权限被授予直到下一次重置，或直到主机请求用户权限； 这样的要求不需要 任何身份验证。

SU 权限提供给用于身份验证的密钥的所有者。 这与默认的 SU 不同 复位后根据 LC 状态授予的权限。

2.4.3 单程执行模式

所有加密服务均以一次性模式执行：在收到服务请求后，HSE 获得 服务参数，读取数据进行处理，最终将结果写入内存并返回 服务结果到主机。对于某些可以在流模式下运行的服务（见下一节），数据字段 accessMode 必须是 设置为 HSE_ACCESS_MODE_ONE_PASS 以强制以一次性模式执行。

2.4.4 流式执行模式

对于涉及处理大数据或流的某些加密服务，主机具有 以流模式运行操作的可能性，即在多个服务调用中而不是一个。 这种灵活性 例如，当 HSE 处理的数据在主机中尚未完全可用时，使用可能会有所帮助 记忆。

流式操作模式分为三个步骤：

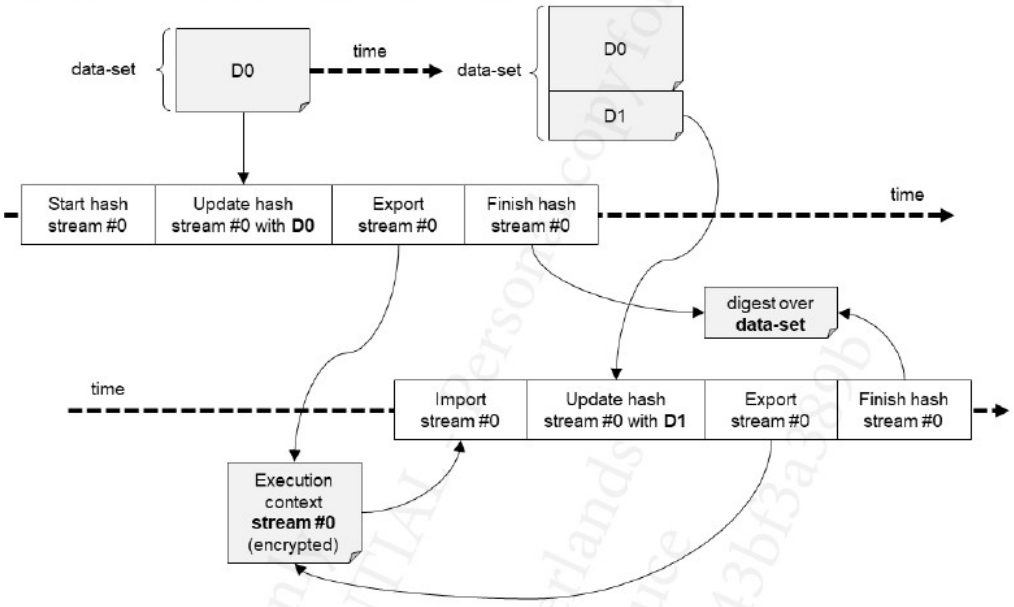
- 初始步骤（“START”）：这是主机对服务进行初始化的第一个服务调用； 只有一个 可以执行 START 服务调用
- 更新步骤（“UPDATE”）：这是来自主机的后续服务调用，用于向 HSE 提供 要处理的“下一个”数据； 可以执行多个更新步骤
- 最后一步（“FINISH”）：这是主机为获取操作结果而进行的最后一次服务调用；只能执行一次 FINISH 服务调用

服务描述符中的数据字段 accessMode 指定 HSE 处理的步骤：

- accessMode = HSE_ACCESS_MODE_START 对于初始步骤
- accessMode = HSE_ACCESS_MODE_UPDATE 用于更新步骤
- accessMode = HSE_ACCESS_MODE_FINISH 用于最后一步

数据字段 streamId 是将每个步骤（即调用）链接在一起的流标识符。 流标识符由主机选择并关联到 MU 实例。 它的值必须介于 0 和 (HSE_STREAM_COUNT – 1)，因此主机最多可以启动 HSE_STREAM_COUNT 每个 MU 实例的流。 一旦在初始步骤 (START) 设置了服务 ID 并将其关联到流标识符，所有后续调用（UPDATE 和 FINISH）必须为该流标识符引用相同的服务 ID。 使用不同的服务 ID 产生服务错误。 START 和 FINISH 步骤必须是唯一的。 如果在 FINISH 之前调用了 START，则对应的流 标识符被重置，之前的执行上下文丢失。 流启动后，其执行上下文可以安全地导出到 HSE 之外 在稍后阶段重新导入。例如，此功能允许根据数据计算摘要或 MAC 随时间顺序增长的集合：不是每次都重新计算整个数据集，而是可以 使用添加到集合中的数据更新计算，从而减少计算时间。 这就是 下图说明

Figure 21: Illustrating streaming contexts import / export



流执行上下文导入和导出服务是记录在案的 HSE 管理服务的一部分。 请注意，在流模式中使用的服务频道没有任何限制：它不必 对于每个呼叫都是相同的。下面的源代码说明了 AES CMAC 如何在流模式下执行。缓冲区 [] 数组是 用作数据输入两次（在 START 和 UPDATE 调用中）。生成的 CMAC 存储在 mac[] FINISH 调用中的数组

```
void test_aes_cmac_in_streaming(void)
{
hseSrvDescriptor_t* pHseSrvDesc;
hseMacSrv_t* pMacSrv;
hseSrvResponse_t srvResponse;
uint8_t buffer[64];
uint8_t mac[16];
uint32_t macLen = sizeof(mac);
uint8_t ch;
uint8_t MU = 0;
uint8_t ID = 0;
…/… (e.g. pointer initializations)
// initialization of the MAC service request
pHseSrvDesc->srvId = HSE_SRV_ID_MAC;
pMacSrv = &(pHseSrvDesc->hseSrv.macReq);
macScheme.macAlgo = HSE_MAC_ALGO_CMAC;
```

```
macScheme.sch.cmac.cipherAlgo = HSE_CIPHER_ALGO_AES;
pMacSrv->macScheme.macAlgo = HSE_MAC_ALGO_CMAC;
pMacSrv->macScheme.sch.cmac.cipherAlgo = HSE_CIPHER_ALGO_AES;
pMacSrv->authDir = HSE_AUTH_DIR_GENERATE;
pMacSrv->keyHandle = GET_KEY_HANDLE(HSE_KEY_CATALOG_ID_NVM, 2, 1); // 2nd key slot in 3rd key group
pMacSrv->inputLength = sizeof(buffer);
pMacSrv->pInput = buffer;
    pMacSrv->sgtOption = 0;
// streaming mode: initial step
pMacSrv->streamId = ID;
pMacSrv->accessMode = HSE_ACCESS_MODE_START;
srvResponse = runSrv(pHseSrvDesc);
if(HSE_SRV_RSP_OK != srvResponse) goto error;
…/… (e.g. waiting for further data to be available in array buffer)
// streaming mode: update step
pMacSrv->streamId = ID;
pMacSrv->accessMode = HSE_ACCESS_MODE_UPDATE;
srvResponse = runSrv(pHseSrvDesc);
if(HSE_SRV_RSP_OK != srvResponse) goto error;
…/… (e.g. waiting for further data to be available in array buffer)
// streaming mode: final step
// retrieves the authentication tag (AES-CMAC)
pMacSrv->inputLength = 0;
pMacSrv->pTagLength = &macLen;
pMacSrv->pTag = mac;
pMacSrv->streamId = ID;
pMacSrv->accessMode = HSE_ACCESS_MODE_FINISH;
srvResponse = runSrv(pHseSrvDesc);
if(HSE_SRV_RSP_OK != srvResponse) goto error;
error:
…/…(e.g. analysing the response)
return;
    }
```

2.4.5 取消服务请求

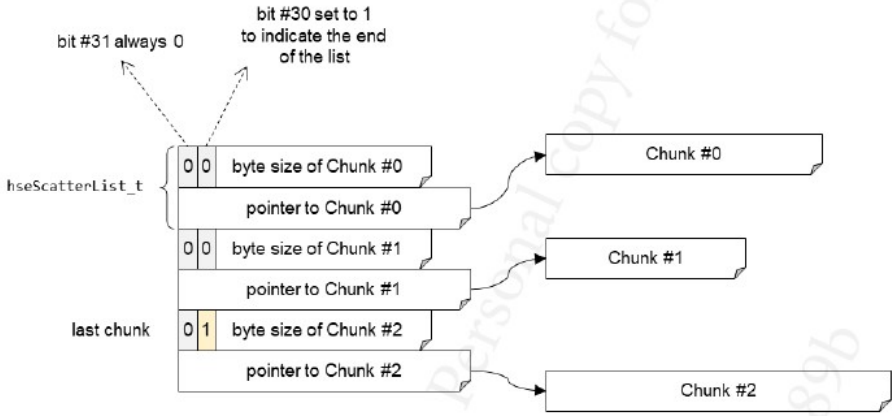
只有在以下情况下，HSE 接受的服务请求才能取消：

- ID=0x00A5XXXX 的服务不可取消
- 其相关作业仍在待处理作业队列中（即 HSE 尚未开始处理）

2.4.6 分散/聚集输入和输出

HSE 服务的输入/输出数据作为指向字节数组的指针提供。 此外，HSE 支持，对于特定的加密服务，提供指向分散/聚集列表的指针的可能性 指向字节数组的指针。 这允许在单个服务请求中处理分布式数据（即 “分散”）跨越内存中的多个字节数组（或块）。 HSE 支持的分散/聚集列表格式是 hseScatterList_t 类型的数组，如 下图。

Figure 22: Illustrating a scatter/gather list



第一个数据字段长度是一个 32 位字，它提供：

- 一个控制位（位#30），当设置为 1 时指示分散/聚集列表的结尾； 无论如何 一个列表中的最大块数不能超过 HSE_MAX_NUM_OF_SGT_ENTRIES
- 第二个数据字段 pPtr 指向的块的字节大小（从位#0 到位#29 编码）

支持分散/聚集输入和/或输出的 HSE 服务具有类型为 sgtOption 的数据字段 hseSGTOption_t。 此字段指示可用时 pInput 和 pOutput 指向的数据类型：

- 当 pInput 和 pOutput 都指向字节数组时，sgtOption 必须设置为 HSE_SGT_OPTION_NONE (0)
- 当 pInput 指向分散/聚集列表时，sgtOption 必须设置为 HSE_SGT_OPTION_INPUT
- 当 pInput 和 pOutput 都指向分散/聚集列表时，sgtOption 必须设置为 HSE_SGT_OPTION_INPUT | HSE_SGT_OPTION_OUTPUT 下表总结了所有可能的组合。

表：分散/聚集选项与输入/输出参数类型

pInput points to…	**pOutput points to…**	**sgtOption set to…**
Byte array	Byte array or not provided	HSE_SGT_OPTION_NONE (0)
Scatter/gather list	Byte array or not provided	HSE_SGT_OPTION_INPUT
Byte array	Scatter/gather list	HSE_SGT_OPTION_OUTPUT
Scatter/gather list	Scatter/gather list	HSE_SGT_OPTION_INPUT HSE_SGT_OPTION_OUTPUT

此外，服务请求中指定的相应输入和输出大小必须占总数 要处理的字节数。 这意味着如果 inputLength 是散点图/聚集列表的输入大小，那么它 必须等于列表中提供的所有字节大小的总和。

2.5 Crypto 驱动 AES 示例使用到的服务

本程序涉及到以下服务：

```
//                          HSE Service IDs
/*  HSE Service IDs of type hseSrvId_t. It's a concatenation of 4 bytes:
   "Service Version | Cancellable | Class | ID" (refer to hseSrvDescriptor_t) */
/*-------- Service class 0x00: administrative services ----------*/
   #define HSE_SRV_ID_SET_ATTR              ((hseSrvId_t)(HSE_SRV_VER_0 | 0x00000001UL))  /**< @brief Set HSE attribute. */
   #define HSE_SRV_ID_GET_ATTR              ((hseSrvId_t)(HSE_SRV_VER_0 | 0x00A50002UL))  /**< @brief Get HSE attribute. */
#define HSE_SRV_ID_FORMAT_KEY_CATALOGS      ((hseSrvId_t)(HSE_SRV_VER_0 | 0x00000101UL))  /**< @brief Format key catalogs (NVM or RAM). */
   #define HSE_SRV_ID_SHE_LOAD_PLAIN_KEY    ((hseSrvId_t)(HSE_SRV_VER_0 | 0x0000A102UL))  /**< @brief Load the SHE RAM key as plain text */
#define HSE_SRV_ID_SYM_CIPHER               ((hseSrvId_t)(HSE_SRV_VER_0 | 0x00A50203UL))  /**< @brief Symmetric encryption/decryption */
```

Table 16: Service ID format

Bit field	31 ~ 24	23 ~ 16	15 ~ 8	7 ~ 0
Description	Service Version	Service Type	Service class	Service subclass

服务版本	服务类型	描述: 服务类 ID	描述:服务子类
0x00	0x00 0xA5	HSE 固件管理服务：0x00	设置系统属性：0x01 读取系统属性：0x02
0x00	0x00	加密密钥管理： 0x01	格式化密钥目录：0x01
0x00	0xA5	加密功能：0x02	加密：0x03
0x00	0x00	SHE：0xA1	加载密钥，加载普通密钥：0x02

3 环境搭建

3.1 安装与编译

安装 HSW FW 和 RTD 3.0.2 后，MCAL Crypto 驱动有以下三个示例：

C:\NXP\SW32G_RTD_4.4_3.0.2\eclipse\plugins\Crypto_TS_T40D11M30I2R0\examples\EBT

- Crypto_CmacCtr_S32G274A_M7

  Readme.txt:

  Example Description

  The example application showcases the usage of the 'Generate/Verify CMAC with counter' feature offered by HSE Firmware.

- Crypto_SymmetricPrimitives_S32G274A_M7

  Readme.txt:

  Example Description

  The example application showcases the usage of 3 features offered by the Crypto driver:

  - AES128 ECB Encryption and Decryption

  - CMAC Generate/Verify

  - HASH

- Hse_Ip_AesEncAsyncIrq_S32G274A_M7

  Readme.txt:

  Example Description

Checks that HSE FW is up and running, formats the key catalogs, imports an AES key and sends synchronous and asynchronous encryption requests to HSE on one MU instance

本文以 Hse_Ip_AesEncAsyncIrq_S32G274A_M7 为例来说明 Mcal Crypto 驱动的使用，其它两个可以参考自行分析。

参考文档 Hse_Ip_AesEncAsyncIrq_S32G274A_M7/readme.txt 来搭建环境：

打开EB Tresos 27.1.0，File->Import…->General->Existing Projects into Workspace:->Next Select root directory->Browse…-> C:\NXP\SW32G_RTD_4.4_3.0.2\eclipse\plugins\Crypto_TS_T40D11M30I2R0\examples\EBT\Hse_Ip_AesEncAsyncIrq_S32G274A_M7

从而打开工程Hse_Ip_AesEncAsyncIrq_S32G274A_M7。(勾选 Copy projects into workspace)。

在工程Hse_Ip_AesEncAsyncIrq_S32G274A_M7右击，选择Properties->Resource->Linked Rsources: PROJECT_LOC:C:\EB\tresos\workspace\ Hse_Ip_AesEncAsyncIrq_S32G274A_M7。此为工程所在目录。

Configuration Porject->Code Generation: Default Generation Path= ..\..\generate。此为生成代码的地方。

所以：

在工程Hse_Ip_AesEncAsyncIrq_S32G274A_M7右击，选择generate project，使用EB生成代码后，将C:\EB\tresos\generate*(注意清空之前的代码)拷贝到：C:\NXP\SW32G_RTD_4.4_3.0.2\eclipse\plugins\Crypto_TS_T40D11M30I2R0\examples\EBT\Hse_Ip_AesEncAsyncIrq_S32G274A_M7\generate。

修改编译相关文件：C:\NXP\SW32G_RTD_4.4_3.0.2\eclipse\plugins\Crypto_TS_T40D11M30I2R0\examples\EBT\Hse_Ip_AesEncAsyncIrq_S32G274A_M7\project_parameters.mk：

```
GCC_DIR = C:/NXP/S32DS.3.4/S32DS/build_tools/gcc_v9.2/gcc-9.2-arm32-eabi #编译器位置
TRESOS_DIR = C:/EB/tresos  #指向EB安装 目录
PLUGINS_DIR = ../../../../ #RTD plugins安装目录
ADDITIONAL_INCLUDE_DIRS = C:/NXP/HSE_FW_S32G274_0_1_0_1/interface \
                          C:/NXP/HSE_FW_S32G2_0_1_0_1/interface/config \
                          C:/NXP/HSE_FW_S32G2_0_1_0_1/interface/inc_common \
                          C:/NXP/HSE_FW_S32G2_0_1_0_1/interface/inc_custom \
                          C:/NXP/HSE_FW_S32G2_0_1_0_1/interface/inc_services
```

Cygwin中编译Hse_Ip_AesEncAsyncIrq_S32G274A_M7的命令：

```
cd C:
NXP/SW32G_RTD_4.4_3.0.2/eclipse/plugins/Crypto_TS_T40D11M30I2R0/examples/EBT/Hse_Ip_AesEncAsyncIrq_S32G274A_M7/
$ make clean
$ make build
```

输出镜像为：

```
-o output/main.elf
```

3.2 运行 Demo

注意，Crypto 驱动需要 HSE Core+FW 的支持，所以事实上无法仅使用 lauterbach 直接下载运行，需要 FW 已经正确安装，并且 HSE Core 已经运行起来。这可以先从 QSPI NOR 启动运行 HSE 的 Demo App，本文使用运行 secure bootloader 的办法。

注意一下，由于我们不需要运行 bootloader 加载其它 APP，只需要完成 FW 的安装工作，所以我们将 Bootloader 停止在加载 APP 前，方便调试之后的 Crypto MCAL 驱动镜像：

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

```
int main(void)
{…
#if (BL_ENABLE_SECURE_BOOT == STD_ON)
  Bl_ConfigureSecureBoot();
#endif /* BL_ENABLE_SECURE_BOOT == STD_ON */
  while(1); //add in here to stop the bootloader
if (E_OK != Bl_Run())
```

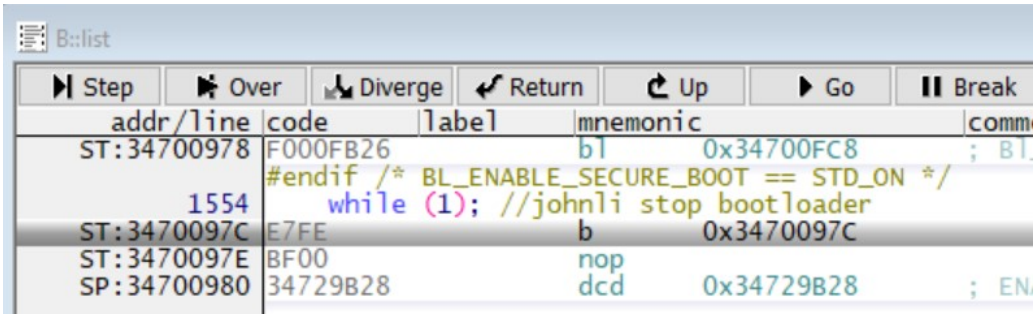
根据文档《AN13750.pdf：Enabling Multicore Application on S32G2 using S32G2 Platform Software Integration》和《S32G_Bootloader_V*.pdf》说明，制造一个带 HSE FW 的 bootloader 启动镜像：bootloader_hse_blob_g2.bin，然后用 flash tools 烧录到 QSPI Nor 中，配置为 QSPI NOR 启动起来。这样，HSE core+FW 就启动起来了，启动后会完成一次 pink FW 到 blue FW 的安装工作，然后重启。

准备 lauterbach 运行脚本：C:\NXP\SW32G_RTD_4.4_3.0.2\eclipse\plugins\Crypto_TS_T40D11M30I2R0\examples\EBT\Hse_Ip_AesEncAsyncIrq_S32G274A_M7\debug\connect_s32gxx_m7.cmm 如下：

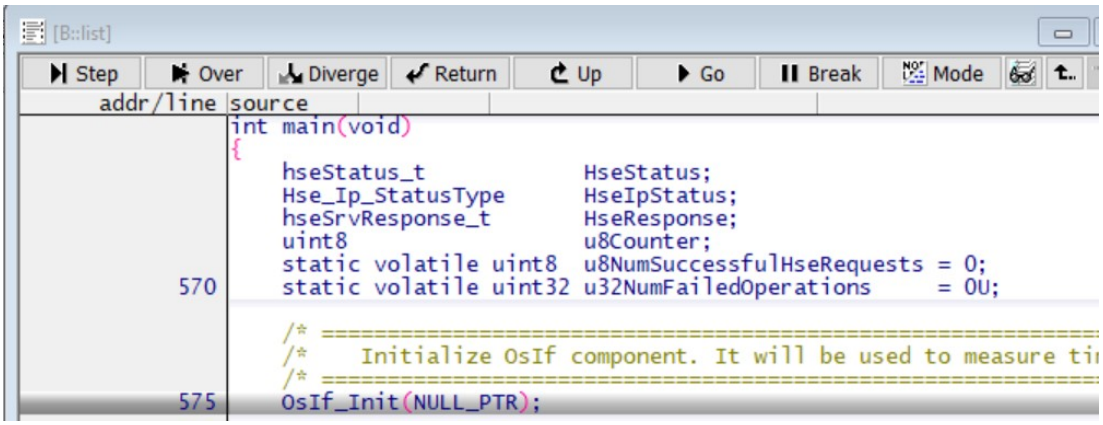
```
sys.cpu S32G-M7_0
SYStem.config.debugporttype JTAG
SYStem.Option TRST OFF
sys.attach
Data.Load.Elf ../out/main.elf
break
list
go main
ENDDO
```

设置为 QSPI NOR 启动 Bootloader，然后启动后用 lauterbach 先运行：

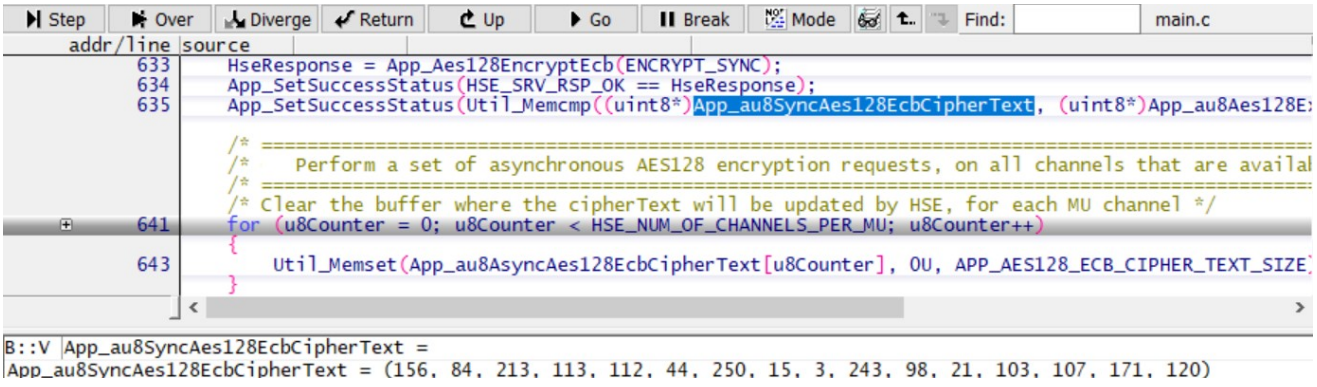
C:\NXP\Integration_Reference_Examples_S32G2_2022_06\code\framework\realtime\swc\bootloader\platforms\S32G2XX\build\cmm\connect_s32gxx_m7.cmm，可以看到 bootloader 停止在 while(1); 处。



然后再直接运行 C:\NXP\SW32G_RTD_4.4_3.0.2\eclipse\plugins\Crypto_TS_T40D11M30I2R0\examples\EBT\Hse_Ip_AesEncAsyncIrq_S32G274A_M7\debug\connect_s32gxx_m7.cmm，就可以加载并运行 Crypto 的 mcal 镜像了，切换模式为源代码模式，窗口如下：



然后运行到：



可以看到加密输出结果与期望结果对比成功。

使用 Openssl 来验证，在 Linux PC 上使用以下命令验证结果：

nxa08200@lsv11049:/opt/samba/nxa08200/test$ echo 000102030405060708090a0b0c0d0e0f | xxd -r -ps > mingwen.bin

nxa08200@lsv11049:/opt/samba/nxa08200/test$ openssl aes-128-ecb -e -in mingwen.bin -out miwen.bin -K 101112131415161718191a1b1c1d1e1f

nxa08200@lsv11049:/opt/samba/nxa08200/test$ xxd miwen.bin

00000000: 9c54 d571 702c fa0f 03f3 6215 676b ab78 .T.qp,....b.gk.x

…

与 HSE 硬件加密结果一样。

4 Crypto 驱动代码与功能说明

本工程使用 EB 配置不多，大部分是使用代码实现，所以直接分析源代码：

C:\NXP\SW32G_RTD_4.4_3.0.2\eclipse\plugins\Crypto_TS_T40D11M30I2R0\examples\EBT\Hse_Ip_AesEncAsyncIrq_S32G274A_M7\src\main.c

int main(void)

* - Initializes the OsIf component. It is used by Hse_Ip layer to determine the timeout while waiting for responses from HSE for synchronous requests

* - Installs the handler for treating OredRx interrupt for MU0 instance

* - Checks the status of the HSE FW by calling the Hse_Ip_GetHseStatus() API and checking the returned value

* - Initializes the Hse Ip layer to work with MU0 instance

* - Sends a synchronous request over MU0 to read the HSE FW capabilities and checks that the RANDOM and SHE capabilities are enabled

|->HseResponse = App_GetHseAttrCapabilities(MU0_INSTANCE_U8, &Hse_AttrCapabilities);

/* Fill the descriptor for the HSE request. Because the request to get an attribute is an administrative one,

it should be sent over channel 0 */

pHseSrvDescriptor = &Hse_aSrvDescriptor[MU_ADMIN_CHANNEL_U8]; ///* Identifier of the MU channel used by HSE to listen for administrative requests (eg. read/write attributes) */ #define MU_ADMIN_CHANNEL_U8 ((uint8)0U)

pHseSrvDescriptor->srvId = HSE_SRV_ID_GET_ATTR; // #define HSE_SRV_ID_GET_ATTR ((hseSrvId_t)(HSE_SRV_VER_0 | 0x00A50002UL)) /**< @brief Get HSE attribute. */

pHseSrvDescriptor->hseSrv.getAttrReq.attrId = HSE_CAPABILITIES_ATTR_ID;

pHseSrvDescriptor->hseSrv.getAttrReq.attrLen = sizeof(hseAttrCapabilities_t);

pHseSrvDescriptor->hseSrv.getAttrReq.pAttr = HSE_PTR_TO_HOST_ADDR(pHseAttrCapabilities);

/* Activate all MU instances. By default, HSE only listens for requests coming on MU0 instance */

|-> HseResponse = App_ActivateAllMuInstances();

```
//如下 enable 所有 MU
/* Variable where application will store the request of configuring (enabling/disabling) the MU instances */
static hseAttrMUConfig_t     Hse_MuConfig;
   /* MU 0 is by default activated and should stay like this */
   Hse_MuConfig.muInstances[0U].muConfig = HSE_MU_ACTIVATED;

   /* Activate the other MU instances */ //#define HSE_NUM_OF_MU_INSTANCES      (4U)     /**< @brief  The maxim number of MU interfaces */
   for(u8Index = 1U; u8Index < HSE_NUM_OF_MU_INSTANCES; u8Index++)
   {
      Hse_MuConfig.muInstances[u8Index].muConfig = HSE_MU_ACTIVATED;
   }

   /* Fill the descriptor for the HSE request. Because the request to set an attribute is an administrative one,
   it should be sent over channel 0. */
   pHseSrvDescriptor = &Hse_aSrvDescriptor[MU_ADMIN_CHANNEL_U8];

   pHseSrvDescriptor->srvId                    = HSE_SRV_ID_SET_ATTR; //#define HSE_SRV_ID_SET_ATTR ((hseSrvId_t)(HSE_SRV_VER_0 | 0x00000001UL))  /**< @brief Set HSE attribute. */
   pHseSrvDescriptor->hseSrv.setAttrReq.attrId  = HSE_MU_CONFIG_ATTR_ID;// #define HSE_MU_CONFIG_ATTR_ID                ((hseAttrId_t)20U)   /**< @brief NVM-RW-ATTR; MU configuration (see #hseAttrMUConfig_t) */
   pHseSrvDescriptor->hseSrv.setAttrReq.attrLen = sizeof(hseAttrMUConfig_t);
   pHseSrvDescriptor->hseSrv.setAttrReq.pAttr   = HSE_PTR_TO_HOST_ADDR(&Hse_MuConfig);

//  Format HSE Nvm and Ram key catalogs
|->HseResponse = App_FormatHseKeyCatalogs();
  /* Create the service request for HSE by setting the descriptor's members */
      pHseSrvDescriptor->srvId                                = HSE_SRV_ID_FORMAT_KEY_CATALOGS; //#define HSE_SRV_ID_FORMAT_KEY_CATALOGS      ((hseSrvId_t)(HSE_SRV_VER_0 | 0x00000101UL))  /**< @brief Format key catalogs (NVM or RAM). */
/*
/* Table containing NVM key catalog entries */
const hseKeyGroupCfgEntry_t Hse_aNvmKeyCatalog[] =
{
   /* NvmKeyGroup_MASTER_ECU_KEY__BOOT_MAC_KEY__Key1_To_Key10 */
```

```
    {(HSE_MU0_MASK), HSE_KEY_OWNER_ANY, HSE_KEY_TYPE_SHE, 12U, 128U, {0U, 0U}},
    /* Marker to end the key catalog */
    {0U, 0U, 0U, 0U, 0U, {0U, 0U}}
};

/* Table containing RAM key catalog entries */
const hseKeyGroupCfgEntry_t Hse_aRamKeyCatalog[] =
{
    /* RamKeyGroup_RamKey */
    {(HSE_MU0_MASK), HSE_KEY_OWNER_ANY, HSE_KEY_TYPE_SHE, 1U, 128U, {0U, 0U}},
    /* Marker to end the key catalog */
    {0U, 0U, 0U, 0U, 0U, {0U, 0U}}
};
*/
        pHseSrvDescriptor->hseSrv.formatKeyCatalogsReq.pNvmKeyCatalogCfg =
HSE_PTR_TO_HOST_ADDR(Hse_aNvmKeyCatalog);
        pHseSrvDescriptor->hseSrv.formatKeyCatalogsReq.pRamKeyCatalogCfg =
HSE_PTR_TO_HOST_ADDR(Hse_aRamKeyCatalog);

//  Import a key in the SHE RAM key slot
|->HseResponse = App_SheLoadPlainRamKey();
  pSheLoadPlainKeyReq =  &(pHseSrvDescriptor->hseSrv.sheLoadPlainKeyReq);
        /* Create the service request for HSE by setting the descriptor's members */
        pHseSrvDescriptor->srvId  = HSE_SRV_ID_SHE_LOAD_PLAIN_KEY; //#define
HSE_SRV_ID_SHE_LOAD_PLAIN_KEY     ((hseSrvId_t)(HSE_SRV_VER_0 | 0x0000A102UL))  /**< @brief Load
the SHE RAM key as plain text */
/* Array storing the value of the key */ //SHE AES 密钥
static uint8 App_au8SheRamKey[APP_SHE_KEY_SIZE] __attribute__((aligned)) =
{
  0x10, 0x11, 0x12, 0x13, 0x14, 0x15, 0x16, 0x17, 0x18, 0x19, 0x1a, 0x1b, 0x1c, 0x1d, 0x1e, 0x1f
};
        pSheLoadPlainKeyReq->pKey = HSE_PTR_TO_HOST_ADDR(App_au8SheRamKey);

//   Perform a synchronous AES128 encryption
//  Clear the buffer where the cipherText will be updated by HSE
```

|->Util_Memset(App_au8SyncAes128EcbCipherText, 0U, APP_AES128_ECB_CIPHER_TEXT_SIZE);\\此为输出 HSE 加密文件数组

```
/* Array storing the value of the cipherText resulted after a synchronous encryption request */
static uint8  App_au8SyncAes128EcbCipherText[APP_AES128_ECB_CIPHER_TEXT_SIZE];
```

|->HseResponse = App_Aes128EncryptEcb(ENCRYPT_SYNC);

```
  /* Create the service request for HSE by setting the descriptor's members */
      pHseSrvDescriptor->srvId    = HSE_SRV_ID_SYM_CIPHER; //#define HSE_SRV_ID_SYM_CIPHER ((hseSrvId_t)(HSE_SRV_VER_0 | 0x00A50203UL))  /**< @brief Symmetric encryption/decryption */
      pHseEncrypt->accessMode     = HSE_ACCESS_MODE_ONE_PASS; /** @brief    HSE access modes.*/
#define HSE_ACCESS_MODE_ONE_PASS    ((hseAccessMode_t)0U)      /**< @brief ONE-PASS access mode */
      pHseEncrypt->sgtOption      = HSE_SGT_OPTION_NONE; //#define HSE_SGT_OPTION_NONE ((hseSGTOption_t)0U)       /**< @brief Scatter list is not used.*/
      pHseEncrypt->cipherAlgo     = HSE_CIPHER_ALGO_AES; #define HSE_CIPHER_ALGO_AES ((hseCipherAlgo_t)0x10U)    /**< @brief AES cipher */ //使用 AES 算法
      pHseEncrypt->cipherDir      = HSE_CIPHER_DIR_ENCRYPT; #define HSE_CIPHER_DIR_ENCRYPT ((hseCipherDir_t)1U)        /**< @brief Encrypt */ //为加密
      pHseEncrypt->cipherBlockMode = HSE_CIPHER_BLOCK_MODE_ECB; #define HSE_CIPHER_BLOCK_MODE_ECB    ((hseCipherBlockMode_t)3U)   /**< @brief ECB mode (AES) */ //使用块加密模式为 ECB
      pHseEncrypt->inputLength    = (uint32)APP_AES128_ECB_PLAIN_TEXT_SIZE;
      pHseEncrypt->pInput         = HSE_PTR_TO_HOST_ADDR(App_au8Aes128EcbPlaintext);//此为输入明文数组：
/* Array storing the value of the plainText to encrypt */
static uint8 App_au8Aes128EcbPlaintext[APP_AES128_ECB_PLAIN_TEXT_SIZE] =
{
  0x00, 0x01, 0x02, 0x03, 0x04, 0x05, 0x06, 0x07, 0x08, 0x09, 0x0a, 0x0b, 0x0c, 0x0d, 0x0e, 0x0f
};
/* Identifier of the HSE key in the key catalog: RAM catalog, group 0, slot 0 */
#define SHE_RAM_KEY_HSE_KEY_HANDLE          (0x00020000U)
      pHseEncrypt->keyHandle      = SHE_RAM_KEY_HSE_KEY_HANDLE;
case ENCRYPT_SYNC:
            pHseEncrypt->pOutput = HSE_PTR_TO_HOST_ADDR(App_au8SyncAes128EcbCipherText);

            /* Build the request to be sent to Hse Ip layer */
            HseIp_aRequest[u8MuChannel].eReqType   = HSE_IP_REQTYPE_SYNC;
```

HseIp_aRequest[u8MuChannel].u32Timeout = TIMEOUT_TICKS_U32;

break;

|-> App_SetSuccessStatus(Util_Memcmp((uint8*)App_au8SyncAes128EcbCipherText, (uint8*)App_au8Aes128ExpectedEcbCiphertext, APP_AES128_ECB_CIPHER_TEXT_SIZE)); //比较加密结果：

/* Array storing the expected cipherText resulted when encrypting the plainText stored in App_au8Aes128EcbPlaintext[] array with

the key stored in App_au8SheRamKey[] array */

static const uint8 App_au8Aes128ExpectedEcbCiphertext[APP_AES128_ECB_CIPHER_TEXT_SIZE] =

{

0x9c, 0x54, 0xd5, 0x71, 0x70, 0x2c, 0xfa, 0x0f, 0x03, 0xf3, 0x62, 0x15, 0x67, 0x6b, 0xab, 0x78

};

之后的异步加密方法与之类似，不同是：

|->App_Aes128EncryptEcb(ENCRYPT_ASYNC_IRQ);

case ENCRYPT_ASYNC_IRQ:

pHseEncrypt->pOutput = HSE_PTR_TO_HOST_ADDR(App_au8AsyncAes128EcbCipherText[u8MuChannel]);

/* Build the request to be sent to Hse Ip layer */

HseIp_aRequest[u8MuChannel].eReqType = HSE_IP_REQTYPE_ASYNC_IRQ;// HSE_IP_REQTYPE_ASYNC_IRQ, /*!< Asynchronous using interrupts - the service request function returns right after sending the request to HSE; an interrupt is triggered when HSE completes the request

(application can be notified through the channel callback) */

HseIp_aRequest[u8MuChannel].pfCallback = Hse_Ip_RxCallback;

break;

| |-> Hse_Ip_RxCallback//此函数比较加密后的密文，如果成功，则加密成功计数器+1

/* Callback for asynchronous requests sent to Hse_Ip layer */

static void Hse_Ip_RxCallback

(

uint8 u8MuInstance,

uint8 u8MuChannel,

hseSrvResponse_t HseResponse,

void* pCallbackParam

)

{

if((HSE_SRV_RSP_OK == HseResponse) &&

```
        (Util_Memcmp((uint8*)App_au8AsyncAes128EcbCipherText[u8MuChannel],
(uint8*)App_au8Aes128ExpectedEcbCiphertext, APP_AES128_ECB_CIPHER_TEXT_SIZE)))
    {
        u8NumSuccessfulHseResponses++;
    }

    /* Clear the buffer where the cipherText will be updated by HSE, in case this channel will be re-used */
    Util_Memset(App_au8AsyncAes128EcbCipherText[u8MuChannel], 0U,
APP_AES128_ECB_CIPHER_TEXT_SIZE);

    /* Ignore compiler warning for not used parameters */
    (void)u8MuInstance;
    (void)pCallbackParam;
}
```

5 定制 1：增加 GetAttribute 功能

考虑更多增加 Get Attribute 功能，根据文档 C:\NXP\HSE_FW_S32G2_0_1_0_1\docs\《HSE_FW_H_S32G2XX_0.1.0.1_HSE_Service_API_Reference_Manual.pdf》说明：

Chapter 3: Administration Services-> 3.2 HSE Set/Get Attribute Services：

Type: hseAttrId_t	
Name	**Value**
HSE_NONE_ATTR_ID	0U
HSE_FW_VERSION_ATTR_ID	1U
HSE_CAPABILITIES_ATTR_ID	2U
HSE_SMR_CORE_BOOT_STATUS_ATTR_ID	3U
HSE_DEBUG_AUTH_MODE_ATTR_ID	10U
HSE_APP_DEBUG_KEY_ATTR_ID	11U
HSE_SECURE_LIFECYCLE_ATTR_ID	12U
HSE_ENABLE_BOOT_AUTH_ATTR_ID	13U
HSE_EXTEND_CUST_SECURITY_POLICY_ATTR_ID	14U
HSE_MU_CONFIG_ATTR_ID	20U
HSE_EXTEND_OEM_SECURITY_POLICY_ATTR_ID	21U
HSE_FAST_CMAC_MIN_TAG_BIT_LEN_ATTR_ID	22U
HSE_CORE_RESET_RELEASE_ATTR_ID	23U
HSE_RAM_PUB_KEY_IMPORT_POLICY_ATTR_ID	24U
HSE_PHYSICAL_TAMPER_ATTR_ID	30U
HSE_MEM_REGIONS_PROTECT_ATTR_ID	31U
HSE_FW_SIZE_ATTR_ID	100U
HSE_AVAIL_ANTI_ROLLBACK_COUNTER_ATTR_ID	101U
HSE_FW_PARTITION_ATTR_ID	102U
HSE_OTFAD_CTX_STATUS_ATTR_ID	103U
HSE_APP_DEBUG_DIS_ATTR_ID	200U
HSE_TEMP_SENSOR_VIO_CONFIG_ATTR_ID	400U

以 HSE_AVAIL_ANTI_ROLLBACK_COUNTER_ATTR_ID 为例，结构体说明如下：

hseAvailAntiRollbackCounter_t

typedef uint32_t hseAvailAntiRollbackCounter_t

Anti-rollback counter updates left.

There are available 158 anti-rollback counter updates (fuses) for the key store and HSE firmware. After 158 updates, the key store and HSE firmware are not protected against rollbacks.

源代码头文件定义如下：hse_fw_s32g2_0_1_0_1\interface\inc_services\Hse_srv_attr.h

```
/*==========================================================================================
                                Anti-rollback counter updates left
==========================================================================================*/

/** @brief    Anti-rollback counter updates left.
 * @details   There are available 158 anti-rollback counter updates (fuses) for the key store and HSE firmware.
 *            After 158 updates, the key store and HSE firmware are not protected against rollbacks.
 */
typedef uint32_t hseAvailAntiRollbackCounter_t;
```

```
#define HSE_AVAIL_ANTI_ROLLBACK_COUNTER_ATTR_ID ((hseAttrId_t)101U) /**< @brief RO-ATTR; The anti-rollback counter updates left (see #hseAvailAntiRollbackCounter_t) */
```

源文件 crypto_ts_*/examples/ebt/hse_ip_aesencasyncirq_s32g274a_m7/src/main.c 中本有获取 ATTR 的示例：

```
    /*
================================================================================================================================== */
    /*    Read HSE FW Capabilities, by sending a request over MU0                                                                   */
    /*
================================================================================================================================== */
    HseResponse = App_GetHseAttrCapabilities(MU0_INSTANCE_U8, &Hse_AttrCapabilities);
    App_SetSuccessStatus(HSE_SRV_RSP_OK == HseResponse);
    App_SetSuccessStatus(0U != (Hse_AttrCapabilities & (HSE_ALGO_CAP_MASK(HSE_CAP_IDX_RANDOM) | HSE_ALGO_CAP_MASK(HSE_CAP_IDX_SHE))));
```

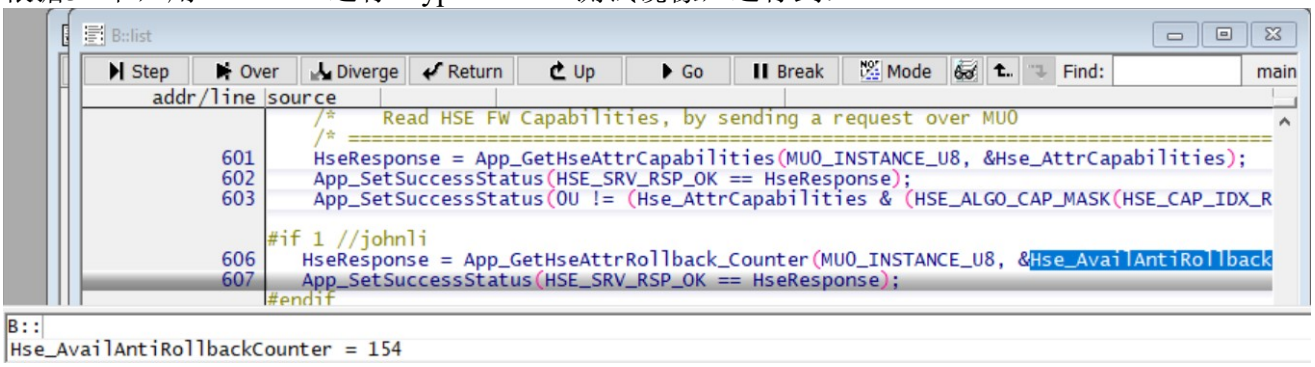
参考增加：

```
static hseAvailAntiRollbackCounter_t Hse_AvailAntiRollbackCounter;
int main(void)
{…
    HseResponse = App_GetHseAttrRollback_Counter(MU0_INSTANCE_U8, &Hse_AvailAntiRollbackCounter);
  App_SetSuccessStatus(HSE_SRV_RSP_OK == HseResponse);
…}
static hseSrvResponse_t App_GetHseAttrRollback_Counter
(
  const uint8          u8MuInstance,
  hseAvailAntiRollbackCounter_t* pHseAttrRollbackCounter
)
{
  hseSrvDescriptor_t* pHseSrvDescriptor;

  /* Fill the descriptor for the HSE request. Because the request to get an attribute is an administrative one,
  it should be sent over channel 0 */
  pHseSrvDescriptor = &Hse_aSrvDescriptor[MU_ADMIN_CHANNEL_U8];

  pHseSrvDescriptor->srvId                    = HSE_SRV_ID_GET_ATTR;
  pHseSrvDescriptor->hseSrv.getAttrReq.attrId  = HSE_AVAIL_ANTI_ROLLBACK_COUNTER_ATTR_ID;
  pHseSrvDescriptor->hseSrv.getAttrReq.attrLen = sizeof(hseSrvDescriptor_t);
  pHseSrvDescriptor->hseSrv.getAttrReq.pAttr   = HSE_PTR_TO_HOST_ADDR(pHseAttrRollbackCounter);

  /* Build the request to be sent to Hse Ip layer */
  HseIp_aRequest[MU_ADMIN_CHANNEL_U8].eReqType   = HSE_IP_REQTYPE_SYNC;
  HseIp_aRequest[MU_ADMIN_CHANNEL_U8].u32Timeout = TIMEOUT_TICKS_U32;

  /* Send the request to Hse Ip layer */
  return Hse_Ip_ServiceRequest(u8MuInstance, MU_ADMIN_CHANNEL_U8,
&HseIp_aRequest[MU_ADMIN_CHANNEL_U8], pHseSrvDescriptor);
```

}

测试方法与结果如下：

根据3.2节，用lauterbach运行Crypto MCAL测试镜像，运行到：



可以读出目前可使用的anti-rollbackCounter的数目，而且 HseResponse=1436920371=0x55A5AA33 //#define HSE_SRV_RSP_OK ((hseSrvResponse_t)0x55A5AA33UL) /**< @brief HSE service successfully executed with no error. */

返回值定义在文件：hse_fw_s32g2_0_1_0_1\interface\hse_srv_responses.h中。

6 CmacCtr Demo 简介

此 Demo 的核心用法是使用 Crypto_KeyElementGet()函数来提交同步 HSE server 完成 CMAC 生成与验证。

先用 Crypto_KeyElementSet()->Crypto_Hse_ImportKey()来注入密钥，然后以生成 CMAC 为例：

```
App_u32CmacGenerateCtrVolatileVal = 0x7654U;
/* Prepare the HSE descriptor */
Hse_SrvDescriptor.srvId = HSE_SRV_ID_CMAC_WITH_COUNTER;// #define HSE_SRV_ID_CMAC_WITH_COUNTER ((hseSrvId_t)(HSE_SRV_VER_0 | 0x00A5020BUL)) /**< @brief CMAC with counter service ID */
Hse_SrvDescriptor.hseSrv.cmacWithCounterReq.authDir = HSE_AUTH_DIR_GENERATE; // #define HSE_AUTH_DIR_GENERATE ((hseAuthDir_t)1U) /**< @brief Generate authentication tag */
Hse_SrvDescriptor.hseSrv.cmacWithCounterReq.keyHandle = APP_CMAC_KEY_HANDLE; // #define HSE_KEY_CATALOG_ID_RAM ((hseKeyCatalogId_t)2U) /**< @brief RAM key catalog */
Hse_SrvDescriptor.hseSrv.cmacWithCounterReq.counterIdx = APP_CMAC_GENERATE_CTR_IDX;
Hse_SrvDescriptor.hseSrv.cmacWithCounterReq.inputBitLength = HSE_BYTES_TO_BITS(APP_CMAC_DATA_SIZE);
// /* Data used to compute the CMAC tag over */static const uint8 App_au8CmacData[APP_CMAC_DATA_SIZE]
Hse_SrvDescriptor.hseSrv.cmacWithCounterReq.pInput = HSE_PTR_TO_HOST_ADDR(App_au8CmacData);
Hse_SrvDescriptor.hseSrv.cmacWithCounterReq.tagBitLength = HSE_BYTES_TO_BITS(APP_CMAC_TAG_SIZE);
```

```
/* Buffer where HSE FW will upload the tag of the CMAC generate operation */static uint8
App_au8CmacGenerateTagBuffer[APP_CMAC_TAG_SIZE];
  Hse_SrvDescriptor.hseSrv.cmacWithCounterReq.pTag           =
HSE_PTR_TO_HOST_ADDR(App_au8CmacGenerateTagBuffer);
  Hse_SrvDescriptor.hseSrv.cmacWithCounterReq.pVolatileCounter =
HSE_PTR_TO_HOST_ADDR(&App_u32CmacGenerateCtrVolatileVal);
  /* Send the HSE descriptor to the firmware for synchronous processing */
  RetVal = Crypto_KeyElementGet(CryptoConf_CryptoKey_Crypto_Key_FEEDDE5C,    /* uint32 keyId        */
                    0xFEEDDE5CU,                    /* uint32  keyElementId   */
                    (uint8*)&Hse_SrvDescriptor,          /* uint8*  resultPtr      */
                    (uint32*)&HseResponse                /* uint32* resultLengthPtr */
|-> Crypto_Ipw_FeedHseDescriptor= Crypto_Hse_FeedHseDescriptor
|   |-> pHseIpReq = &Crypto_Hse_apMuState[u8MuInstance]->Hse_Ip_aRequest[u8MuChannel];

    pHseIpReq->eReqType   = HSE_IP_REQTYPE_SYNC;
    pHseIpReq->u32Timeout = Crypto_Hse_apMuState[u8MuInstance]->u32HseSyncRequestsTimeout;

    /* Send the request to Hse Ip layer */
    HseResponse = Hse_Ip_ServiceRequest(u8MuInstance, u8MuChannel, pHseIpReq,
(hseSrvDescriptor_t*)pu8ResultPtr); //提起 service request
```

7 SymmetricPrimitive Demo 简介

此 Demo 的核心用法是使用 Crypto_ProcessJob()的办法以流式接口来访问 HSE：

```
/* --- Structure of the job to be passed to Crypto driver, requesting Aes128 ECB Encrypt --------- */
static Crypto_JobType App_JobAes128EcbEncrypt =
{…
/*
typedef enum
{
  CRYPTO_OPERATIONMODE_START      = 0x01U, /* Operation Mode is "Start". The job's state shall be reset,
i.e. previous input data and intermediate results shall be deleted. */
  CRYPTO_OPERATIONMODE_UPDATE     = 0x02U, /* Operation Mode is "Update". Used to calculate
intermediate results. */
  CRYPTO_OPERATIONMODE_STREAMSTART = 0x03U, /* Operation Mode is "Stream Start". Mixture of
"Start" and "Update". Used for streaming. */
```

```
    CRYPTO_OPERATIONMODE_FINISH      = 0x04U, /* Operation Mode is "Finish". The calculations shall be finalized */
    CRYPTO_OPERATIONMODE_SINGLECALL  = 0x07U  /* Operation Mode is "Single Call". Mixture of "Start", "Update" and "Finish". */
} Crypto_OperationModeType;
*/
        CRYPTO_OPERATIONMODE_SINGLECALL,   /* mode                  - Indicator of the mode(s)/operation(s) to be performed */
…
};
Crypto_ProcessJob(APP_SYMMETRIC_CDO_ID, &App_JobAes128EcbEncrypt);
|->Crypto_Ipw_ProcessJob=Crypto_Hse_ProcessJob
|   |->Crypto_Hse_ProcessOperation
|   |   |->Hse_Ip_GetFreeChannel
|   |   |->Crypto_Hse_OperationModeControl
      case CRYPTO_OPERATIONMODE_SINGLECALL:
        /** [SWS_Crypto_00020] If Crypto_ProcessJob() is called while in Idle or Active state and
        with the operation mode START, the previous request shall be cancelled.
        That means, that all previously buffered data for this job shall be reset,
        and the job shall switch to Active state and process the new one.*/
        pJob->jobState = CRYPTO_JOBSTATE_ACTIVE;
|   |   |->Crypto_Hse_AllocStream
|   |   |->Crypto_Hse_ServiceRequest
|   |   |   |->Crypto_Hse_EncDec
|   |   |   |   |->Crypto_Hse_SendMsg
  if (CRYPTO_PROCESSING_SYNC == eProcessingType)
  {
    pHseIpReq->eReqType   = HSE_IP_REQTYPE_SYNC;
    pHseIpReq->u32Timeout = Crypto_Hse_apMuState[u8MuInstance]->u32HseSyncRequestsTimeout;
  }
  else
  {
#if   (STD_ON == CRYPTO_USE_INTERRUPTS_FOR_ASYNC_JOBS)
    pHseIpReq->eReqType = HSE_IP_REQTYPE_ASYNC_IRQ;
#else
```

```
        pHseIpReq->eReqType = HSE_IP_REQTYPE_ASYNC_POLL;
#endif /*(STD_ON == CRYPTO_USE_INTERRUPTS_FOR_ASYNC_JOBS)*/

        pHseIpReq->pfCallback     = Crypto_Hse_ProcessMuChannelResponse;
        /* Set the pointer to the job as parameter to the callback */
        pHseIpReq->pCallbackParam = (void*)pJob;
    }

    /* Send the request to HSE driver */
    HseResponse = Hse_Ip_ServiceRequest(u8MuInstance, u8MuChannel, pHseIpReq, pHseSrvDescriptor);
```

8 总结

Crypto MCAL 驱动使用总结如下：

- 驱动的核心文件 crypto_ts_*/src/crypto_hse.c，参考文件 RTD_CRYPTO_UM.pdf：

3.2 Driver Design Summary

The CRYPTO driver supports cryptographic primitives, key storage, key configuration and key management for cryptographic services by accessing the HSE core functionalities. The CRYPTO driver communicates with the HSE via descriptors whose addresses are passed via a messaging unit to the HSE. To issue any request to the HSE, the host must first fill out a descriptor associated with the request then pass the descriptor address to the HSE via one of the MU channels. When the request has been fulfilled, the HSE will write the response in the same MU channel used by the host to trigger the request. The MU is a NXP IP which is present on this platform in 4 instances, each instance having a number of 16 channels.

The CRYPTO module provides the following major features for this release
:...

了解驱动支持的服务。

- MCAL 附带三个 Demo，以不同的操作层次，不同的操作方式来调用 HSE 提供的底层服务 API。
- 从www.autosar.org下载 AUTOSAR_SWS_CryptoDriver.pdf，了解 autosar 对 Crypto MCAL 驱动的要求。

9 其它注意事项

注意以下errata：

In our S32G2 errata, we explain an item as follows:

ERR051655: OCOTP: Incorrect data may be read from fuses after software power down and up of the fusebox

Description

If the system fusebox is powered down and then up by software by setting and clearing CTRL_FBX[FBX_PD] in the OCOTP, subsequent direct fuse word reads may contain incorrect data. A bit may also fail to program without flagging an error.

Reads from the shadow registers are not impacted and will maintain correct data.

In addition to the system fusebox older versions of the HSE FW which utilize the HSE fusebox do not comply with this erratum.

Workaround

If the system fusebox is powered down by writing a one to CTRL_FBX[FBX_PD], a destructive reset or a standby exit must be issued before fuse words are read or programmed.

Additionally, the HSE FW release notes should be reviewed to ensure the version in use complies with this erratum.

And 2 items should be take attention:

1: “A bit may also fail to program without flagging an error”

2: “In addition to the system fusebox older versions of the HSE FW which utilize the HSE fusebox do not comply with this erratum.”, which means the former FW do not power up/down fuse box, but the version

0.1.0.0/0.1.0.1 you used have this power up/down. Checked with internal, from 0.1.0.9, power up/down be removed again.

So suggest you:

1: for your current production, use the FW before 0.1.0.9, pls use destructive reset or standby between you publish sys-img or changed HSE OTP attributes.

2: for your future develop project, pls update the FW to 0.1.0.9.

3: G3 FW fix version is 0.2.22.0.