

TDA4 HS Prime 密钥烧录以及 vHSM 的集成

王力 (Neo Wang)

Central FAE

摘要

Jacinto™ 7 TDA4 系列处理器是 TI 公司基于 Keystone 架构推出的最新一代汽车处理器，主要致力于辅助驾驶系统 (ADAS) 和自动驾驶 (AD) 领域芯片解决方案。TDA4 系列汽车处理器基于片上多核异构芯片架构，16nm 系统工艺不仅为芯片带来了高系统集成度，而且大幅降低了多功能高级汽车平台设计所需的外设复杂度以及成本。性能上，在提供高达 ASIL-D 的系统功能安全等级支持的同时，以领先于市场的性能/功耗比为深度学习/视觉加速提供了卓越的处理能力。

在车载应用中，出于系统安全以及功能隐私考虑，为了防止应用镜像被恶意篡改、复制以及删除，与日常开发过程中的芯片不同的是，量产产品中的芯片类型一般采用高安全类型芯片。TDA4 系列处理器针对此考虑，也推出了 GP 以及 HS 芯片类型，在 TDA4 HS 芯片中，加入了 OTP 标识 eFuse 以及多个硬件安全加速器，增加了芯片在启动过程中对系统镜像的加解密以及签名验签验证，能够避免系统遭受外围的恶意篡改。更进一步的，随着车辆数据以及对数据信息安全需求的猛增，为了能在 SOC 内部基于既有的硬件 IP 加速器，实现对交互数据进行复杂且实时的加解密操作，TDA4 HS 推出了能够集成例如 vHSM 的第三方 HSM 栈功能代码芯片类型 TDA4 HS Prime，从而能够进一步的加强芯片在实时运行过程中的安全保证。

本文将对 TDA4 HS Prime 芯片进行介绍，并详细描述了从搭建密钥烧录环境、烧写密钥、集成 vHSM、编译签名 TIFS 以及最后的系统验证。能够帮助用户基于 TDA4 HS Prime 芯片，使能并调用芯片内部的安全机制，从而保证整机系统的安全运行。

目录

| | |
|---|-----------|
| 1. 系统介绍 | 3 |
| 1.1. TDA4 HS Prime 芯片..... | 3 |
| 1.2. DMSC 中的 vHSM..... | 4 |
| 2. TDA4 HS Prime 密钥烧写 | 4 |
| 2.1. 安装并编译 keywriter..... | 4 |
| 2.2. 烧写 keys 到 eFuse 中..... | 5 |
| 3. 集成 vHSM 到 TDA4 HS Prime | 6 |
| 3.1. 编译 TIFS + vHSM..... | 7 |
| 3.2. TIFS 签名..... | 8 |
| 3.3. SBL 签名..... | 9 |
| 4. TDA4 HS Prime 运行状态验证 | 10 |
| 4.1. TDA4 HS Prime FS 验证..... | 10 |
| 4.2. TDA4 HS Prime SE 验证..... | 12 |
| 4.3. 整机及 vHSM 运行验证..... | 12 |
| 5. 总结 | 13 |
| 6. 参考文献 | 13 |

图

| | |
|------------------------------------|----|
| 图 1 不同 Security 需求的 TDA4 芯片类型..... | 3 |
| 图 2 DMSC 集成 vHSM 前后架构示意图..... | 4 |
| 图 3 Keywriter 安装目录..... | 5 |
| 图 4 OTP eFuse 烧写流程..... | 6 |
| 图 5 TIFS 源码编译链环境搭建..... | 7 |
| 图 6 TIFS 编译环境搭建..... | 7 |
| 图 7 TDA4 HS Prime 整机启动测试..... | 13 |

表

| | |
|-------------------------------|---|
| 表 1 OTP eFuse 烧录过程中的电压取值..... | 6 |
| 表 2 RTOS_SDK7.1 Patch 说明..... | 8 |

1. 系统介绍

针对不同的算力/配置需求，TDA4 系列芯片提供了不同 IP 配置的芯片种类，例如 TDA4VL，TDA4VM，TDA4VH 等，用以支持不同应用场景以及成本需求下的系统方案。

同时，针对同一芯片种类，以 TDA4VM 为例，又提供了不同 PN 的芯片类型，用以支持不同 security 要求场景下的安全需求。如图 1 所示为同一 TDA4 芯片种类下，根据不同 security 需求支持进行划分的芯片类型。

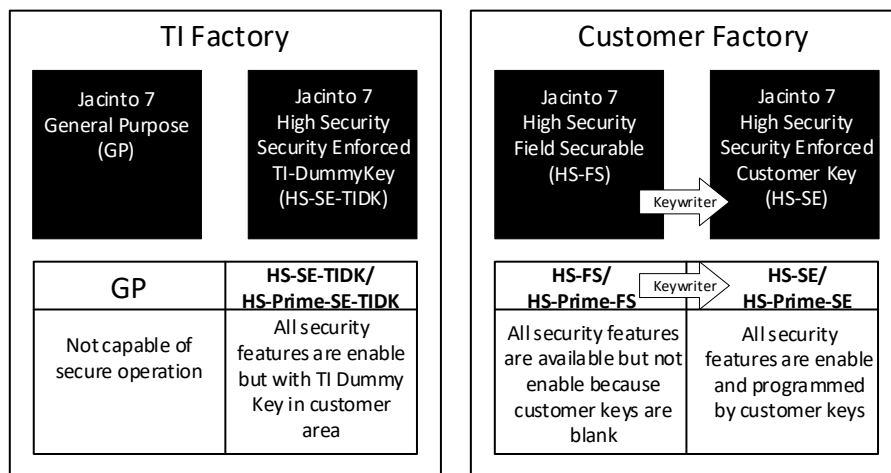


图 1 不同 Security 需求的 TDA4 芯片类型

GP Device: General Purpose 芯片类型。不支持 security 相关操作，常用以日常开发，因为缺乏安全保证，可以但不推荐用于量产系统。

HS-SE-TIDK/HS-Prime-SE-TIDK device: High Security Enforced 芯片类型。支持 security 相关操作，在芯片出厂时 OTP 中已经默认烧录了 TI Dummy Keys，故常用于 security 工作的前期开发以及验证工作，由于 TI Dummy Keys 的私钥随 SDK 公开，故此芯片类型仅推荐用于调试，而非量产。

HS-FS/HS-Prime-FS device: HS-Field Securable 芯片类型。支持 security 相关操作但没有使用，其中用户密钥烧写区域 OTP 为空，用户可自行烧录用户密钥到芯片中，用以支持 security 相关控制。

HS-SE/HS-Prime-SE device: HS-Security Enforced 芯片类型。支持 security 相关操作，其中用户密钥烧写区域 OTP 已经完成了密钥烧录，可用于量产产品中。

上述不同芯片类型中，带/不带 Prime 的区别在于，芯片内部会有 eFuse 标识芯片是否为 Prime 类型，ROM code 会根据读取到 eFuse 的值，来执行对 TIFS 不同的验签流程。功能上体现为是否支持集成 vHSM 栈代码到 TIFS 中。

1.1. TDA4 HS Prime 芯片

通常来说，对于 TDA4 中的 HS Non-Prime 芯片类型，DMSC 中运行的 TIFS 固件是由 TI 的产品密钥进行签名的，而为了安全考虑，此产品密钥是保密且仅由 TI 保留的，故通常 HS Non-Prime 的 TIFS 以二进制文件提供在 SDK 中。通过对执行镜像文件的签名加密，可以防止 HS Non-Prime 芯片中镜像文件被修改，拷贝以及删除。

但是随着车辆和 ECU 网络化和互联化程度的加深，对于车辆数据信息安全的需求在持续增加。为了开发一种安全的车辆 E/E 架构，需要在 ECU 内部实现一套复杂的加密计算过程。TI 为了解决这个问题，在 DMSC 中集成了 AES，RNG，SHA 等安全加速器，用来对启动过程中的加密，验签提供硬件加速，这些硬件 IP 对上层提供的服务可以通过标准的 crypto driver 被调用。

而在当前 ADAS 应用中，许多应用出于功能安全考虑会使用 AUTOSAR 作为 MCU 的 OS，为了支持在 AUTOSAR 在 MCU 中调用 HSM，我们需要一个可以与 AUTOSAR 软件无缝集成并且提供标准接口的固件。Vector

基于自己在 AUTOSAR 基础软件和网络信息安全方面的经验，开发了用以支持 HSM 的固件 vHSM。vHSM 用于控制 HSM 中的硬件加速 IP 核，其模块接口与 AUTOSAR 的 crypto driver 接口一致。

所以为了提供更加完善的安全服务支持，TI 推出了 TDA4 HS Prime 芯片类型，并开源了 TIFS 的源码，用户可以基于 TIFS 源码对 vHSM 的栈代码进行集成，从而实现了对安全加速器的调用。用户可以在运行 AUTOSAR 的 MCU 核心中，使用 SCI message 与 DMSC 进行通信，并通过 vHSM 对硬件安全加速器进行实时调用，例如随机数生成，AES/DES 加密加速，SHA1/SHA2/MD5 哈希值计算以及验签加速等。

1.2. DMSC 中的 vHSM

相对于软件实现加密验签等操作，通过在 DMSC 中集成 vHSM 栈功能，能够直接调用硬件模块进行运算，从而大幅提高效率，如图 2 所示为 DMSC 集成 vHSM 前后的架构变化示意图。

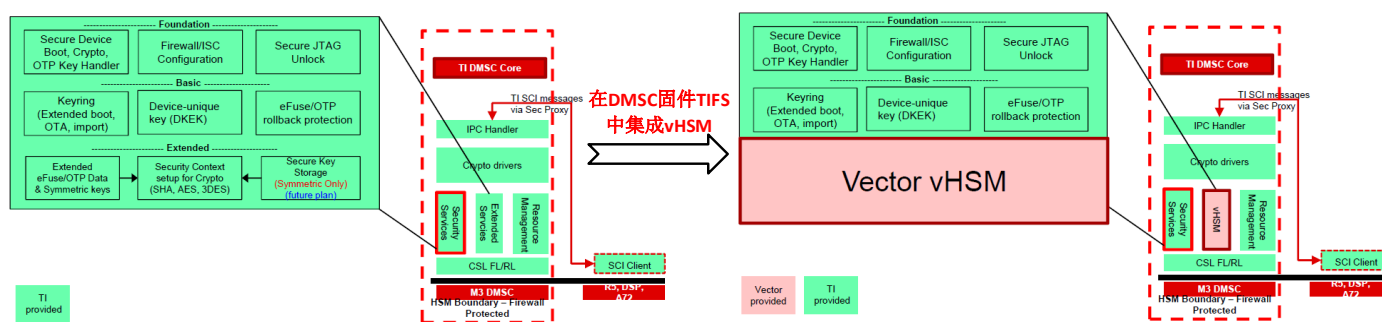


图 2 DMSC 集成 vHSM 前后架构示意图

对于 DMSC 模块，无论是否集成 vHSM，都能够支持安全启动中的 AES 对称密钥验签，PKA/SHA2/AES 的加解密，DKEK 校验以及实时运行过程中的随机数/哈希值/对称密钥产生等。

在 DMSC 中集成 vHSM 等第三方 HSM 栈代码后，除了上述基础功能，不仅能使得 MCU 能够通过标准的 crypto driver 接口对 IP 进行调用，还能够实现例如拓展 OTP 密钥库，使用带有安全密钥的 HWA 以及调用 DKEK 保护外部 flash 中的 HSM 密钥库等功能。

2. TDA4 HS Prime 密钥烧写

Keywriter 是用于将用户密钥烧写到 HS-FS 芯片 eFuse 中，从而将其转换为 HS-SE 芯片的工具软件包，其基于标准 SDK 目录下的 pdk 文件夹进行功能构建。在 Keywriter 软件包中，提供了用于 HS-FS 芯片的 TIFS、标准 256/512bit 密钥生成脚本以及相关源代码等一系列工具，用户可以基于此软件包生成一个二进制文件，作为 TDA4 芯片中的 bootloader 加载程序，此程序将会把指定密钥烧写到 eFuse 中。本章节将基于 Ubuntu 18.04 开发环境，RTOS_SDK7.1 软件版本，以及 TDA4VM HS Prime 芯片硬件平台进行测试，Keywriter 安装包请联系对应 TI 支持窗口进行获取。

值得注意的是，密钥烧写过程为不可逆过程，密钥一旦烧写完成，则无法再次进行密钥修改以及擦除等操作。

值得注意的是，本小节描述的密钥烧写过程并不局限于 TDA4 HS Prime 芯片，对于 TDA4 HS Non Prime 芯片也同样适用。

2.1. 安装并编译 keywriter

一旦 Keywriter 解压安装完成之后，其目录结构如图 3 所示，需要手动将 Keywriter 的安装文件与默认 pdk 目录下的文件进行合并以及更新。

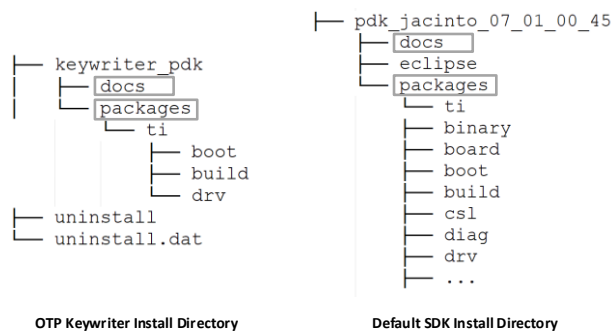


图 3 Keywriter 安装目录

Keywriter 编译过程需要安装 OpenSSL 插件，用以对密钥进行处理，使用下述命令在 Linux OS 中安装 OpenSSL。

```
$ sudo apt-get install openssl
```

在开始编译 Keywriter 烧录工具之前，需要用户拥有自己的密钥用于烧录以及后续的文件签名，同样的，Keywriter 软件包中也提供了密钥生成工具，参照下列流程生成对应密钥。在下列密钥中，bmek.key 以及 bmpk.pem 是可选的备用密钥，其它密钥均为编译过程中必要的，其中若用户拥有自己的密钥，也需要执行下述流程来生成 keys 文件夹以及其它密钥，但需要用自己的密钥替换掉 smpk.pem 即可。

```
$ cd packages/ti/boot/sbl/example/k3MulticoreApp/keywriter/scripts
$ ./gen_keywr_cert.sh -g
$ ls packages/ti/boot/sbl/example/k3MulticoreApp/keywriter/scripts/keys/
aes256.key bmek.key bmpk.pem smek.key smpk.pem tifekeypub.pem
```

密钥准备完成之后，按照下述流程来生成对应的 X509 认证文件。

```
$ cd packages/ti/boot/sbl/example/k3MulticoreApp/keywriter/scripts
$ ./gen_keywr_cert.sh -s keys/smpk.pem --smek keys/smek.key -t keys/tifekeypub.pem -a keys/aes256.key
```

生成的对应 X509 认证文件在 x509cert/final_certificate.bin 目录下，按照下述指令编译 Keywriter 并将生成的 X509 认证文件附加到 Keywriter 文件头上用以解析。

```
$ cd package/ti/build
$ make keywriter_img -j8
```

生成的 Keywriter 烧录工具在 packages/ti/boot/sbl/example/k3MulticoreApp/binary/keywriter_img_j721e_release.bin 目录下。

2.2. 烧写 keys 到 eFuse 中

烧录程序编译完成之后，在正式启动烧录之前，还需要准备硬件以及软件烧写环境。

对于软件烧写环境，除了 2.1 中生成的 Keywriter 文件外，还需要 TIFS 文件来唤醒 TDA4 中的 DMSC，其中 HS FS 芯片的 TIFS 与 GP 和 HS SE 芯片的 TIFS 都不同，其以二进制文件已经提供在了 Keywriter 软件包中。将此两文件拷贝至启动介质并重名，流程如下所示。

```
$ cp packages/ti/boot/sbl/example/k3MulticoreApp/binary/keywriter_img_j721e_release.bin /media/${USER}/BOOT/tiboot3.bin
$ cp packages/ti/drv/sci/client/soc/V1/tifs-hs-enc.bin /media/${USER}/BOOT/tifs.bin
```

对于硬件烧写环境，在上电烧写前，硬件需要对指定 PIN 脚进行控制，大致流程为：

- 在使用软件对 OTP 寄存器进行烧写前，VPP_CORE and VPP_MCU 管脚的供电必须置为低电平。
- 在使用软件对 OTP 寄存器进行烧写时，在系统按照正确的电源轨上电后，VPP_CORE and VPP_MCU 管脚电压必须置为高电平。其电压取值如表 1 所示。

表 1 OTP eFuse 烧录过程中的电压取值

| Parameter | Description | Min | Type | Max | Unit |
|-----------|--|------|------|------|------|
| VPP_CORE | Supply voltage range for the eFuse ROM domain during normal operation | N/A | | | V |
| | Supply voltage range for the eFuse ROM domain during OTP programming | 1.71 | 1.8 | 1.89 | V |
| VPP_MCU | Supply voltage range for the eFuse ROM domain during normal operation | N/A | | | V |
| | Supply voltage range for the eFuse ROM domain during OTP programming | 1.71 | 1.8 | 1.89 | V |
| Tj | Temperature | 0 | 25 | 85 | C° |

- 在使用软件对 OTP 寄存烧写完成后，VPP_CORE and VPP_MCU 管脚的供电必须置为低电平。
- 针对上述软硬件烧写需求以及步骤，其完整的密钥烧写流程如图 4 所示。

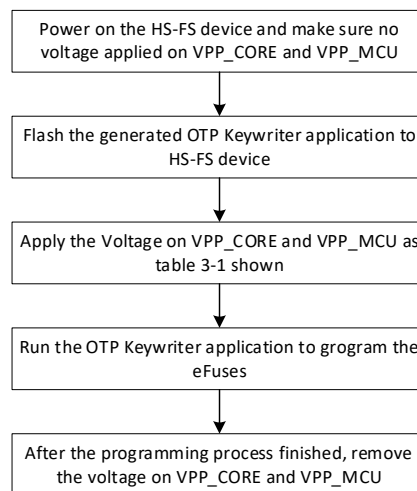


图 4 OTP eFuse 烧写流程

按照图 4 执行烧写流程后，如果看到下述 log 打印，则代表密钥烧写完毕，其验证方法参考第四章节。

```

OTP Keywriter Revision: 01.00.00.00 (Mar 7 2021 - 20:15:01)
OTP Keywriter ver: 20.8.5-w2020.23-am64x-14-g7409e
Beginning key programming sequence
Taking OTP configuration from 0x41c7e000
Debug response: 0x0
Key programming is complete
  
```

3. 集成 vHSM 到 TDA4 HS Prime

如本文第一章所述，TDA4 HS Prime 的 TIFS 需要由用户自行加入第三方的 HSM 代码，所以 TI 公开了 TIFS 的源码并支持用户自行集成，编译以及签名，请联系对应 TI 窗口获取 TIFS 源码。vHSM 为第三方提供的 HSM 固件，vHSM 在 TIFS 中的安装请需求第三方支持。本章会介绍如何在安装了 vHSM 的 TIFS 源码中编译以及生成 TIFS 固件并使用用户 key 签名，最终通过用户签名的 SBL 进行加载，进而驱动整个片上系统。

3.1. 编译 TIFS + vHSM

TIFS 源码安装完成后，用户需要在“tifs_v2020.08e/docs/BUILD.rst”中按照对应版本链接下载并安装编译工具链到 TIFS 源码包中，如图所示，下载完成后需要将 TI CGT ARM Compiler, XDC Tools 以及 SYS/BIOS 的源码安装在 TIFS 一级子目录下。

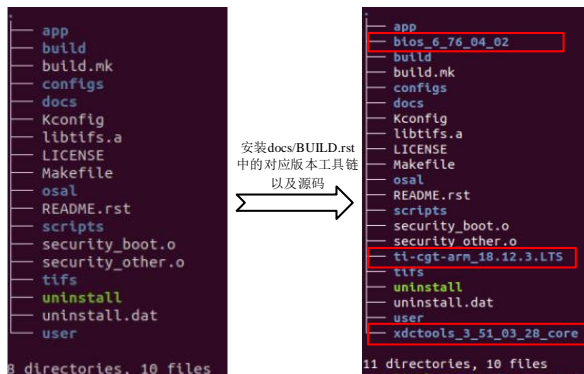


图 5 TIFS 源码编译链环境搭建

安装完对应编译链后，需要在“tifs_v2020.08e/build/build.paths”文件中，将 CGT_ARM_VERSION, BIOS_VERSION, XDC_VERSION 三个宏变量的值修改为与所下工具包版本号一致，如图所示。

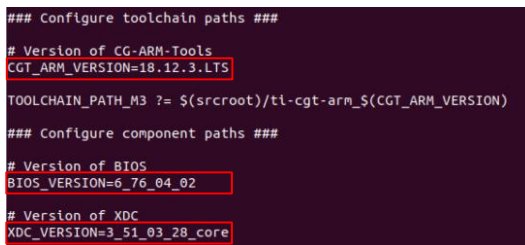


图 6 TIFS 编译环境搭建

至此，TIFS 的编译环境搭建完成，用户可在此基础上集成安装 vHSM 代码。

值得注意的是，不建议用户直接在 TIFS 源码目录下建立 git，因为会因为 git 路径错误而导致后续编译 TIFS 源码时报错找不到对应文件。用户可选择在上级目录中建立自己的 git 仓库。

但庆幸的是，就算报出找不到对应文件，其实并不影响源码的编译以及 TIFS 的生成，忽略即可。

由于 TIFS 源码包含了众多家族 SOC 芯片的 TIFS 源码，所以在编译 TIFS 固件之前，需要设置编译 target 为 TDA4 系列 SOC，然后执行 make 编译命令即可，如下所示。

```
$make j721e_dmsc_defconfig
$make
CHK tifs/include/version.h
GEN tifs/include/config.h
...
CPP app/linkercmd/arm_bin.cmd
BIN ti-fs-firmware-j721e-hs.bin
```

生成的 ti-fs-firmware-j721e-hs.bin 在 TIFS 源码目录下，此文件即为纯净的 TIFS 二进制文件，其需要使用用户的私钥进行签名后，才能在烧录的用户对应公钥的 HS Prime 芯片中被 SBL 加载到 DMSC 并运行。

3.2. TIFS 签名

如第一章中所介绍的，HS Prime 芯片与 HS Non-Prime 芯片使用的是 TDA4 芯片中 ROM code 中不同的执行逻辑，但是 TIFS 作为 DMSC 的固件，都是通过 SBL 进行加载并被 secure ROM 进行验签。对于 HS Non-Prime 芯片来说，其 TIFS 在出厂时已经通过 TI 的产品密钥进行了内部证书的签名，用户在 HS Non-Prime 芯片上的签名其实是对外部证书的签名；但在 HS Prime 芯片中，由于 TIFS 由用户自行生成，所以其内部证书的签名也由用户进行，故需要在默认 SDK 的签名流程基础上用户手动对 TIFS 进行签名，才能在 HS Prime 芯片中运行。

本节基于 SDK7.1 软件版本以及 TDA4VM HS Prime PG1.1 版本硬件进行测试，推荐用户安装[补丁](#)，用以兼容不同版本硬件芯片。在上述 SDK7.1 下载超链接中，对应的 patch 以及其作用如下表所示。所有 patch 打上后，均向下兼容。

表 2 RTOS_SDK7.1 Patch 说明

| POST RELEASE PATCHES | Comments |
|--|---|
| psdk_rtos_j721e_7.1_es11_src.tar.gz | 在 SDK7.1 软件基础上，用以支持 PG1.1 版本芯片，例如 XJ721ERBALF。 |
| psdk_rtos_j721e_7.1_es11_src_patch2.tar.gz | 在 SDK7.1 软件基础上，用以支持 RTM 量产版本芯片，例如 TDA4VM88T5BALF。 |

一般地，在 SDK 中，使用脚本“`packages/ti/drv/sciclient/tools/firmwareHeaderGen.sh`”对 TIFS 进行签名，在默认 SDK 的基础上，需要打上下述补丁，用以支持对 TDA4 HS Prime 芯片 TIFS 的签名。

```
diff --git a/packages/ti/drv/sciclient/tools/firmwareHeaderGen.sh b/packages/ti/drv/sciclient/tools/firmwareHeaderGen.sh
index 963c1ea..70a0aab 100755
--- a/packages/ti/drv/sciclient/tools/firmwareHeaderGen.sh
+++ b/packages/ti/drv/sciclient/tools/firmwareHeaderGen.sh
@@ -187,13 +187,10 @@ $ECHO "Generating the Header file for " $FIRMWARE_SILICON
export SBL_CERT_KEY=$ROOTDIR/ti/build/makerules/rom_degenerateKey.pem
$SBL_CERT_GEN -b $FIRMWARE_SILICON -o $SYSFW_SE_SIGNED -c DMSC_I -I $SYSFW_LOAD_ADDR -k $SBL_CERT_KEY
else
-$ECHO "Generating outer certificate for " $SYSFW_SE_INNER_CERT
+$ECHO "Generating the Header file for " $FIRMWARE_SILICON
export SBL_CERT_KEY=$ROOTDIR/ti/build/makerules/k3_dev_mpk.pem
-$SBL_CERT_GEN -b $SYSFW_SE_INNER_CERT -o $SYSFW_SE_CUST_CERT -c DMSC_O -I $SYSFW_LOAD_ADDR -k $SBL_CERT_KEY
+$SBL_CERT_GEN -b $FIRMWARE_SILICON -o $SYSFW_SE_SIGNED -c DMSC_I -I $SYSFW_LOAD_ADDR -k $SBL_CERT_KEY

-$ECHO "Generating the Header file for " $FIRMWARE_SILICON
-$CAT $SYSFW_SE_CUST_CERT $FIRMWARE_SILICON > $SYSFW_SE_SIGNED
-$RM -f $SYSFW_SE_CUST_CERT
fi
$ECHO "Generating the Header file for the soc in the folder"
```

安装完补丁后，首先需要将用户的私钥拷贝至“`packages/ti/build/makerules`”目录下并重命名为 `k3_dev_mpk.pem`，上述脚本将默认调用此密钥对 TIFS 以及后续 SBL 等镜像进行签名操作。

然后将 3.1 小节中生成的 TIFS 拷贝至 SDK “`drv/sciclient/soc/sysfw/binaries`”目录下并重命名为 `ti-fs-firmware-j721e_sr1_1-hs-enc.bin`，默认签名脚本将会调用上述密钥对其进行签名，最终生成的签名后的 TIFS 在目录“`drv/sciclient/soc/V1/tifs_sr1.1-hs-enc.bin`”下，大致签名流程如下所示。


```

$ cp TIFS-SRC-Release/tifs_v2020.08e/ti-fs-firmware-j721e-hs.bin ../drv/sciclient/soc/sysfw/binaries/ti-fs-firmware-j721e_sr1_1-hs-enc.bin
$ ./firmwareHeaderGen.sh j721e_sr1_1-hs
/home/wangli/ti-processor-sdk-rtos-j721e-evm-07_01_00_11/pdk_jacinto_07_01_00_45/packages/ti/drv/sciclient/tools
Building the bin2c generation c tool
/home/wangli/ti-processor-sdk-rtos-j721e-evm-07_01_00_11/pdk_jacinto_07_01_00_45
Generating the Header file for /home/wangli/ti-processor-sdk-rtos-j721e-evm-
07_01_00_11/pdk_jacinto_07_01_00_45/packages/ti/drv/sciclient/soc/sysfw/binaries/ti-fs-firmware-j721e_sr1_1-hs-enc.bin
~/ti-processor-sdk-rtos-j721e-evm-07_01_00_11/pdk_jacinto_07_01_00_45~/ti-processor-sdk-rtos-j721e-evm-
07_01_00_11/pdk_jacinto_07_01_00_45
DMSC_I Certificate being generated :
    X509_CFG = ./x509-temp.cfg
    KEY = /home/wangli/ti-processor-sdk-rtos-j721e-evm-
07_01_00_11/pdk_jacinto_07_01_00_45/packages/ti/build/makerules/k3_dev_mpk.pem
    BIN = /home/wangli/ti-processor-sdk-rtos-j721e-evm-
07_01_00_11/pdk_jacinto_07_01_00_45/packages/ti/drv/sciclient/soc/sysfw/binaries/ti-fs-firmware-j721e_sr1_1-hs-enc.bin
    CERT TYPE = DMSC_I, 2
    CORE ID = 0
    LOADADDR = 0x00040000
    IMAGE_SIZE = 262112
    BOOT_OPTIONS = 0
SUCCESS: Image /home/wangli/ti-processor-sdk-rtos-j721e-evm-
07_01_00_11/pdk_jacinto_07_01_00_45/packages/ti/drv/sciclient/soc/V1/tifs_sr1.1-hs-enc.bin generated. Good to boot
Generating the Header file for the soc in the folder

    Converting binary file [/home/wangli/ti-processor-sdk-rtos-j721e-evm-
07_01_00_11/pdk_jacinto_07_01_00_45/packages/ti/drv/sciclient/soc/V1/tifs_sr1.1-hs-enc.bin] to C array
    .. Done. (263858 bytes)
    Done.
$ cp ../drv/sciclient/soc/V1/tifs_sr1.1-hs-enc.bin /media/wangli/BOOT/tifs.bin

```

最后将生成并签名后的 TIFS 拷贝至启动设备中并重命名为 tifs.bin。

3.3. SBL 签名

SBL 用于在 MCU1_0 中对系统时钟内存等进行初始化并为 DMSC 加载 TIFS 固件，在 HS Prime SE 芯片中，SBL 需要使用对应密钥进行签名后，才能通过 ROM code 的验签并执行。同样的，默认 SBL 编译指令也会使用 k3_dev_mpk.pem 对 SBL 进行签名。

额外的，在 SBL 中也会对后续的系统镜像，例如 APP image，Linux/QNX image 等进行加载，解析以及验签。用户可通过下述流程生成签名后的在 MMCSd 中运行的 SBL 镜像，然后拷贝至启动设备中并重命名为 tiboot3.bin。

```

$ cp customer_keys.pem packages/ti/build/makerules/k3_dev_mpk.pem
$ make pdk_libs_clean -j8
$ make sciclient_boardcfg BOARD=j721e_evm BUILD_HS=yes -j8
$ make pdk_libs -j8
$ make -j8 BOARD=j721e_evm CORE=mcu1_0 BUILD_PROFILE=release sbl_mmcsd_img_hs
$ cp packages/ti/boot/sbl/binary/j721e_evm_hs/mmcsd/bin/sbl_mmcsd_img_mcu1_0_release.tiimage /media/wangli/BOOT/tiboot3.bin

```

值得注意的是，默认的 SBL HS 编译指令仅会对 SBL 进行签名，而非加密，若要对 SBL 进行加密，请参考另一篇应用手册，此处不进行赘述。

值得注意的是，若用户在编译完 PDK 后，有进行密钥，或者 board config 相关改动，必须按照上述流程进行 clean & build，否则 SBL 运行可能会出错。

4. TDA4 HS Prime 运行状态验证

如本手册 1.1 小节中所述，TDA4 HS Prime 芯片根据密钥的烧写状态，划分为 TDA4 HS Prime FS 以及 TDA4 HS Prime SE。本小节将介绍如何验证不同阶段 TDA4 HS Prime FS 芯片的烧录状态以及公钥 hash 值，用以辅助验证 vHSM 以及整机系统的运行状态。

值得注意的是，此方法可以但不局限于 TDA4 HS Prime 芯片，所有 TDA4 系列不同阶段的芯片，都可使用此方法来检查芯片状态以及密钥烧录情况。

4.1. TDA4 HS Prime FS 验证

一般地，用户可以按照下述流程来获取芯片状态 log 并转换为可读明文。

- 配置板子启动模式为 UART BOOT 并通过串口连接 MCU 域的第二个 UART 串口，然后板子上电，具体操作可参考 [SDK guide](#)。
- 串口将会打印一系列字母数字 log，然后不断打印 CCC，需要手动将 CCC 删除并将之前的 log 另存为文本文件。
- 拷贝下述转换脚本并另存为 python 文件，用以解析步骤 b 中的状态 log。在此之前，Ubuntu 系统需要安装 python3.0 及以上开发环境。

```
#!/usr/bin/env python3
import binascii
import struct
import string
import sys
filename=sys.argv[1]
fp = open(filename, 'rt')
lines= fp.readlines()
fp.close()
bin_arr = [ binascii.unhexlify(x.rstrip()) for x in lines ]
bin_str = b"".join(bin_arr)
pubInfoStr='BB2B12B4B4B4B'
secInfoStr='BBHHH64B64B32B'
numBlocks = list(struct.unpack('l', bin_str[0:4]))
pubROMInfo = struct.unpack(pubInfoStr, bin_str[4:32])
if numBlocks > 1:
    secROMInfo = struct.unpack(secInfoStr, bin_str[32:200])
print ('-----')
print ('SoC ID Header Info:')
print ('-----')
print "NumBlocks      :", numBlocks
print ('-----')
print ('SoC ID Public ROM Info:')
print ('-----')
print "SubBlockId      :", pubROMInfo[0]
print "SubBlockSize   :", pubROMInfo[1]
tmpList = list(pubROMInfo[4:15])
hexList = [hex(i) for i in tmpList]
deviceName = "".join(chr(int(c, 16)) for c in hexList[0:])
print "DeviceName     :", deviceName
tmpList = list(pubROMInfo[16:20])
hexList = [hex(i) for i in tmpList]
deviceType = "".join(chr(int(c, 16)) for c in hexList[0:])
print "DeviceType      :", deviceType
```


通过上述解析出来的明文可以看出，芯片属于 HS FS 以及 Sec Key Revision/Count 都为 0，即此 HS 芯片属于 eFuse 未烧写状态。同样的，Sec Prime 值为 1，表明此芯片类型属于 TDA4 HS Prime 芯片。

4.2. TDA4 HS Prime SE 验证

同样的，通过本手册第二章介绍的步骤对 HS Prime 芯片进行密钥烧录后，其转换为 HS Prime SE 阶段，同样可通过 4.1 小节中方法对芯片状态进行验证，以 TDA4 HS Prime SE 为例，如下所示。

```
$ python uart_parse.py uart_SE.log
-----
SoC ID Header Info:
-----
NumBlocks      : [2]
-----
SoC ID Public ROM Info:
-----
SubBlockId     : 1
SubBlockSize   : 26
DeviceName     : j7es
DeviceType     : HSSE
DMSC ROM Version : [0, 1, 1, 1]
R5 ROM Version  : [0, 1, 1, 1]
-----
SoC ID Secure ROM Info:
-----
Sec SubBlockId : 2
Sec SubBlockSize : 166
Sec Prime      : 1
Sec Key Revision : 1
Sec Key Count  : 1
Sec TI MPK Hash : aa1f8e3095042e5c71ac40ede5b4e8c85fa87e03305ae0ea4f47933e89f4164aeb5a12ae13778f49de0622c1a578e6e747981d
8c44a130f89a336a887a7955ee
Sec Cust MPK Hash : 1f6002b07cd9b0b7c47d9ca8d1aae57b8e8784a12f636b2b760d7d98a18f189760dfd0f23e2b0cb10ec7edc7c6edac3d9bdfc
fe0eddc3fff7fe9ad875195527d
Sec Unique ID   : fb3cf5d9d3aba219f1a8535ece5c8b9ff86f86da624be3eed0a0c35fadfd73e0
```

通过上述解析出来的明文可以看出，芯片属于 HS SE 以及 Sec Key Revision/Count 都为 1，即此 HS 芯片属于 eFuse 已烧写状态。同样的，Sec Prime 值为 1，表明此芯片类型属于 TDA4 HS Prime 芯片。其次，可以得到用户烧写到 eFuse 中公钥的哈希值为 Sec Cust MPK Hash 值，用户可通过 openssl 对用户私钥进行转换并与此 hash 值进行对比，从而判断烧写密钥的类型。

4.3. 整机及 vHSM 运行验证

确认芯片已经烧录成功，并生成对应签名后的 TIFS 以及 SBL 之后，系统即可以成功初始化，为了验证整机系统以及 vHSM 的启动，需要编译对应其它核心的镜像并对其进行签名。本节以 A72 运行 Linux，MCU1_0 运行 CAN APP 等为例，在 HS Prime 芯片上对整个 TDA4 系统启动进行验证，其余镜像的编译请参考 [SDK Guide](#)。

生成对应 binary 之后，需要进行手动签名才能在 HS Prime SE 芯片中运行，进行签名的脚本在 SDK 中默认提供，用户需要调用用户密钥对 binary 进行签名，大致流程如下所示。

```
packages/ti/build/makerules/x509CertificateGen.sh -b input_app_binary -o output_app_binary.signed -c R5 -l 0x0 -k customer_smpk.pem -d
DEBUG -j DBG_FULL_ENABLE -m SPLIT_MODE -s 1
```

通过 SD 卡进行测试，TDA4 HS Prime SE 整机启动 log 如下图所示。

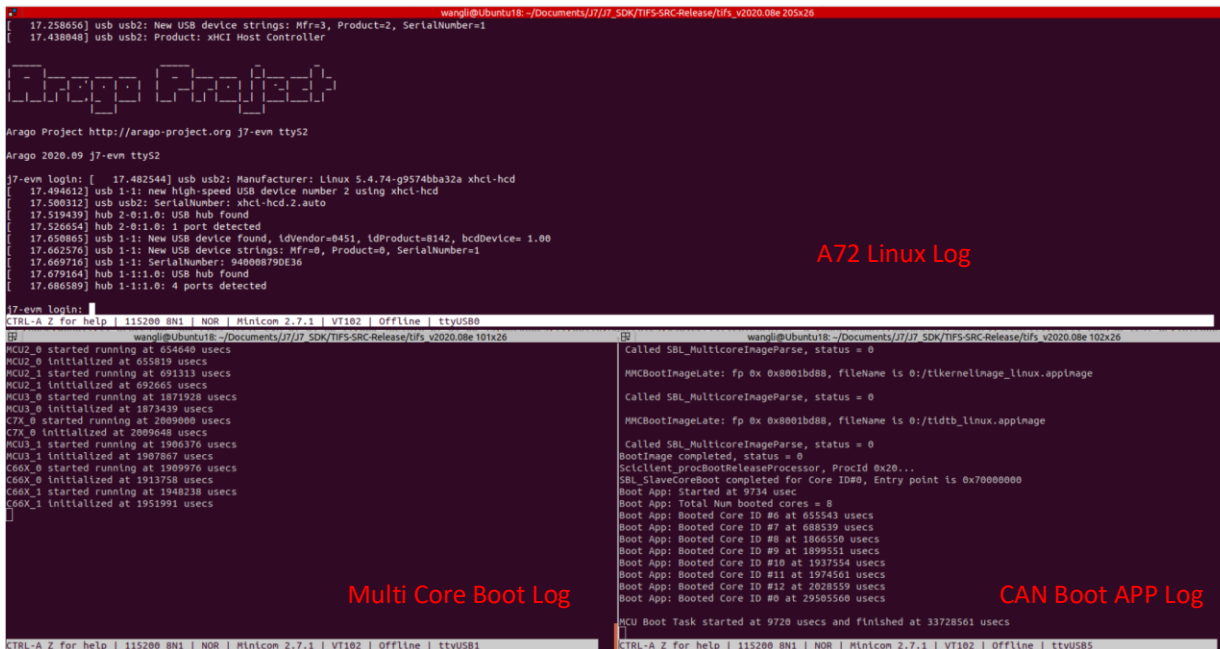


图 7 TDA4 HS Prime 整机启动测试

可以看出在 HS Prime SE 芯片中，所有核心正常运行。对于 vHSM 的功能验证，需要客户在 TIFS 中利用 vHSM 函数接口控制 AES，RNG 等硬件加密加速器来进行随机数产生等操作，此处不做赘述。

值得注意的是，对于 A72 上运行的 HLOS：

若是 Linux 操作系统，需要签名的至少有 `atf_optee.appimage`，`tidtb_linux.appimage` 以及 `tikernelimage_linux.appimage`

若是 QNX 操作系统，需要签名的至少有 `atf_optee.appimage`，`ifs_qnx.appimage`

5. 总结

为了防止系统镜像被恶意篡改、复制以及删除，以及实现系统运行时交互数据复杂且实时的加解密操作，Security 已经成为了汽车量产产品中不可或缺的保证。本文针对 TDA4 HS Prime 芯片，对密钥烧录环境的搭建、密钥的烧写、vHSM 与 TIFS 的集成和编译签名以及最后的系统验证进行了详细的描述。

6. 参考文献

1. [TDA4VM Jacinto™ Processors for ADAS and Autonomous Vehicles Silicon Revisions 1.0 and 1.1 datasheet \(Rev. J\)](#)
2. [DRA829/TDA4VM/AM752x Technical Reference Manual \(Rev. B\)](#)
3. [Developing with High Security Devices of TDA4.](#)

重要声明和免责声明

TI“按原样”提供技术和可靠性数据（包括数据表）、设计资源（包括参考设计）、应用或其他设计建议、网络工具、安全信息和其他资源，不保证没有瑕疵且不做任何明示或暗示的担保，包括但不限于对适销性、某特定用途方面的适用性或不侵犯任何第三方知识产权的暗示担保。

这些资源可供使用 TI 产品进行设计的熟练开发人员使用。您将自行承担以下全部责任：(1) 针对您的应用选择合适的 TI 产品，(2) 设计、验证并测试您的应用，(3) 确保您的应用满足相应标准以及任何其他功能安全、信息安全、监管或其他要求。

这些资源如有变更，恕不另行通知。TI 授权您仅可将这些资源用于研发本资源所述的 TI 产品的应用。严禁对这些资源进行其他复制或展示。您无权使用任何其他 TI 知识产权或任何第三方知识产权。您应全额赔偿因在这些资源的使用中对 TI 及其代表造成的任何索赔、损害、成本、损失和债务，TI 对此概不负责。

TI 提供的产品受 [TI 的销售条款](#) 或 [ti.com](#) 上其他适用条款/TI 产品随附的其他适用条款的约束。TI 提供这些资源并不会扩展或以其他方式更改 TI 针对 TI 产品发布的适用的担保或担保免责声明。

TI 反对并拒绝您可能提出的任何其他或不同的条款。

邮寄地址：Texas Instruments, Post Office Box 655303, Dallas, Texas 75265
Copyright © 2021，德州仪器 (TI) 公司