Application Note

Jacinto7 高安全性器件开发

Neo Wang, Joe Shen, Brijesh Jadav, and Keerthy J

摘要

Jacinto7（J7）SoC 具有两种类型的器件：通用（GP）器件和高安全性（HS）器件。HS 器件还具有两种表示 HS 器件状态的子类型：高安全性 - 现场安全（HS-FS）型和高安全性 - 强制安全（HS-SE）型。所有安全特性在 HS-SE 器件中均处于启用状态，以确保终端客户系统和软件具有完整性、机密性和防克隆保护。在使用 HS-SE 器件开始批量生产之前，您应该确保 HS 器件中每个步骤的正确性。本文档向您介绍了如何在 TIDK 器件和客户产品 HS 器件中完成安全流程。

内容

1 引言 2
2 TIDK 器件验证 3
2.1 对次级引导加载程序（SBL）签名和加密 3
2.2 为系统映像签名和加密 4
3 密钥编程 5
3.1 安装 Keywriter 5
3.2 密钥生成 6
3.3 编译 Keywriter 应用 7
3.4 在 HS-FS 器件中对密钥进行编程 7
4 密钥编程验证 9
5 使用 Linux SDK 对 HS 器件进行编译和引导 10
6 总结 10

插图清单

图 1-1. Jacinto7 SoC 器件类型 2
图 1-2. Jacinto7 高安全性开发 2
图 3-1. Keywriter 安装目录 5
图 3-2. X509 证书生成过程 6
图 3-3. eFuses 编程流程 8

表格清单

表 1-1. 高安全性开发中的密钥 3
表 2-1. 启动系统所需的二进制 4
表 3-1. 建议的 OTP eFuse 编程操作条件 8

商标

1 引言

在从开发到批量生产的不同客户阶段，J7 系列 SoC 可根据不同的安全要求提供多种类型的器件，如图 1-1 中所示。

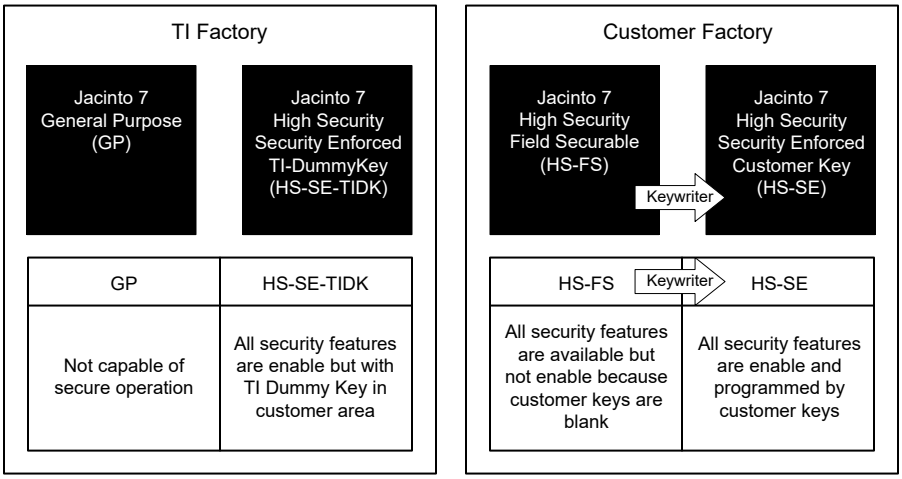

图 1-1. Jacinto7 SoC 器件类型

- **GP 器件**：通用器件类型可以是生产器件，但通常在考虑安全性的开发流程中使用。
- **HS-SE-TIDK 器件**：高安全性器件包含 TI 密钥，并且客户区域中有 TI 虚拟密钥。TIDK 器件绝不可能是生产器件。
- **HS-FS 器件**：高安全性 - 现场安全型器件，客户区域为空。HS-FS 器件需要先进行密钥编程，然后才能交付给终端客户。
- **HS-SE 器件**：高安全性 - 强制安全型器件。客户已将其密钥编程到客户区域中。HS-SE 器件可以是生产器件。

不同类型的器件提供不同的安全特性，但也有不同的限制。GP 器件没有安全特性，JTAG 端口未锁定，并且所有二进制均不需要签名和加密，因此，它通常用作开发器件。HS 器件强制执行安全特性，但 JTAG 端口已锁定，并且所有二进制均必须签名和加密，因此，它通常用作生产器件。客户可以查看图 1-2 中的开发流程以完成 HS 流程。

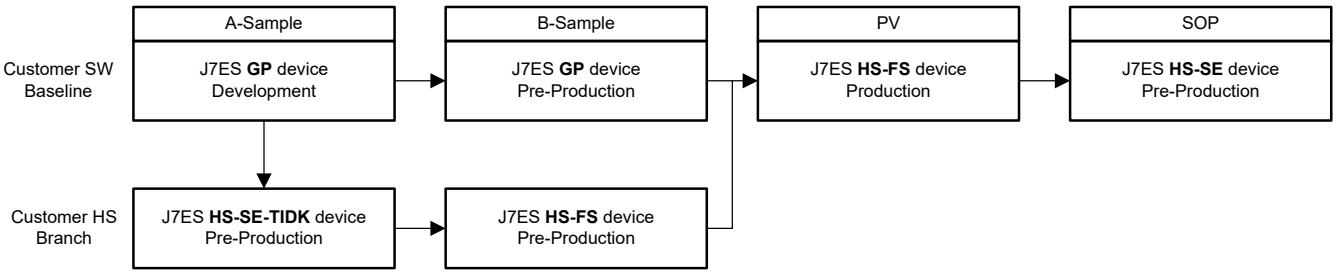

图 1-2. Jacinto7 高安全性开发

建议的流程为：

- 客户工程师针对 GP 器件进行开发；同时，HS 工程师开始针对 HS-SE-TIDK 器件进行安全开发，以确保 HS 签名和加密操作的正确性。
- HS 工程师购买 HS-FS 器件，并使用 TI 虚拟密钥或随机虚拟密钥复查 HS 流程，以便确认密钥生成、Keywriter 设置和 eFuse 编程流程。
- 合成 GP 的工作和 HS-FS 的安全分支，而且客户还需要设置其 HSM，将客户密钥编程到 eFuse 中，并使用相同的客户密钥对系统映像签名和加密。

在 J7 器件的安全流程中会用到许多密钥，如表 1-1 中所示。注释介绍了 HS 开发期间的重要密钥。SMPKH、SMEK、BMPKH 和 BMEK 被编程到 eFuse 中，以对系统映像进行验证和解密。AES-256 密钥和 TI FEK 公钥用于保护密钥编程流程。

TI 还在 Keywriter 包中提供测试密钥。客户可以使用这些测试密钥完成整个 HS 流程，接着设置其 HSM 并使用客户密钥完成任务，然后再进行生产。

表 1-1. 高安全性开发中的密钥

首字母缩写词	名称	状态	所有者	注意事项
KEK	密钥加密密钥	必需	器件	每个器件具有一个 256 位的在统计上唯一的随机数字
MPK 哈希	制造商公钥哈希	必需	TI	MPK 的 512 位 SHA2 哈希。MPK 是 TI 在工厂中编程的 4096 位密钥。
MEK	制造商加密密钥	必需	TI	器件的 256 位初始加密密钥，用于加密引导，由 TI 在工厂中编程。
SMPK 哈希	次级制造商公钥哈希	必需	客户	SMPK 的 512 位 SHA2 哈希。SMPK 是用于验证已签名二进制的 4096 位密钥。
SMEK	次级制造商加密密钥	必需	客户	加密引导的 256 位客户加密密钥，用于将加密二进制解密。
BMPK 哈希	备用制造商公钥哈希	可选	客户	SMPK 的备用 512 位 SHA2 哈希。SMPK 是用于验证已签名二进制的 4096 位密钥。
BMEK	备用制造商加密密钥	可选	客户	加密引导的备用 256 位客户加密密钥，用于将加密二进制解密。
AES-256	高级加密标准 256 位密钥	可选	客户	要用作临时 AES 加密密钥以保护 OTP 扩展数据的随机 256 位数字。
TI FEK Pub	TI 工厂加密密钥	必需	TI	在客户密钥资料被写至 eFuses 之前对其进行保护的 RSA 4K 加密密钥。

2 TIDK 器件验证

HS-SE-TIDK 器件是强制安全型器件，它在客户区域对 TI 虚拟密钥进行了编程。如果客户想要验证 HS-SE-TIDK 器件的功能，则需要使用 TI 虚拟密钥为其系统映像签名和加密。要帮助客户熟悉使用安全密钥为二进制签名和加密的流程，需要执行此步骤。但 TI 虚拟密钥对所有客户公开，因此，HS-SE-TIDK 器件绝不会成为生产器件。

TI 虚拟密钥包含在默认 RTOS SDK 中。您可以从适用于 DRA829 & TDA4VM Jacinto™ 处理器的 RTOS SDK 下载 SDK。您可以在以下文件夹中找到 TI 虚拟密钥。在 SDK7.1 软件中测试的所有流程均可正常运行。

```
# cp ${PSDKRA_PATH}/pdk/packages/ti/build/makerules/k3_dev_mpk.pem ~/TIDummyKey/smpk.pem
# cp ${PSDKRA_PATH}/pdk/packages/ti/build/makerules/k3_dev_mek.txt  ~/TIDummyKey/smek.txt
```

2.1 对次级引导加载程序（SBL）签名和加密

SBL（次级引导加载程序）由 MCU R5F ROM 加载并由 DMSC 验证。MCU R5F 运行此代码并为其他内核启动引导流程。使用以下命令为 HS 器件编译已签名的 SBL。

```
# cd ~/ti-processor-sdk-rtos-j721e-evm-07_01_00_11/pdk_jacinto_07_01_00_45/packages/ti/build
# make -j BOARD=j721e_evm CORE=mcu1_0 BUILD_PROFILE=release sbl_mmcsd_img_hs
#ls ${PSDKRA_PATH}/pdk/packages/ti/boot/sbl/binary/j721e_evm_hs/mmcsd/bin/
sbl_mmcsd_img_mcu1_0_release.tiimage
```

默认 makefile 仅对 SBL 签名，但不会对其加密，应用以下补丁并重新编译 SBL_HS 以获取签名并对 SBL 映像加密。

```
diff --git a/packages/ti/build/makerules/common.mk b/packages/ti/build/makerules/common.mk
index f56e069..e9ec0d9 100644
--- a/packages/ti/build/makerules/common.mk
+++ b/packages/ti/build/makerules/common.mk
@@ -635,7 +635,7 @@ else ifeq ($(SOC),$(filter $(SOC), am65xx am64x j721e j7200))
        $(CHMOD) a+x $(SBL_CERT_GEN)
 endif
-       $(SBL_CERT_GEN) -b $(SBL_BIN_PATH) -o $(SBL_TIIMAGE_PATH) -c R5 -l $(SBL_RUN_ADDRESS) -k $($
(APP_NAME)_SBL_CERT_KEY) -d DEBUG -j DBG_FULL_ENABLE -m $(SBL_MCU_STARTUP_MODE)
+       $(SBL_CERT_GEN) -b $(SBL_BIN_PATH) -o $(SBL_TIIMAGE_PATH) -c R5 -l $(SBL_RUN_ADDRESS) -k $($
(APP_NAME)_SBL_CERT_KEY)  -y ENCRYPT -e $(SBL_ENCRYPT_KEY_HS) -d DEBUG -j DBG_FULL_ENABLE -m $
(SBL_MCU_STARTUP_MODE)
diff --git a/packages/ti/build/makerules/platform.mk b/packages/ti/build/makerules/platform.mk
index cc6b905..381f1dd 100644
--- a/packages/ti/build/makerules/platform.mk
+++ b/packages/ti/build/makerules/platform.mk
@@ -200,7 +200,7 @@ endif
export SBL_CERT_KEY=$(ROOTDIR)/ti/build/makerules/rom_degenerateKey.pem
-
+export SBL_ENCRYPT_KEY_HS=~/TIDummyKey/smek.txt
```

上一个 makefile 和编译命令 将首先生成 GP SBL 二进制，然后使用 TI 虚拟密钥对其签名和 加密。此外，在编译 SBL_HS 期间，它还会 对板配置、安全配置、RM（资源管理）和 PM（电源管理）签名， 然后将其集成到 SBL 中。最后，可由 HS-SE-TIDK 器件 对 SBL 进行验证和解密。

在开始对二进制加密之前， SDK8.0 和之前的 SDK 版本中存在已知错误。首先应用以下补丁， 然后为二进制加密。

```
diff --git a/packages/ti/build/makerules/x509CertificateGen.sh b/packages/ti/build/makerules/
x509CertificateGen.sh
index 20fe23b..4c906e5 100755
--- a/packages/ti/build/makerules/x509CertificateGen.sh
+++ b/packages/ti/build/makerules/x509CertificateGen.sh
@@ -116,7 +116,7 @@ image_encrypt() {
        truncate -s %16 enc_tmp.bin
        xxd -r -p $ENC_RS enc_rs.bin
        cat enc_tmp.bin  enc_rs.bin > enc_bin_rs.bin
-       ENC_BIN=$CERT_SIGN"-ENC-"$BIN
+       ENC_BIN=$BIN"-ENC-"$CERT_SIGN
        echo "$ENC_BIN"
        if [ "$IMG_ENC" == "ENCRYPT" ];then
```

2.2 为系统映像签名和加密

CAN 响应和引导加载程序演示应用中提供了 GP 器件的引导加载程序演示。启动系统所需的二进制显示在表 2-1 中。

表 2-1. 启动系统所需的二进制

	需要签名和加密的映像		
系统结构	**引导多核 RTOS**	**引导 A72 上的 Linux 和其他内核上的 RTOS**	**引导 A72 上的 QNX 和其他内核上的 RTOS**
二进制	tiboot3.bin tifs.bin app lateapp1 lateapp2 lateapp3	tiboot3.bin tifs.bin app lateapp1 lateapp2 atf_optee.appimage tidtb_linux.appimage tikernelimage_linux.appimage	tiboot3.bin tifs.bin app lateapp1 lateapp2 atf_optee.appimage ifs_qnx.appimage

区别在于，如果在 A72 内核中运行 Linux，并且不需要为文件系统签名，则应对 ATF（Arm Trusted Firmware）映像、DTB（Device Tree Binary）映像和 Linux 内核映像签名和加密。此外，如果在 A72 内核中实现 QNX，则只需对 ATF 和 IFS（Image FileSystem）映像签名。

对于表 2-1 中的二进制，可由默认 SDK 中的 x509Certificate 脚本使用以下命令签名和加密。

```
# ${PSDKRA_PATH}/pdk/packages/ti/build/makerules/x509CertificateGen.sh -b binary_need_sign&encrypt -
o signed_encrypted_binary -c R5 -l 0x0 -k ${PSDKRA_PATH}/pdk/packages/ti/build/makerules/
k3_dev_mpk.pem -y ENCRYPT -e ${PSDKRA_PATH}/pdk/packages/ti/build/makerules/k3_dev_mek.txt  -d
DEBUG -j DBG_FULL_ENABLE -m SPLIT_MODE
```

除了为 SBL 和系统映像签名和加密外， 还需要使用 TI 虚拟密钥为 TIFS 签名和加密。使用 TI 虚拟密钥签名的 TIFS 已在 SDK 中提供，可直接在 HS-SE-TIDK 器件中 使用。

```
$ cp ${PSDKRA_PATH}/pdk/packages/ti/drv/sciclient/soc/V1/tifs-hs-enc.bin    /media/user/boot/tifs.bin
```

3 密钥编程

Keywriter 是可将客户密钥融入到 HS-FS 器件中并将该器件转换为 HS-SE 器件的软件包。Keywriter 可基于 PDK 进行编译并提供特殊的 TIFS 二进制、TI 密钥生成工具和相关源代码。客户可以使用此 Keywriter 包创建可作为引导加载程序在目标上运行的二进制，其将自动完成密钥编程。

3.1 安装 Keywriter

安装完 Keywriter 包后，它包含两个文件夹，如图 3-1 中所示。您需要将这两个原始文件夹或 PDK 中的文件合并，并替换为 Keywriter 文件夹。

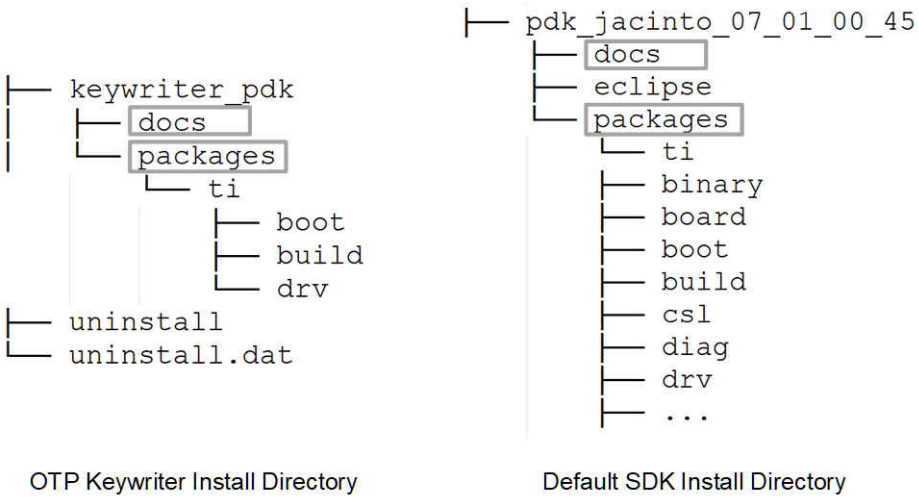

图 3-1. Keywriter 安装目录

OpenSSL 是编译 OTP Keywriter 所需的，请先确保在 Linux OS 上安装了 OpenSSL，然后再执行下一步。

```
# sudo apt-get install openssl
```

3.2 密钥生成

安装 SDK 之后，Keywriter 包应包括在编译 Keywriter 应用之前用于生成 x509 证书的密钥，然后将该 x509 证书附加到最终的 Keywriter 二进制，如图 3-2 中所示。

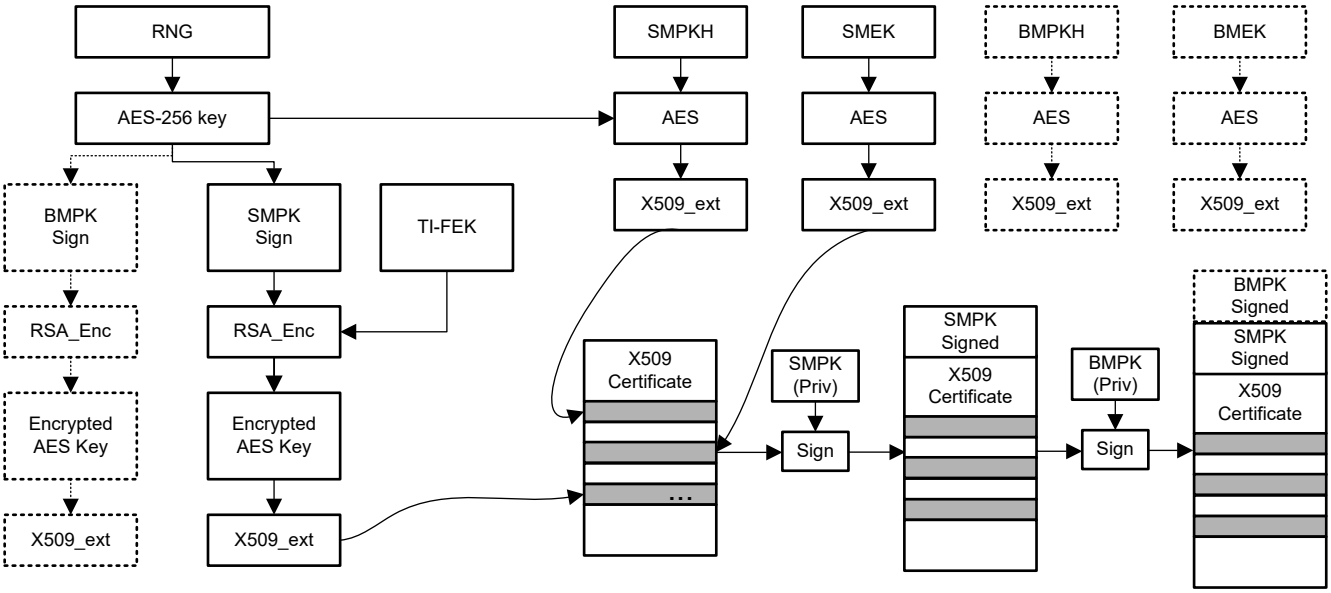

图 3-2. X509 证书生成过程

- 客户 HSM 中的 RNG（随机数生成器）会生成随机 AES-256 密钥。该 AES-256 密钥是临时的，用于为每个字段加密。对于每个字段，AES-256 密钥是相同的，但要使用不同的 IV（初始化矢量）来调用。该 AES-256 密钥还由 TI FEK 公钥加密，并且包括在 x509 扩展中。
- SMPKH、SMEK、BMPKH 和 BMEK 密钥由 AES-256 密钥加密，并且包括在不同的 x509 扩展中。这些密钥以纯文本形式编程到 eFuses 中。
- x509 配置由 SMPK 私钥和 BMPK 私钥（可选）签名，之后将这些字段附加到原始 x509 扩展中。

一旦 eFuse 由客户密钥进行了编程，此流程便不可逆转。为了确保操作和流程的正确性，在客户开始将客户密钥编程到 HS-FS 器件中之前，最好首先使用 TI 虚拟密钥或随机虚拟密钥作为测试密钥；然而，此步骤不是强制性的。

Keywriter 包提供一个可生成随机虚拟密钥来帮助客户进行测试的脚本。TI 虚拟密钥是公钥，而客户密钥应安全地保存在客户 HSM 中，以便客户可以选择在生产前使用此随机虚拟密钥进行测试。要生成随机虚拟密钥，请执行以下步骤：

```
# cd ${PSDKRA_PATH} /pdk/packages/ti/boot/sbl/example/k3MulticoreApp/keywriter/scripts
# ./gen_keywr_cert.sh -g
# cp ${PSDKRA_PATH} /pdk/packages/ti/boot/sbl/example/k3MulticoreApp/keywriter/scripts/
ti_fek_public.pem
${PSDKRA_PATH} /pdk/packages/ti/boot/sbl/example/k3MulticoreApp/keywriter/scripts/keys/tifekpub.pem
# ls ${PSDKRA_PATH} /pdk/packages/ti/boot/sbl/example/k3MulticoreApp/keywriter/scripts/keys
aes256.key bmek.key bmpk.pem smek.key smpk.pem tifekpub.pem
```

此外，TI 虚拟密钥在 TI SDK 中发布。一旦客户使用 TI 虚拟密钥完成密钥编程，器件将转换为 HS-SE-TIDK 器件。因此，客户可以使用节 2 中的方法对其系统映像签名和加密，然后验证由 TI 虚拟密钥编程的器件中的二进制。

```
# cd ${PSDKRA_PATH}/pdk/packages/ti/boot/sbl/example/k3MulticoreApp/keywriter/scripts
# ./gen_keywr_cert.sh -g
# rm ${PSDKRA_PATH}/pdk/packages/ti/boot/sbl/example/k3MulticoreApp/keywriter/scripts/keys/bmek.key
bmpk.pem
smek.key smpk.pem
# cp ${PSDKRA_PATH}/pdk/packages/ti/build/makerules/k3_dev_mpk.pem
${PSDKRA_PATH}/pdk/packages/ti/boot/sbl/example/k3MulticoreApp/keywriter/scripts/keys/smpk.pem
# xxd -p -r ${PSDKRA_PATH}/pdk/packages/ti/build/makerules/k3_dev_mek.txt ${PSDKRA_PATH}/pdk
/packages/ti/boot/sbl/example/k3MulticoreApp/keywriter/scripts/keys/smek.key
# cp ${PSDKRA_PATH} /pdk/packages/ti/boot/sbl/example/k3MulticoreApp/keywriter/scripts/
ti_fek_public.pem
${PSDKRA_PATH} /pdk/packages/ti/boot/sbl/example/k3MulticoreApp/keywriter/scripts/keys/tifekpub.pem
# ls ${PSDKRA_PATH} /pdk/packages/ti/boot/sbl/example/k3MulticoreApp/keywriter/scripts/keys
aes256.key smek.key smpk.pem tifekpub.pem
```

k3_dev_mek.txt 是 32 字节的数据文件，用于将二进制作为对称密钥进行加密，如节 2 中所示。相应的解密密钥需要由 Keywriter 编程到 eFuse 中，但 Keywriter 要求文件采用二进制形式。用于转换格式的“xxd”命令如下所示。此外，k3_dev_mpk.pem 是用于对二进制签名的对称密钥的密钥，并且相应的公钥会计算哈希值并由 Keywriter 编程到 eFuse 中。

更加可靠的是使用以下命令检查 smek.key。执行下一步之前，确保这两个结果完全相同。

```
# cat ${PSDKRA_PATH}/pdk/packages/ti/build/makerules/k3_dev_mek.txt
c143f03568798964d4a5769bd5a27d3adc0d6bdd8f3cc47b84229e50a54ab043
# xxd -p /home/wangli/ti-processor-sdk-rtos-j721e-evm-
07_01_00_11/pdk_jacinto_07_01_00_45/packages/ti/boot/sbl/example/k3MulticoreApp/keywriter/scripts/
keys/smek.key
c143f03568798964d4a5769bd5a27d3adc0d6bdd8f3cc47b84229e50a54ab043
```

如果客户决定使用其客户密钥，他们还需要将其密钥复制到以下文件夹中：
${PSDKRA_PATH}/pdk/packages/ti/boot/sbl/example/k3MulticoreApp/keywriter/scripts/keys。

3.3 编译 Keywriter 应用

在生成要编程到 SoC 中的密钥之后，请执行以下命令以生成 x509 证书。

```
# cd ${PSDKRA_PATH}/pdk/packages/ti/boot/sbl/example/k3MulticoreApp/keywriter/scripts
# ./gen_keywr_cert.sh -s keys/smpk.pem --smek keys/smek.key -t keys/tifekpub.pem -a keys/aes256.key
```

您可以在以下文件夹中找到生成的证书：x509cert/final_certificate.bin。执行以下命令以编译 Keywriter 源代码并将 x509 证书附加到 keywriter 应用。

```
# cd ${PSDKRA_PATH}/pdk/packages/ti/build
# make keywriter_img -j8
```

GP 器件和 HS-FS 器件中的 TIFS 是不同的，因为 TI 生产密钥已在 TI 工厂内编程到 HS-FS 器件中，但它在 GP 器件中为空。然而，编译 Keywriter 应用时，PDK 中的 keywriter.mk 将加载 HS-FS TIFS 二进制并将其转换为数组形式，该数组可由 Keywriter 应用源代码加载。所有这些步骤将在编译 Keywriter 源代码时自动执行，因此不需要执行额外操作。

生成的 Keywriter 应用位于以下文件夹中：
${PSDKRA_PATH}/pdk/packages/ti/boot/sbl/example/k3MulticoreApp/binary/keywriter_img_j721e_release.bin

3.4 在 HS-FS 器件中对密钥进行编程

在软件准备就绪后，必须满足以下硬件要求方可在 SoC OTP eFuse 中对密钥进行编程：

- 当不对 OTP 寄存器进行编程时，VPP_CORE 和 VPP_MCU 电源必须禁用。
- VPP_CORE 和 VPP_MCU 电源的电压必须在正确的器件上电序列完成后上升，并且需要设置处于以下范围（如表 3-1 中所列）内的电压值。

表 3-1. 建议的 OTP eFuse 编程操作条件

参数	说明	最小值	典型值	最大值	单位
VPP_CORE	**正常工作**期间 eFuse ROM 域的电源电压范围	不适用			
	OTP 编程期间 eFuse ROM 域的电源电压范围	1.71	1.8	1.89	V
VPP_MCU	**正常工作**期间 eFuse ROM 域的电源电压范围	不适用			
	OTP 编程期间 eFuse ROM 域的电源电压范围	1.71	1.8	1.89	V
Tj	温度	0	25	85	℃

有关开始刷写工作的步骤，请参阅图 3-3。

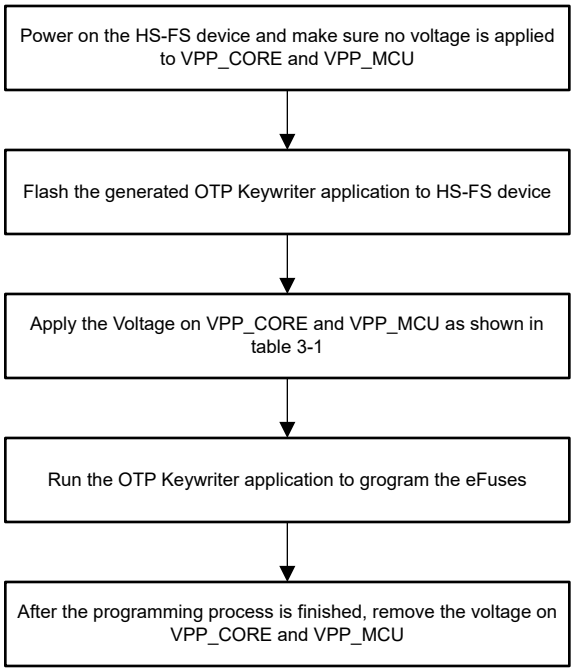

图 3-3. eFuses 编程流程

OTP Keywriter 可作为器件引导介质中的引导加载程序运行。一旦 Keywriter 应用开始进行密钥编程，便可以在 MCU 域 UART1 中找到如下所示的日志：

```
$ \0OTP 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
```

完成前面的步骤后，应使用特定密钥 将 HS-FS 器件转换为 HS-SE 器件；然而， 如果客户想要在 HS-SE 器件中运行这些二进制， 则需要使用相同的密钥对其进行签名和加密。

4 密钥编程验证

将密钥编程到 eFuse 中之后，客户可以先通过以下步骤验证编程结果并检查器件状态，然后在工厂中启动生产：

1. 将板的引导模式配置为 UART 引导并将板的第二个 MCU UART 串行端口连接到主机 PC，参阅 J721E 的 EVM 设置，然后启动 EVM。
2. 终端会显示如下所示的一些日志。您需要移除末尾的额外 CCC 并另存为日志文件。

```
# cat default_uart_hs.log
02000000011a00006a3765730000000000000000485353450200010002002000100033c74f0c8631aa67a
56d53b06f250d75cb2a9cf7a52d6eb5e21b5e824250d7e09c22d997f09dc9389ecaa3f7d2b64d3a76d6163aa09e928ea0
50e1da9550
7e661f6002b07cd9b0b7c47d9ca8d1aae57b8e8784a12f636b2b760d7d98a18f189760dfd0f23e2b0cb10ec7edc7c6eda
c3d9bdfefe0eddc3fff7fe9ad875195527df02f2a23c0ed9d5fcf6dfb3a097ee4207cb1e2a5956e07ba144b73fe711439
82
```

3. 复制以下代码并将其另存为 python 文件，以在步骤 2 中用于解析日志。

```
#!/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('I', 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
dmscROMVer = list(pubROMInfo[20:24])
dmscROMVer.reverse()
print "DMSC ROM Version      :", dmscROMVer
r5ROMVer = list(pubROMInfo[24:28])
r5ROMVer.reverse()
print "R5 ROM Version        :", r5ROMVer
print ('-----------------------')
print ('SoC ID Secure ROM Info:')
print ('-----------------------')
print "Sec SubBlockId        :", secROMInfo[0]
print "Sec SubBlockSize      :", secROMInfo[1]
print "Sec Prime             :", secROMInfo[2]
print "Sec Key Revision      :", secROMInfo[3]
print "Sec Key Count         :", secROMInfo[4]
tmpList = list(secROMInfo[5:69])
tiMPKHash = ''.join('{:02x}'.format(x) for x in tmpList)
print "Sec TI MPK Hash       :", tiMPKHash
tmpList = list(secROMInfo[69:133])
custMPKHash = ''.join('{:02x}'.format(x) for x in tmpList)
print "Sec Cust MPK Hash     :", custMPKHash
tmpList = list(secROMInfo[133:167])
uID = ''.join('{:02x}'.format(x) for x in
print "Sec Unique ID         :", uID
```

4. 在获取上述两个文件后，使用以下命令解析日志。解析后的信息如下所示：

```
# python uart_boot_socid.py default_uart_hs.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, 0, 2]
R5 ROM Version       : [0, 1, 0, 2]
-----------------------
SoC ID Secure ROM Info:
-----------------------
Sec SubBlockId       : 2
Sec SubBlockSize     : 166
Sec Prime            : 0
Sec Key Revision     : 1
Sec Key Count        : 1
Sec TI MPK Hash      :
33c74f0c8631aa67a56d53b06f250d75cb2a9cf7a52d6eb5e21b5e824250d7e09c22d997f09dc9389ecaa3f7d2b64d3a7
6d6163aa09e928ea050e1da95507e66
Sec Cust MPK Hash    :
1f6002b07cd9b0b7c47d9ca8d1aae57b8e8784a12f636b2b760d7d98a18f189760dfd0f23e2b0cb10ec7edc7c6edac3d9
bdfefe0eddc3fff7fe9ad875195527d
Sec Unique ID        : f02f2a23c0ed9d5fcf6dfb3a097ee4207cb1e2a5956e07ba144b73fe71143982
```

日志显示，器件类型已转换为 HS-SE，并且密钥版本和密钥数量均为 1，这意味着只会对 SMPK 进行编程并使用，而不会对 BMPK 这样操作。当可以通过日志获取客户的 SMPK 哈希值时，客户可以使用以下方法检查与客户自己的密钥的一致性。

```
# openssl rsa -in k3_dev_mpk.pem -pubout -outform DER -out /tmp/k3_dev_mpk_pub.der
writing RSA key
# sha512sum /tmp/k3_dev_mpk_pub.der
1f6002b07cd9b0b7c47d9ca8d1aae57b8e8784a12f636b2b760d7d98a18f189760dfd0f23e2b0cb10ec7edc7c6edac3d9bdf
efe0eddc3fff7fe9ad875195527d  /tmp/k3_dev_mpk_pub.der
```

比较发现，客户所编程的密钥的哈希值与从器件读取的哈希值完全相同。因此，我们可以根据特定客户密钥验证器件是否已成功转换为 HS-SE。

5 使用 Linux SDK 对 HS 器件进行编译和引导

您可以使用 Linux SDK 8.0 或更高版本对 TDA4VM D 样片（虚拟密钥样片）进行编译和引导。SDK 8.0 中添加了 HS 支持。有关 J721e HS 编译/引导指令，请参阅：https://e2e.ti.com/support/processors/f/791/t/1061584。

6 总结

处理 HS 器件期间要小心，因为有些步骤不可逆转。本文档可以帮助客户完成 HS 开发。

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出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