Linux 安全
开发指南

版本号: 2.1
发布日期: 2025.04.23

版本历史

版本号	日期	制/修订人	内容描述
1.0	2020.11.10	AWA1551	初始版本。
1.1	2021.03.02	AWA1551	兼容 4.9 版本 kernel。
1.2	2021.05.13	AWA1551	修正适用范围描述。
1.3	2022.03.03	AWA1832	1. 调整文档格式。2. 修正适用范围描述。
1.4	2023.08.02	XAA0175	1. 补充部分细节。2. 修正格式问题。
1.5	2023.08.28	XAA0175	1. 更新 demo 的打印信息 2. 添加 “安卓环境下 CA 编译” 和 “解密密钥源” 章节 3. 添加 FAQ
1.6	2023.11.22	XAA0175	1. 完善 “TA/CA 开发指引” 2. 补充 FAQ
1.7	2023.11.25	XAA0248	调整文档格式。
1.8	2023.11.25	XAA0193	1. 调整文档中的语句，使其清晰、易懂、无歧义 2. 修改文档的段落结构，使其清晰、易懂、无歧义 3. 修改内核配置部分过时的介绍 4. 修改部分小错误：文件地址不准确、笔误等 5. 优化了 TA 加密、验签部分功能的使用介绍
1.9	2024.09.05	AWA2275	1. 添加 “安全固件配置” 章节 2. 修正部分描述
2.0	2025.03.12	XAA0309	调整 tee_kit 路径以及运行 demo。
2.1	2025.04.23	XAA0348	FAQ 章节增加 demo 运行报错检查步骤。

目　录

插　图

1 前言

1.1 文档简介

本文主要介绍了 Allwinner 安全方案的组成与功能。安全完整的方案基于 normal 方案扩展, 覆盖硬件安全、安全启动（Secure Boot）、安全系统（Secure OS）、安全应用（Trusted apps）等方面。本文从硬件安全、安全启动（Secure Boot）、安全系统（Secure OS）、安全应用的开发（TA/CA 开发指引）、固件密钥存储、安全系统在 Flash 上的加密保存几个方面进行介绍。

1.2 目标读者

Allwinner 软件平台的相关技术人员。

1.3 适用范围

Allwinner 软件平台。

Allwinner 使用 4.9 或更高版本的 kernel 的硬件平台。

2 安全系统基础

2.1 安全系统介绍

安全系统是基于硬件配合软件的安全解决方案。其主要目的是保障系统资源的完整性、保密性、可用性, 从而为系统提供一个可信的运行环境。

2.2 密码学基础介绍

2.2.1 数据加密模型

1. 明文 P。准备加密的文本, 称为明文。
2. 密文 Y。加密后的文本, 称为密文。
3. 加解密算法 E(D)。用于实现从明文到密文或从密文到明文的一种转换关系。
4. 密钥 K。密钥是加密和解密算法中的关键参数。

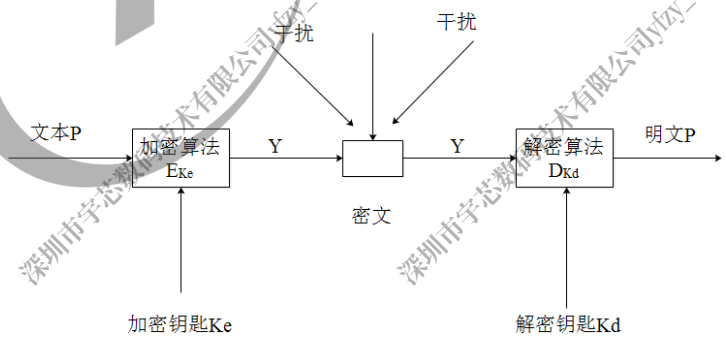

图 2-1: 加密交互过程

2.2.2 加密算法

1. 对称加密算法: 加密、解密用的是同一个密钥。比如 AES 算法。
2. 非对称加密算法: 加密、解密用的是不同的密钥, 一个密钥 “公开” , 即公钥, 另一个密钥持有, 即私钥。其中一把用于加密, 另一把用于解密。比如 RSA 算法。

3. 散列（hash）算法: 一种摘要算法, 把一笔任意长度的数据通过计算得到固定长度的输出, 但不能通过这个输出得到原始计算的数据。

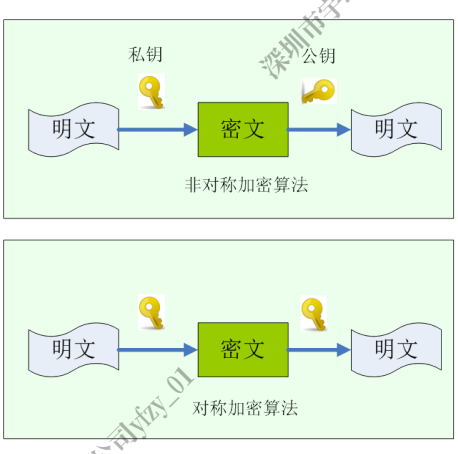

图 2-2: 对称及非对称加密

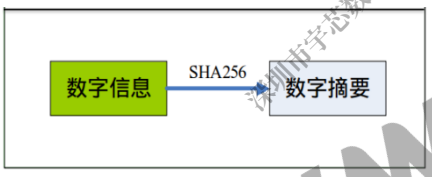

图 2-3: 摘要计算

2.2.3 签名与证书

数字签名: 数字签名是非对称密钥加密技术与数字摘要技术的应用。数字签名保证信息是由签名者自己签名发送的, 签名者不能否认或难以否认；可保证信息自签发后到收到为止未曾作过任何修改, 签发的文件是真实文件。

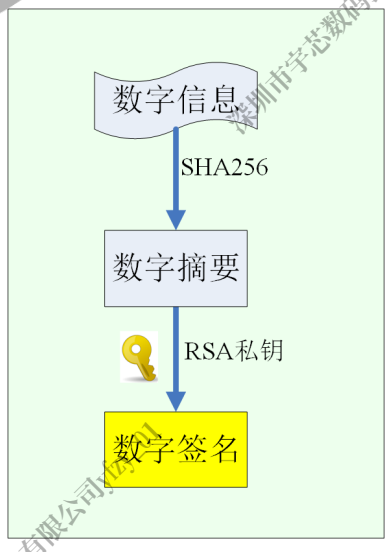

图 2-4: 数字签名

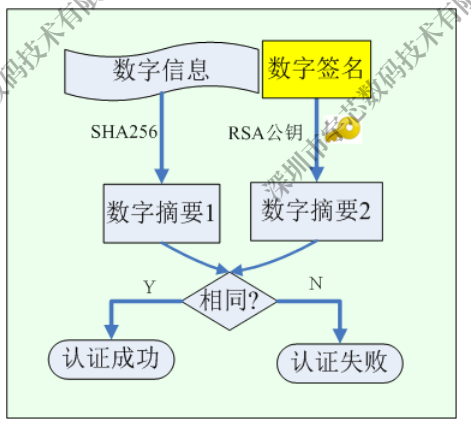

图 2-5: 验证签名

数字证书: 是一个经证书授权中心数字签名的包含公开密钥拥有者信息以及公开密钥的文件, 是一种权威性的电子文档。

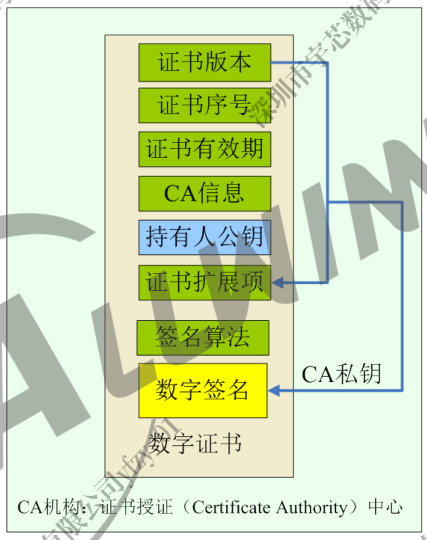

图 2-6: 证书结构

2.2.4 efuse

efuse: 一次性可编程熔丝技术。有些 SoC 集成了一个 efuse 电编程熔丝作为 OTP（One-Time Programmable, 一次性可编程）存储器。efuse 内部数据只能从 0 变成 1, 不能从 1 变成 0, 只能写入一次。

2.3 TrustZone

TrustZone 是 ARM 提出的安全解决方案, 旨在提供独立的安全操作系统及硬件虚拟化技术, 提供可信的执行环境（Trust Execution Environment）。TrustZone 系统模型如下图所示。

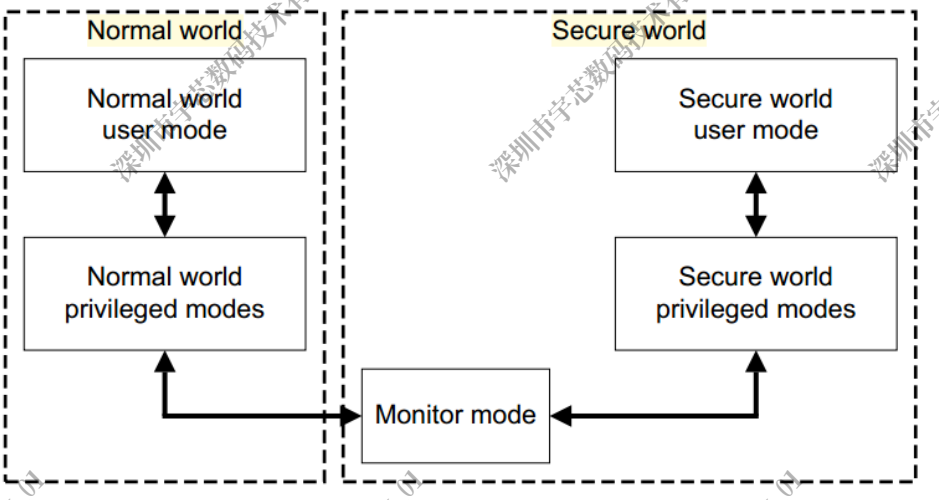

图 2-7: TurstZone 系统模型

TrustZone 技术将软硬件资源隔离成两个环境, 分别为安全世界（Secure World）和非安全世界（Normal World）, 所有需要保密的操作在安全世界执行, 其余操作在非安全世界执行, 安全世界与非安全世界通过 monitor mode 来进行切换。具体可参考《TrustZone security whitepaper.pdf》。

2.3.1 OPTEE

很多公司基于 TrustZone 推出了自己的安全操作系统, 各自有各自的实现方式, 但是基本都会遵循 GP（GlobalPlatform）标准。GlobalPlatform 是一个跨行业的国际标准组织, 致力于开发、制定并发布安全芯片的技术标准, 以促进多应用产业环境的管理及其安全、可互操作的业务部署。

OPTEE 是 Linaro 联合其他几个公司一起合作开发的基于 ARM TrustZone 技术实现的 TEE 方案, 遵循 GP 标准。

2.4 相关术语

- SMC：Secure Monitor Call,ARM 给出的一条指令, 可以让 CPU 从 Linux（非安全）直接跳转到 Monitor（安全）模式执行。
- RPC：Remote Procedure Control Protocol。OPTEE 中, 用于操作 Linux 下资源的一种机制。比如,OPTEE 中不能读写文件, 就通过 RPC 调用 Linux 下的文件系统来完成。
- REE：Rich Execution Environment。顾名思义，是资源丰富的执行环境，比如常见的 Linux,Android 系统等。
- TEE：Trusted Execution Environment。可信执行环境, 即安全执行环境, 在这个区域内, 所有的代码, 资源都是用户可以信任的。

- TA：Trusted Apps, 在 TEE 下执行的应用程序, 完成用户需要保护的任务, 比如对密码的保护。
- CA：Client Apps, 在 REE 下执行的应用程序, 完成普通的, 不需要保护的任务, 比如看普通视频。
- UUID：Universally Unique Identifier, 通用唯一识别码。由当前日期和时间, 时钟序列, 机器识别码（如 MAC）组成。

3 硬件安全模块

TrustZone 技术要求安全非安全使用独立的外设资源。allwinner 平台上, 安全非安全的资源的隔离通过指定的外设进行控制, 具体有下面三个:

- spc(Secure Peripherals Control)

指定外设的安全属性, 某外设被设定为安全后, 该外设只有在安全世界才能正常访问, 非安全世界写无效, 读为 0。

- sid(Secure ID)

控制 efuse 的访问。efuse 的访问只能通过 sid 模块进行。sid 本身非安全, 安全非安全均可访问。但通过 sid 访问 efuse 时, 安全的 efuse 只有安全世界才可以访问, 非安全世界访问的范围结果固定为 0。

- smc(Secure Memory Control)

控制内存地址空间的访问。某地址空间的内存被设定为安全后, 该空间内的内存只有安全世界可访问, 非安全世界写无效, 读为 0。

4 secure boot

4.1 基本原理

secure boot 启动过程中，芯片在启动时会先对系统做安全性检验，检验通过后才引导系统。检查不通过则认为系统已经被修改，拒绝引导系统并进入烧录模式。

安全性校验主要针对两个项目进行:

4.1.1 基于证书的哈希校验

固件中会包含证书，证书记录了固件的哈希值。芯片验证证书有效后，会使用证书记录的固件哈希值，和实际计算得到的固件哈希值对比，两者匹配则认为固件检验 OK。固件由多个子固件组成，brom->sboot->atf(64 位平台，32 位平台则跳过)->OPTEE->uboot->boot.img 的启动过程中，每个子固件都有对应的证书用于该子固件的哈希校验，确保整个方案的安全启动。

固件中各子固件的具体验证流程如下:

1. 使用 efuse 中的 rotpk 哈希验证证书记录的 rotpk 的有效性。
2. 使用 rotpk 验证证书的有效性。
3. 使用证书中记录的公钥验证子固件对应证书的有效性。
4. 使用子固件对应证书记录的哈希值验证子固件的有效性。

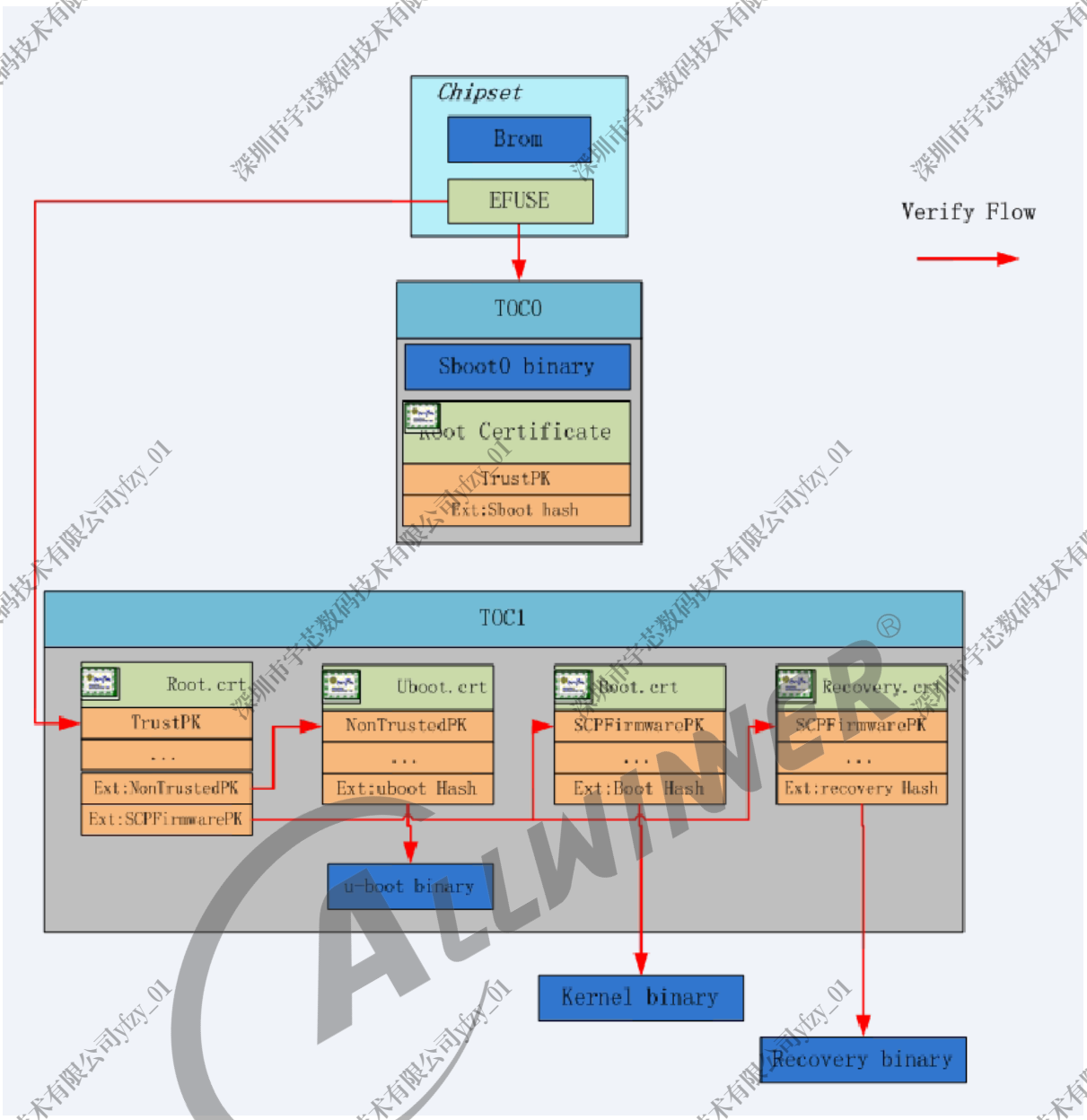

图 4-1: 固件验证过程

4.1.2 防回滚版本号校验

固件会含有一个用于防回滚检验的版本号，芯片会对比内部存储的版本号与固件的版本号。固件的版本号大于等于芯片记录的版本号时，认为固件的防回滚检验通过。

芯片会按需更新内部存储的版本号，确保存储的版本号为所有运行过的安全固件中，版本号最大的值。

下面介绍使用不同签名工具时如何配置两个检验项目并生成安全固件，使能 secure boot 功能。

4.2 安全固件配置

4.2.1 dragon_securetool 签名工具

4.2.1.1 签名密钥配置

根据不同的平台，支持不同算法类型的签名密钥，可根据当前方案中{SDK}/devices/configs/chips/\${chip}/configs/default/dragon_toc.cfg配置文件确定具体支持何种签名算法。

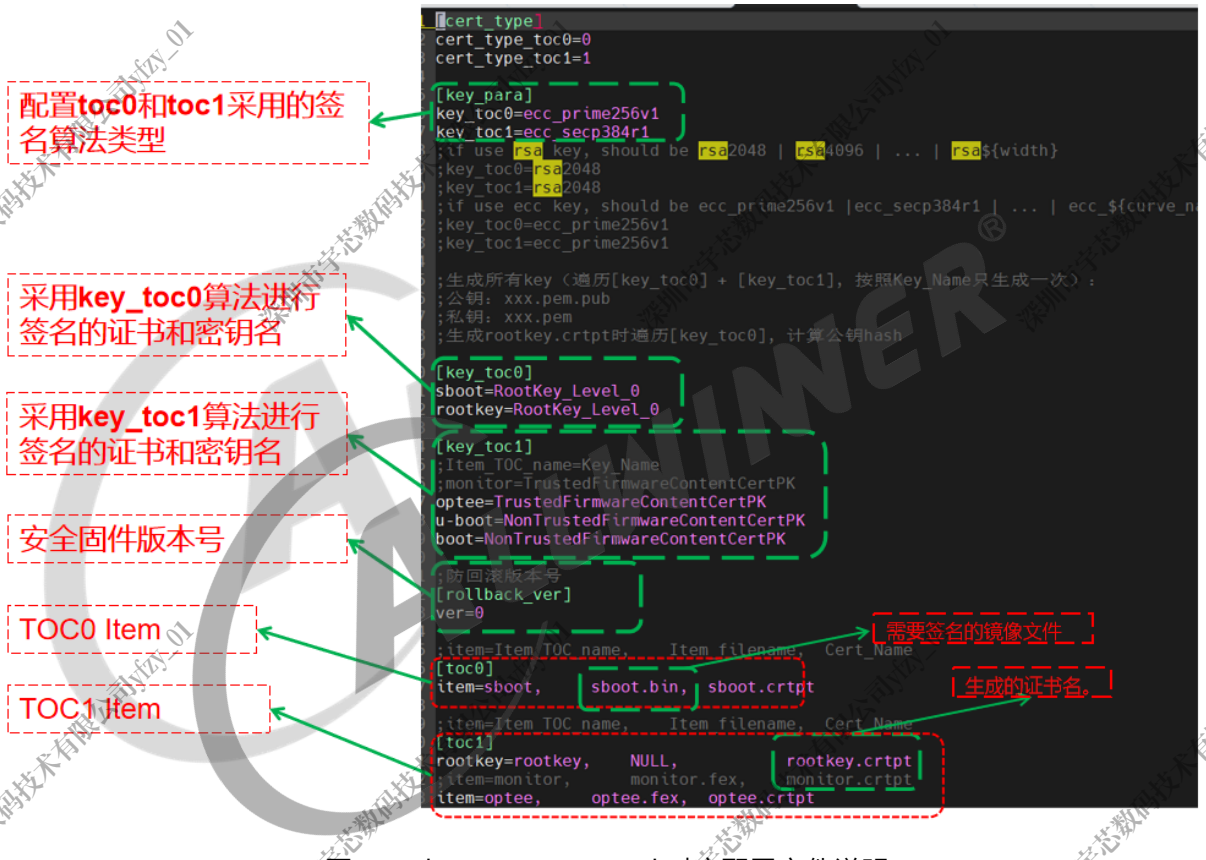

图 4-2: dragon_securetool 对应配置文件说明

若当前方案 SDK 中的dragon_toc.cfg配置文件与上图一致，则表示当前方案支持 toc0 和 toc1 采用不同算法签名。具体配置如上图所示。

配置完dragon_toc.cfg文件，运行 {SDK}/build/createkeys工具，选择对应平台后，会自动生成一组用于签名的密钥，生成的密钥位于 {SDK}/out/$(platform)/common/keys 目录下。打包安全固件时，会使用这些密钥分别对相应的镜像进行签名并生成证书。

说明

如果之前已经生成过密钥，将密钥文件放到{SDK}/out/$(platform)/common/keys目录下即可。

其中：

（1）rotpk.bin 为烧录到芯片中用于验证根证书的公钥；

（2）rotpk.bin 需要在烧录了安全固件的设备上才能烧录到芯片中，使用方法后述，见 rotpk 烧写；

（3）其他为打包固件时用于为固件包签名的私钥，一个固件由多个部分组成，每个部分使用单独的密钥对进行签名认证。

⚠ 注意

1. **这些密钥都是相互关联的，必须配套使用。生成的密钥请成套妥善保存。**
2. **密钥的文件数量及名称都是根据平台固件打包的过程做过适配的。A 平台生成的密钥不可用于 B 平台安全固件的打包，否则打包过程可能会因为找不到指定的密钥而失败。**

4.2.1.2 防回滚版本号配置

芯片在引导固件的时候，会对比固件的版本号与芯片内存保留的版本号。

防回滚版本号在 {SDK}/devices/configs/chips/\${chip}/configs/default/dragon_toc.cfg中进行配置：

```
[rollback_ver]
ver=1
```

- ver：表示固件的防回滚版本号，可用范围为 0-31。

4.2.2 dragonsecboot 签名工具

4.2.2.1 签名密钥配置

根据不同的平台，支持不同算法类型的签名密钥，可根据当前方案中{SDK}/devices/configs/chips/\${chip}/configs/default/dragon_toc.cfg配置文件确定具体支持何种签名算法。

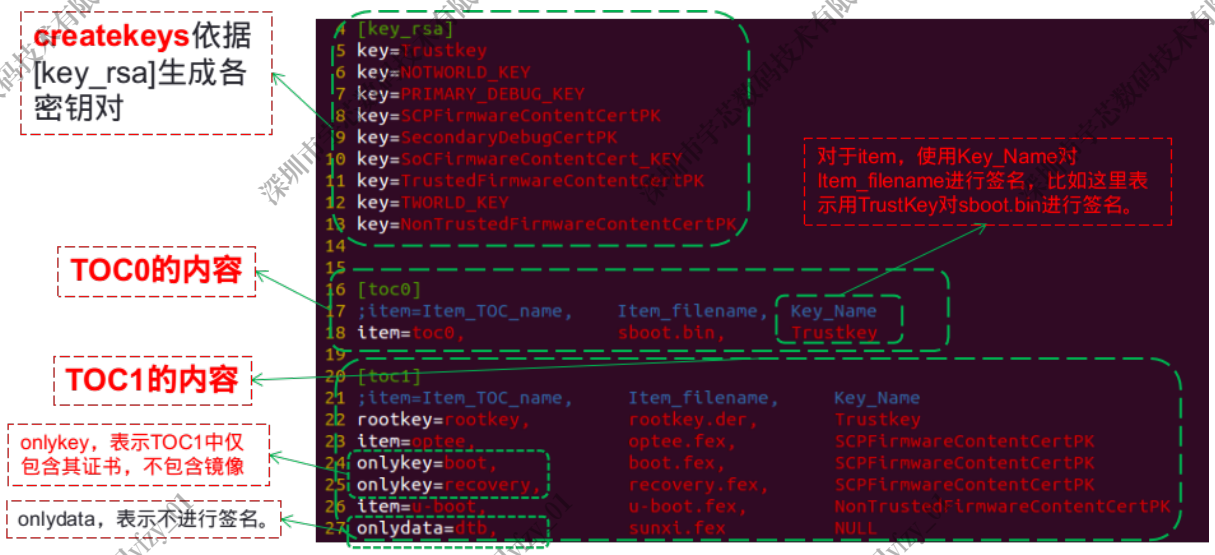

图 4-3: dragonsecboot 对应配置文件说明

若当前方案 SDK 中的dragon_toc.cfg配置文件与上图一致，则表示当前方案仅支持 toc0 和 toc1 采用 RSA 算法签名。具体配置如上图所示。

配置完dragon_toc.cfg文件，运行 {SDK}/build/createkeys工具，选择对应平台后，会自动生成一组用于签名的密钥，生成的密钥位于 {SDK}/out/$(platform)/common/keys 目录下。打包安全固件时，会使用这些密钥分别对相应的镜像进行签名并生成证书。

说明

如果之前已经生成过密钥, 将密钥文件放到{SDK}/out/$(platform)/common/keys目录下即可。

其中：

（1）rotpk.bin 为烧录到芯片中用于验证根证书的公钥；

（2）rotpk.bin 需要在烧录了安全固件的设备上才能烧录到芯片中, 使用方法后述, 见 rotpk 烧写；

（3）其他为打包固件时用于为固件包签名的私钥，一个固件由多个部分组成，每个部分使用单独的密钥对进行签名认证。

⚠ 注意

1. **这些密钥都是相互关联的，必须配套使用。生成的密钥请成套妥善保存。**
2. **密钥的文件数量及名称都是根据平台固件打包的过程做过适配的。A 平台生成的密钥不可用于 B 平台安全固件的打包，否则打包过程可能会因为找不到指定的密钥而失败。**

4.2.2.2 防回滚版本号配置

芯片在引导固件的时候, 会对比固件的版本号与芯片内存保留的版本号。

防回滚版本号在 {SDK}/devices/configs/chips/\${chip}/configs/default/version_base.mk中进行配置：

```
# define the verions of the image
# format: main
# such as 1, 2
# NOTICE: the range of main version is from 0 to 31,
# when you change the version, you must increase main version, and never reduce the versions.
# the default version is 0

MAIN_VERSION = 1
```

- MAIN_VERSION：表示固件的防回滚版本号，可用范围为 0-31，配置其他值芯片会直接认为固件版本号检验失败。

4.3 安全固件生成

在 {SDK} 目录下运行，即可打包安全固件：

```
{SDK}$ ./build.sh pack_secure
```

生成的固件在 {SDK}/out/目录下，然后使用 phoneSuit 工具进行烧录。

4.4 ROTPK 烧写

Rotpk 通过 PC 端工具 DragonSN 进行烧录。DragonSN 工具通过 usb 与设备通信, 控制设备烧录指定的 rotpk 信息，如果未显式配置，按 burn_key=0 处理。具体烧录步骤如下:

1. 配置 burn_key 属性：设置 burn_key 属性值为 1 后，设备才会接收 DragonSN 通过 usb 传输的信息, 进行相应的烧录工作。该属性有三种修改方式。

- 方式一：此方式用于 uboot 与内核共用设备树的情况，v853 方案上不支持，在文件devices/config/chips/${chips}/configs/${board}/sys_config.fex中,[target] 项, 如下图。如果未显式配置, 按 burn_key=0 处理。

```
22 ;----------------------------------------------------------------
23 [target]
24 boot_clock    >--= 1008
25 storage_type     = -1
26 burn_key = 1
27 ;----------------------------------------------------------------
28 ; power setting
```

图 4-4: 配置烧 key 属性方式一

- 方式二：此方式用于 uboot 有独立设备树的情况，在文件{SDK}/device/config/chips/v853/configs/perf1/uboot-board.dts中的 &target 项, 如下图。

```
&platform {
    eraseflag = <1>;
};

&target {
    boot_clock = <900>;     /*CPU boot frequency, Unit: MHz*/
    burn_key = <1>;
};
```

图 4-5: 配置烧 key 属性方式二

- 方式三：此方式的优先级最高，在文件{SDK}/device/config/chips/v853/configs/default/env.cfg中的 burn_key 项, 如下图。

```
#kernel command arguments
earlyprintk=sunxi-uart,0x02500000
initcall_debug=0
console=ttyS0,115200
nand_root=ubi0_4
mmc_root=/dev/mmcblk0p4
nor_root=/dev/mtdblock3
init=/init
loglevel=8
coherent_pool=16K
burn_key=2
#reserve_list=30M@64M,78M@128M,200M@512M
```

图 4-6: 配置烧 key 属性方式三

2. 打包安全固件并烧录，打包时使用的密钥必须与烧录的 rotpk 匹配, 具体原因详见 rotpk 烧录时设备端的处理过程。
3. 在 PC 端配置 DragonSN 工具后运行。
4. 设备通过 usb 与 pc 连接后开机。
5. DragonSN 显示设备已连接后开始烧录（详见 DragonSN 工具的使用说明）。

备注：rotpk 烧录时设备端的处理过程中，为了保证不会误烧错误的 rotpk，烧录时设备端会做额外的处理, 具体流程如下：

（1）pc 工具下发的 rotpk；

（2）uboot 会在确认与当前固件的 rotpk 匹配后才请求 Monitor 烧录该 rotpk。（对于 64 为平台 Monitor 为 ATF，对于 32 位平台，Monitor 为 OPTEE）

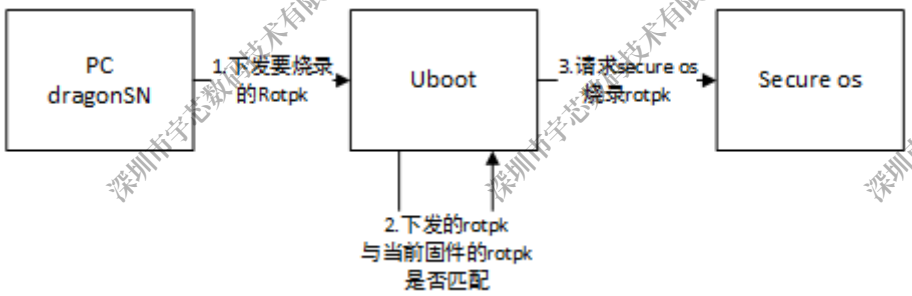

图 4-7: rotpk 烧录时设备端处理

4.5 注意事项

1. rotpk 只能烧录一次, 烧录后不可再修改。请妥善保管《配置密钥-a. 生成密钥》中生成的密钥文件。
2. 烧录过安全固件后, 芯片每次上电都会对固件进行安全性检查, 这时候烧录普通固件, 会因为无法通过检查而不能启动。故安全固件和普通固件不可混合使用。
3. 芯片未烧录 rotpk 时, 跳过根证书上公钥的验证, 继续进行后续的验证流程。
4. 芯片出厂时, 芯片内记录的防回滚版本号为 0。

5 secure os

5.1 OPTEE 介绍

Allwinner 软件平台采用 OPTEE 作为 secure os 的实现。OPTEE 严格遵循 ARM Trust-Zone 和 TEE/GP 等产业标准。加入 OPTEE 后，运行时系统的总体构成如下图所示。

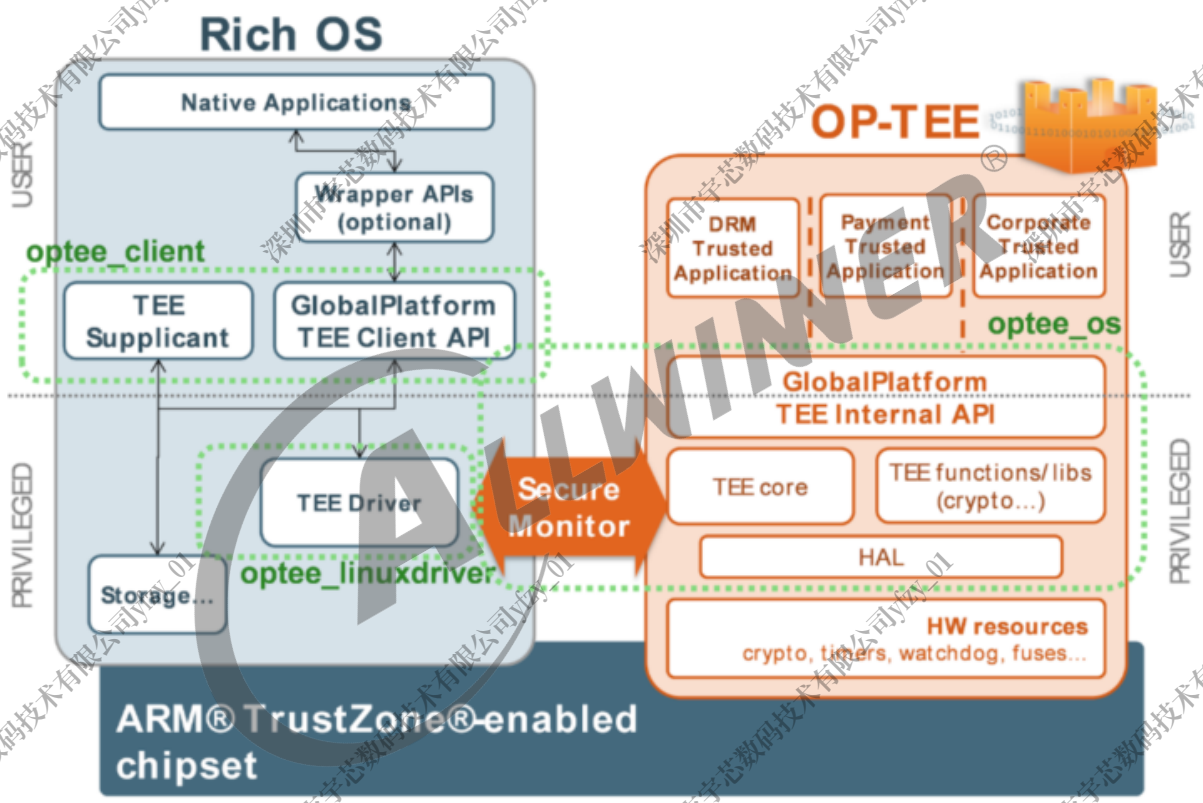

图 5-1: OPTEE 总体架构

目前 optee os 整体作为闭源代码，以镜像的方式提供给各位客户，命名方式为：optee_xxx.bin，存放的位置为：${SDK}/device/config/chips/${chip}/bin/optee_xxx.bin

不同的芯片，使用的 optee 版本略有差异，可以通过如下方式确认其版本：

```
${SDK}/device/config/chips/${chip}/bin$ strings -n3 optee_xxx.bin | head
optee
3.7    # 如果是3.7 则表示是opteee-os-3.7版本；反则如果是2.5则表示是opteee-os-2.5版本
CHKv1.0
AW_PUBK!
BootInfo
```

```
ADs
    '@
      7@
,0@
p@-
```

下面介绍使用 OPTEE OS 后，系统中新增的关键构件。

5.1.1 OPTEE OS

运行在 TEE 环境的 os。主要负责:

1. TEE 下的任务调度。
2. TEE 下的资源分配。
3. 为 TEE 下的应用程序提供系统调用，这些系统调用包含 GP 规定的 TEE internal API。

5.1.2 OPTEE linux driver

REE 环境下的程序向 TEE 环境下的程序请求服务时，需要通过 TrustZone 指定的 smc 请求的形式进行。OPTEE linux driver 封装了这些请求。REE 对 TEE 环境的请求不直接操作硬件, 通过对 OPTEE linux driver 的 IO ctrl 接口下发。

5.1.3 OPTEE client

OPTEE client 有两个用途:

- 对于 REE 向 TEE 发起的请求，GP 有规定标准接口，称为 TEE client API。OPTEE client 实现了这些接口。REE 环境下的应用程序只需要针对 TEE client API 接口进行编程即可完成对 TEE 的请求。
- OPTEE OS 在运行时需要在请求 REE 环境协助进行文件操作。OPTEE client 接收这些请求并进行处理。如 TA（trusted apps，在 TEE 环境运行的应用程序）的加载。TA 存储在 REE 环境的文件系统，需要运行某个 TA 时，OPTEE OS 请求 REE 读取 TA 的数据后进行加载（已有针对 REE 下 TA 被窜改的预防措施）。

5.1.4 TA/CA

- TA: Trusted apps，运行在 TEE 环境的应用程序，敏感信息的直接处理在 TA 进行。
- CA: Client apps，也称为 NA（Normal apps）。在 REE 环境下运行的应用程序，即传统的应用程序。

TrustZone 技术的使用基本以此形式实现：REE 下的应用程序请求 TEE 下的应用程序执行某些涉及敏感信息的操作。如验证用户密码时，REE 提供用户输入的密码，TEE 比对用户输入的密码与设置的密码是否一致。返回比对通过/不通过。这样就在 REE 完全不接触设置的密码的明文的情况下实现了对输入的密码的验证。保证了已保存密码的安全。

这些功能都需要 TA/CA 配合实现，CA 作为 client 发起请求，TA 处理。

5.2 TEE 运行环境配置

为了 TA/CA 能够正常配合工作，需要妥当配置 TEE 运行环境，包括 OPTEE OS、OPTEE linux driver、OPTEE client 等。下面逐个说明配置方法。

5.2.1 OPTEE OS

安全固件在编译时已经包含 OPTEE OS，在 secure boot 的过程中也会引导 OPTEE OS。无需额外配置。

5.2.2 OPTEE linux driver

1. 在 {SDK} 目录下输入./build.sh menuconfig进入内核配置界面，
2. 使能配置：Device Drivers --> 选中Trusted Execution Environment support --> TEE drivers ---> 选中OP-TEE。即打开如下 2 个基本配置，便可在内核编译时启用 OPTEE linux driver：

```
CONFIG_TEE
CONFIG_OPTEE
```

3. 正确启用 OPTEE linux driver 后，在板子上可以看到对应的设备节点：

```
/ # ls /dev/tee*
/dev/tee0    /dev/teepriv0
```

5.2.3 OPTEE client

- 集成 tee-supplicant 及 libteec

将下表中的文件拷贝到{SDK}/out/{chip}/{board}/bsp/rootfs_def/ 文件路径下，或者直接使用 adb push命令将如下文件直接推送到小机端。

（1）optee_os-3.7：

文件名	拷贝到
tee-supplicant	/usr/sbin/
libteec.so*	/usr/lib/ 或 /usr/lib64/

（2）optee_os-2.5：

文件名	拷贝到
tee-supplicant	/bin/
libteec.so*	/lib/

⚠ 注意

拷贝到文件系统路径后，要在 {SDK} 目录下运行./build.sh才会将上述文件打包到镜像中，然后重新烧录启动即可。

- 运行 tee-supplicant

（1）方法一：手动启动，在 linux 环境下执行：

```
tee-supplicant &
```

⚠ 注意

tee-supplicant运行后会无限循环监听 TEE 的请求，不会返回。因此请不要漏掉 ``&''。

（2）方法二：自动启动：修改 linux 初始化流程，开机时自动启动，具体方式不在此展开。

5.2.4 TEE 环境内存配置

5.2.4.1 TEE 环境的内存使用

TEE 环境使用的内存有 3 部分:

- optee os 内存（TEE_MEM）：optee_os 专用的内存。optee_os 加载到此处运行。
- 共享内存（SHARED_MEM）：用于 REE 与 TEE 的数据交换。由于 TrustZone 技术中 REE 与 TEE 的交互是通过 smc 指令进行的，但 smc 只能通过寄存器交换有限的数据，因此需要使用共享内存实现更多的数据交换。
- TA 内存堆（TA_MEM）：用于放置 TA 程序以及 TA 的堆和栈的内存空间。由 optee_os 进行分配。分配给某一个 TA 的内存只能由 TA 或 optee_os 访问，其他 TA 无法访问。

可通过如下 3 种方式，来指定上述内存段的地址与大小，**任选其一即可**。

5.2.4.2 方式 1：optee os 编译指定

在编译 optee os 时指定 **，通过 optee os 的platform_config.h里的宏：TEE_SHMEM_START，TEE_SHMEM_SIZE，TA_RAM_START和TA_RAM_SIZE来指定 TEE 使用的内存。

```
diff --git a/core/arch/arm/plat-sun50iw12p1/conf.mk b/core/arch/arm/plat-sun50iw12p1/conf.mk
 index 2ea4602..46192f5 100644
 --- a/core/arch/arm/plat-sun50iw12p1/conf.mk
 +++ b/core/arch/arm/plat-sun50iw12p1/conf.mk
 @@ -35,3 +35,5 @@
 CFG_NUM_THREADS = 2
 CFG_SUNXI_KL_SUPPORT = y
 CFG_SUNXI_MIPS_SETUP = y
+  CFG_SUNXI_FDT = y
+  CFG_EXTERN_DEFINED_MM_LAYOUT = y
diff --git a/core/arch/arm/plat-sun50iw12p1/main.c b/core/arch/arm/plat-sun50iw12p1/main.c
 index c182a6b..1683318 100644
 --- a/core/arch/arm/plat-sun50iw12p1/main.c
 +++ b/core/arch/arm/plat-sun50iw12p1/main.c
 @@ -83,7 +83,6 @@
   sunxi_hash_install();
   /*sboot did not use all sram, clean part of it should be enough*/
   memset((void *)sunxi_sram_base, 0, SRAM_SIZE);
+   sunxi_smc_config(TA_RAM_START, TA_RAM_SIZE);
 }

 void console_init(void)
diff --git a/core/arch/arm/plat-sun50iw12p1/platform_config.h b/core/arch/arm/plat-sun50iw12p1/platform_config.h
 index c58e8b5..80e5358 100644
 --- a/core/arch/arm/plat-sun50iw12p1/platform_config.h
 +++ b/core/arch/arm/plat-sun50iw12p1/platform_config.h
 @@ -71,9 +71,6 @@

 #define CFG_TEE_CORE_NB_CORE   2

-/* Full GlobalPlatform test suite requires TEE_SHMEM_SIZE to be at least 2MB */
-#define TEE_SHMEM_SIZE      0x400000
-#define TEE_SHMEM_START     (TZDRAM_BASE - TEE_SHMEM_SIZE) /* 1M for ATF */

 #define HEAP_SIZE      (128 * 1024)

@@ -94,10 +91,6 @@
 /* TA: 1M */
 #define TEE_RAM_PH_SIZE   TEE_RAM_VA_SIZE
 #define TEE_RAM_START   TZDRAM_BASE
-#define TA_RAM_START   ROUNDUP((TZDRAM_BASE + TEE_RAM_VA_SIZE), \
-                       CORE_MMU_PGDIR_SIZE)

-#define TA_RAM_SIZE    ROUNDDOWN((TZDRAM_SIZE - TEE_RAM_VA_SIZE), \
-                       CORE_MMU_PGDIR_SIZE)

 /*devices area:0x100 0000 ~ 0xA00 0000*/
 #define DEVICE0_PA_BASE      0x02000000
```

5.2.4.3 方式 2：uboot dts 配置指定

通过 uboot 的 fdt 指定：optee_os-3.7 支持通过 uboot 的 fdt 修改 memory layout，从而指定**共享内存**和 **TA 内存堆**的大小。uboot 在运行期间会将设备树里配置的内存信息传递给 optee，并传递参数给内核，预留相关的内存。此功能默认关闭，需要 uboot 的支持方能启用。启用方法如下：

1. 在 UBOOT 的 defconfig 里打开相应的宏：

```
diff --git a/configs/sun50iw12p1_defconfig b/configs/sun50iw12p1_defconfig
 index 670a2ef..d1d6fc4 100644
 --- a/configs/sun50iw12p1_defconfig
 +++ b/configs/sun50iw12p1_defconfig
 @@ -96,5 +96,6 @@
 CONFIG_SUNXI_IMAGE_VERIFIER=y
 CONFIG_SUNXI_KEYBOX=y
 CONFIG_SUNXI_DRM_SUPPORT=y
+ CONFIG_SUNXI_EXTERN_SECURE_MM_LAYOUT=y
 CONFIG_SUNXI_MIPS=y
```

2. 在 dts 中添加相关属性：

```
/* {SDK}/device/config/chips/{chip}/configs/{board}/uboot-board.dts */
/{
    firmware {
        optee {
            shm_base = <0x48400000>;     /* 0x48000000: 表示预留地址的起始地址，需根据不同IC的实际情况而定制 */
            shm_size = <0x00200000>;     /* 2MB，表示预留地址的大小，需根据不同IC的实际情况而定制 */
            ta_ram_base = <0x48700000>;  /* 0x48700000: 表示预留地址的起始地址，需根据不同IC的实际情况而定制 */
            ta_ram_size = <0x00500000>;  /* 5MB，表示预留地址的大小，需根据不同IC的实际情况而定制 */
        };
    };
};
```

```
// 0x48000000: 表示预留地址的起始地址，需根据不同IC的实际情况而定制
// 0x01000000：表示预留地址的大小，需根据不同IC的实际情况而定制

reserved-memory {
    #address-cells = <2>;
    #size-cells = <2>;
    ranges;

    bl31 { /* 新版本的内核里，此节点的名称变成了 optee_reserve */
        reg = <0x0 0x48000000 0x0 0x01000000>;
    };
};
```

- 其他版本 linux：

```
// 0x48000000: 表示预留地址的起始地址，可根据不同IC的实际情况而定制
// 0x01000000：表示预留地址的大小，可根据不同IC的实际情况而定制
```

```
/memreserve/ 0x48000000 0x01000000;
```

说明

（1）SDK 里已将上述内容配置就绪，无需再手动配置或修改。

（2）预留内存的大小一般为 SHARED_MEM + TA_MEM + TEE_MEM。

（3）若 uboot 的设备树已经配置了shm_*和ta_ram_*属性，那么 uboot 在运行时会将 SHARED_MEM + TA_MEM 的预留信息传递给内核，因此内核 dts 里仅需预留 TEE_MEM 即可。

部分 linux-4.9 平台（如 v853）也使用与 linux-5.4 相同的方式。

5.3 TEE 运行环境的使用

TEE 运行环境配置完毕后，即可开发 TA 和 CA，使用 TrustZone 所提供的强大功能了。

6 TA/CA 开发指引

6.1 tee_kit 开发环境介绍

开发环境位于{SDK}/platform/allwinner/security/optee/，目录结构如下：

文件（夹）	说明
build.sh	编译脚本
plat/	编译依赖（平台相关）
demo/	示例程序
platform_config.mk	平台配置文件
tools/	编译工具链

- build.sh使用说明：

参数	功能
-h	显示帮助消息。
-t	解包 tools 目录中的编译工具链。
clean	清除所有 demo 编译输出。
config	选择编译平台。

6.2 编译

注意：编译tee_kit之前务必使用{SDK}/build.sh config选正确对应的平台

a. cd {SDK}/platform/allwinner/security/optee/。

b. 运行./build.sh -t，解压编译工具链（只需运行一次）。

c. 运行./build.sh config，选择配置

d. 运行./build.sh，编译所有 DEMO。

6.3 拷贝

CA/TA 运行时依赖如下文件，请将其拷贝到设备端：

- optee_os-3.7：

From （{SDK}/platform/allwinner/security/optee/）	拷贝到（设备端的文件系统）
dev_kit/arm-plat-${platform}/export-ca/bin/tee-supplicant	/usr/sbin
dev_kit/arm-plat-${platform}/export-ca/exportlib/libteec.so*	/usr/lib 或 /usr/lib64

- optee_os-2.5：

From （{SDK}/platform/allwinner/security/optee/）	拷贝到（设备端的文件系统）
dev_kit/arm-plat-${platform}/export-ca/bin/tee-supplicant	任意位置，如/usr/bin
dev_kit/arm-plat-${platform}/export-ca/exportlib/libteec.so*	/lib

6.4 运行

在设备端执行如下操作：

6.4.1 运行 tee-supplicant

将 tee-supplicant 作为守护进程运行：

1. 方式 1：tee-supplicant &
2. 方式 2：tee-supplicant -d

6.4.2 运行 DEMO

6.4.2.1 DEMO1: hello_world

本 demo 展示 CA 如何调用 TA，以及 CA 如何通过共享内存向 TA 传输数据。

1. 拷贝

From （{SDK}/platform/allwinner/security/optee/）	拷贝到（设备端）
demo/optee_helloworld/out/ca/hello_world_ca	任意位置，如/usr/bin
demo/optee_helloworld/out/ta/12345678-4321-8765-9b74f3fc357c7c61.ta	/lib/optee_armtz/

2. 运行

- 不带参数运行：

```
/ # hello_world_ca
CA: init context
CA: open session
TA: TA_CreateEntryPoint()
TA: TA_OpenSessionEntryPoint()
CA: allocate memory
CA: invoke command with MSG = NULL
TA: TA_InvokeCommandEntryPoint(): nCommandID = 0x210
TA: TA_InvokeCommandEntryPoint(): Received MSG = NULL
TA: TA_CloseSessionEntryPoint()
TA: TA_DestroyEntryPoint()
CA: finish with 0
```

- 带参数运行

```
/ # hello_world_ca 1234
CA: init context
CA: open session
TA: TA_CreateEntryPoint()
TA: TA_OpenSessionEntryPoint()
CA: allocate memory
CA: invoke command with MSG = '1234'
TA: TA_InvokeCommandEntryPoint(): nCommandID = 0x210
TA: TA_InvokeCommandEntryPoint(): Received MSG = '1234'
TA: TA_CloseSessionEntryPoint()
TA: TA_DestroyEntryPoint()
CA: finish with 0
```

6.4.2.2 DEMO2: optee-secure-storage

本 demo 展示如何通过 TA 在 REE 的文件系统中创建、读、写和删除一个加密文件。

1. 拷贝

From（{SDK}/platform/allwinner/security/optee/）	拷贝到（设备端）
demo/optee-secure-storage/out/ca/demo/demo	任意位置，如/usr/bin
demo/optee-secure-storage/out/ta/2977f028-30d8-478b-975c-beeb3c134c34.ta	/lib/optee_armtz

2. 运行

- 创建文件

```
/ # demo ree_fs -c ./test
```

- 写文件（写的数据由 demo 随机生成）

```
/ # demo ree_fs -w ./test
---- Write file:test with 256 Bytes data: ----
22 93 33 bd a2 46 10 9a b5 24 8c 9c 1d ac d6 f3
6c 6a 28 7a bf 7a 20 0f 47 ba b2 ea 78 71 72 1b
06 a5 58 28 6b e8 c2 a1 0d ce bd a9 7c 95 9e 68
00 c6 e2 bf 41 82 4e 08 bc 80 f2 35 f2 65 cf 78
8a a8 a0 f6 91 e2 18 9e b2 d5 c7 2f 6b 66 17 ea
ad f9 ab ee fb 79 f7 b8 fa 6a 6d ed d0 3e 66 da
e6 86 51 78 69 69 96 9b 40 5e ca 2b 45 61 17 f2
5b 42 61 57 bb d8 8f b6 44 fd 24 94 3c 8a 6f a2
11 41 1b fa aa 31 96 6a 0f e1 96 54 43 2d c6 1f
6f a8 f5 2b 81 86 62 45 84 86 d9 40 12 c9 62 a2
0b 7d 1d b5 2e b4 a0 3d 16 37 12 59 64 d8 78 d3
01 6f 7e 83 75 e0 c8 79 68 23 b9 f9 ec 1c 1c 77
19 3a 2d c6 6e cd 04 84 84 16 5d 68 6f d6 bb 70
c5 3b 73 ba 9b bc 34 83 5f 6d fc 4c 09 1a 43 a1
d3 70 68 42 be 6c 46 c2 03 a3 ab f1 f9 67 e1 bf
22 56 f9 3e 92 2e c1 f1 9b bf bd 24 59 01 c5 2d

---- Write file:test end! ----
```

- 读文件

```
/ # demo ree_fs -r ./test
---- Read file:test 256 Bytes data: ----
22 93 33 bd a2 46 10 9a b5 24 8c 9c 1d ac d6 f3
6c 6a 28 7a bf 7a 20 0f 47 ba b2 ea 78 71 72 1b
06 a5 58 28 6b e8 c2 a1 0d ce bd a9 7c 95 9e 68
00 c6 e2 bf 41 82 4e 08 bc 80 f2 35 f2 65 cf 78
8a a8 a0 f6 91 e2 18 9e b2 d5 c7 2f 6b 66 17 ea
ad f9 ab ee fb 79 f7 b8 fa 6a 6d ed d0 3e 66 da
e6 86 51 78 69 69 96 9b 40 5e ca 2b 45 61 17 f2
5b 42 61 57 bb d8 8f b6 44 fd 24 94 3c 8a 6f a2
11 41 1b fa aa 31 96 6a 0f e1 96 54 43 2d c6 1f
6f a8 f5 2b 81 86 62 45 84 86 d9 40 12 c9 62 a2
0b 7d 1d b5 2e b4 a0 3d 16 37 12 59 64 d8 78 d3
01 6f 7e 83 75 e0 c8 79 68 23 b9 f9 ec 1c 1c 77
19 3a 2d c6 6e cd 04 84 84 16 5d 68 6f d6 bb 70
c5 3b 73 ba 9b bc 34 83 5f 6d fc 4c 09 1a 43 a1
```

```
d3 70 68 42 be 6c 46 c2 03 a3 ab f1 f9 67 e1 bf
22 56 f9 3e 92 2e c1 f1 9b bf bd 24 59 01 c5 2d

---- Read file:test end! ----
```

- 删除文件

```
/ # demo ree_fs -d ./test
```

6.5 二次开发指引

下面介绍开发新的 TA 和 CA 时如何配置依赖，由于篇幅有限，这里只介绍一些关键点，具体请参考 demo 的源码。

6.5.1 TA

6.5.1.1 目录结构

以platform/allwinner/security/optee/demo/optee_helloworld/ta为例：

```
tina-5.0/platform/allwinner/security/optee/demo/optee_helloworld/ta$ tree
.
├── include
│   └── user_ta_header_defines.h
├── Makefile
├── optee_helloworld_ta.c
└── sub.mk
```

6.5.1.2 Makefile

在目标 TA 源码根目录下创建Makefile文件，包含以下内容：

内容	说明
BINARY=$(UUID) include $(TA_DEV_KIT_DIR)/mk/ta_dev_kit.mk	$UUID$‘.ta‘‘(UUID) ‘必须与 TA 的 UUID 一致。 TA 代码编译的二进制需特殊处理，因此使用$(TA_DEV_KIT_DIR)/mk/ta_dev_kit.mk进行编译。这里的$(TA_DEV_KIT_DIR)就是{SDK}/platform/allwinner/security/optee/plat/arm-plat-{chip}/export-ta_arm32/。

例如：

```
# Makefile
BINARY = 12345678-4321-8765-9b74f3fc357c7c61
include $(TA_DEV_KIT_DIR)/mk/ta_dev_kit.mk
```

6.5.1.3 sub.mk

在 TA 源码根目录创建sub.mk，提供编译信息：

命令	作用
srcs-y += {c_filename}	将 C 文件c_filename加入编译
subdirs-y += {dirname}	包含{dirname}下的{sub.mk}
global-incdirs-y += {dirname}	编译 TA 时的头文件搜索路径
cflags-{filename}-y +=	c 文件的编译 flag。{filename}为生效的文件。若是cflags-y，则表示对所有 c 文件有效
aflags-{filename}-y +=	s 文件的编译 flag
cppflags-{filename}-y +=	c 及 s 文件的编译 flag
TA_PRIVATE_FLAGS +=	链接生成 TA 的 elf 时使用的 flag

例如：

```
# sub.mk
global-incdirs-y += include
srcs-y += optee_helloworld_ta.c
```

6.5.1.4 user_ta_header_defines.h

platform/allwinner/security/optee/demo/demo-xxx/ta/include/user_ta_header_defines.h文件提供了 TA 的基本信息。OPTEE OS 在加载 TA 时，会根据这些信息，为 TA 配置运行环境。

比如，可以通过调整TA_STACK_SIZE、TA_DATA_SIZE的值，让 OPTEE OS 为 TA 分区更多堆栈空间。

宏	作用
TA_UUID	TA 的 UUID，与$(BINARY)一致
TA_FLAGS	运行配置。使用 demo 默认值即可，完整选项 见dev_kit/arm-plat-${chip}/export-ta_arm32/include/user_ta_header.h
TA_STACK_SIZE	此 TA 使用的栈大小
TA_DATA_SIZE	此 TA 使用的堆大小

例如：

```
/* user_ta_header_defines.h */
#ifndef USER_TA_HEADER_DEFINES_H
#define USER_TA_HEADER_DEFINES_H
```

```
#define TA_UUID { 0x12345678, 0x4321, 0x8765, \
        {0x9b,0x74,0xf3,0xfc,0x35,0x7c,0x7c,0x61} }

/*
 * This is important to have TA_FLAG_SINGLE_INSTANCE && !TA_FLAG_MULTI_SESSION
 * as it is used by the ytest
 */
#define TA_FLAGS            (TA_FLAG_USER_MODE | TA_FLAG_EXEC_DDR | \
                             TA_FLAG_MULTI_SESSION)
#define TA_STACK_SIZE       (2 * 1024)
#define TA_DATA_SIZE        (32 * 1024)

#endif
```

6.5.1.5 optee_helloworld_ta.c

1. 在 ta 所在的 C 代码里包含 2 个必须的头文件：

```
#include <tee_ta_api.h>
#include <stdio.h>
```

2. 根据具体的业务需求，填充对应的回调函数，实现此头文件里所要求的 TEE TA API，已达到对应的差异化功能：

TEE TA API	功能
TEE_Result TA_CreateEntryPoint(void)	TA 启动时的回调函数
void TA_DestroyEntryPoint(void)	TA 退出时的回调函数
TEE_Result TA_OpenSessionEntryPoint(uint32_t nParamTypes, TEE_Param pParams[4], void ** ppSessionContext)	创建 session 时的回调函数
void TA_CloseSessionEntryPoint(void *pSessionContext)	关闭 session 时的回调函数
TEE_Result TA_InvokeCommandEntryPoint(void *pSessionContext, uint32_t nCommandID, uint32_t nParamTypes, TEE_Param pParams[4])	执行命令的回调

3. 示例如下：

```
/* optee_helloworld_ta.c */
#include <tee_ta_api.h>
#include <stdio.h>

/* Called each time a new instance is created */
TEE_Result TA_CreateEntryPoint(void)
{
```

```
}

/* Called each time an instance is destroyed */
void TA_DestroyEntryPoint(void)
{
  ...
}

/* Called each time a session is opened */
TEE_Result TA_OpenSessionEntryPoint(uint32_t nParamTypes,
      TEE_Param pParams[4],
      void **ppSessionContext)
{
  ...
}

/* Called each time a session is closed */
void TA_CloseSessionEntryPoint(void *pSessionContext)
{
  ...
}

/* Called when a command is invoked */
TEE_Result TA_InvokeCommandEntryPoint(void *pSessionContext,
            uint32_t nCommandID, uint32_t nParamTypes,
            TEE_Param pParams[4])
{
  ...
}
```

6.5.2 CA

6.5.2.1 目录结构

以platform/allwinner/security/optee/demo/optee_helloworld/ca为例：

```
tina-5.0/platform/allwinner/security/optee/demo/optee_helloworld/ca$ tree
.
├── Makefile
└── optee_helloworld_ca.c
```

6.5.2.2 Makefile

- 将optee_helloworld_ca.c加入源文件列表
- 将dev_kit/arm-plat-${platform}/export-ca/include加入头文件搜索路径
- 将dev_kit/arm-plat-${platform}/export-ca/exportlib加入库文件搜索路径
- 使用-lteec选项，链接动态库libteec.so

```
srcs += optee_helloworld_ca.c
...
CFLAGS += -I./
CFLAGS += -I$(OPTEE_CLIENT_EXPORT)/include
...
LDFLAGS += -L$(OPTEE_CLIENT_EXPORT)/exportlib -lteec
...
```

6.5.2.3 optee_helloworld_ca.c

1. 包含 TEE client API 标准头文件：#include <tee_client_api.h>
2. 调用 TEE client API 与 TA 交互，详见 demo。

6.5.2.4 安卓环境下 CA 编译

对于安卓固件，ca 需要使用安卓的编译环境进行编译，具体操作方法如下：

- 手动编译：（1）将demo/文件夹拷贝到安卓环境；（2）在安卓环境执行 lunch命令；（3）最后在demo/*/ca下执行mm命令
- 统一编译：（1）将 demo ca 加入安卓产品 mk 的PRODUCT_PACKAGES，package 名字见demo/*/Android.bp；（2）执行 make 编译命令即可完成编译

6.5.3 TA 加密功能

基本概念：加密功能是为了保证**保密性（Confidentiality）**：加密功能通过对数据进行加密，确保只有授权的用户能够解密和阅读数据。即使数据被未经授权的人访问，也无法理解其内容。

使用步骤：打开/关闭 TA 加密：对于支持 TA 加密的平台，在可以通过如下配置打开该功功能，在随后该 ta 被 secure os 加载时 optee os 会自动判断 TA 是否需要解密，进行相应的处理

```
${SDK}/platform/allwinner/security/optee$ ./build.sh config
encrypt TA (y/n): y

# 其中`y`、`n`代表是否对TA进行加密
```

修改配置加密秘钥：

1. 配置加密密钥：TA 加密使用通过 aes 进行，密钥长度为 128bit（16 字节），存放在${SDK}/dev_kit/arm-plat-${platform}/export-ta_arm32/keys/ta_aes_key.bin文件中。如果需要修改 TA 加密使用的密钥，请修改此 bin 文件。
2. 解密密钥源：目前支持使用两种 key 解密 TA，如下：

（1）ssk：对应using derive key TA(y/n): n，芯片使用 efuse 中的 ssk key 对 TA 进行解密。请确保ta_aes_key.bin的内容与芯片的 efuse 中的 ssk key 一致。
（2）rotpk 派生的密钥：对应using derive key TA(y/n): y芯片使用 rotpk 派生的 key 对 TA 进行解密。请使用以下命令生成 aes key bin 文件并覆盖ta_aes_key.bin：${SDK}/platform/allwinner/security/optee/plat/${platform}/export-ta-arm32/scripts/generate_ta_key.py --rotpk longan/out/${product}/common/keys/rotpk.bin --out ${out_put_file}

6.5.4 TA 签名-验签功能

基本概念：验签功能（Digital Signature）是为了保证**完整性（Integrity）和认证性（Authentication）**：验签功能通过数字签名对数据进行签名，确保数据的完整性和真实性。接收方可以使用签名来验证数据的完整性，以及确认数据的发送方是合法和可信的。

使用步骤：此功能是 TA 的基础功能，optee os 强制打开，故无需额外配置

修改配置签名秘钥：OPTEE 在运行 TA 前会对 TA 进行验签，可以使用以下工具更新签名 TA、验签 TA 使用的公钥私钥。

1. 通过 openssl 或者其他方式生成 RSA 的密钥对：openssl genrsa -out new_ta_priv_key.pem 2048
2. 生成自己的公钥：openssl rsa -in new_ta_priv_key.pem -pubout -out new_ta_public_key.pem
3. 更新 opteeos 固件中的公钥（用于 ta 加载时的验签）：

```
${SDK}$ ./platform/allwinner/security/optee/plat/arm-plat-${chip}/export-ta_arm32/scripts/update_optee_key ./device
        /config/chips/${chip}/bin/optee_xxx.bin new_ta_public_key.pem

# ./device/config/chips/${chip}/bin/optee_xxx.bin 表示SDK中欲重新签名的optee bin文件
# new_ta_public_key.pem 表示新的RSA公钥
```

4. 替换 tee_kit 环境中 TA 的签名所使用的私钥platform/allwinner/security/optee/plat/arm-plat-${chip}/export-ta_arm32/keys/default_ta.pem为目标私钥new_ta_priv_key.pem，然后在 tee_kit 环境下重新编译 TA CA，然后即可自动使用更换后的新私钥完成签名

```
# 先清理，保证环境干净
${SDK}/platform/allwinner/security/optee$ ./build.sh clean

# 再次重新编译所有TA CA 即可
${SDK}/platform/allwinner/security/optee$./build.sh

# 备注：签名成功的标志打印如下：
SIGN AND AES ENCRYPT    ${SDK}/platform/allwinner/security/optee/demo/api_demo/out/ta/17bce0c9-260c-48e8-b0dd
        -36c95d3cadba.ta
```

5. 同时我们还支持通过工具一键完成 TA 的重新签名，而无需再次完整编译 TA CA，使用范例如下：

```
${SDK}/platform/allwinner/security/optee$ ./dev_kit/arm-plat-${chip}/export-ta_arm32/scripts/resign.py --key
      new_ta_priv_key.pem --in demo/api_demo/out/ta/17bce0c9-260c-48e8-b0dd-36c95d3cadba.ta --out 17bce0c9
      -260c-48e8-b0dd-36c95d3cadba.ta

# new_ta_priv_key.pem 前文中提到的新的私钥
# demo/api_demo/out/ta/17bce0c9-260c-48e8-b0dd-36c95d3cadba.ta 表示欲被重新签名的TA
# 17bce0c9-260c-48e8-b0dd-36c95d3cadba.ta 表示重新签名后的新TA
```

现象说明：

1. 如果 TA 验签成功，则可以顺利执行 TA 中的安全操作逻辑，同时也可以通过下面的反例来反向证明验签成功；
2. 如果签名验证的秘钥不一致，导致 TA 加载失败的标志性打印：

```
CA: finish with ffff000f
```

工具说明：

1. 工具 1：更新 optee os 固件中公钥的工具

```
${SDK}/platform/allwinner/security/optee/plat/arm-plat-${chip}/export-ta_arm32/scripts/update_optee_key

# 使用办法：
${SDK}/platform/allwinner/security/optee/plat/arm-plat-sun8iw20p1/export-ta_arm32/scripts$ ./update_optee_key

update optee key for TA verifing

usage:
  update_optee_key optee_bin new_key
    optee_bin    optee bin that need update key
    new_key new   key used to verify TA, pem format
```

2. 工具 2：更新 TA 签名所使用私钥的工具

```
${SDK}/platform/allwinner/security/optee/plat/arm-plat-${chip}/export-ta_arm32/scripts/resign.py

# 使用办法：
${SDK}/t-coffer/optee_os/optee_os-3.7/scripts$ ./resign.py -h
usage: resign.py [-h] --key KEY --in INF --out OUT [--old_aes_key AES_KEYF]
           [--new_aes_key AES_KEYF_NEW]

optional arguments:
 -h, --help         show this help message and exit
 --key KEY          Name of key file
 --in INF           Name of in file
 --out OUT          Name of out file
 --old_aes_key AES_KEYF
                Name of aes key file
 --new_aes_key AES_KEYF_NEW
                Name of aes key file
```

注意，安卓固件本身已经含有预编译的 TA，如果更新密钥，请同步更新安卓的 TA，TA 路径为：

```
android/device/softwinner/common/optee_ta/
```

7 密钥存储

无论是使用密钥进行加密解密，还是使用哈希校验固件，都涉及到对密钥/哈希的保存。这些信息通过 dragonSN 工具完成烧录，并以 key 的形式烧录到设备中。

key 保存的方式有多种，每种方式的特性各不相同。下面按照 key 存储的分别介绍：

7.1 efuse 存储

保存在芯片内的 efuse 上，不可擦除。通过 sid 模块访问。efuse 中的内容分成安全和非安全两部分。访问 efuse 的读取结果视芯片状态而定。具体如下：

芯片状态	访问非安全 efuse	访问安全 efuse
安全	正常	正常
非安全	正常	返回 0

- 烧录配置

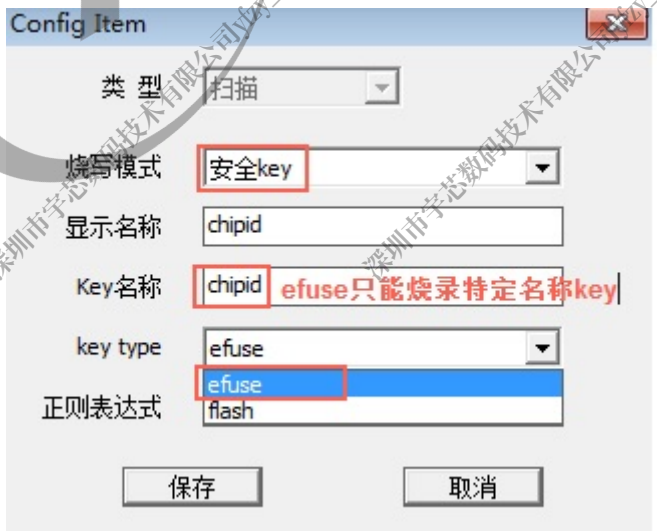

图 7-1: DragonSN 烧录安全 efuse 配置

- efuse map：efuse 空间在芯片设计时划分，只能烧录特定名称的 key。每个 key 在每个芯片 efuse 的存储位置请查阅芯片的 efuse map。

7.2 flash 存储

保存芯片外的 flash 上，根据保存的 flash 区域的访问方式，可分为安全 key 和私有 key 两种。

7.2.1 安全 key（secure storage）

保存在 flash 上，使用的扇区未映射到逻辑扇区。通常的 flash 操作无法访问，且正常量产不会被擦除。

使用烧录工具烧录时，需要按照下图方式进行配置：

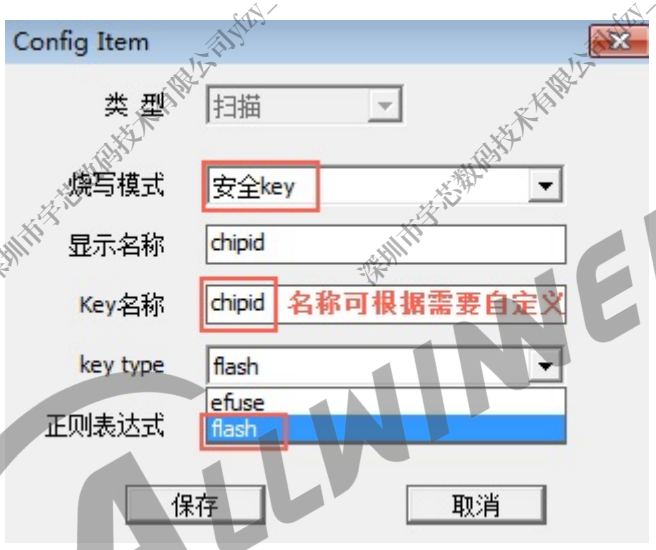

图 7-2: DragonSN 烧录安全 key 配置

7.2.1.1 keybox

keybox 位于 secure storage 中，用于保存特殊的安全 key，且这里保存的 key 不会保存明文，而是保存由 secure os 加密过的密文。（请注意，OPTEE OS 使用 efuse 中的 SSK key 来加密明文数据，为了保证数据安全，请在烧录 SSK 后进行 keybox 数据的烧录、使用。）

dragonSN 烧录工具只能指定 key 烧录到 secure storage，而 key 是否烧录到 keybox 中，由 uboot 根据 key 的名称决定。具体的实现是通过 keybox 列表。

- keybox 列表：uboot 通过环境变量keybox_list判断当前烧录的 key 是否为需要保存到 keybox 的 key，若是则在烧录、加载时做相应的处理。keybox_list环境变量位于devides/config/chips/${chip}/configs/default/env.cfg中，使用逗号分各 key。下面的例子中，烧录时名称为rsa_key或ecc_key或testkey的 key 会保存到 keybox 供 secure os 解密使用。

```
keybox_list=rsa_key, ecc_key, testkey
```

- 烧录：依旧通过 dragonSN 工具进行烧录，烧录原理如下所示：

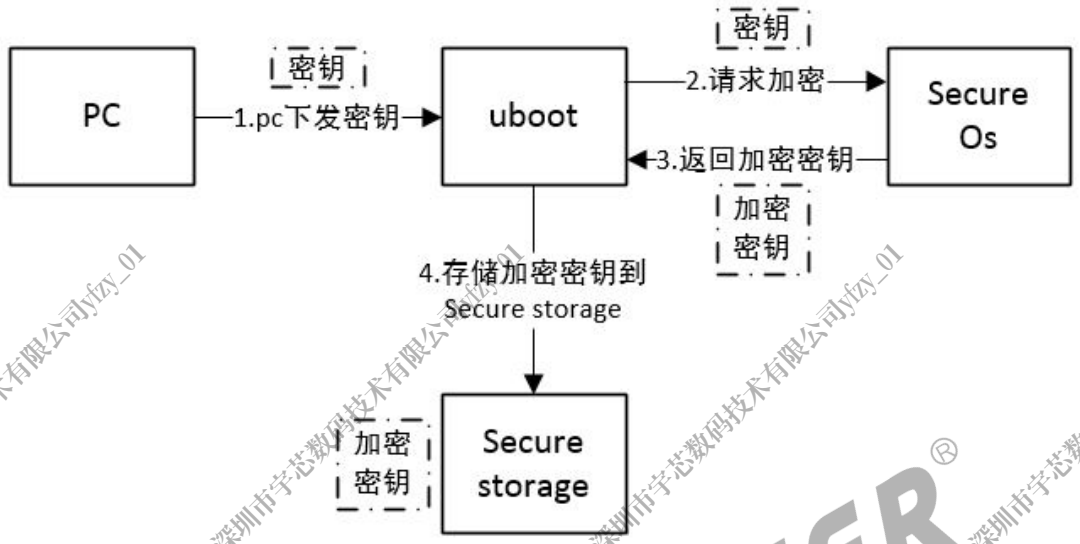

图 7-3: key 烧录到 keybox 的过程

- 上电加载：uboot 读取 key 后送到 secure os，secure os 解密后缓存，供 TA 通过接口访问。

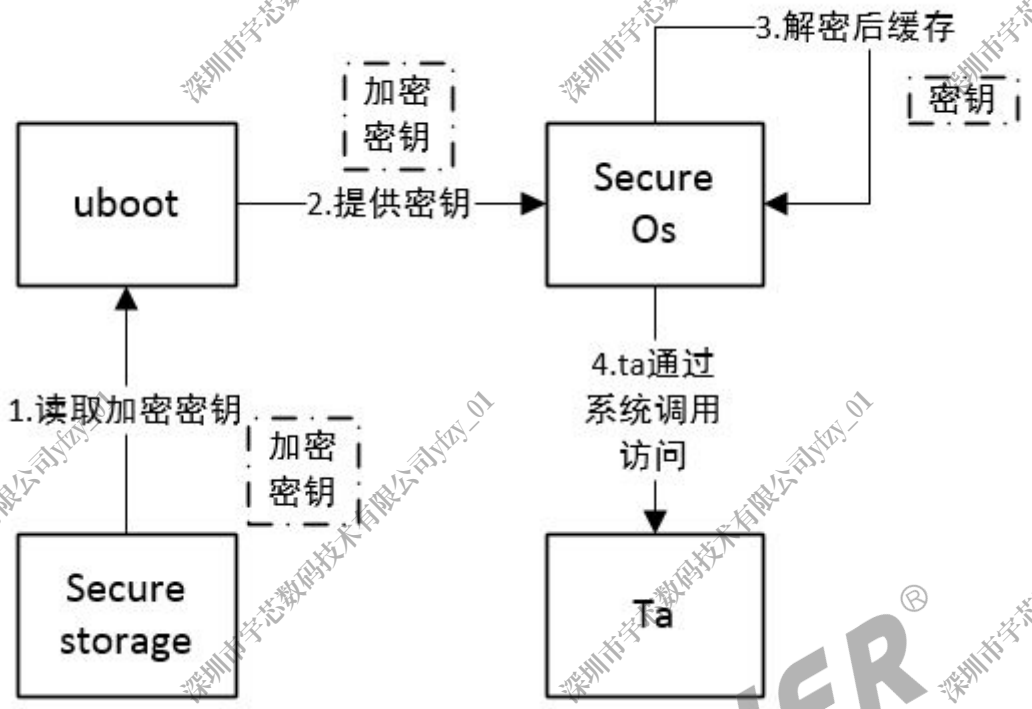

图 7-4: key 从 keybox 中加载的过程

7.2.2 私有 key(private storage)

此部分 key 保存在 private 分区。量产时会把此分区内容读取到内存，量产完毕后从内存恢复到 flash 中，达到量产后分区内容保留的目的。

添加私有分区需要在 sys_partition.fex 里添加如下内容：

```
[partition]
  name      = private
  size      = 1024
  user_type  = 0x8000
```

key 保存到文件名为 key 名称的文件中，可以通过文件系统访问到 key 的内容。

- 烧录配置：

图 7-5: DragonSn 烧录私有 key 配置

- 分区格式化：量产后 private 分区内容保持不变。故 private 分区无法通过量产进行格式化。对于未格式化的分区，烧 key 时需要进行分区格式化。

图 7-6: 格式化私有分区

- 文件路径：key 保存到文件名为 key 名称的文件中，文件路径可以在烧录时配置。

图 7-7: 私有 key 烧录路径

⚠ 注意

部分平台可能会读取特定私有 key，读取时使用的路径为烧录私有 key 的默认路径。如果修改过 key 的烧录路径，请注意同步修改涉及到的配置项。

在sys_config.fex中使用$(keyname)_filename= "$(path_to_key_file)" 的形式指定 key 的读取。

8 TEE 环境中数据的掉电保存

TrustZone 要求安全与非安全资源在硬件上进行隔离，其安全资源集成到芯片内以应对替换 pcb 上非芯片元件的攻击。

但由于成本原因，大部分设备都没有集成到芯片内部的、大容量的掉电不丢失的、可擦写存储介质（如 nand），因此 TEE 中需要掉电保存的数据都需要保存到芯片外的存储介质中。对此 TrustZone 技术规范也有相应的方案，TEE 的中的数据需要保存到芯片外存储介质上时，需要满足指定要求。optee 提供了满足该要求的实现，可以用于 TEE 环境中数据的掉电保存。

8.1 OPTEE Secure Storage 功能框架

OPTEE Secure Storage 的软件架构如下:

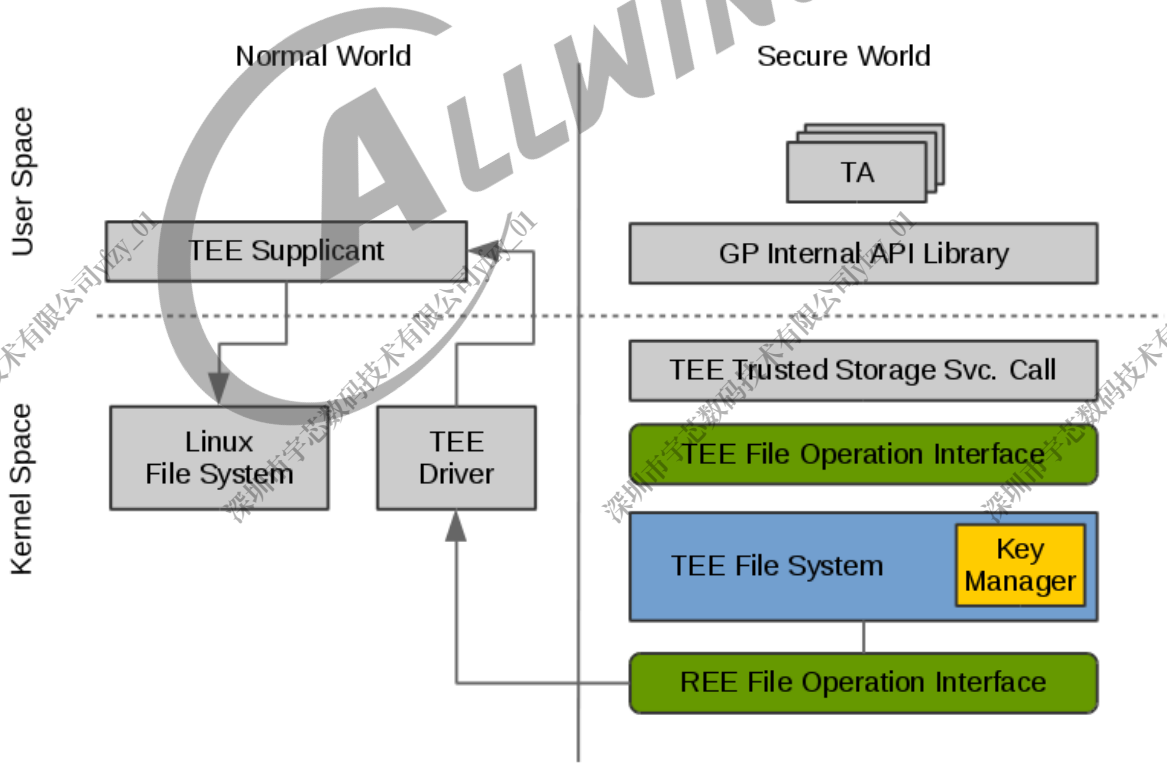

图 8-1: OPTEE Secure Storage 软件架构

8.2 文件操作流程

TA 调用 GP Trusted Storage API 提供的写接口，此接口会调用 TEE Trusted Storage Service 中的相关 syscall，陷入到 OPTEE 的 kernel space 中，该 syscall 会调用一系列的 TEE File Operation Interface 接口来存储写入的数据。

TEE 文件系统会将写入的数据进行加密，然后通过一系列的 RPC 消息向 TEE supplicant 发送 REE 文件操作命令以及已加密的数据。

TEE supplicant 对这些消息进行解析，按照参数的定义将加密的数据存放到对应的 Linux 文件系统中（默认是/data/tee目录）。

以上是对写数据的处理，对读数据的处理类似。

8.3 使用 OPTEE Secure Storage 为 REE 提供加密文件存储

配置 TA 作为服务端调用 OPTEE OS 提供的存储接口，可以让 REE 也使用此接口，完成数据的加密存储。

具体可参考 demo 中的encrypt_file_storage。

9 FAQ

9.1 运行 tee-supplicant 时报错：“failed to find an OP-TEE supplicant device”

这种情况一般是因为没有启用 Linux 内核里的 optee driver。请参考 “Optee linux driver” 章节，正确启用驱动。

9.1.1 确认节点是否存在，内核 config 是否打开

```
/ # ls /dev/tee*
/dev/tee0     /dev/teepriv0
/ # zcat /proc/config.gz | grep TEE
# CONFIG_HID_STEELSERIES is not set
CONFIG_TEE=y
# TEE drivers
CONFIG_OPTEE=y
```

9.1.2 确认 optee.bin 是否加载成功

若 OP-TEE version 未打印说明 optee.bin 未加载成功，查看 boot0 阶段 BL31 打印是否有报错，optee_base 应不为 0

如下是正常启动日志参考

```
[377]Jump to ATF: monitor_base = 0x48000000, uboot_base = 0x4a000000, optee_base = 0x48600000 #确认optee_base
    不为0
NOTICE:  BL31: OP-TEE 64bit detected
NOTICE:  BL31: U-BOOT 32bit detected
NOTICE:  BL31: dram size is 4294967296 bytes
NOTICE:  BL31: v2.5(debug):cf4a4e026
NOTICE:  BL31: Built : 20:40:18, Mar 25 2025
NOTICE:  BL31: No DTB found.
NOTICE:  cpuidle init version V2.1
I/TC:
M/TC: OP-TEE version: 67581e36 (gcc version 9.2.1 20191025 (GNU Toolchain for the A-profile Architecture 9.2-2019.12 (
    arm-9.10))) #1 Mon Apr 14 11:51:29 UTC 2025 aarch64 #正常启动会打印OP-TEE version,若无此部分打印则表示optee
    并未加载成功
M/TC: OP-TEE 64bit
E/TC:0 0 plat_rng_init:460 prng seed by trng
I/TC: Initialized
```

若 optee_base 为 0 可能是 optee.bin 没有正确打包进去

1. 确认{SDK}/device/config/chips/{chip}/bin/optee_sun*iw*p*.bin是否存在且版本正确
2. 确认打包脚本是否正确，安全固件打包配置路径：{SDK}/device/config/chips/{chip}/configs/default/dragon_toc.cfg中有如下行：

```
shell    item=optee,        optee.fex,    optee.crtpt
```

9.2 运行 ca demo 时，卡在 TEEC_OpenSession() 里无响应

这种情况一般是因为tee-supplicant没有运行。请运行tee-supplicant后重试。

9.3 运行 tee-supplicant 时报错：找不到动态库

报错如下：

```
/ # /tee-supplicant &
tee-supplicant: error while loading shared libraries: libteec.so.1: cannot open shared object file: No such file or directory
```

解决办法：请检查动态库libteec.so*是否被正确地集成在文件系统中了。动态库的路径一般为/usr/lib或/usr/lib64或/lib或/lib64。

9.4 编译 TEE Demo 时报错：no_avaliable_toolchain_found

- 报错如下：

```
Build demo 'api_demo' ...
make: Entering directory '~/tina-5.0/platform/allwinner/security/optee/demo/api_demo'
make[1]: Entering directory '~/tina-5.0/platform/allwinner/security/optee/demo/api_demo/ca'
  LD     ~/tina-5.0/platform/allwinner/security/optee/demo/api_demo/out/ca/api_demo_ca
/bin/sh: 1: no_avaliable_toolchain_found!: not found
Makefile:57: recipe for target 'api_demo_ca' failed
make[1]: *** [api_demo_ca] Error 127
make[1]: Leaving directory '~/tina-5.0/platform/allwinner/security/optee/demo/api_demo/ca'
Makefile:61: recipe for target 'ca' failed
make: *** [ca] Error 2
make: Leaving directory '~/tina-5.0/platform/allwinner/security/optee/demo/api_demo'
#### make failed to build some targets  ####
```

- 解决办法：请先执行./build.sh -t，安装编译工具链。

9.5 签名 ta 失败

- 问题现象：

```
${SDK}/platform/allwinner/security/optee$ ./build.sh demo
0. api_demo --- show how to do HAMC and AES with TEE API in TA

# 省略中间编译过程

 CPP    ${SDK}platform/allwinner/security/optee/demo/api_demo/out/ta/ta.lds
 LD     ${SDK}platform/allwinner/security/optee/demo/api_demo/out/ta/17bce0c9-260c-48e8-b0dd-36c95d3cadba.elf
 OBJDUMP ${SDK}platform/allwinner/security/optee/demo/api_demo/out/ta/17bce0c9-260c-48e8-b0dd-36
        c95d3cadba.dmp
 OBJCOPY ${SDK}platform/allwinner/security/optee/demo/api_demo/out/ta/17bce0c9-260c-48e8-b0dd-36c95d3cadba.
        stripped.elf
 SIGN AND AES ENCRYPT    ${SDK}platform/allwinner/security/optee/demo/api_demo/out/ta/17bce0c9-260c-48e8-b0dd
        -36c95d3cadba.ta
Traceback (most recent call last):
 File "${SDK}platform/allwinner/security/optee/plat/arm-plat-sun8iw20p1/export-ta_arm32/scripts/sign.py", line 269,
        in <module>
   main()
 File "${SDK}platform/allwinner/security/optee/plat/arm-plat-sun8iw20p1/export-ta_arm32/scripts/sign.py", line 253,
        in main
   img=pad(img,AES.block_size)
 File "${SDK}platform/allwinner/security/optee/plat/arm-plat-sun8iw20p1/export-ta_arm32/scripts/sign.py", line 132,
        in pad
   data = data + chr(pad) * pad

# 重点是这个报错
TypeError: can't concat str to bytes

${SDK}platform/allwinner/security/optee/plat/arm-plat-sun8iw20p1/export-ta_arm32/mk/link.mk:109: recipe for target
        '${SDK}platform/allwinner/security/optee/demo/api_demo/out/ta/17bce0c9-260c-48e8-b0dd-36c95d3cadba.ta'
        failed
make[1]: *** [${SDK}platform/allwinner/security/optee/demo/api_demo/out/ta/17bce0c9-260c-48e8-b0dd-36
        c95d3cadba.ta] Error 1
make[1]: Leaving directory '${SDK}platform/allwinner/security/optee/demo/api_demo/ta'
Makefile:67: recipe for target 'ta' failed
 make: *** [ta] Error 2
```

- 解决办法：此问题是由于在签名的工具 sign.py 中尝试将字符串与字节串进行连接而引起的。而在 Python 3 中，字符串和字节串是不同的数据类型，不能直接进行连接操作，所以在运行强制指定 sign.py 的 Python 版本为 2.7 版本即可：

```
-- a/dev_kit/arm-plat-sun8iw20p1/export-ta_arm32/scripts/sign.py
+++ b/dev_kit/arm-plat-sun8iw20p1/export-ta_arm32/scripts/sign.py
@@ -1,4 +1,4 @@
-#!/usr/bin/env python
+#!/usr/bin/env python2.7
```

9.6 运行 demo CA 提示 Derive TA key failed

阅读本文TA加密功能章节，确认 TA 加密使用的 key 是否配置正确

```
/ # /usr/bin/hello_world_ca
NA:init context
NA:open session
E/TC:? 0 decrypt_ta_data:84 Derive TA key failed
E/LD:  init_elf:259 sys_open_ta_bin(12345678-4321-8765-9b74-f3fc357c7c61)
E/TC:? 0 init_with_ldelf:237 ldelf failed with res: 0xf0100003
E/TC:? 0 tee_ta_open_session:727 Failed. Return error 0xf0100003
NA:finish with -267386877
```

9.7 运行 demo CA 提示 init_elf:259 sys_open_ta_bin

加载 TA 报错，检查：

1. 使用ps命令检查 tee-supplicant是否正在运行
2. /lib/optee_armtz/{UUID}.ta文件是否存在，且与/usr/bin/hello_world_ca版本对应（./build.sh config配置编译的 ta 和 ca 同为加密或非加密）
3. 如果最后的报错是CA: finish with ffff000f，则说明 TA 签名与验签的密钥不匹配，根据TA签名-验签功能章节更新密钥

```
/ # /usr/bin/hello_world_ca
NA:init context
NA:open session
CE/LD:  init_elf:259 sys_open_ta_bin(12345678-4321-8765-9b74-f3fc357c7c61)
E/TC:? 0 init_with_ldelf:237 ldelf failed with res: 0xffff000c
E/TC:? 0 tee_ta_open_session:727 Failed. Return error 0xffff000c
```

10 参考资料

10.1 TrustZone

- PRD29-GENC-009492C_trustzone_security_whitepaper.pdf

10.2 GlobalPlatform

- GPD_TEE_SystemArch_v1.1.pdf
- GPD_TEE_Client_API_v1.0_EP_v1.0.pdf
- GPD_TEE_Internal_Core_API_Specification_v1.1.pdf
- GPD_TEE_TA_Debug_Spec_v1.0.pdf

10.3 OPTE

- https://www.op-tee.org/documentation/
- optee_os/documentation/secure_storage.md

著作权声明

本文档及内容受著作权法保护，其著作权由珠海全志科技股份有限公司（“全志”）拥有并保留一切权利。

本文档是全志的原创作品和版权财产，未经全志书面许可，任何单位和个人不得擅自摘抄、复制、修改、发表或传播本文档内容的部分或全部，且不得以任何形式传播。

商标声明


、 
、全志科技、  （不完全列举）均为珠海全志科技股份有限公司的商标或者注册商标。在本文档描述的产品中出现的其它商标，产品名称，和服务名称，均由其各自所有人拥有。

免责声明

您购买的产品、服务或特性应受您与珠海全志科技股份有限公司（“全志”）之间签署的商业合同和条款的约束。本文档中描述的全部或部分产品、服务或特性可能不在您所购买或使用的范围内。使用前请认真阅读合同条款和相关说明，并严格遵循本文档的使用说明。您将自行承担任何不当使用行为（包括但不限于如超压，超频，超温使用）造成的不利后果，全志概不负责。

本文档作为使用指导仅供参考。由于产品版本升级或其他原因，本文档内容有可能修改，如有变更，恕不另行通知。全志尽全力在本文档中提供准确的信息，但并不确保内容完全没有错误，因使用本文档而发生损害（包括但不限于间接的、偶然的、特殊的损失）或发生侵犯第三方权利事件，全志概不负责。本文档中的所有陈述、信息和建议并不构成任何明示或暗示的保证或承诺。

本文档未以明示或暗示或其他方式授予全志的任何专利或知识产权。在您实施方案或使用产品的过程中，可能需要获得第三方的权利许可。请您自行向第三方权利人获取相关的许可。全志不承担也不代为支付任何关于获取第三方许可的许可费或版税（专利税）。全志不对您所使用的第三方许可技术做出任何保证、赔偿或承担其他义务。