Tina Linux 安全
开发指南

版本号: 2.5
发布日期: 2021.04.19

版本历史

版本号	日期	制/修订人	内容描述
1.0	2017.10.27	AWA0916	初始版本
1.1	2018.04.10	AWA0916	新增 AW 签名，新增量产工具
1.2	2018.09.26	AWA0916	新增 dm-verity，新增 secure storage
1.3	2019.01.25	AWA0916	新增 TA/CA 开发环境，测试 demo
1.4	2019.02.25	AWA0916	新增 keybox，TA 加密，efuse 烧写注意事项
1.5	2019.04.19	AWA0916	新增 uboot 检验 rootfs，rotpk 烧写方法
1.6	2019.07.17	AWA0916	新增 TA 签名，更新 uboot 校验 rootfs
1.7	2019.08.02	AWA0916	更新 dragonsn 烧写 rotpk 说明
1.8	2019.09.20	AWA0916	完善 TA 加密
1.9	2020.02.24	AWA0916	新增 MR813，新增 dm-crypt
2.0	2020.04.01	AWA0916	新增 optee-3.7，新增 RPMB 安全存储
2.1	2020.04.17	AWA0916	新增 R329，新增 ubi 下 dm-crypt 支持
2.2	2020.05.25	AWA0916	新增 R329 TA 加密，新增 MR813 TA 加密、用户空间写 keybox
2.3	2020.06.30	AWA0916	新增 R818，调整 keybox 到 Secure Storage
2.4	2020.10.28	AWA0916	调整文档格式
2.5	2021.04.19	AWA0916	新增 R528 平台支持，新增 key 说明，新增各平台安全功能支持说明

目　录

1 概述 1

1.1 编写目的 . . . 1
1.2 适用范围 . . . 1
1.3 相关人员 . . . 2
1.4 配置文件 . . . 2

2 安全系统基础 4

2.1 安全系统介绍 . . . 4
2.2 密码学基础介绍 . . . 4
2.2.1 数据加密模型 . . . 4
2.2.2 加密算法 . . . 4
2.2.3 签名与证书 . . . 5
2.3 TrustZone . . . 6
2.3.1 OP-TEE . . . 7
2.3.2 ARM Trusted Firmware . . . 7
2.4 硬件安全模块 . . . 7
2.4.1 SPC . . . 7
2.4.2 SMC . . . 8
2.4.3 SID . . . 8
2.4.4 efuse . . . 8
2.4.5 CE . . . 8
2.4.6 TZMA . . . 8
2.5 相关术语 . . . 8

3 Secure Boot 10

3.1 安全启动原理 . . . 10
3.2 生成安全固件 . . . 10
3.2.1 安全固件配置 . . . 11
3.2.1.1 内核镜像格式配置 . . . 11
3.2.1.2 安全世界内存配置 . . . 11
3.2.2 签名密钥 . . . 12
3.2.3 安全固件版本管理 . . . 12
3.3 开启安全启动 . . . 13
3.4 烧写 rotpk.bin 与 secure enable bit . . . 14
3.4.1 方法一 . . . 14
3.4.1.1 DragonSN 烧写 efuse 流程 . . . 14
3.4.1.2 DragonSN 烧写 rotpk.bin 步骤 . . . 15
3.4.2 方法二 . . . 16
3.4.3 方法三 . . . 16
3.4.3.1 API 说明 . . . 16
3.4.3.2 开启方法 . . . 18
3.4.3.3 使用例子 . . . 18

3.5 校验 rootfs . . . 18
3.5.1 uboot 校验 rootfs . . . 19
3.5.1.1 uboot 校验 squashfs rootfs 功能实现 . . . 19
3.5.1.2 uboot 校验 squashfs rootfs 开启 . . . 19
3.5.2 dm-verity 机制 . . . 20
3.5.2.1 Initramfs 构建 . . . 21
3.5.2.2 dm-verity 启用 . . . 22
3.5.2.3 dm-verity 测试 . . . 23
3.5.2.4 dm-verity 影响 . . . 23
3.6 安全启动代价 . . . 24
3.6.1 启动时间增加 . . . 24
3.6.2 ota 升级的变化 . . . 24

4 Secure OS 25
4.1 optee 总体框架 . . . 25
4.2 开启 Secure OS . . . 26
4.2.1 Secure OS 镜像 . . . 26
4.2.2 内核支持 optee 驱动 . . . 26

5 TA/CA 开发环境 27
5.1 TA/CA 开发环境使用 . . . 27
5.2 TA/CA 开发及编译 . . . 28
5.3 TA 签名 . . . 28
5.4 TA 加密 . . . 29
5.5 安全应用 demo . . . 30
5.5.1 optee-helloworld 效果 . . . 30
5.5.2 optee-efuse-read 效果 . . . 30

6 Secure Storage 31
6.1 keybox Secure Storage . . . 31
6.1.1 keybox 烧写及读取流程 . . . 31
6.1.1.1 DragonSN 烧写 keybox . . . 31
6.1.1.2 keybox_na 烧写 keybox . . . 32
6.1.1.3 keybox 读取流程 . . . 32
6.1.1.4 keybox 列表 . . . 33
6.1.2 DragonSN 烧写 efuse 与 keybox 的配置 . . . 33
6.2 OP-TEE Secure Storage . . . 34
6.2.1 OP-TEE REE FS Secure Storage . . . 35
6.2.1.1 REE FS Secure Storage 功能框架 . . . 35
6.2.1.2 REE FS Secure Storage 文件操作流程 . . . 35
6.2.1.3 REE FS Secure Storage 密钥管理 Key Manager . . . 35
6.2.1.4 REE FS Secure Storage Meta Data 加密流程 . . . 36
6.2.1.5 REE FS Secure Storage Block data 加密流程 . . . 37
6.2.2 OP-TEE RPMB Secure Storage . . . 37

6.2.2.1 RPMB Secure Storage 功能框架 . . . 37
6.2.2.2 RPMB Secure Storage 密钥管理与加解密 . . . 37
6.2.2.3 RPMB Secure Storage 功能启用 . . . 38
6.2.2.4 RPMB 调试工具 . . . 38
6.2.3 Tina OP-TEE Secure Storage demo . . . 39
6.2.3.1 Tina OP-TEE Secure Storage TA . . . 39
6.2.3.2 Tina OP-TEE Secure Storage Library . . . 39
6.2.3.3 Tina OP-TEE Secure Storage Demo . . . 42
6.2.4 Tina OP-TEE Secure Storage 开启 . . . 42
6.2.4.1 OP-TEE Secure Storage 配置 . . . 42
6.2.4.2 编译安全固件 . . . 43
6.2.5 OP-TEE Secure Storage 使用 . . . 43
6.3 dm-crypt Seucre Storage . . . 44
6.3.1 Tina dm-crypt . . . 45
6.3.1.1 dm-crypt 配置 . . . 45
6.3.1.2 dm-crypt 使用 . . . 46
6.3.1.3 dm-crypt key . . . 48

7 SELinux 49
7.1 基本概念 . . . 49
7.1.1 主体 Subject . . . 49
7.1.2 客体 Object . . . 49
7.1.3 安全上下文 Secure Context . . . 49
7.1.4 策略 Policy . . . 49
7.1.5 SELinux 的运行模式 . . . 50
7.2 LSM 框架 . . . 50
7.3 Tina SELinux 开启 . . . 51
7.3.1 menuconfig 配置 . . . 51
7.3.2 kernel_menuconfig 配置 . . . 52
7.3.3 SELinux 初始化 . . . 52
7.4 策略开发 . . . 53
7.4.1 限制主体的权限 . . . 54
7.4.1.1 fork_test 源代码 . . . 54
7.4.1.2 添加策略 . . . 54
7.4.1.3 测试 . . . 56
7.4.2 限制客体的访问权限 . . . 56
7.4.2.1 sunxi_info 源代码 . . . 56
7.4.2.2 添加策略 . . . 57
7.4.2.3 测试 . . . 58

8 量产工具 59
8.1 RSA 密钥对生成工具 . . . 59
8.2 安全固件版本管理 . . . 59

8.3 数据封包工具 59
8.4 烧 key 工具 59
8.5 关闭 jtag 60
8.6 密钥说明 60
8.6.1 固件签名密钥 60
8.6.2 efuse 中密钥 60
8.6.3 dm-verity 密钥 61
8.6.4 TA 签名密钥 62
8.6.5 TA 加密密钥 62
8.6.6 dm-crypt 密钥 62
8.6.7 rpmb 密钥 63

9 参考资料 64

9.1 TrustZone 64
9.2 GlobalPlatform 64
9.3 OP-TEE 64
9.4 Dm-verity 64
9.5 SELinux 64

插　图

2-1 数据加密模型 4
2-2 对称/非对称加密算法 5
2-3 SHA256 算法 5
2-4 数字签名与认证 5
2-5 数字证书 6
2-6 TrustZone 模型 6
3-1 dragon-toc 配置文件说明 12
3-2 DragonSN 烧写 efuse 流程 15
3-3 rotpk 烧写配置 15
3-4 烧写 rotpk 的 API 源码 17
3-5 dm-verity 验证 rootfs 流程 20
4-1 optee 总体架构 25
4-2 内核预留内存 26
5-1 编译选项设置 28
6-1 keybox 烧写流程 32
6-2 keybox 读取流程 32
6-3 efuse key 烧写参考配置 33
6-4 keybox key 烧写参考配置 34
6-5 OP-TEE REE FS Secure Storage 软件架构 35
6-6 Meta Data 加密流程 36
6-7 Block Data 加密流程 37
6-8 RPMB Secure Storage 软件框架 37
6-9 dm-crypt 架构 45
7-1 SELinux 决策流程 50
7-2 禁止特定主体的 fork 权限 55
7-3 selinux-fork 测试 log 56
7-4 selinux-file 测试 log 58

1 概述

1.1 编写目的

介绍 TinaLinux 下安全方案的功能。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）、安全存储（Secure Storage）、安全应用（Trust Application）、完整性保护（Dm-Verity）、强制访问控制（MAC）等方面。

1.2 适用范围

适用于基于硬件平台: 全志 R18/R30/R311/MR133/R328/MR813/R329/R818/R528 芯片。

软件平台: Tina V3.5 及其后续版本。

功能	R18	R30	R311/MR133	R328	R329	MR813/R818/R528
安全启动	√	√	√	√	√	√
uboot 校验 rootfs	×	×	×	√	√	√
dm-verity	×	×	×	√	√	√
optee os	√	√	√	√	√	√
TA/CA 开发环境	√	×	√	√	√	√
optee-2.5.0	√	×	√	√	×	×
optee-3.7.0	×	×	×	×	√	√
TA 签名	√	×	√	√	√	√
TA 加密	×	×	×	√	√	√
keybox 安全存储	√	×	√	√	√	√
keybox_na 工具	×	×	×	×	×	√
OPTEE REE 安全存储	√	×	×	√	√	√
OPTEE RPMB 安全存储	×	×	×	×	×	√
dm-crypt 安全存储	×	×	√	√	√	√
SELinux	×	×	×	√	√	√

1.3 相关人员

适用于 TinaLinux 平台的客户及相关技术人员。

1.4 配置文件

本文涉及到一些配置文件，在此进行说明。

⚠ 警告

新 SDK 配置文件优先级高于旧 SDK 文件优先级。

- env*.cfg配置文件路径：

```
tina/device/config/chips/<chip>/configs/<board>/env.cfg  #新SDK，优先级高
tina/device/config/chips/<chip>/configs/<board>/linux/env-<kernel-version>.cfg  #新SDK，优先
    级中
tina/device/config/chips/<chip>/configs/default/env.cfg  #新SDK，优先级低
tina/target/allwinner/<board>/configs/env-<kernel-version>.cfg #旧SDK，优先级最低
```

- sys_config.fex路径：

```
tina/device/config/chips/<chip>/configs/<board>/sys_config.fex  #新SDK
tina/target/allwinner/<board>/configs/sys_config.fex #旧SDK，优先级最低
```

- uboot-board.dts路径：

```
tina/device/config/chips/<chip>/configs/<board>/uboot-board.dts
```

说明

如果存在 uboot-board.dts，uboot 会使用 uboot-board.dts 中配置。不存在 uboot-board.dts，uboot 会使用 sys_config.fex 中的配置。

- dragon_toc*.cfg配置文件路径：

```
tina/device/config/chips/<chip>/configs/default/dragon_toc*.cfg  #新SDK，优先级高
tina/device/config/common/sign_config/dragon_toc*.cfg  #新SDK，优先级低
tina/target/allwinner/<chip>-common/sign_config/dragon_toc*.cfg  #旧SDK，优先级高
tina/target/allwinner/generic/sign_config/dragon_toc*.cfg  #旧SDK，优先级低
```

- version_base.mk配置文件路径：

```
tina/device/config/chips/<chip>/configs/default/version_base.mk  #新SDK，优先级高
tina/device/config/common/version/version_base.mk  #新SDK，优先级低
tina/target/allwinner/<chip>-common/version/version_base.mk  #旧SDK，优先级高
tina/target/allwinner/generic/version/version_base.mk  #旧SDK，优先级低
```

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。

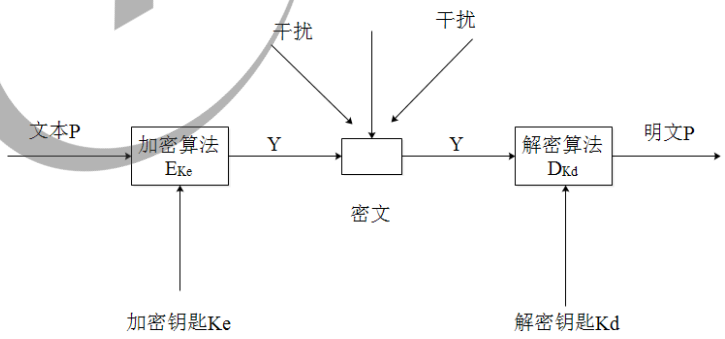

图 2-1: 数据加密模型

2.2.2 加密算法

对称加密算法：加密、解密用的是同一个密钥。比如 AES 算法。

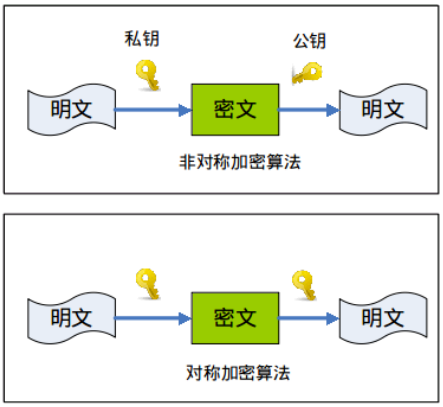

图 2-2: 对称/非对称加密算法

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。

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。

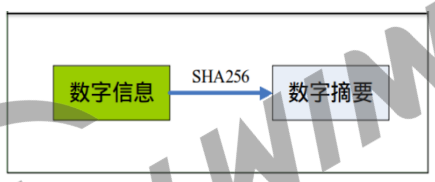

图 2-3: SHA256 算法

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。

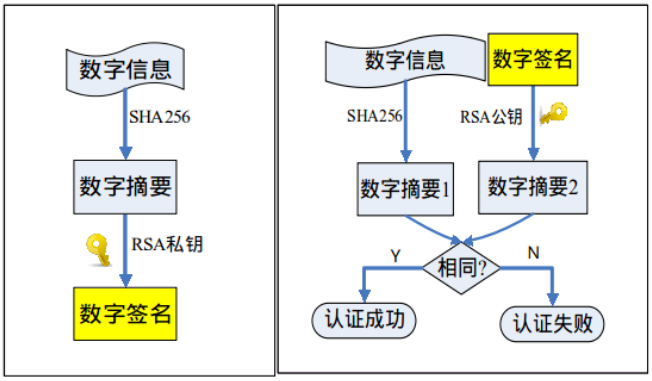

图 2-4: 数字签名与认证

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。

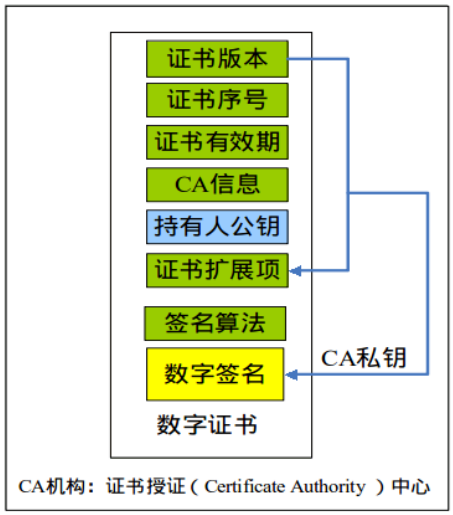

图 2-5: 数字证书

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。

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》。

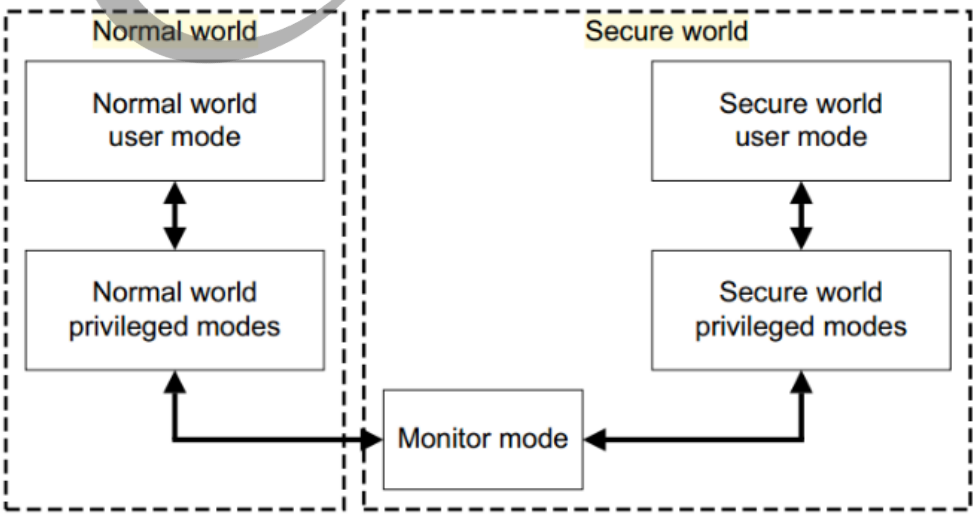

图 2-6: TrustZone 模型

2.3.1 OP-TEE

运行在安全世界的系统称为安全操作系统。

很多机构基于 TrustZone 推出了自己的安全操作系统，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。

当前 Tina 中采用的是 OP-TEE 安全系统。OP-TEE 是 Linaro 联合其他公司合作开发的基于 ARM TrustZone 技术实现的 TEE 方案，遵循 GP 标准，主要由三部分组成：

- OP-TEE client (optee_client)：运行在非安全世界用户空间的客户端 API。
- OP-TEE Linux Kernel device driver (optee_linuxdriver)：用以控制非安全世界用户空间和安全世界通信的设备驱动。此部分代码在 Linux-4.9 mainline 上已经包含。
- OP-TEE Trusted OS (optee_os)：运行在安全世界的可信操作系统。

2.3.2 ARM Trusted Firmware

ARM Trusted Firmware(ATF) 是 ARM 官方提供的安全世界软件的参考实现。它统一了 ARM 底层接口标准，包括电源状态控制接口 (Power Status Control Interface, PSCI)，安全启动需求 (Trusted Board Boot Requirements, TTBR)，安全监控模式调用 (Secure Monitor Call, SMC) 等。它还提供了 ARMv8 架构下 Exception Level 3(EL3) Secure Monitor 的参考实现。

说明

***AW ARM 64** 位平台使用 **ATF** 中的 **bl31** 作为 **Secure Monitor** 实现，**AW ARM 32** 位平台使用 **OP-TEE** 中的 **Secure Monitor** 实现。*

2.4 硬件安全模块

ARM TrustZone 技术要求安全非安全使用独立的外设资源。在 Tina SOC 系列方案中，我们设计了相关硬件模块来控制资源的安全属性。

2.4.1 SPC

Secure Peripherals Control，配置外设的安全属性，只有在安全环境才可以使用该模块。某外设被设定为安全后，该外设只有在安全世界下才能正常访问，非安全世界写无效，读为 0 。

2.4.2 SMC

这里指的是 Secure Memory Control（注意与 ARM 指令 Secure Monitor Call 区分开），配置内存地址的安全属性，只有在安全环境才可以使用该模块。某地址空间的内存被设定为安全后，该空间的内存只有安全世界可访问，非安全世界写无效，读为 0。

2.4.3 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，非安全世界访问的结果为 0。

2.4.4 efuse

efuse：一次性可编程熔丝技术，是一种 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。efuse 中区域的划分详见各 SOC 的 SID spec。

2.4.5 CE

Crypto Engine，硬件加解密加速引擎。支持多种对称加密、非对称加密、摘要以及随机数生成算法等。具体见各 SOC 的 CE spec。

2.4.6 TZMA

TrustZone Memory Access，TZMA 是配置存储器存在安全区域的控制模块。

目前仅 R528 包含 TZMA 模块，用于配置 SRAM 区域的安全属性。

2.5 相关术语

- SMC：Secure Monitor Call，ARM 给出的一条指令，可以让 CPU 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。
- PTA：Pesudo Trusted Apps，伪 TA，OPTEE 中的一个概念，表明该 TA 被集成到了 OPTEE OS 中。
- NA：Normal Apps，或称为 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。
- PRNG：Pesudo Random Number Generator，伪随机数生成器。
- TRNG：True Random Number Generator，真随机数生成器。
- RPMB：Replay Protected Memory Block，是 eMMC 中的一个具有安全特性的分区。

3 Secure Boot

Secure Boot，即安全启动，是一个**安全系统必不可少**的组成部分，是本文后续安全功能的基础。通常来说，Secure Boot 从 brom 执行开始，到 Linux 启动结束。Secure Boot 主要设计目的：

- 建立完整的安全信任链，确保启动阶段加载的各种镜像是可信的。
- 相关 key 的烧写。
- 安全固件版本管理。
- 设置安全的硬件环境，加载并运行 Secure OS 等。

3.1 安全启动原理

Tina 安全方案基于私钥签名-公钥验签的业界公认非对称算法实现完整的安全启动方案，具体来说，选择的是 RSA2048-SHA256。

先使用私钥给固件进行签名生成安全固件，再将根密钥公钥的 SHA256 值即 rotpk.bin 烧写至芯片中 efuse 特定区域。启动时，固化在芯片的 brom 程序首先会读取 efuse 中的 rotpk 值，将该值与保存在 flash 上的根证书中公钥进行 SHA256 运算后的值进行比对，验证根证书中公钥的可信任性。然后会使用 flash 上存储的证书链中的一系列公钥来对各个子镜像进行逐级安全校验。验证顺序为 brom->sboot->monitor(仅 aarch64)->secure_os->uboot->kernel。efuse 的不可更改性确保了证书链的可信任，整个流程的设计确保了整个 Linux 方案的安全启动。

3.2 生成安全固件

Tina SDK 已经将安全固件制作流程中密钥的生成和必要的签名过程集成在打包脚本内部，所以安全固件的编译及打包流程与非安全固件的几乎一致，只是在最后的打包的时候有差异。非安全固件的打包可参考用户《TinaLinux SDK 开发指南》文档，安全固件的打包步骤如下：

```
$ source build/envsetup.sh
==> 设置环境变量。
$ lunch
==> 选择方案。
$ make [-jN]
==> 编译，-jN 参数选择并行编译进程数量。
$ ./scripts/createkeys
==> 生成一组用于签名的密钥，不需要每次执行，详见3.2.2小节。生成的密钥路径位于out/{BOARD}/keys/。
$ pack -s [-d]
```

```
==> 打包固件。-s 表示制作安全固件；-d 表示生成的固件包串口信息转到tf卡座输出（可选）。
```

后续几个小节将对安全固件生成过程中一些注意事项进行说明。

3.2.1 安全固件配置

在执行 make 进行编译前，请确保包含如下配置。

3.2.1.1 内核镜像格式配置

执行 make menuconfig，确保如下选项配置正确。

```
Tina Configuration
└─> Target Images
    └─> [ ] Build filesystem for Boot (SD Card) partition
    └─> Boot (SD Card) Kernel format (boot.img)
    └─> [ ] Build filesystem for Boot-Recovery initramfs partition
    └─> Boot-Recovery initramfs Kernel format (boot.img)
```

3.2.1.2 安全世界内存配置

安全世界独享一片内存，因此在 Linux 中需要将这一片内存设为保留内存。安全世界使用的内存在 secure_os 镜像编译时就确定了，但是 secure_os 没有开源，因此保留内存具体的地址与大小，请咨询 AW 安全接口。

```
diff --git a/arch/arm/boot/dts/sun8iw18p1.dtsi b/arch/arm/boot/dts/sun8iw18p1.dtsi
index 589466f..90c4131 100644
--- a/arch/arm/boot/dts/sun8iw18p1.dtsi
+++ b/arch/arm/boot/dts/sun8iw18p1.dtsi
@@ -5,7 +5,7 @@
  */

 /* optee used */
-/memreserve/ 0x41a00000 0x00100000; /* optee range   : [0x41a00000~0x41b00000], size = 1M
    */
+/memreserve/ 0x41900000 0x00400000; /* optee range   : [0x41900000~0x41D00000], size = 4M
    */

 #include <dt-bindings/interrupt-controller/arm-gic.h>
 #include <dt-bindings/gpio/gpio.h>
```

以 R328 为例，假设安全世界使用了 4M 内存（Shmem 1M，secure os 1M，TA 2M），需按照如上补丁修改 tina/lichee/linux-4.9/arch/arm/boot/dts/sun8iw18p1.dtsi 文件。

3.2.2 签名密钥

⚠ 警告

客户在首次进行安全固件打包之前，必须运行一次./scripts/createkeys 创建自己的签名密钥，并将创建的秘钥妥善保存。每次执行 createkeys 后都会生成新的密钥，因此不用每次都执行，除非需要更换密钥。

createkeys 脚本会根据dragon_toc*.cfg生成一组用于签名的密钥，生成的密钥保存在out/{BOARD}/keys/目录下。执行 pack -s 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。

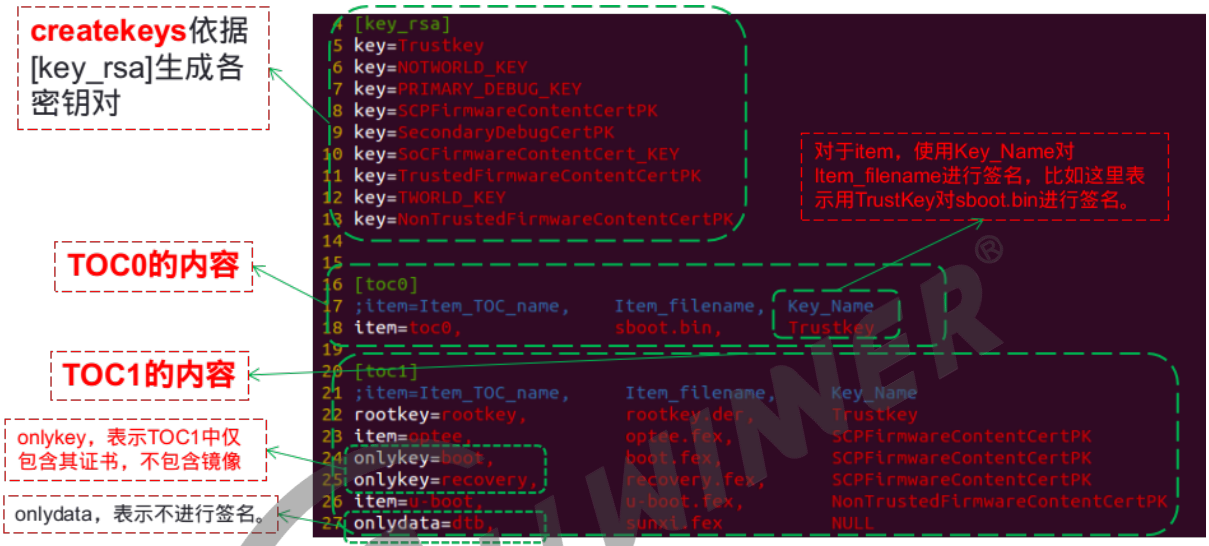

图 3-1: dragon-toc 配置文件说明

以 R328 为例，其dragon_toc*.cfg文件内容如上图所示。createkeys 依据 [key_rsa] 下的 key-value 生成密钥对。打包过程中会将 sboot.bin 封装成 toc0.fex，将 optee/uboot/dts 等封装成 toc1.fex。

请将生成的密钥保存到自己的私密目录，其中 Trustkey.bin，Trustkey.pem 与 rotpk.bin 三个文件（有的方案为 RootKey_Level_0.bin, RootKey_Level_0.pem 与 rotpk.bin）为根密钥相关文件，要重点保护。

Trustkey.bin 与 Trustkey.pem（RootKey_Level_0.bin 与 RootKey_Level_0.pem）是根密钥私钥，不能泄漏和丢失。丢失与泄露会导致一系列问题，比如：生成的安全固件无法在芯片上启动、失去防刷机功能等。

3.2.3 安全固件版本管理

注：pack -s 打包完成后，生成的安全镜像位于out/{BOARD}/keys/目录下，文件名为tina_{BOARD}_<uart0/card0>_secure_v[NUM].img。其中 NUM 为固件版本号，由version_base.mk文件决定。

Tina 提供了一种安全固件防回退机制，具体实现是：在设备启动过程中会比较当前 flash 上固件版本与 efuse 中版本信息，如果 efuse 中版本信息更高，启动失败；如果 flash 上固件的版本更高，将此版本信息写入 efuse 中，继续启动；如果版本信息一致，正常启动。

说明

最多支持更新 *32* 个版本。另外此功能需要烧写 *efuse*，所以务必保证 *efuse* 供电充足。对于部分方案 *(R328)*，由于考虑硬件成本，出厂的设备 *efuse* 供电不足，此功能不可用。

3.3 开启安全启动

完全开启安全启动**共需三个前提**：

1. 烧写 efuse 中的 secure enable bit。
2. 烧写 rotpk.bin 到 efuse 中 rotpk 区域。
3. 烧写安全固件到 flash 中。

⚠ 警告

- **不同的 IC，efuse 大小不同。efuse 的硬件特性决定了 efuse 中每个 bit 仅能烧写一次。此外，efuse 中会划分出很多区域，大部分区域也只能烧写一次。详细请参考芯片 SID 规范。**
- **烧写 secure enable bit 后，会让设备变成安全设备，此操作是不可逆的。后续将只能启动安全固件，启动不了非安全固件。**
- **默认情况下，通过 LiveSuit/PhoenixSuit 烧写安全固件完成时会自动烧写 secure enable bit。**
- **如果既烧写了 secure enable bit，又烧写了 rotpk.bin，设备就只能启动与 rotpk.bin 对应密钥签名的安全固件；如果只烧写 secure enable bit，没有烧写 rotpk.bin，此设备上烧写的任何安全固件都可以启动。调试时可只烧写 secure enable bit，但是设备出厂前必须要烧写 rotpk.bin。**
- **为节省成本，某些硬件（R328）方案上 efuse 供电不足，导致不能写入。因此，在所有需要写 efuse 操作的时候，请注意给 efuse 供电。通常在烧写安全固件、升级 sboot/uboot、DragonSN 烧 key 到 efuse 等场景会写 efuse。**

如何判断 secure enable bit 是否烧写?

- 因为只有 secure enable bit 烧写后才能启动安全固件，所以如果是安全启动，secure enable bit 就一定烧写了。安全启动过程中有一些特有的打印，如 “SBOOT is starting!”、“sboot commit...”、“OLD version:...”、“NEW version: ...”、“secure enable bit: 1” 等等，可用来进行判断。
- 执行`cat /sys/class/sunxi_info/sys_info`，如果输出的结果中 sunxi_secure 为 secure，则表明 secure enable bit 已经烧写。

如何判断 rotpk.bin 是否烧写?

- 执行`cat /proc/cmdline`，查看输出结果中的 rotpk_status 值，如果为 1 表明已经烧写。注：当前仅 R328/MR813/R818 有开发支持。需要 uboot 配置中打开 SID_ROTPK_CTRL，同时 env 文件中`setargs_nor`，`setargs_nand`，`setargs_mmc`的定义中包含`rotpk_status=${rotpk_status}`。
- 反证法。烧录使用其他 key 签名的安全固件（安全版本号一致），如果不能启动，则表明已经烧写 rotpk。
- 执行`cat /sys/class/sunxi_info/sys_info`，如果输出的结果中 sunxi_rotpk 为 1，则表明 rotpk.bin 已经烧写。仅 R329/R818/MR813/R528 支持。

3.4 烧写 rotpk.bin 与 secure enable bit

3.4.1 方法一

方法一为通用方法，所有 IC 都支持，主要包含两个步骤：

1. 使用 LiveSuit/PhoenixSuit 烧写安全固件，安全固件烧写完毕时自动烧写 efuse 中的 secure enable bit 位。
2. 使用 DragonSN 工具将 rotpk.bin 烧写到设备的 efuse 中。

DragonSN 是 AW 开发的 PC 端烧 key（SN 号、MAC 地址、rotpk 等）工具，可以将 key 烧录到 private 分区、efuse 或 keybox 中，当前仅支持在 windows 上运行。DragonSN 与设备之间通过 USB 通信，控制设备烧录配置好的 key 信息。

方法一的优缺点：

- 优点：所有 IC 都支持；方便调试；
- 缺点：需要使用 Windows 端工具；量产时通常需要两个工位。

3.4.1.1 DragonSN 烧写 efuse 流程

DragonSN 烧写 efuse 流程如下图所示。

uboot 获取到 DragonSN 下发的 key 数据，将其传送到 ATF（aarch64）或者 Secure OS（arm32），ATF 或者 Secure OS 调用 efuse 驱动将 key 数据写入到 efuse 中。

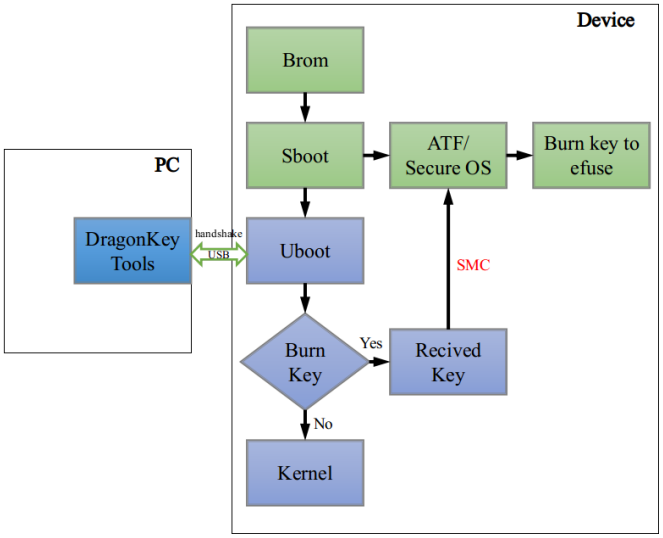

图 3-2: DragonSN 烧写 efuse 流程

3.4.1.2 DragonSN 烧写 rotpk.bin 步骤

DragonSN 烧 rotpk.bin 具体步骤如下：

- 设置 burn_key 属性为 1。只有 burn_key 的值为 1，设备才会接收 DragonSN 通过 usb 传过来的信息，进行烧录动作。该属性位于`uboot-board.dts`或者`sys_config.fex`文件中 [target] 项下。如果未显式配置，按照 burn_key=0 来处理。
- 打包安全固件，烧写到 flash 中。

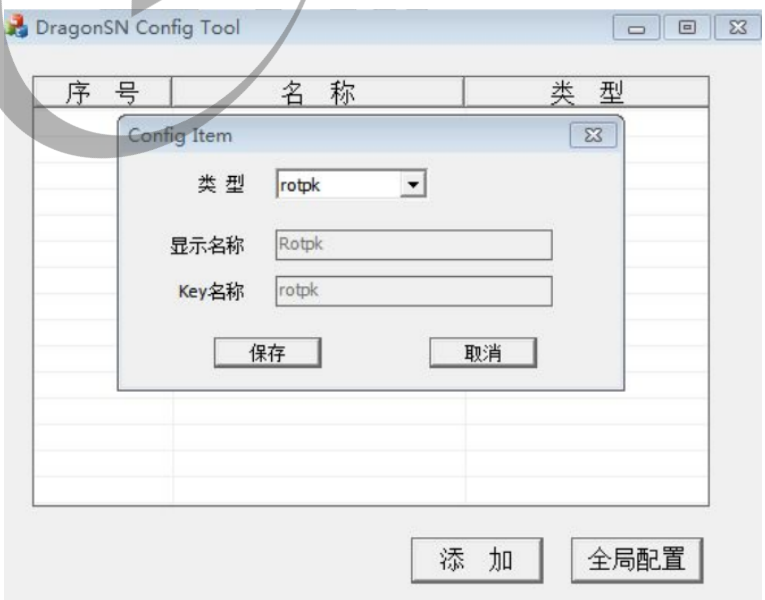

图 3-3: rotpk 烧写配置

- 在 PC 端对 DragonSN 工具进行配置。打开 DragonSNConfig.exe，如上图所示，点击 “添加”，在 “类型” 一栏下拉菜单中选择 rotpk，点击 “保存”、“确定”。点击 “全局配置”, 设置 “烧写模式” 为 “安全 key”。配置完成后，关闭配置工具。
- 运行 DragonSN.exe 工具，配置 rotpk.bin 所在的路径。然后将设备通过 usb 与 PC 连接，重启设备。当 DragonSN 提示框显示设备已连接后，开始烧录。为了保证不会烧录错误的 rotpk.bin，在烧录过程中，会将 PC 端下发的 rotpk.bin 与当前 flash 上安全固件中根证书公钥的 SHA256 值进行对比，匹配后才烧录该 rotpk.bin。

3.4.2 方法二

方法二在烧写安全固件完毕时，解析安全固件获取 rotpk.bin 并写入 efuse，然后再将 efuse 中的 secure enable bit 置 1。**当前仅 R328/MR813/R329/R818/R528 有开发支持。**

说明

- **要支持此功能，需要在 *uboot* 中*configs/{CHIP}_defconfig*或者*configs/{CHIP}_tina_defconfig*文件中打开如下宏：*CONFIG_SUNXI_BURN_ROTPK_ON_SPRITE=y***
- **此功能仅在首次烧写安全固件时生效。**

方法二的优缺点：

- 优点：量产时比较方便。
- 缺点：烧写固件时默认就烧写了 rotpk.bin。

3.4.3 方法三

方法三是在 Linux 用户空间烧写 rotpk.bin 与 secure enable bit。由于 rotpk.bin 与 secure enable bit 只能在安全环境下读写，而 Linux 环境属于非安全环境，因此在用户空间的程序会发送相关命令至安全环境下的 TA，TA 收到命令后，在安全环境下对 efuse 中的 rotpk.bin 和 secure enable bit 进行读写。**当前仅 R328 有开发支持。**

方法三的优缺点：

- 优点：量产比较快，比较适合固件离线烧录（即固件事先已经保存在 flash 上，需要时直接组装到设备）。
- 缺点：需要支持 secure os、TA 等，增大内存消耗。

3.4.3.1 API 说明

相关源码位于 tina/package/security/optee-rotpk 目录下。

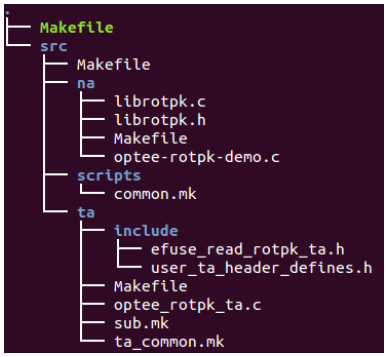

图 3-4: 烧写 rotpk 的 API 源码

其中 librotpk.c 会被编译成库文件，该库提供如下三个 API。

```
/**
 * write_rotpk_hash() - write rotpk hash to efuse.
 * @buf: input c-style string, should be 32byte hash, with a nul terminated.
 *
 * return value: zero, write success; non-zero, write failed.
 */
int write_rotpk_hash(const char *buf);

/**
 * read_rotpk_hash() - read rotpk hash from efuse.
 * @buf: buf used to contain the rotpk hash value.
 *
 * return value: size of hash length.
 */
int read_rotpk_hash(char *buf);
```

其中 optee-rotpk-demo.c 是一个调用上述 API 的 demo 程序，被编译成可执行文件 rotpk_na，使用说明如下：

```
  usage: rotpk_na [options] [hex-string]
  [options]:
    r           read rotpk from efuse.
    w           write rotpk to efuse.
    hex-string: input hex-string to burn to efuse.
```

常见的四种使用方法：

```
  "rotpk_na w", 烧写 90fa80f15449512a8a042397066f5f780b6c8f892198e8d1baa42eb6ced176f3 到efuse
     的 rotpk 区域。
  "rotpk_na w 1234567890abcdef1234567890abcdef1234567890abcdef1234567890abcdef", 烧写 自定义
    的字符串 1234567890abcdef1234567890abcdef1234567890abcdef1234567890abcdef 到 efuse的
    rotpk 区域。
  "rotpk_na r", 读取 efuse 中的 rotpk 内容。
```

注：客户可依据自身需要修改应用程序，库，TA。

3.4.3.2 开启方法

首先，需要开启 secure os 与 TA/CA 开发环境支持，具体参考第 4、5 章。

其次，执行 make menuconfig，开启如下选项。

```
Tina Configuration
  Global build settings  --->
    [*] OP-TEE Support
        choose OP-TEE version (optee version x.x.0)  --->
  Security --->
    OPTEE --->
      -*- optee-client-x.x
      -*- optee-os-dev-kit
      <*> optee-rotpk
```

然后重新打包安全固件并烧写。

3.4.3.3 使用例子

```
root@TinaLinux:/# tee-supplicant &
root@TinaLinux:/# rotpk_na w
buf_in: 90fa80f15449512a8a042397066f5f780b6c8f892198e8d1baa42eb6ced176f3, size: 64
NA: write efuse hash
NA: init context
NA: open session
TA: create entry!
TA: open session!
NA: allocate memory
NA: invoke command
TA: rec cmd 0x221
TA: keyname:rotpk,key len:32,keydata:
0x90 0xfa 0x80 0xf1 0x54 0x49 0x51 0x2a
0x8a 0x04 0x23 0x97 0x06 0x6f 0x5f 0x78
0x0b 0x6c 0x8f 0x89 0x21 0x98 0xe8 0xd1
0xba 0xa4 0x2e 0xb6 0xce 0xd1 0x76 0xf3

NA: finish with 0
```

3.5 校验 rootfs

3.1 节中提到，Secure Boot 从 brom 执行开始，到 Linux 启动结束。但是 rootfs 没有进行校验，为了校验 rootfs 的完整性，将 Secure Boot 延展至 rootfs，Tina 引入两种方法：uboot 校验 rootfs 与 dm-verity。

⚠ 警告

- **rootfs 必须为只读才能进行校验。**
- **rootfs 类型必须是 squashfs。**

3.5.1 uboot 校验 rootfs

由于 rootfs 通常来说较大，从 flash 中读取以及校验时间都比较长。Tina 上提供了一种在 uboot 阶段校验 rootfs 的方法，可以提取部分 rootfs 的数据来进行校验，有效减少校验时间。

注：仅 R328/R329/MR813/R818/R528 有开发支持该功能。

3.5.1.1 uboot 校验 squashfs rootfs 功能实现

主要思路是：

- 使用 extract_suqashfs 工具对 squashfs rootfs 进行采样，具体为每 1M 取前面 rootfs_per_MB 字节的数据，最后不足 1M 的不采样。rootfs_per_MB 在 env 中设置，必须为 4096 的倍数或者 full，其中 full 表示对整个 rootfs 进行校验；如未设置，默认取 4096 字节。
- 将所有采集的数据组合成新的文件，对该文件进行签名，生成证书。
- 使用 update_squashfs 工具将证书附着在 squashfs rootfs 的结尾处。

具体来说，使用 extract_squashfs 将 out/{BOARD}/image/rootfs.fex 进行采样，获取文件 out/{BOARD}/image/rootfs-extract.fex。使用秘钥 SCPFirmwareContentCertPK 对该 rootfs-extract.fex 进行签名，生成证书 out/{BOARD}/image/toc1/cert/rootfs.der。然后使用工具 update_squashfs 将该 rootfs.der 证书附着在 out/{BOARD}/image/rootfs.fex 的结尾处。启动过程，在 uboot 中按照相反的步骤对 rootfs 进行校验。

以上操作都是在打包脚本 scripts/pack_img.sh 中实现。

3.5.1.2 uboot 校验 squashfs rootfs 开启

首先，执行 make menuconfig，打开 CONFIG_USE_UBOOT_VERIFY_SQUASHFS 选项。

```
Tina Configuration
└─> Global build settings  --->
    └─> [*] Verify squashfs rootfs in uboot
```

其次，确保 uboot 文件`lichee/brandy-2.0/u-boot-2018*/configs/{CHIP}_defconfig`中开启了 CONFIG_SUNXI_PART_VERIFY=y 的配置。

使能该功能后，在启动过程中，uboot 会出现类似如下的 log。

```
pubkey rootfs valid
partition rootfs verify pass
```

3.5.2 dm-verity 机制

Tina dm-verity 是为了在启动过程中验证特定分区（通常是 rootfs 分区）的完整性而设计的一套解决方案。dm-verity 从启动开始，在整个设备运行过程中，提供对特定分区数据的验证。

dm-verity 在开机过程中，依靠内核提供的 device mapper 机制，验证特定分区 hash tree 数据。验证通过后，在设备节点上添加 dm-verity 设备。以后任何对该特定分区上数据的操作，都会映射到 dm-verity 设备节点上，首先对待操作数据所在的 block 计算一次 hash，将此 hash 值与该 block 在初始 hash tree 中对应的 hash 进行对比，一旦对比失败，dm-verity 就会返回失败给此次操作的调用者。

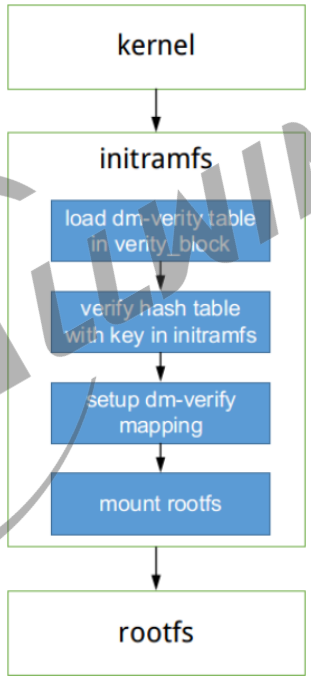

图 3-5: dm-verity 验证 rootfs 流程

Tina dm-verity 主要用在安全平台，是 Secure Boot 最后一个环节，目的是校验根文件系统分区的完整性，确保根文件系统的数据没有被篡改。

dm-verity 验证根文件系统分区的流程如上图所示。

3.5.2.1 Initramfs 构建

Tina 启动时，在 initramfs 中验证 dm-verity table 签名完整性，并挂载 dm-verity 分区，因此必须要使用 initramfs，同时 initramfs 中需要包含有相关的工具，如 openssl、veritysetup 等。

Tina 3.0 版本之后提供了一个 initramfs 生成办法。

下面以 cowbell-perf1 为例给出构建步骤，其他方案类似。

1. `source build/envsetup.sh`
2. `lunch cowbell_perf1-tina`
3. `make ramfs_menuconfig`

说明

***make ramfs_menuconfig* 命令对 *target/allwinner/cowbell-perf1/defconfig_ramfs* 进行配置修改，并基于该配置生成 *initramfs*。如果该方案没有 *defconfig_ramfs*，请复制 *defconfig* 为 *defconfig_ramfs*，为节省空间，对于不需要的配置请尽可能关闭。**

请查看如下选项是否配置正确：

```
Tina Configuration
└─> Target Images
    └─> [*] customize image name
        └─> --- customize image name
            └─>Boot Image(kernel) name suffix (boot_ramfs.img/boot_initramfs_ramfs.img)
            └─> Rootfs Image name suffix (rootfs_ramfs.img)
└─> System init (busybox-init)
└─> Base system
    └─> -*- busybox-init-base-files
        └─> [*]   Customize busybox init base files options
        └─> (busybox-init-base-files_ramfs) PATH for busybox base files
└─> Security  --->
    └─> Device Mapper  --->
        └─> <*> cryptsetup
            └─> use crypt lib (use libopenssl)  --->
        └─> <*> dm-verity
└─> Utilities  --->
    └─> <*> openssl-util
```

4. `make_ramfs [-jN]`

⚠ 警告

此处必须是 `make_ramfs`，不能是 `make`。

以上步骤执行完后，生成的 initramfs 位于 out/cowbell-perf1/compile_dir/target/rootfs_ramfs 目录下。

通常来说，initramfs 只需要构建一次，可以将构建好的 initramfs 保存在一个地方，以免以后重新生成。如需更改 initramfs 的内容，才需要重新构建。

3.5.2.2 dm-verity 启用

前提是参考 3.5.2.1 小节的说明构建好 initramfs。

下面以 cowbell-perf1 为例给出构建步骤，其他方案类似。

1. `source build/envsetup.sh`
2. `lunch cowbell_perf1-tina`
3. `make menuconfig`

```
Tina Configuration
└─> target Images
    └─> [*] ramdisk
        └─> --- ramdisk
            └─> Compression (gzip)
            └─> (../../out/cowbell-perf1/compile_dir/target/rootfs_ramfs) Use external cpio
└─> Global build settings  --->
    └─> [*] Device Mapper Verity
```

4. `make kernel_menuconfig`

选中如下配置。

```
Linux/arm 4.9.118 Kernel Configuration
└─> General setup  --->
    └─> [*] Initial RAM filesystem and RAM disk (initramfs/initrd) support
    └─> [*]   Support initial ramdisks compressed using gzip
└─> Device Drivers  --->
    └─> [*] Multiple devices driver support (RAID and LVM)  --->
        └─> <*>   Device mapper support
            └─> <*>     Verity target support
```

5. `./scripts/dm-verity-key.sh`

说明

执行该脚本，将在 *out/cowbell-perf1/verity/keys* 下生成一组 *key*，为 *dm-verity-pri.pem* 与 *dm-verity-pub.pem*，并复制到 *package/security/dm-verity/files/*目录下，同时将公钥 *dm-verity-pub.pem* 复制到 *out/cowbell-perf1/compile_dir/target/rootfs_ramfs* 下，重命名为 *verity_key*，设备启动时需要使用此公钥来验证 *rootfs*。

⚠ 警告

- **dm-verity-pri.pem 是私钥，非常重要的隐私数据，用以对 dm-verity table 进行签名，请妥善保存。**
- **如果不执行此脚本，将使用 package/security/dm-verity/files/下默认的 key，因此请务必执行一次来替换此目录下的 key。每运行一次脚本，就会更新一次 key。**

6. `make -j`
7. `pack -s [-d]`

经过以上步骤，可以生成一个支持 dm-verity 的安全固件。

3.5.2.3 dm-verity 测试

- 方法一：查看设备节点

```
root@TinaLinux:/# ls -l /dev/dm-0 /dev/mapper/rootfs
brw-r--r--    1 root     root      254,   0 Jan  1 08:21 /dev/dm-0
brw-------    1 root     root      254,   0 Jan  1 08:21 /dev/mapper/rootfs
```

如果没有/dev/mapper/rootfs 文件，执行`mount -t devtmpfs devtmpfs /dev`后即可看到。

- 方法二：查看挂载

```
root@TinaLinux:/# mount
/dev/mapper/rootfs on /rom type squashfs (ro,relatime)
```

3.5.2.4 dm-verity 影响

- 占用 flash 空间

> 需要新增 dm-verity 相关信息。同时会用到 initramfs，导致内核镜像增大。

- 系统性能影响

> dm-verity 功能可以提高 Tina 系统安全性能，但是从其实现机制来讲，会延长启动时间，降低 rootfs 分区的读取速度。

3.6 安全启动代价

3.6.1 启动时间增加

安全启动过程中会逐级对下一阶段运行的镜像进行校验，会增加启动时间。相对于非安全启动，整体会增长 500ms 左右（不包括 rootfs 的校验）。实际增加时间会因存储介质、硬件 CE 版本、cpu/dram 频率等因素的影响而不同。

3.6.2 ota 升级的变化

由 3.2 小节可知，安全固件封包与非安全固件有一定的差异，因此在 ota 升级时，请确保使用正确的文件。

- 升级 optee/uboot/dts/sys_config，需要使用 tina/out/{BOARD}/image/toc1.fex 文件；
- 升级 sboot，需要使用 tina/out/{BOARD}/image/toc0.fex 文件；
- 升级 linux kernel，需要使用 tina/out/{BOARD}/image/boot.fex 文件；
- 升级 rootfs，需要使用 tina/out/{BOARD}/image/rootfs.fex 文件。

4 Secure OS

ARM 利用 CPU 分时复用的思路，设计了 SMC 指令切换到另外一个特殊状态再结合 SOC 级别的硬件 IP 构建了被称为 ARM TrustZone 的安全技术。

Tina 从 SOC 层面支持 ARM Trustzone，但要设计满足 Linux 系统安全标准和需求的安全方案，除了实现 ARM TrustZone，还必须有一套软件可信执行环境 TEE。Tina 采用的 OP-TEE 便是一种特定安全系统实现，它严格遵循 ARM TrustZone 和 TEE/GP 等产业标准。

4.1 optee 总体框架

optee 系统，是由运行在 TEE 环境下的 optee os、TA、以及运行在 REE 环境下的 client、driver、NA 组成，一共五个部分。optee 总体架构如下图所示：

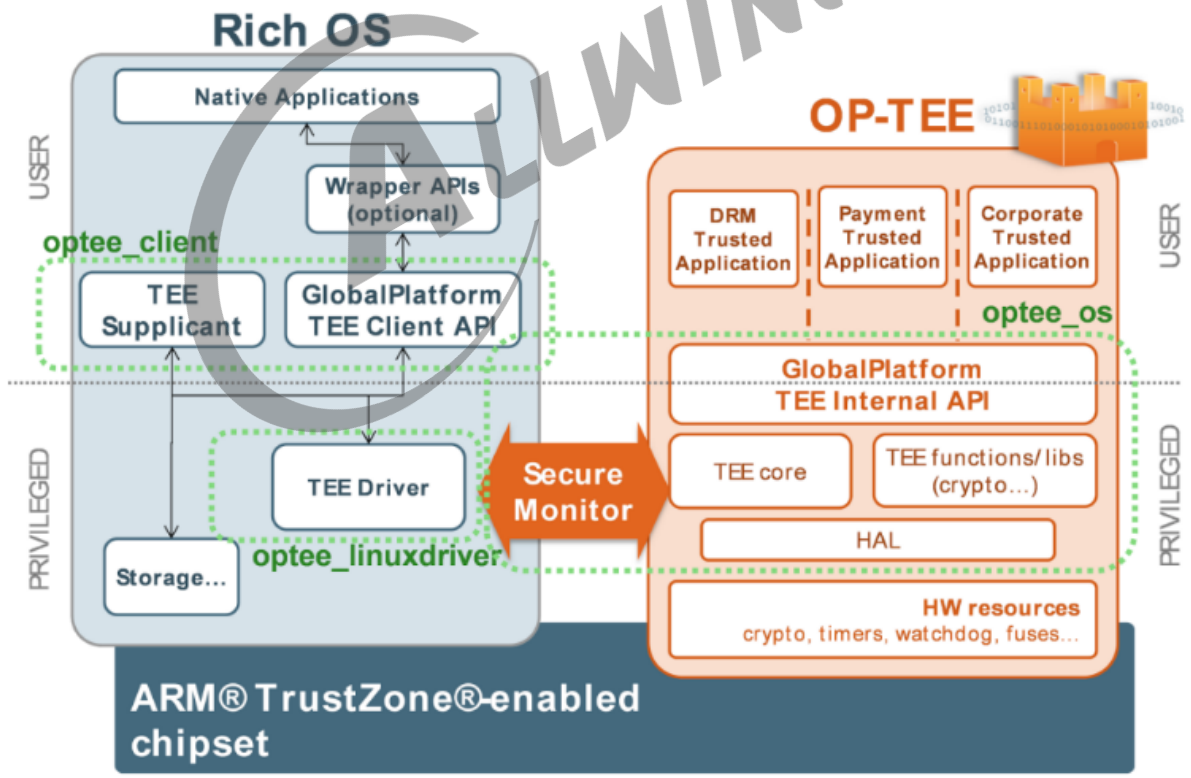

图 4-1: optee 总体架构

4.2 开启 Secure OS

4.2.1 Secure OS 镜像

Tina 固件在打包会自动把 Secure OS 镜像打包到安全固件中。Secure OS 镜像位于`device/config/chips/{IC}/bin/optee_{CHIP}.bin`。

TEE 环境使用的内存有 3 个部分，各部分大小与起始地址在 Secure OS 编译时指定。各部分作用如下：

- 共享内存。REE 与 TEE 通过 smc 指令进行交互，smc 只能通过寄存器交换有限的数据，更多的数据通过共享内存进行交换。REE 和 TEE 都有访问权限。
- optee os 内存。optee_os 专用的内存。optee_os 被加载到此处开始运行。REE 无权访问。
- TA 内存堆。加载 TA、放置 TA 堆、栈的内存空间。由 optee_os 进行分配。分配给某一个 TA 的内存只能由该 TA 或 optee_os 访问，其他 TA 无法访问。REE 无权访问。

⚠ 警告

在内核中需要为 TEE 环境预留内存，预留内存的大小与地址需要按照 optee_{CHIP}.bin 编译时指定的大小与地址来设置。

假设 R328 Secure 环境需要使用的内存如下：

1. SHARE MEM: 0x41900000-0x41A00000
2. OPTEE OS: 0x41A00000-0x41B00000
3. OPTEE TA: 0x41B00000-0x41C00000

在文件 tina/lichee/linux-4.9/arch/arm/boot/dts/sun8iw18p1.dtsi 也必须预留 3M 的内存。

```
/* optee used 3M: SHM 1M, OS 1M, TA 1M*/
/memreserve/ 0x41900000 0x00300000;
```

图 4-2: 内核预留内存

4.2.2 内核支持 optee 驱动

在内核中使能 optee 驱动，执行 make kernel_menuconfig，选中如下几项：

```
Device Drivers  --->
    <*> Trusted Execution Environment support
          TEE drivers  --->
              <*> OP-TEE
```

5 TA/CA 开发环境

Tina 上包含了 TA/CA 开发环境，便于用户在 Tina 上开发 TA 与 CA 应用程序。

Tina 上 TA/CA 开发环境主要涉及如下几个 packages:

1. tina/package/security/optee-client-x.x，提供 CA 所需的 tee-supplicant 以及 libteec 库，其中 x.x 为不同的版本。
2. tina/package/security/optee-os-dev-kit，提供 TA 端编译环境。
3. tina/package/security/optee-helloworld，关于 helloworld 的 TA/CA demo 程序。
4. tina/package/security/optee-secure-storage，关于 optee Seucre Storage 的 TA/CA demo 程序。
5. tina/package/security/optee-base64，关于 base64 算法的 TA/CA demo 程序。
6. tina/package/security/optee-efuse-read，关于读取 efuse 中 CHIPID，ROTPK，SSK，OEM 或 OEM_SEC 等区域的 TA/CA demo 程序。
7. tina/package/security/optee-getdmkey，关于从 keybox 中读取 dm-crypt 加密 key 的程序。

上面 1 与 2 是开发 TA/CA 所需的环境，3-7 分别是一些 TA/CA demo 程序，这些 demo 程序需要依赖 1、2 这两个包。

⚠ 警告

- **要使用 TA/CA 开发环境，前提是要支持 Secure boot 以及 Secure OS。**
- **demo 程序仅用于开发测试，实际产品根据需要选中。**

5.1 TA/CA 开发环境使用

CA 属于 Linux 端应用程序，同其他应用程序一样，编译比较简单，只需要依赖 optee-client 所提供的库，即可编译完成。

TA 属于安全应用程序，编译需要借助 TA dev-kit。

如要使用 TA/CA 开发环境，执行 make menuconfig，开启如下选项：

```
Tina Configuration
└─> Global build settings  --->
    └─> [*] OP-TEE Support
    └>   choose OP-TEE version (optee version x.x.0)  --->
```

```
└> Security  --->
    └> OPTEE  --->
        └> <*> optee-client-x.x..................................... optee-client
        └> <*> optee-os-dev-kit..................................... optee-os-dev-kit
```

说明

开启选项时，建议使用默认的 *OP-TEE* 版本。当前仅 *MR813/R329/R818/R528* 使用 *3.7.0*。

编译时，将会把 TA 所需的编译环境从 tina/package/security/optee-os-dev-kit/dev_kit 复制到 tina/out/{BOARD}/staging_dir/target/usr/dev_kit。

5.2 TA/CA 开发及编译

TA/CA 的开发需要参考 GlobalPlatform 提供的标准接口说明文档。

编译 TA/CA 的关键点在设置编译环境变量，如 CROSS_COMPILE_HOST, CROSS_COMPILE_TA 以及 TA_DEV_KIT_DIR 等。

TA/CA 开发环境使用可参考 Tina 上的 optee-helloworld 包 tina/package/security/optee-helloworld/src 的实现。相关编译选项设置可参考下图：

```
define Build/Compile/Source
	$(MAKE) -C $(PKG_BUILD_DIR)/ \
		ARCH="$(TARGET_ARCH)" \
		AR="$(TARGET_AR)" \
		CC="$(TARGET_CC)" \
		CROSS_COMPILE_TA="$(TARGET_CROSS)" \
		CROSS_COMPILE_HOST="$(TARGET_CROSS)" \
		PLATFORM="$(TARGET_CHIP)" \
		TA_LDFLAGS="$(TARGET_LDFLAGS)"\
		DEV_KIT_DIR="$(STAGING_DIR)/usr/dev_kit"\
		CA_DEV_KIT_DIR="$(STAGING_DIR)/usr"
endef
```

图 5-1: 编译选项设置

说明

***OPTEE* 中通过 *UUID* 唯一标识系统中的 *TA*，因此开发 *TA* 时需要在 *ta/include/user_ta_header_defines.h* 文件中设置 *TA_UUID*。*UUID* 可使用 *uuidgen* 工具生成。**

5.3 TA 签名

Tina 上支持更换 TA 签名 key。在编译 TA 之前，务必使用`openssl genrsa -out default_ta.pem 2048` 命令重新生成一个 key，对默认 key 进行替换；默认 key 的路径位于`package/security/optee-os-dev-kit/dev_kit/arm-plat-{CHIP}/export-ta_arm32/keys/default_ta.pem`。

编译 TA 的过程中，会使用该 key 对 TA 进行签名；在打包过程中，会通过 scripts/update_optee_pubkey.py 脚本提取该 key 的公钥并将其保存到 optee_os 的 image 中；这样就

保证了只有经过该 key 签名后的 TA 才可以运行在包含该 key 公钥的 optee_os 上。因此请注意妥善保存该 key。

scripts/update_optee_pubkey.py 脚本使用说明如下：

```
usage: update_optee_pubkey.py [-h] --in_file IN_FILE --out_file OUT_FILE --key KEY

optional arguments:
  -h, --help            show this help message and exit
  --in_file IN_FILE     Name of in file
  --out_file OUT_FILE   Name of out file
  --key KEY             Name of key file
```

5.4 TA 加密

默认情况下，Tina 编译的 TA 只进行了签名，不进行加密，TA 二进制文件以明文形式存放在 rootfs 中的/lib/optee_armtz 目录下。

当前，Tina 支持在 **R328/MR813/R329/R818/R528** 方案上将 TA 加密后再签名，其他方案暂未开发此功能。

执行 make menuconfig，开启如下选项使能 TA 加密（一旦开启，所有的 TA 都会进行加密）。

```
Tina Configuration
└─> Security  --->
    └─> OPTEE  --->
        └─> -*- optee-os-dev-kit
        └─> [*]   whether encrypt ta
            └─> [*]   encrypt ta with which key (ssk)  --->
```

目前加密密钥来源有两种（加密密钥长度为 128bit）：

- 使用 ssk 来作为加密秘钥。此方法需要烧写 efuse 上的 ssk 区域，然后将 ssk 的内容复制到`tina/package/security/optee-os-dev-kit/dev_kit/arm-plat-{CHIP}/export-ta_arm32/keys/ta_aes_key.bin`中，重新编译 TA。有些方案 efuse 中 ssk 区域的长度为 256bit，那么仅取其中前 128bit 作为 ta_aes_key.bin 文件。
- 使用 rotpk 派生的 key 作为加密密钥。需要借助 tina/scripts/generate_ta_key.py 工具来生成。其使用方法如下，将生成的 OUT 文件重命名为`tina/package/security/optee-os-dev-kit/dev_kit/arm-plat-{CHIP}/export-ta_arm32/keys/ta_aes_key.bin`，重新编译系统。

```
  usage: generate_ta_key.py [-h] --rotpk ROTPK --out OUT
```

5.5 安全应用 demo

5.5.1 optee-helloworld 效果

该 demo 展示 CA 如何调用 TA，以及如何通过共享内容向 TA 传输数据。

```
root@TinaLinux:/# tee-supplicant &
root@TinaLinux:/# hello_world_na 1234
NA:init context
NA:open session
TA:creatyentry!
TA:open session!
NA:allocate memoryTA:rec cmd 0x210
NA:invoke command: hello       1234
TA:hello       1234
NA:finish with 0
```

5.5.2 optee-efuse-read 效果

该 demo 中 TA 通过系统调用 utee_sunxi_read_efuse 与 utee_sunxi_keybox 来获取 efuse 与 keybox 中的内容。

⚠ 警告

本 demo 只是演示作用，实际使用时不要将获取的内容打印或传递到 CA。

```
root@TinaLinux:/# tee-supplicant &
root@TinaLinux:/# efuse_read_demo_na rotpk
NA:init context
NA:open session
TA:creatyentry!
TA:open session!
NA:allocate memory
NA:invoke command
TA:rec cmd 0x210
read efuse:rotpk
read result:
0x90 0xfa 0x80 0xf1 0x54 0x49 0x51 0x2a
0x8a 0x04 0x23 0x97 0x06 0x6f 0x5f 0x78
0x0b 0x6c 0x8f 0x89 0x21 0x98 0xe8 0xd1
0xba 0xa4 0x2e 0xb6 0xce 0xd1 0x76 0xf3
NA:finish with 0
```

6 Secure Storage

数据是最核心资产，存储系统作为数据的保存空间，是数据保护的最后一道防线。当前 Tina 上提供了三种 Secure Storage 参考实现：

- keybox Secure Storage
- OP-TEE Secure Storage
- dm-crypt Secure Storage

6.1 keybox Secure Storage

由于 efuse 空间受限，Tina 上支持了 keybox Secure Storage 功能，该功能默认开启。keybox 是 Tina 上实现的一种安全存储技术，它将待烧写的 key 传递到 secure os，在 secure os 中使用 efuse 中的 ssk 或 huk 对 key 进行加密，然后将加密后的 key 保存在 flash 上一片特定预留的区域。该区域未映射到逻辑扇区，通常的数据操作无法访问，正常量产也不会被擦除。

6.1.1 keybox 烧写及读取流程

写 keybox 有两种方式，一种是使用 DragonSN，写入的数据被 efuse 中的 ssk 进行加密，所有安全方案均支持；另一种是使用 keybox_na，仅 MR813/R818/R528 有开发支持，写入的数据会被 efuse 中的 huk 加密，如果 efuse 没有 huk 区域，则通过 chipid 派生出一个 key 来进行加密。

说明

烧写 *keybox* 之前，请注意提前烧写 *efuse* 中的 *ssk* 或 *huk* 区域，烧写方法参考 *6.1.2* 小节。

6.1.1.1 DragonSN 烧写 keybox

使用 DragonSN 烧写 keybox 流程如下图所示。

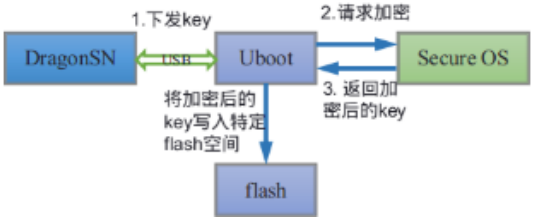

图 6-1: keybox 烧写流程

6.1.1.2 keybox_na 烧写 keybox

keybox_na 可以在用户空间烧写 keybox，其源码位于 tina/package/security/optee-keybox。keybox_na 是一个 NA，它发送 key data 到 optee os 的 PTA，PTA 将其加密后，再将加密后的数据返回给 NA 端，写入到 keybox 中。

make menuconfig 选中如下配置，编译生成 keybox_na。

```
Tina Configuration
  --> Security
    --> OPTEE
      --> <*> optee-keybox
```

keybox_na 使用方法如下。

```
usage: keybox_na [-rw] [-k key_name] <-f key_file>
 [options]:
        -r      read key named 'dm_crypt_key'
        -w      write key named [key_name] with binary [key_file]
        -k      key name
        -f      key file, binary
```

6.1.1.3 keybox 读取流程

keybox 读取流程如下图所示。启动过程中 uboot 会按照一定的条件（见 6.1.1.4 小节）将 flash 上加密的 key 读取到 secure os 进行解密，并一直保存在 secure os 的内存中，供 TA 调用。

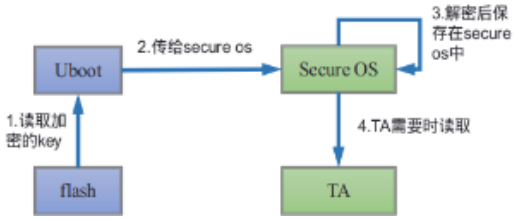

图 6-2: keybox 读取流程

6.1.1.4 keybox 列表

对于非 R328/MR813/R329/R818/R528 方案，uboot 会将所有加密的 key 加载至 secure os 中进行解密。

对于 R328/MR813/R329/R818/R528，uboot 会根据环境变量 keybox_list 来选择加载至 secure os 中的 key。keybox_list 环境变量在 env 文件中进行配置，使用逗号分隔各 key。比如下面的例子中，名称为 rsa_key，ecc_key 与 testkey 的 key 会被加载至 secure os 中进行解密。

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

说明

对于 *R328/MR813/R329/R818/R528*，使用 *DragonSN* 烧 *key* 到 *keybox* 之前，必须要配置好 *keybox_list*，否则烧写的 *key* 不会经过 *secure os* 加密，只会以明文保存。

6.1.2 DragonSN 烧写 efuse 与 keybox 的配置

前面已经介绍了烧写 rotpk 时的配置，下图给出烧录 efuse 中其他 key 的配置。

图 6-3: efuse key 烧写参考配置

烧录 efuse 时配置“烧写模式”为“安全 key”。

其中“显示名称”只是显示在 DragonSN 工具上的名字，不会影响设备端。

其中的“Key 名称”只能是特定的字符串。对于 R328 来说包括 chipid、oem、rotpk、ssk、oem_secure 五种，其他方案有一些差异，通常 chipid、rotpk 等都是可行的。

烧录 efuse 时，“key type”需要选成 efuse。

烧写 keybox key 时，DragonSN 的关键配置如下图所示。

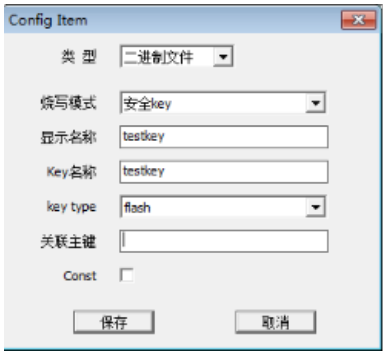

图 6-4: keybox key 烧写参考配置

烧录 keybox 时配置“烧写模式”为“安全 key”。

其中“显示名称”只是显示在 DragonSN 工具上的名字，不会影响设备端。

其中的“Key 名称”对于不同的 R 系列有不同的配置。对于 R328/MR813/R329/R818/R528，可以自己定义。对于其他方案，必须是 widevine、ec_key、rsa_key、ec_cert1、ec_cert2、ec_cert3、rsa_cert1、rsa_cert2、rsa_cert3 这些特定的字符串，如果希望自定义名字，则需要修改 uboot、monitor/secure os 等。

烧录 keybox 时，“key type”需要选成 flash。

说明

如果“类型”选择为“二进制文件”，那么待烧写的 *key* 文件名必须要以*.bin*为后缀。

6.2 OP-TEE Secure Storage

OP-TEE Secure Storage 是根据 GP TEE Internal API 规范实现的安全存储技术。它借助 Secure OS 将数据进行加密，然后保存到文件系统（/data/tee）或 RPMB 中。此功能可以与具体的设备绑定，充分保证了数据的私密性与完整性。

根据数据存储位置的不同，Tina 上支持两种 OP-TEE Secure Storage：

- REE FS Secure Storage。加密后的数据保存在 linux 文件系统中（/data/tee）。
- RPMB Secure Storage。加密后的数据保存在 eMMC 设备的 RPMB（Replay Protected Memory Block）分区中。

说明

- ***Secure Storage* 依赖 *Secure OS*，因此只有安全固件中才包含 *OP-TEE Secure Storage* 功能。**
- ***RPMB* 是 *eMMC* 中的一个具有安全特性的分区，因此只有 *eMMC* 才支持。**
- **当前仅 *R18*、*R328*、*MR813*、*R329*、*R818*、*R528* 支持 *OP-TEE REE FS Secure Storage* 功能，具体原因见 *6.2.1.3* 小节。仅 *MR813/R818/R528* 支持 *OP-TEE RPMB Secure Storage* 功能。**

6.2.1 OP-TEE REE FS Secure Storage

6.2.1.1 REE FS Secure Storage 功能框架

OP-TEE REE FS Secure Storage 的软件架构如下图所示。

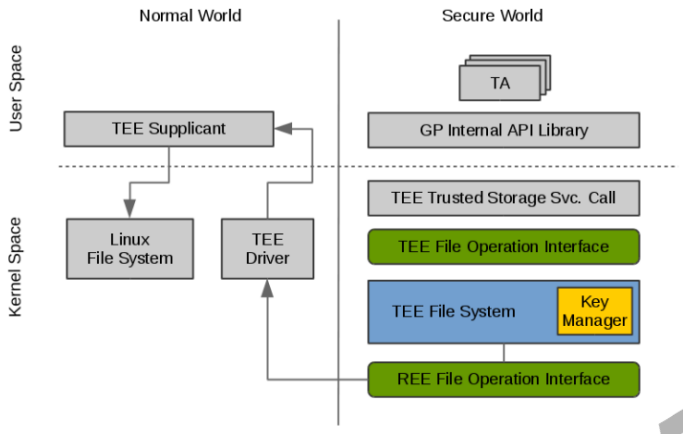

图 6-5: OP-TEE REE FS Secure Storage 软件架构

6.2.1.2 REE FS Secure Storage 文件操作流程

当要写入数据时，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 实现陷入到 OP-TEE 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。

6.2.1.3 REE FS Secure Storage 密钥管理 Key Manager

Key Manager 是 TEE file system 中的一个组件，它主要是用来处理数据加解密，并对敏感的 key 进行管理。在 Key Manager 中会使用三种类型的 key：Secure Storage Key(SSK)、TA Storage Key(TSK)、File Encryption Key(FEK)。

（1）Secure Storage Key - SSK

SSK 是一个 per-device key，当 OP-TEE 启动时，会生成此 key，并保存在安全内存中。SSK 用来生成 TSK。SSK 由如下公式计算得出：

$SSK = HMAC_{SHA256}(HUK, \text{Chip ID} \,||\, \texttt{"static string"})$

其中 HUK 为 Hardware Unique Key.

说明

- **这里的 *HUK* 是通过 *tee_otp_get_hw_unique_key* 函数获取的。对于 *R18/MR813/R818/R528* 来说，该函数会获取 *efuse* 中 *HUK* 内容的前 *128bit*；对于 *R328/R329* 来说，由于 *efuse* 中不存在 *HUK* 区域，该函数会读取 *efuse* 中 *chipid* 的内容并进行派生；对于其他方案，此函数没有实现，即该函数获取的内容全部为 *0*。**
- **这里的 *SSK* 是由 *HUK* 与 *Chip ID* 等运算得到，与 *efuse* 中的 *ssk* 区域不是同一个意思，要注意区分。**
- **对于 *MR813/R818/R528*，固件第一次启动时，会由 *CE* 模块的 *TRNG* 生成 *192bit* 的随机数，写入到 *efuse* 的 *HUK* 中。对于其他平台，默认 *efuse* 中的 *HUK* 区域（如果 *efuse* 中存在 *HUK* 区域）为全 *0*，需要借助 *DragonSN* 工具来进行烧写。具体烧写说明详见 *6.1.2* 小节。**

（2）TA Storage Key - TSK

TSK 是一个 per-Trusted Application key，用来对 FEK 进行加解密。TSK 公式计算如下：

$TSK = HMAC_{SHA256}(SSK, TA\_UUID)$

（3）File Encryption Key - FEK

当创建一个 TEE 文件时，Key Manager 会通过 PRNG 为此文件生成一个新的 FEK。并将加密之后的 FEK 存放在 meta file 中。而 FEK 本身用来对 TEE 文件进行加解密。

6.2.1.4 REE FS Secure Storage Meta Data 加密流程

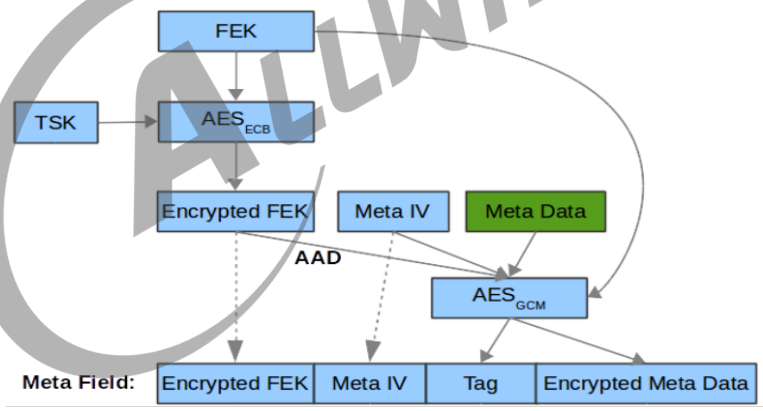

图 6-6: Meta Data 加密流程

6.2.1.5 REE FS Secure Storage Block data 加密流程

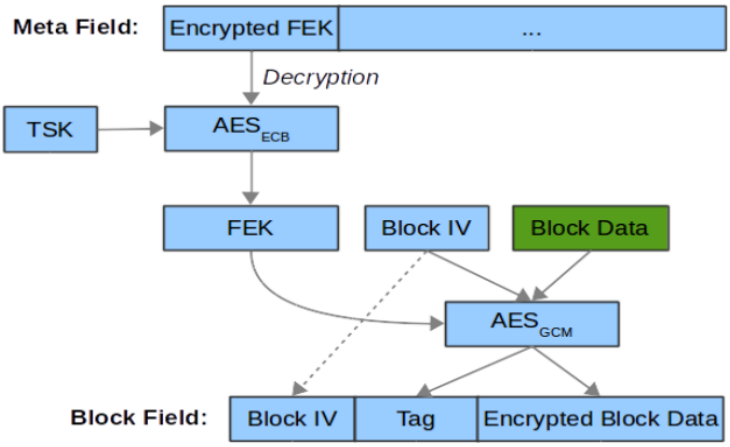

图 6-7: Block Data 加密流程

6.2.2 OP-TEE RPMB Secure Storage

6.2.2.1 RPMB Secure Storage 功能框架

RPMB Secure Storage 软件框架如下图所示。

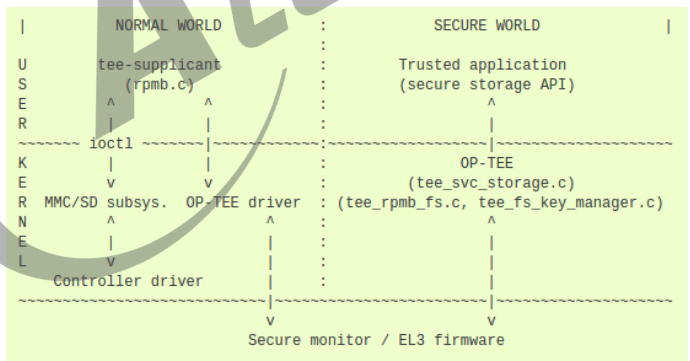

图 6-8: RPMB Secure Storage 软件框架

OP-TEE OS 中并不包含 eMMC 驱动，因此会借助 Linux 端的 tee-supplicant 通过 ioctl 来对 RPMB 分区进行访问。

6.2.2.2 RPMB Secure Storage 密钥管理与加解密

RPMB Secure Storage 文件加解密过程如下：

```
FEK = AES-Decrypt(TSK, encrypted FEK);
k = SHA256(FEK);
IV = AES-Encrypt(128 bits of k, block index padded to 16 bytes)
Encrypted block = AES-CBC-Encrypt(FEK, IV, block data);
Decrypted block = AES-CBC-Decrypt(FEK, IV, encrypted block data);
```

其中 SSK、TSK 与 FEK 的处理与 REE FS Secure Storage 一致，最终的加解密算法有差异，RPMB 用的是 AES-CBC:ESSIV，REE FS 用的是 AES-GCM。

6.2.2.3 RPMB Secure Storage 功能启用

首先需要生成 rpmb key 并写入到 eMMC 的 OTP 区域，同时设备端也需要保留该 key（当前 Tina 上将此 key 写入到 keybox 中）。整个配置步骤如下：

（1）uboot rpmb 支持

uboot 中，默认没有开启 rpmb 的支持，需要手动开启。在平台头文件`tina/lichee/brandy-2.0/u-boot-2018/include/configs/{IC}.h`加入如下宏来使能。

```
    #define CONFIG_SUPPORT_EMMC_RPMB
```

（2）uboot 加载 rpmb_key 到 optee

在 env 文件中的 keybox_list 项中加入`rpmb_key`，uboot 在启动过程中，就会自动读取`rpmb_key`的安全 key，送到 optee os 中。

（3）烧写 rpmb_key

rpmb_key 的烧录，请参考本文档 6.1 小节关于 keybox 的说明，这里需要注意的是，名字必须是`rpmb_key`。

烧写过程中，如果 uboot 或 optee 检测到名字为 rpmb_key 的安全 key，首先将此 key 烧录到 emmc 的 OTP 中，然后再保存到 keybox。

⚠ 警告

- **rpmb_key 是安全 key，长度是 256bit，请注意保密。**
- **RPMB Secure Storage 需要特定的 optee.bin，Tina SDK 默认没有包含此 optee.bin。如需支持此功能，请联系 AW 安全接口人。**

6.2.2.4 RPMB 调试工具

Tina 上集成了 mmc-utils 工具包，用于调试 RPMB。执行`"mmc -h"`查看使用说明。

mmc 工具使用时，在没有传入 rpmb_key 的情况下，可以读取 RPMB 数据，但是并不能保证

这个数据没有被修改过。传入 rpmb_key 才能保证读取的数据没有被修改。RPMB 的写入需要传入 rpmb_key。如果传入的 rpmb_key 不匹配，读写都会报错。

6.2.3 Tina OP-TEE Secure Storage demo

Tina OP-TEE Secure Storage demo 是基于第三方开源库 optee-test 中的测试样例修改而来，客户也可以参考 optee-test 自行编写代码。

相关文件保存在 tina/package/security/optee-secure-storage 目录中，其内容如下：

```
.
├── Makefile
└── src
    ├── Makefile
    ├── na
    │   ├── demo.c
    │   ├── libstorage.c
    │   ├── libstorage.h
    │   ├── Makefile
    │   ├── tee_api_defines_extensions.h
    │   └── tee_api_defines.h
    └── ta
        ├── include
        │   ├── storage.h
        │   ├── ta_storage.h
        │   └── user_ta_header_defines.h
        ├── Makefile
        ├── storage.c
        ├── sub.mk
        ├── ta_common.mk
        └── ta_entry.c
```

6.2.3.1 Tina OP-TEE Secure Storage TA

我们在 Secure World 端实现了一个 TA demo，用来调用 Secure OS 中的 TEE File System 对数据进行加解密等操作。TA 的源码位于 optee-secure-storage/src/ta 下。当 Normal World 中有应用程序发起请求时，此 TA 会被加载到 Secure World 并运行。

Ta 目录下还包含了 ta_storage.h 头文件，此文件中包含了 TA 的 UUID 以及相关的 command 编号。

6.2.3.2 Tina OP-TEE Secure Storage Library

我们将 Normal World 中同 Secure Storage TA 交互的接口进行了封装，具体实现在 optee-secure-storage/src/na/libstorage.c 文件中，默认编译成库文件。Linux 端应用程序可以直接调用封装好的接口，便于开发。包含如下五个 API。

(1) 创建文件

```
TEEC_Result OP-TEE_fs_create(TEEC_Context ctx, TEEC_Session *sess, void *file_name,
    uint32_t file_size, uint32_t flags, uint32_t *obj, uint32_t storage_id);
```

函数功能：创建一个文件。

参数说明：

- TEEC_Context ctx：NA 端打开 TA 前创建初始化的一个 TEE context，主要用于申请共享内存。
- TEEC_Session *sess：NA 端创建一个 TA 连接的一个 session 结构体。
- void *file_name：创建文件的索引指针。
- uint32_t file_size：创建文件的大小。
- uint32_t flags：打开文件的权限，一般配置如下三种：TEE_DATA_FLAG_ACCESS_WRITE | TEE_DATA_FLAG_ACCESS_READ | TEE_DATA_FLAG_ACCESS_META 其中分别对应对文件的写、读、擦除权限。
- uint32_t *obj：文件描述符指针，成功创建文件时，会赋予 obj 打开文件的文件描述符，供后面读写擦除等操作使用。
- uint32_t storage_id：配置存储属性。默认有三种：

  1. TEE_STORAGE_PRIVATE
  2. TEE_STORAGE_PRIVATE_RE_REE
  3. TEE_STORAGE_PRIVATE_RPMB

  前面两种支持文件加密存储在 REE 端/data/tee 目录，最后一种表示存储在 eMMC 的 RPMB 分区。

(2) 打开文件

```
TEEC_Result OP-TEE_fs_open(TEEC_Context ctx, TEEC_Session *sess, void *file_name, uint32_t
    file_size, uint32_t flags, uint32_t *obj, uint32_t storage_id);
```

函数功能：打开一个文件，如果文件不存在，返回错误。

参数说明：

- TEEC_Context ctx：NA 端打开 TA 前创建初始化的一个 TEE context，主要用于申请共享内存。
- TEEC_Session *sess：NA 端创建一个 TA 连接的一个 session 结构体。
- void *file_name：打开文件的索引指针。
- uint32_t file_size：打开文件名的大小。
- uint32_t flags：打开文件的权限，一般配置如下三种：TEE_DATA_FLAG_ACCESS_WRITE | TEE_DATA_FLAG_ACCESS_READ | TEE_DATA_FLAG_ACCESS_META 其中分别对应对文件的写、读、擦除权限。

- uint32_t *obj：文件描述符指针，成功打开或者创建文件时，会赋予 obj 打开文件的文件描述符，供后面读写擦除等操作使用。
- uint32_t storage_id：配置存储属性。默认有三种：

  1. TEE_STORAGE_PRIVATE
  2. TEE_STORAGE_PRIVATE_RE_REE
  3. TEE_STORAGE_PRIVATE_RPMB

(3) 读取文件

```
TEEC_Result OP-TEE_fs_read(TEEC_Context ctx, TEEC_Session *sess, uint32_t obj, void *data,
    uint32_t data_size, uint32_t *count);
```

函数功能：读取一个文件指定长度。

参数说明：

- TEEC_Context ctx：NA 端打开 TA 前创建初始化的一个 TEE context，主要用于申请共享内存。
- TEEC_Session *sess：NA 端创建一个 TA 连接的一个 session 结构体。
- uint32_t obj：文件描述符。
- void *data：承载读取文件数据的 buffer 地址。
- uint32_t data_size：读取文件数据长度。
- uint32_t *count：实际读取文件的长度。

(4) 写文件

```
TEEC_Result OP-TEE_fs_write(TEEC_Context ctx, TEEC_Session *sess, uint32_t obj, void *data,
     uint32_t data_size);
```

函数功能：向文件写入指定长度数据。

参数说明：

- TEEC_Context ctx：NA 端打开 TA 前创建初始化的一个 TEE context，主要用于申请共享内存。
- TEEC_Session *sess：NA 端创建一个 TA 连接的一个 session 结构体。
- uint32_t obj：文件描述符。
- void *data：写入文件数据的 buffer 地址。
- uint32_t data_size：写入文件数据长度。

(5) 删除文件

```
TEEC_Result OP-TEE_fs_unlink(TEEC_Session *sess, uint32_t obj);
```

函数功能：关闭并删除文件

参数说明：

- TEEC_Session *sess：NA 端创建一个 TA 连接的一个 session 结构体。
- uint32_t obj：文件描述符。

6.2.3.3 Tina OP-TEE Secure Storage Demo

此为 Linux 端的 demo 程序，源文件为 demo.c，默认编译成 ss_demo。使用方法如下：

```
usage: ss_demo [type] [options] [file name]
 [type]: 'ree_fs' or 'rpmb_fs'
 [options]:
        -c       create a file named [file name] to secure storage
        -r       read a file named [file name] from secure storage
        -w       write a file named [file name]  to secure storage
                 content is 256 bytes random number
        -d       delete a file named [file name] from secure storage

 [file name]: file name
```

比如，当运行"ss_demo ree_fs -w 1.file"，会随机生成 256 个字节的数据，保存到 Secure Storage 中的 1.file 文件中。

6.2.4 Tina OP-TEE Secure Storage 开启

6.2.4.1 OP-TEE Secure Storage 配置

(1) 开启 Tina 相关配置

在 Tina 环境下，执行"make menuconfig"，确保如下选项已经开启。

```
Tina Configuration
    Global build settings  --->
        [*] OP-TEE Support
              choose OP-TEE version (optee version x.x.0)  --->
    Security  --->
        OPTEE  --->
            -*- optee-os-dev-kit
            -*- optee-client-x.x
            <*> optee-secure-storage
```

(2) 开启内核相关配置

在 Tina 环境下，执行"make kernel_menuconfig"，确保如下选项已经开启。

```
Linux/arm 4.9.118 Kernel Configuration
    Device Drivers  --->
        <*> Trusted Execution Environment support
            TEE drivers  --->
                <*> OP-TEE
```

(3) 设置 dts

在 Tina 环境下，确保tina/lichee/linux-<kernel_version>/arch/arm*/boot/dts/sunxi/{CHIP}.dtsi文件中的 firmware 下包含如下内容：

```
optee {
         compatible = "linaro,optee-tz";
         method = "smc";
};
```

6.2.4.2 编译安全固件

在 Tina 环境下，按照第 3 章说明来编译安全固件。

6.2.5 OP-TEE Secure Storage 使用

```
root@tulip-mozart:/# tee-supplicant &
root@TinaLinux:/# ss_demo rpmb_fs -c test.file
root@TinaLinux:/# ss_demo rpmb_fs -r test.file
---- Read file:test.file 0 Bytes data: ----
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00

---- Read file:test.file end! ----
root@TinaLinux:/# ss_demo rpmb_fs -w test.file
---- Write file:test.file with 256 Bytes data: ----
0d 84 14 34 76 19 a9 c2 98 76 86 f9 2f c7 07 29
77 3b 9b 98 cb dd 57 f4 5f d5 b3 f6 d1 01 f4 5e
```

```
05 88 12 fa 22 3c be 3a b2 c4 34 61 8d ba 8b 84
76 27 9d c1 84 f4 b7 e4 4a 6b db 1c ec 51 f9 f1
d9 0c ed 7b c7 2c b5 7b f0 6a 5c 7e 25 e7 83 9b
8e 21 5e 14 16 95 f8 60 01 54 fb ed 25 75 60 7f
01 cd fa c9 f9 b1 c4 ea 9b 21 e9 40 89 6d dc 18
8e ba 2c 24 cf a4 84 d0 79 00 3f 9e 75 9f 1e f5
6d 1a bf e6 4b 04 d1 66 26 bb a6 af a8 03 47 37
bd 74 5b 0d 98 5f de 12 de 9d b1 d3 bc 4f ca a9
e8 0a 90 b3 0f e1 1a 35 9e c1 64 c6 c4 ab fe 03
9f d9 10 39 b9 6e ca 18 8b fb ec 48 4c b7 f1 b4
41 82 69 50 65 03 05 83 44 e8 4a 89 95 c8 8c b4
23 1c ed 5c 0b b9 74 96 b5 61 df 81 98 51 37 da
d4 20 aa b9 23 b0 bc e7 99 86 71 ae cf 7d 64 72
99 d1 ce 24 0b c2 bb 41 a4 1b c2 3d 6c 79 97 c0

---- Write file:test.file end! ----
root@TinaLinux:/# ss_demo rpmb_fs -r test.file
---- Read file:test.file 256 Bytes data: ----
0d 84 14 34 76 19 a9 c2 98 76 86 f9 2f c7 07 29
77 3b 9b 98 cb dd 57 f4 5f d5 b3 f6 d1 01 f4 5e
05 88 12 fa 22 3c be 3a b2 c4 34 61 8d ba 8b 84
76 27 9d c1 84 f4 b7 e4 4a 6b db 1c ec 51 f9 f1
d9 0c ed 7b c7 2c b5 7b f0 6a 5c 7e 25 e7 83 9b
8e 21 5e 14 16 95 f8 60 01 54 fb ed 25 75 60 7f
01 cd fa c9 f9 b1 c4 ea 9b 21 e9 40 89 6d dc 18
8e ba 2c 24 cf a4 84 d0 79 00 3f 9e 75 9f 1e f5
6d 1a bf e6 4b 04 d1 66 26 bb a6 af a8 03 47 37
bd 74 5b 0d 98 5f de 12 de 9d b1 d3 bc 4f ca a9
e8 0a 90 b3 0f e1 1a 35 9e c1 64 c6 c4 ab fe 03
9f d9 10 39 b9 6e ca 18 8b fb ec 48 4c b7 f1 b4
41 82 69 50 65 03 05 83 44 e8 4a 89 95 c8 8c b4
23 1c ed 5c 0b b9 74 96 b5 61 df 81 98 51 37 da
d4 20 aa b9 23 b0 bc e7 99 86 71 ae cf 7d 64 72
99 d1 ce 24 0b c2 bb 41 a4 1b c2 3d 6c 79 97 c0

---- Read file:test.file end! ----
root@TinaLinux:/# ss_demo rpmb_fs -d test.file
Delete file:test.file !
root@TinaLinux:/# ss_demo rpmb_fs -r test.file
Failed to optee_fs_open: test.file, ret = 0xffff0008
```

6.3 dm-crypt Seucre Storage

为防止未授权用户通过对设备进行物理攻击（如直接读取 Flash）来获取敏感信息，造成用户数据泄露，Tina 引入 dm-crypt 机制，对用户文件系统的数据提供加密保护。

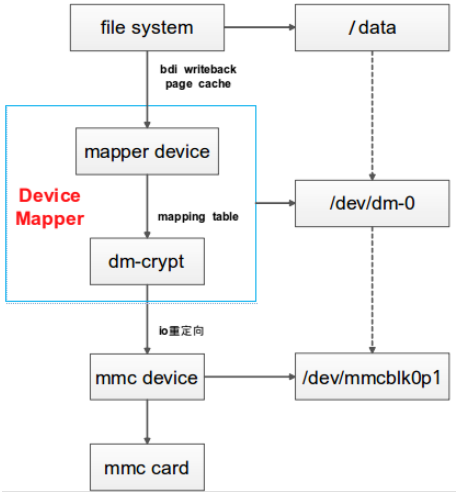

图 6-9: dm-crypt 架构

dm-crypt 是使用 linux 内核加密 API 框架和设备映射（device mapper）子系统的磁盘加密技术。Device mapper 在内核中作为一个块设备驱动被注册的，它包含三个重要的对象概念：mapped device、映射表、target device。Mapped device 是一个逻辑抽象，可以理解成为内核向外提供的逻辑设备，它通过映射表描述的映射关系和 target device 建立映射。这里的映射关系可以是 verity（完整性校验），也可以是 crypt（加密）。

上图示例中，将/dev/mmcblk0p1 通过 device mapper 映射称/dev/dm-0 设备，对/dev/dm-0 进行文件系统格式化后可将/dev/dm-0 挂载至/data 目录。

6.3.1 Tina dm-crypt

dm-crypt 中的加解密可使用内核原生的软件加解密实现，也可以使用 AW SOC 自带的硬件加密引擎（CE Crypto Engine）来实现。

当前 Tina dm-crypt 分区的初始化、挂载与卸载借助 package/security/dm-crypt/dm-crypt.sh 脚本来实现。该脚本默认将映射后的分区格式化为 ext4。

说明

该脚本是一个 *demo*，客户可依据需求自行开发。

6.3.1.1 dm-crypt 配置

使用 Tina dm-crypt 需要三个先决条件：

(1) 配置 Linux 内核。

执行 make kernel_menuconfig，开启内核 dm-crypt 相关功能以及加解密 API：

```
Device Drivers --->
    [*] Multiple devices driver support (RAID and LVM) --->
        <*> Device mapper support
            <*>   Crypt target support
File systems  --->
    <*> The Extended 4 (ext4) filesystem
    [*]   Use ext4 for ext2 file systems
-*- Cryptographic API  --->
    <*> XTS support
    <*> SHA224 and SHA256 digest algorithm
    <*> AES cipher algorithms
    <*> User-space interface for hash algorithms
    <*> User-space interface for symmetric key cipher algorithms
```

如果希望使用硬件加密引擎，开启如下配置。

```
-*- Cryptographic API  --->
    [*]   Hardware crypto devices  --->
        <*>   Support for Allwinner Sunxi CryptoEngine
```

如果方案使用了 UBI，即 Linux 内核中开启了 UBI 相关选项，还需开启如下配置。

```
Device Drivers --->
    <*> Memory Technology Device (MTD) support  --->
        <*>   Caching block device access to MTD devices
        -*-   Enable UBI - Unsorted block images  --->
            <*>   MTD devices emulation driver (gluebi)
```

(2) 配置 rootfs。

执行 make menuconfig，开启如下选项。

```
Tina Configuration
     Security  --->
        Device Mapper  --->
            <*> dm-crypt
```

(3) 配置分区表，新增一个需要加密的分区。

修改 sys_partition*.fex 文件，新增 secret 分区，用来对其进行加密，分区 size 可以自定义。

```
[partition]
    name         = secret
    size         = 40960
    user_type    = 0x8000
```

6.3.1.2 dm-crypt 使用

开启如上配置之后，rootfs 中或包含相关工具及脚本。其中 dm-crypt.sh 脚本会借助 crypt-setup 与 openssl 相关工具来执行映射、格式化、打开、挂载 dm-crypt 分区等操作，其使用说

明如下（当前加密算法为 aes-xts-plain64）：

```
dm-crypt.sh - this script helps you to use the secret partition.
Usage: /usr/bin/dm-crypt.sh <op_flag> <type> <keyfile>
        <op_flag>:
               'c' - create & format secret partition;
               'm' - mount secret partition;
               'u' - unmount secret partition and close mapper device
        <type>: Device type, can be 'plain' or 'luks'.
        <keyfile>: Key, can be 'keyfile' or 'pass' or 'optee-pass'
```

cryptsetup 支持使用 keyfile、pass 或 optee-pass。

- keyfile，这可以是任何文件，但建议使用具有适当保护的随机数据的文件（考虑到访问此密钥文件将意味着访问加密数据）。
- pass, 此模式下需要手动输入 key。
- optee-pass，此模式下会调用 getdmkey_na 程序从 optee 中获取一个 256bit 的 key，此部分将在下一小结详细说明。

cryptsetup 还支持多次加密操作模式，如 luks，plain，loopaes 等，当前 dm-crypt.sh 支持 luks 与 plain 模式。

(1) 格式化 dm-crypt 分区

执行`dm-crypt.sh c luks pass`，创建并格式化 dm-crypt 分区。

```
root@TinaLinux:/# dm-crypt.sh c luks pass
Enter passphrase:
Enter same passphrase again:

Creating Filesystems...

mke2fs 1.42.12 (29-Aug-2014)
```

(2) 挂载 dm-crypt 分区

执行`dm-crypt.sh m luks pass`。

```
root@TinaLinux:/# dm-crypt.sh m luks pass
Enter passphrase:
Enter same passphrase again:
[  412.744846] EXT4-fs (dm-0): mounted filesystem with ordered data mode. Opts: (null)
mount /dev/mapper/secret to /mnt/secret
```

查看 secret 分区是否挂载成功，是否可以读写。

```
root@TinaLinux:/# mount | grep secret
/dev/mapper/secret on /mnt/secret type ext4 (rw,relatime,data=ordered)
root@TinaLinux:/# ls /mnt/secret/
lost+found
```

6.3.1.3 dm-crypt key

dm-crypt 的 key 可以是两种模式，一种是 passphrase，最大长度是 512B；另一种是 key-file，文件的最大长度是 8192KB。

为增加 key 的安全性，Tina 上支持从 optee os 中获取用于 dm-crypt 的 key。该 key 需要预先烧录到 keybox 中，具体烧录方法请参考本文档 6.1 小节关于 keybox 的说明，这里需要注意的是，名字必须是`dm_crypt_key`，key 长度为 256bit。

Linux 端对应的应用程序是 getdmkey_na，源码位于 tina/package/security/optee-getdmkey/目录下，具体使用方法参考第 5 章关于 TA/CA 开发环境的说明。

7 SELinux

SELinux (Security-Enhanced Linux) 是美国国家安全局（NSA）对强制访问控制（MAC，Mandatory Access Control）的实现。

强制访问控制是相对于 Linux 传统的自主访问控制（DAC,Discretionary Access Control）一种访问控制机制。

DAC 控制的主体是用户，其最大的缺点是它无法分离用户与进程，进程能够继承用户的访问控制。由于程序是存在漏洞的，一旦被入侵，则入侵者具有该用户在系统上的所有权限。

MAC 中所有的访问权限是由访问控制策略来定义的，用户无法超越策略的限制。SELinux 以最小权限原则（principle of least privilege）为基础，在策略之外的访问都是无权的。

7.1 基本概念

7.1.1 主体 Subject

可完全等同于进程。

7.1.2 客体 Object

被主体访问的系统资源。可以是文件、目录、共享内存、套接字、端口、设备等。

7.1.3 安全上下文 Secure Context

SELinux 对主体与客体的标记称做安全上下文，也称为安全标签，或标签。安全上下文的构成是一个四元组，包含 User：Role：Type：MLS/MCS。每个字段都可以用来决定访问控制，其中最重要的是 Type，大多数 Policy 都是针对 Type 来制定的。

进程安全上下文被记录在 task_struct 中，客体安全上下文来源是文件的扩展属性（xattr）。

7.1.4 策略 Policy

安全策略是使用策略语言编写的具体的访问控制规则。

主体访问客体时，SELinux 会根据安全策略来判断访问是否允许。

如策略语句：allow init sshd_exec_t:file {open read execute}; 表明允许 init 类型的主体对 sshd_exec_t 类型的客体执行 file 的 open/read/execute 操作。

策略使用策略语言编写的。为了让策略语言起作用，需要用相关的用户态工具将策略语言编译成二进制文件，通过 selinuxfs 接口，将二进制文件所表示的策略输入到 Security Server 中。

7.1.5 SELinux 的运行模式

SELinux 共有三种运行模式：

- enforcing，默认值，表示会强制禁止违反策略的访问。
- permissive，表示不强制禁止，违反策略会生成一条拒绝访问的信息。
- disable，禁用 SELinux。

7.2 LSM 框架

LSM（Linux Security Module）是内核为支持不同安全机制的实现所设计的一个通用访问控制框架。目前 LSM 框架下访问决策模块包括 SELinux、SAMCK、tomoyo、yama、apparmor。

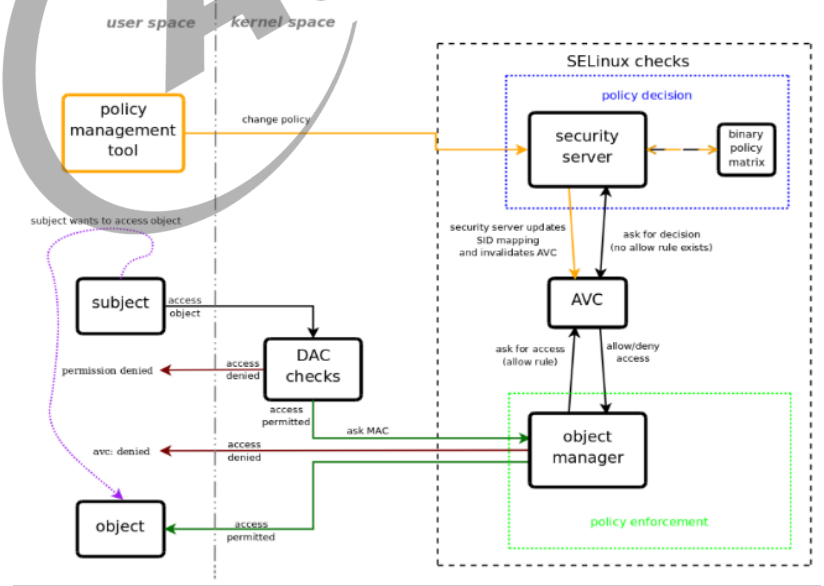

图 7-1: SELinux 决策流程

SELinux 的决策流程如上图所示。策略管理工具将策略文件载入到内核中的 Security server 中。当主体访问客体时，首先进行 DAC 检测，通过后再进行 MAC 检测。

在 Security server 与客体管理器之间有一个缓存 AVC（Access Vector Cache），用来提高检测效率。主体发出访问客体的请求时，内核中的客体管理器首先查看 AVC，如果 AVC 中有缓存策略决策结果，根据缓存情况执行放行或拒绝；如果 AVC 中没有缓存，安全服务器在策略中查找规则，按规则执行放行或拒绝的决策，策略决策的结果缓存到 AVC 中。

SELinux 的策略是允许策略，在策略中没有定义的访问都是被禁止的。

7.3 Tina SELinux 开启

Linux 主线很早就包含了 SELinux 的实现，Tina 上主要是集成了 SELinux 相关库、调试工具、参考策略以及策略加载等组件。

- 库：libsepol、libselinux、libaudit、libcap-ng、libsemanage 等
- 调试工具：policycoreutils、checkpolicy、audit、selinux-python 等。
- 参考策略：refpolicy 与 sepolicy。
- 策略加载：busybox 与 procd 的 init 进程中执行策略加载与安全上下文设置。

⚠ 警告

我们没有提供一个专门为 tina 系统定制的 selinux policy，只是简单的使用 refpolicy 和 sepolicy（基于 android），用户需要根据产品需求开发合适的策略。

7.3.1 menuconfig 配置

进入 Tina 根目录，执行 make menuconfig 进入配置主界面，开启如下配置（以 refpolicy 为例，最小配置）。

```
Tina Configuration
    Global build settings  --->
        [*] NSA SELinux Support
              choose the SELinux Policy (the reference policy)  --->
        [*] Compile the kernel with device tmpfs enabled
        [*]   Automatically mount devtmpfs after root filesystem is mounted
    Base system  --->
        <*> busybox  --->
            [*]   Customize busybox options
                Busybox Settings  --->
                    [*] Support NSA Security Enhanced Linux
                        What kind of applet links to install (as script wrappers)  --->
                        /bin/sh applet link (as script wrapper)  --->
    Security  --->
        SELINUX  --->
            <*> refpolicy
```

可以根据需要加入相关调试工具，如 checkpolicy、policycoreutils 等。

7.3.2 kernel_menuconfig 配置

在命令行中进入 Tina 根目录，执行 make kernel_menuconfig 进入配置主界面，新增如下配置。

```
Linux Kernel Configuration
    General setup  --->
            [*] Auditing support
    File systems  --->
        <*> The Extended 4 (ext4) filesystem
            [*]   Ext4 extended attributes
            [*]     Ext4 Security Labels
        [*] Miscellaneous filesystems  --->
            <*>   SquashFS 4.0 - Squashed file system support
                [*]     Squashfs XATTR support
    Security options  --->
        [*] Enable different security models
        [*] Socket and Networking Security Hooks
        [*] NSA SELinux Support
        [*]   NSA SELinux boot parameter
        (1)     NSA SELinux boot parameter default value (NEW)
        [*]   NSA SELinux runtime disable
        [*]   NSA SELinux Development Support
        [*]   NSA SELinux AVC Statistics
        (1)   NSA SELinux checkreqprot default value
           Default security module (SELinux)  --->
```

注意，不同内核版本可能有细微差异，另上面只列举了 ext4/squashfs 文件系统的配置，如果想支持更多的文件系统，请打开对应文件系统的 xattr 的支持。

对于 Linux-5.4 版本内核，Selinux 配置还需要新增如下配置，在 “Ordered list of enabled LSMs” 选项中加入关键字 selinux。

```
Linux Kernel Configuration
    Security options  --->
    First legacy 'major LSM' to be initialized (SELinux)  --->
    (selinux,lockdown,yama,loadpin,safesetid,integrity) Ordered list of enabled LSMs
```

7.3.3 SELinux 初始化

系统启动时，在 init 进程里，会加载策略文件、文件上下文到系统中，同时根据加载的策略文件初始化系统的安全上下文。具体的初始化过程需要根据实际情况进行适配，当前 Tina 启动后相关 log 如下所示。

```
[    3.734518] device_chose finished 122!
[    5.124774] SELinux: 32768 avtab hash slots, 117923 rules.
[    5.199800] SELinux: 32768 avtab hash slots, 117923 rules.
[    5.234633] SELinux:  6 users, 176 roles, 4773 types, 317 bools
[    5.241333] SELinux:  129 classes, 117923 rules
[    5.262431] SELinux:  Completing initialization.
[    5.267662] SELinux:  Setting up existing superblocks.
```

```
[    5.330382] audit: type=1403 audit(5.280:3): policy loaded auid=4294967295 ses
    =4294967295
SELinux policy load success!
set init context success!
/etc/selinux/targeted/contexts/files/file_contexts load success!
```

启动后，可以执行 sestatus 查看当前 selinux 状态。

```
root@TinaLinux:/# sestatus
SELinux status:                 enabled
SELinuxfs mount:                /sys/fs/selinux
SELinux root directory:         /etc/selinux
Loaded policy name:             targeted
Current mode:                   enforcing
Mode from config file:          enforcing
Policy MLS status:              disabled
Policy deny_unknown status:     denied
Memory protection checking:     requested (insecure)
Max kernel policy version:      30
```

查看文件安全上下文。

```
root@TinaLinux:/# ls -Z
system_u:object_r:bin_t          bin
system_u:object_r:device_t       dev
system_u:object_r:etc_t          etc
system_u:object_r:lib_t          lib
system_u:object_r:mnt_t          mnt
system_u:object_r:unlabeled_t    overlay
system_u:object_r:proc_t         proc
system_u:object_r:default_t      rdinit
system_u:object_r:root_t         rom
root:object_r:user_home_dir_t    root
system_u:object_r:bin_t          sbin
system_u:object_r:sysfs_t        sys
system_u:object_r:tmpfs_t        tmp
system_u:object_r:usr_t          usr
system_u:object_r:default_t      var
system_u:object_r:default_t      www
```

查看进程安全上下文。

```
root@TinaLinux:/# ps -Z
  PID CONTEXT                          STAT COMMAND
1537 system_u:system_r:kernel_t        SW   [RTWHALXT]
1549 system_u:system_r:initrc_t        S    /sbin/swupdate-progress -w
1568 system_u:system_r:init_t          S<   {ntpd} /bin/busybox /usr/sbin/ntpd
1618 system_u:system_r:sysadm_t        R    {ps} /bin/busybox /bin/ps -Z
```

7.4 策略开发

SELinux 的策略是允许策略，在策略中没有定义的操作都是被禁止的。所以对于一个新程序，需要新增对应的策略。

本节以 Tina 上 sepolicy-demo 策略为基础，举例说明如何新增策略来实现需要的强制访问控制。

7.4.1 限制主体的权限

要求：限制进程 fork_test 不能 fork 子进程。

7.4.1.1 fork_test 源代码

写一个简单包含 fork 的程序，命名为 fork_test.c，编译完成后，生成 fork_test 的二进制文件，存放在/usr/bin/下。

```
#include <unistd.h>
#include <stdio.h>
#include <stdlib.h>

int main()
{
    pid_t pid = fork();

    if(pid < 0)
        printf("fork error\n");
    else if(pid == 0)
        printf("this is child\n");
    else
        printf("this is parent, pid = %d\n", pid);

    return 0;
}
```

7.4.1.2 添加策略

首先，定义该文件的安全上下文。在 tina/package/security/sepolicy-demo/src/file_contexts 文件中新增如下行，表明/usr/bin/fork_test 的安全上下文是 u:object_r:fork_test_exec:s0

```
/usr/bin/fork_test                    u:object_r:fork_test_exec:s0
```

其次，定义该进程的安全上下文，以及该进程访问资源的权限。新增 tina/package/security/sepolicy-demo/src/fork_test.te 文件，其内容如下：

```
type fork_test, domain;
type fork_test_exec, exec_type, file_type;
# permissive fork_test;

init_daemon_domain(fork_test)
domain_auto_trans(shell,fork_test_exec,fork_test)
```

```
allow fork_test serial_device:chr_file rw_file_perms;
allow fork_test shell:fd use;

neverallow fork_test self:process fork;
```

- 第一句，定义一个 fork_test 的类型，其属性集是 domain。Sepolicy 中 domain 表示进程属性集。
- 第二句，定义一个 fork_test_exec 的类型，其属性集是 exec_type 和 file_type。表明该类型即是一个可执行的类型，又是一个文件类型。属性集的好处是可以对属性集下的组进行统一处理。
- 第三句，`"#permissive fork_test"`，前面的 # 表示注释。如果不注释，表示 fork_test 在执行的时候，如果没有相关的权限也能继续执行，同时会打印一条提示信息，这在开发阶段有好处，因为很难一开始就知道该主体需要多少权限。但是在最终产品上，不允许任何的 permissive，否则就起不到强制访问控制的作用了。
- 第四句，init_daemon_domain(fork_test)，这里的 init_daemon_domain 是一个宏，该宏的定义位于 te_macros 文件中。这句主要就表明允许 init 类型的进程执行 fork_test_exec 类型的文件，新进程的安全上下文是 fork_test。
- 第五句，domain_auto_trans(shell,fork_test_exec,fork_test)，表示 shell 类型的进程执行 fork_test_exec 类型的文件，新进程的安全上下文自动变为是 fork_test。注：如果不进行设置，默认情况下，子进程的安全上下文继承父进程的安全上下文。
- 第六句，allow fork_test serial_device:chr_file rw_file_perms; 允许 fork_test 类型的进程对 serial_device 的客体进行 chr_file 的 rw 操作。因为程序中有打印，所以需要添加对串口设备的读写权限。
- 第七句，allow fork_test shell:fd use; fd，文件描述符，表示当进程执行完后，domain 改变时继承 fd 的权限。
- 第八句，neverallow fork_test self:process fork; 是不允许 fork_test 对自己进行 process 的 fork 操作。

新增完策略之后，编译，提示有一个冲突。如下图所示，原本在 domain.te 文件中允许 domain 对自己有 process 的 fork 操作，但是在 fork_test.te 中又不允许，所以有冲突，将这里的 domain 减去 fork_test。

```
--- a/security/sepolicy-demo/src/domain.te
+++ b/security/sepolicy-demo/src/domain.te
@@ -15,7 +15,7 @@ allow domain tmpfs:dir r_dir_perms;
 allow domain self:capability sys_nice;

 # Intra-domain accesses.
-allow domain self:process {
+allow { domain - fork_test } self:process {
     fork
     sigchld
     sigkill
```

图 7-2: 禁止特定主体的 fork 权限

7.4.1.3 测试

如下图所示，/usr/bin/fork_test 与/usr/bin/fork_test_bak 内容完全一样，安全上下文有差异。

```
root@TinaLinux:/# md5sum /usr/bin/fork_test /usr/bin/fork_test_bak
5af8db5a4fd23524c3d9fe967d4971ed  /usr/bin/fork_test
5af8db5a4fd23524c3d9fe967d4971ed  /usr/bin/fork_test_bak
root@TinaLinux:/# ls -Z /usr/bin/fork_test /usr/bin/fork_test_bak
u:object_r:fork_test_exec:s0     /usr/bin/fork_test
u:object_r:system_file:s0        /usr/bin/fork_test_bak
root@TinaLinux:/# /usr/bin/fork_test
[14791.037466] type=1400 audit(23696.260:28): avc:  denied  { fork } for  pid=724 com
m="fork_test" scontext=u:r:fork_test:s0 tcontext=u:r:fork_test:s0 tclass=process perm
issive=0
fork error
root@TinaLinux:/# /usr/bin/fork_test_bak
this is parent, pid = 726
this is child
root@TinaLinux:/#
```

图 7-3: selinux-fork 测试 log

fork_test 执行时提示错误，错误信息中的 scontext 表示主体的上下文，tcontext 表示客体的安全上下文，tclass 表示操作的类型，fork 表示具体的操作，permissive=0 表示上述的操作不被允许。（如果 fork_test.te 中包含了 permissive fork_test; 那么这里也会出现提示语句，但是 permissive=1，表示允许 fork）。

fork_test_bak 文件的类型是 system_file，shell 执行该文件时，新进程的安全上下文就是 shell，shell 可以对自己进行 fork 操作，所以执行成功。

7.4.2 限制客体的访问权限

要求：限制只有特定进程 sunxi_info 才能访问/dev/sunxi_soc_info。

7.4.2.1 sunxi_info 源代码

```
#include<stdio.h>
#include<sys/types.h>
#include<sys/stat.h>
#include<fcntl.h>

#define CHECK_SOC_SECURE_ATTR 0x00
#define CHECK_SOC_VERSION     0x01
#define CHECK_SOC_BONDING     0x03

int main(void)
{
    int fd = 0;
    int cmd;
    int arg = 0;
    char Buf[4096];
```

```
    fd = open("/dev/sunxi_soc_info",O_RDWR);
    if(fd < 0)
    {
        printf("Open Dev Mem Erro\n");
        return -1;
    }

    cmd = CHECK_SOC_SECURE_ATTR;
    if(ioctl(fd,cmd,&arg) < 0)
    {
        printf("call cmd CHECK_SOC_SECURE_ATTR fail\n");
        return -1;
    }
    printf("CHECK_SOC_SECURE_ATTR get data is 0x%x\n\n",arg);

    cmd = CHECK_SOC_VERSION;
    if(ioctl(fd,cmd,&arg) < 0)
    {
        printf("call cmd CHECK_SOC_VERSION fail\n");
        return -1;
    }

    printf("CHECK_SOC_VERSION get data is 0x%x\n\n",arg);

    cmd = CHECK_SOC_BONDING;
    if(ioctl(fd,cmd,&Buf) < 0)
    {
        printf("call cmd CHECK_SOC_BONDING fail\n");
        return -1;
    }

    printf("CHECK_SOC_BONDING get data is %s\n\n",Buf);

    close(fd);
    return 0;
}
```

7.4.2.2 添加策略

在 tina/package/security/sepolicy-demo/src/file_contexts 文件中新增如下两行：

```
/usr/bin/sunxi_info                    u:object_r:sunxi_info_exec:s0
/dev/sunxi_soc_info                   u:object_r:sunxi_info_device:s0
```

在 tina/package/security/sepolicy-demo/src/device.te 文件中新增如下行：

```
type sunxi_info_device, dev_type;
```

新增 tina/package/security/sepolicy-demo/src/sunxi_info.te 文件，内容如下：

```
type sunxi_info, domain;
type sunxi_info_exec, exec_type, file_type;

init_daemon_domain(sunxi_info)
```

```
domain_auto_trans(shell,sunxi_info_exec,sunxi_info)

allow sunxi_info { serial_device sunxi_info_device }:chr_file rw_file_perms;
allow sunxi_info shell:fd use;
```

7.4.2.3 测试

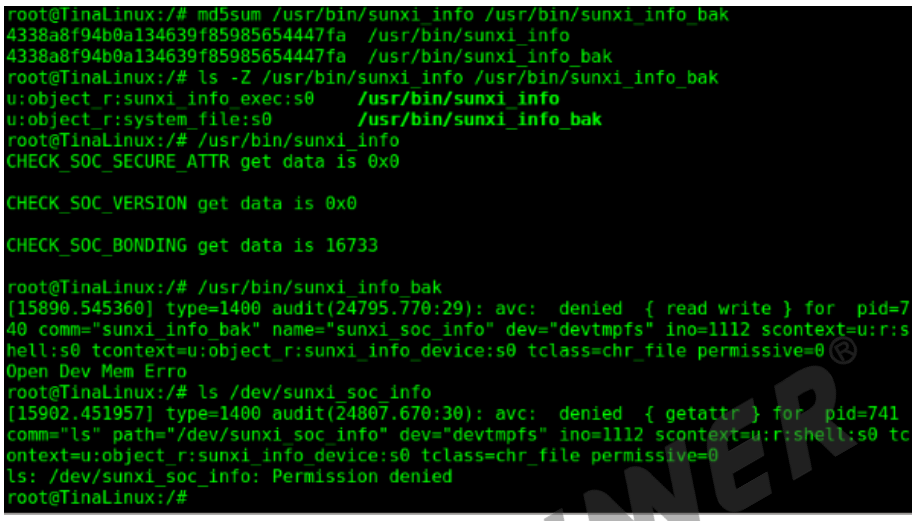

图 7-4: selinux-file 测试 log

测试结果，/usr/bin/sunxi_info 与/usr/bin/sunxi_info_bak 文件内容一样，安全上下文不同，只有/usr/bin/sunxi_info 才能正确执行。/usr/bin/sunxi_info_bak 执行的时候，提示 shell 对 sunxi_info_device 设备没有 chr_file 的 read 与 write 权限。

最后一条指令"ls /dev/sunxi_soc_info"执行也失败，提示 shell 对 sunxi_info_device 设备没有 chr_file 的 getattr 权限（获取属性权限）。

8 量产工具

从整个安全系统的角度看，需要一整套工具来配合完成对应的工作。

8.1 RSA 密钥对生成工具

目前，有公开的密钥对生成工具 openssl，可以生成足够长度的密钥对。

Tina 开发平台 scripts 下提供了一个生成密钥对的脚本 createkeys，该脚本调用 dragonsecboot 工具，解析 dragon_toc*.cfg 中 [key_rsa] 字段，并基于字段的内容生成对应名字的密钥对。

关于 dm-verity 所需要的 keys 是由 tina/scripts/dm-verity-key.sh 生成。

8.2 安全固件版本管理

安全固件打包时会解析`version_base.mk`文件决定。

在 efuse 中会有一块区域用来记录固件版本。

当设备启动时，会将 efuse 中记录的版本号同固件中的版本号比较，如果固件中的版本较低，则不能继续启动；如果固件中的版本比较高，将固件中的版本写入 efuse，继续启动；如果版本相同，正常启动。可防止固件版本回退。

8.3 数据封包工具

Tina 开发平台中提供固件封包工具 dragonsecboot，在安全固件打包过程中会对相关的镜像文件（sboot、uboot、kernel 等）进行签名，并生成证书以及相关信息，以便启动时对这些镜像文件进行校验，验证完整性。

8.4 烧 key 工具

烧 key 工具用来将 rotpk.bin 烧写到设备的 efuse 中，efuse 位于 IC 内部，由于 efuse 中内容一旦写入便不可更改，所以从根源上保证了根证书公钥 hash 的安全性。

可用的烧 key 工具包含 DragonKey 或者 DragonSN，工具的使用说明位于工具包中。

8.5 关闭 jtag

将 sys_config.fex 中 jtag_para 节下的 jtag_enable 设置为 0 即可关闭 jtag 调试功能。

8.6 密钥说明

8.6.1 固件签名密钥

密钥	安全固件签名私钥
功能	RSA2048 类型私钥。对 sboot、monitor、scp、optee、uboot、boot、rootfs 等分区进行签名
SDK 路径	`tina/out/${BOARD}/keys/*.pem`与`tina/out/${BOARD}/keys/*.bin`，除开 rotpk.bin
设备位置	设备上不保存
烧写方式	不烧写
保密	是

密钥	安全固件签名公钥
功能	RSA2048 类型公钥。对 sboot、monitor、scp、optee、uboot、boot、rootfs 等分区进行验签
SDK 路径	位于`tina/out/${BOARD}/image/toc0`以及`tina/out/${BOARD}/image/toc1`目录下的证书中
设备位置	flash 上 TOC0、TOC1、boot、rootfs 等分区中
烧写方式	随固件一起烧写
保密	否

8.6.2 efuse 中密钥

密钥	rotpk
功能	签名根密钥公钥的 sha256 值，用于安全启动中根证书的校验。长度 256bit。
SDK 路径	`tina/out/${BOARD}/keys/rotpk.bin`
设备位置	IC 中 efuse 中的 rotpk 区域
烧写方式	DragonSN 等，参考 3.4 小节
保密	否

密钥	ssk
功能	对称密钥。可用于 DragonSN 烧写 keybox 时对待烧写的内容进行加密，可用于 TA 加密
SDK 路径	SDK 中没有
设备位置	IC 中 efuse 中的 ssk 区域
烧写方式	DragonSN
保密	是

密钥	huk
功能	对称密钥。可用于 rotpk_na 烧写 keybox 时对待烧写的内容进行加密，可用于 OPTEE Secure Storage 对数据进行加密。
SDK 路径	SDK 中没有
设备位置	IC 中 efuse 中的 huk 区域
烧写方式	对于 MR813/R818/R528，安全固件第一次启动时自动使用 CE 产生的随机数进行烧写。其他方案通过 DragonSN 烧写
保密	是

8.6.3 dm-verity 密钥

密钥	dm-verity 私钥
功能	RSA2048 类型私钥。用于对 rootfs 的 hash table 进行签名
SDK 路径	位于`tina/out/${BOARD}/verity/keys/dm-verity-pri.pem`与`tina/package/security/dm-verity/files/dm-verity-pri.pem`
设备位置	设备上不保存
烧写方式	不烧写
保密	是

密钥	dm-verity 公钥
功能	RSA2048 类型公钥。用于在 initramfs 启动脚本中对 rootfs 的 hash table 进行验签
SDK 路径	位于`tina/out/${BOARD}/verity/keys/dm-verity-pub.pem`、`tina/package/security/dm-verity/files/dm-verity-pub.pem`以及`tina/out/${BOARD}/compile_dir/target/rootfs_ramfs/verity_key`
设备位置	flash 上 boot 分区中的 initramfs 文件系统中
烧写方式	随固件 boot 分区一起烧写
保密	否

8.6.4 TA 签名密钥

密钥	TA 签名私钥
功能	RSA2048 类型私钥。用于对 TA 进行签名。没有签名或签名错误的 TA 将不会运行
SDK 路径	位于`tina/package/security/optee-os-dev-kit/dev_kit/arm-plat-{CHIP}/export-ta_arm32/keys/default_ta.pem`与`tina/out/{BOARD}/staging_dir/target/usr/dev_kit/arm-plat-{CHIP}/export-ta_arm32/keys/default_ta.pem`
设备位置	设备上不保存
烧写方式	不烧写
保密	是

密钥	TA 签名公钥
功能	RSA2048 类型公钥。用于 OPTEE 对 TA 进行验签。验签失败的 TA 将不会运行。
SDK 路径	`tina/device/config/chips/${CHIP}/bin/optee_${IC}.bin`与`tina/out/${BOARD}/image/optee.fex`二进制文件中包含 TA 签名公钥
设备位置	flash 上 TOC1 分区中的 optee 内
烧写方式	随固件 TOC1 一起烧写
保密	否

8.6.5 TA 加密密钥

密钥	TA 加密密钥
功能	对称密钥。用于对 TA 进行加密。密钥来源可以是 ssk 或由 rotpk 派生而来。长度 128bit。
SDK 路径	`tina/package/security/optee-os-dev-kit/dev_kit/arm-plat-{CHIP}/export-ta_arm32/keys/ta_aes_key.bin`与`tina/out/{BOARD}/staging_dir/target/usr/dev_kit/arm-plat-{CHIP}/export-ta_arm32/keys/ta_aes_key.bin`
设备位置	IC 中 efuse 中的 ssk 或 rotpk 区域
烧写方式	DragonSN
保密	是

8.6.6 dm-crypt 密钥

密钥	dm-crypt 密钥
功能	对称密钥。用于对 dm-crypt 分区文件系统数据进行加密。dm-crypt.sh 脚本中可使用三种类型的 key：keyfile、pass 与 optee-pass，建议使用 optee-pass。
SDK 路径	SDK 中没有该密钥
设备位置	对于 keyfile 类型，位于根文件系统/encrypt-key-file，此文件经过加密，使用时需要输入 passphrase 进行解密，keyfile 最大 8192kiB；对于 pass 类型，不保存，使用时实时输入 passphrase，passphrase 最大长度 512B；对于 optee-pass 类型，保存在 flash 上的 keybox 中，长度 256bit。
烧写方式	对于 keyfile 类型，执行 dm-crypt.sh 时，写到根文件系统根目录；对于 pass 类型，不需要烧写；对于 optee-pass 类型，通过 DragonSN 或 keybox_na 进行烧写。
保密	是

8.6.7 rpmb 密钥

密钥	rpmb 密钥
功能	对称密钥。用于访问 RPMB 时身份认证。长度 256bit。
SDK 路径	SDK 中没有该密钥
设备位置	保证在 flash 上的 keybox 中，同时保存在 eMMC 中的 OTP 区域中。
烧写方式	通过 DragonSN 或 keybox_na 进行烧写。
保密	是

9 参考资料

9.1 TrustZone

[1] PRD29-GENC-009492C_trustzone_security_whitepaper.pdf

9.2 GlobalPlatform

[1] GPD_TEE_SystemArch_v1.1.pdf

[2] GPD_TEE_Client_API_v1.0_EP_v1.0.pdf

[3] GPD_TEE_Internal_Core_API_Specification_v1.1.pdf

[4] GPD_TEE_TA_Debug_Spec_v1.0.pdf

9.3 OP-TEE

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

9.4 Dm-verity

[1] https://source.android.com/security/verifiedboot/?hl=zh-cn

[2] Documentation/device-mapper/verity.txt

9.5 SELinux

[1] https://github.com/SELinuxProject/selinux

[2] https://github.com/TresysTechnology/refpolicy/wiki

[3] https://source.android.com/security/selinux

著作权声明

商标声明

免责声明