TinaLinux

Security 开发指南

1.6

2019.07.17

文档履历

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 校验 rootfs，rotpk 烧写方法
1.6	2019.07.17	AWA0916	新增 TA 签名，更新 uboot 校验 rootfs

目录

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）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 芯片

软件平台: Tina V3.5 及其后续版本。

1.3 相关人员

适用于 TinaLinux 平台的客户及相关技术人员。

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。

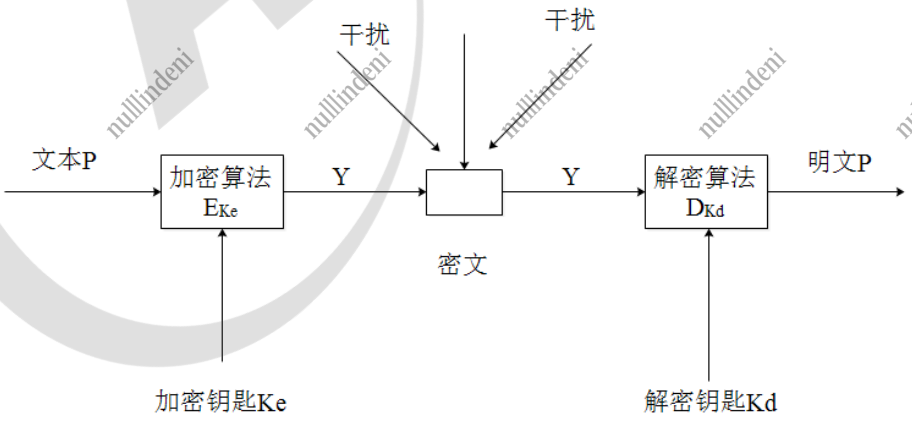

图 1: 数据加密模型

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。

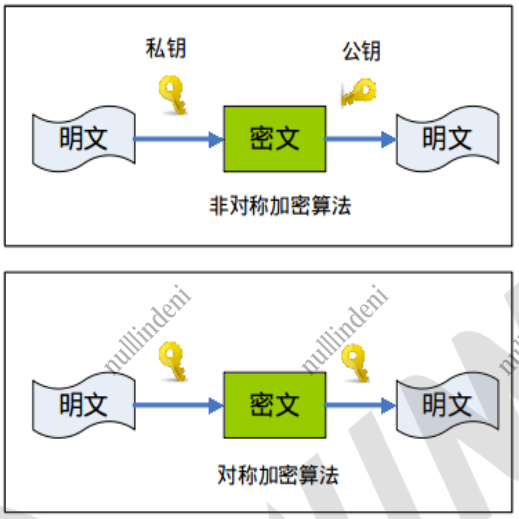

图 2: 对称/非对称加密算法

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。

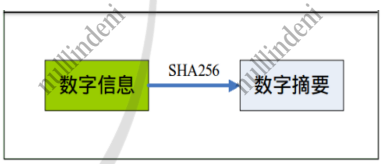

图 3: SHA256 算法

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。

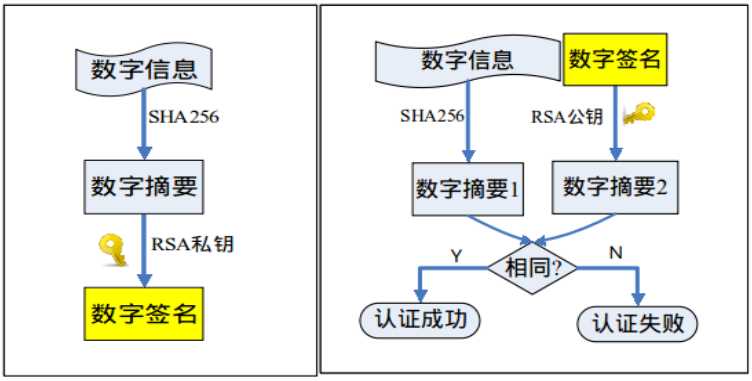

图 4: 数字签名与认证

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。

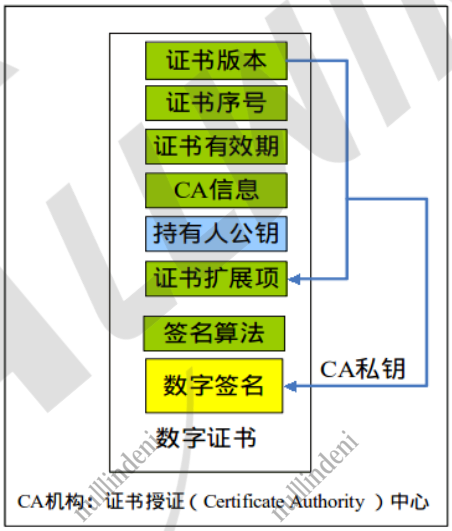

图 5: 数字证书

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》。

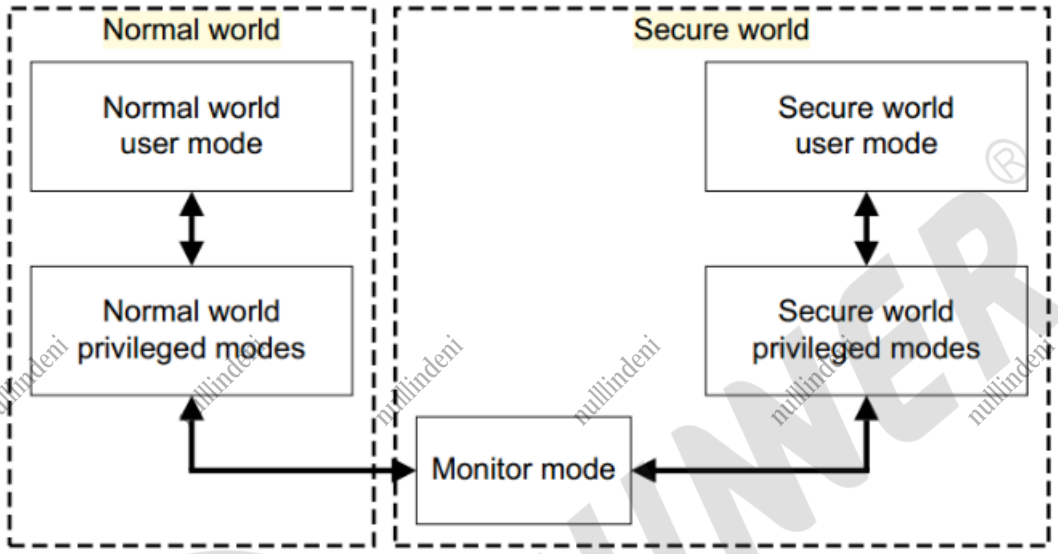

图 6: TrustZone 模型

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。

当前 Tina 中采用的是 OP-TEE 系统。OP-TEE 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，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。

2.4 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 R 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。其中关键的模块描述如下。

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。

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。

2.5 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。
- 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。

3. Secure Boot

安全启动，即 Secure Boot，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。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，再将公钥的 hash 值即 rotpk.bin 烧写至芯片的 efuse 区域。启动时，固化在芯片的 brom 程序首先会读取 efuse 中的 hash 值，将该值与根证书中的公钥 hash 值进行匹配，验证根证书中公钥的可信任性。然后会使用 flash 中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。验证顺序为芯片的 brom->sboot->monitor/secure os->uboot->boot.img。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，如R18开发方案对应的是tulip_*，R30对应koto_*，R328对应cowbell_*等
$ make [-jN]
==> 编译，-jN 参数选择并行编译进程数量
$ ./scripts/createkeys
==> 不需要每次执行，详见下面的注2。创建密钥对，选择对应平台，会在out/<board>/keys/目录下生成签名所需的密钥对。
$ pack -s [-d]
==> 打包，-s 表示制作安全固件。-d参数使生成的固件包串口信息转到tf卡座输出。
```

1. 注：在执行 **make** 前，请确保包含如下配置。

   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)
```

   此外，还需要将 secure 环境所用的内存设置为内核的保留内存（起始地址请咨询项目安全负责人）。

   以 R328 为例，如果 secure 环境使用了 4M 内存（其中共享内存 1M，secure os 1M，TA 2M），按照如下补丁修改 tina/lichee/linux-4.9/arch/arm/boot/dts/sun8iw18p1.dtsi 文件。

```
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>
```

2. 注：客户在首次执行打包动作之前，必须运行一次**./scripts/createkeys** 创建自己的密钥对组，并将创建的秘钥妥善保存。关于 **createkeys** 说明如下：

- 执行 createkeys 后，会根据 tina/target 对应目录下的 dragon_toc*.cfg 生成一组用作签名固件的 keys，这些 keys 保存在 out/<board>/keys/目录下。执行 pack -s 时，会使用这些 keys 对对应的镜像进行签名并生成证书。
- 请复制 out/<board>/keys/目录下 keys 到自己的私密目录，其中 Trustkey.bin，Trustkey.pem 与 rotpk.bin 三个文件要注意重点保护。
- rotpk.bin 是根密钥的 Hash，可借助烧 key（DragonSN）工具烧写到 efuse 里。Trustkey.bin 与 Trustkey.pem 是根密钥，不能泄漏和丢失。丢失与泄露会导致一系列问题，如 SDK 生成的固件无法在芯片上启动、失去防刷机功能等。

3. 注：**pack -s** 打包完成后，生成安全镜像，该镜像在 out/<board>/keys/目录下，文件名为 tina_<board>_<uart0/card0>_secure_v[NUM].img。其中 v[NUM] 表示固件的版本信息，NUM 为版本号，由 target/allwinner/<platform>-common/version/version_base.mk 或者 target/allwinner/generic/version/version_base.mk 文件决定，前者优先级更高。特别注意的是：**R** 系列最高支持 **32** 个版本。在系统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。如果希望启用安全固件版本管理功能，请确保板子的 efuse 有供电，对于部分 R328 方案，由于考虑硬件成本，出厂的设备 efuse 没有供电，此功能不能用。

生成秘钥与打包过程中涉及的 dragon_toc.cfg 文件说明如下:

- 安全固件会将 sboot.bin 封装成 toc0.fex;
- 安全固件将 optee/uboot/dts 等封装成 toc1.fex;

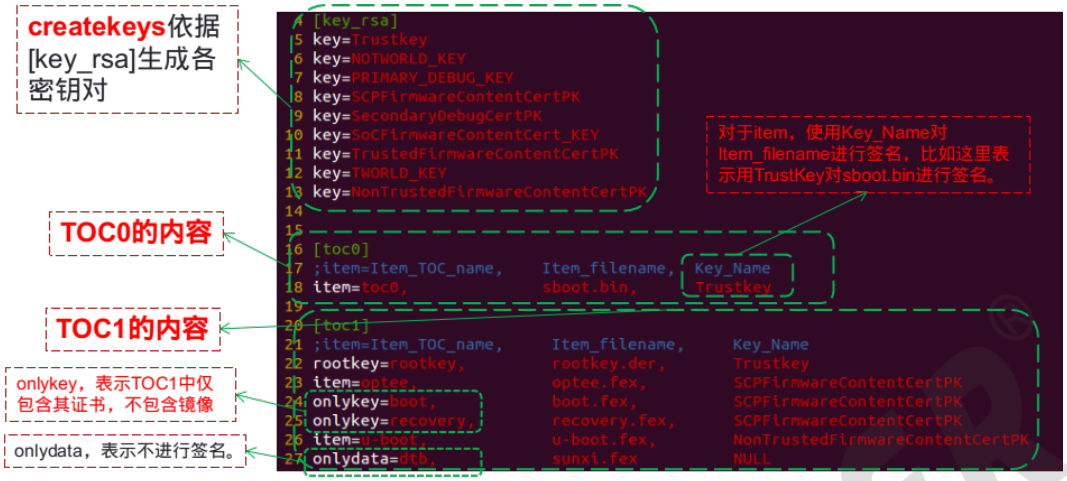

图 7: dragon-toc 配置文件说明

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。
3. 烧写安全固件到 flash 中。

注 **1**：不同的 **IC**，**efuse** 大小不同，如 **R328 efuse** 为 **1024bit**。**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，每个区域也只能烧写一次。

注 **2**：烧写 **secure enable bit** 后，会让设备变成安全设备，此操作是不可逆的。后续只能启动安全固件，启动不了非安全固件。

注 **3**：正常情况下，通过 **LiveSuit/PhoenixSuit** 烧写安全固件完成时会自动烧写 **secure enable bit**。

注 **4**：如果既烧写 **secure enable bit**，又烧写 **rotpk.bin**，设备就只能启动与 **rotpk.bin** 对应 **keys** 签名的安全固件；如果只烧写 **secure enable bit**，没有烧写 **rotpk.bin**，此设备上烧写任何安全固件都可以启动。调试时可只烧写 **secure enable bit**，但是客户产品出厂前必须要烧写 **rotpk.bin**。所以请务必注意保存好 **keys**，尤其是 **rotpk.bin, Trustkey.bin** 和 **Trustkey.pem**。

注 **5**：为节省成本，某些硬件方案上 **efuse** 供电不足，导致不能写入。因此，在所有写 **efuse** 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** 等场景会写数

据到 **efuse** 中。

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 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 有开发支持。需要 uboot 源码中有定义 SID_ROTPK_CTRL，同时 tina/allwinner/<board>/configs/env-x.x.cfg 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（版本号一致），如果不能启动，则表明已经烧写 rotpk。

3.4 烧写 rotpk 与烧写 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将 efuse 中的 secure enable bit 置 1。
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 工具，可以将 key（SN 号、MAC 地址、rotpk 等）烧录到 private 分区、efuse 或 keybox 中，当前仅支持 windows。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 流程如图所示。uboot 获取到 DragonSN 下发的 key 数据，将其传送的 Secure OS，Secure OS 调用 efuse 驱动将 key 数据写入到 efuse 中。

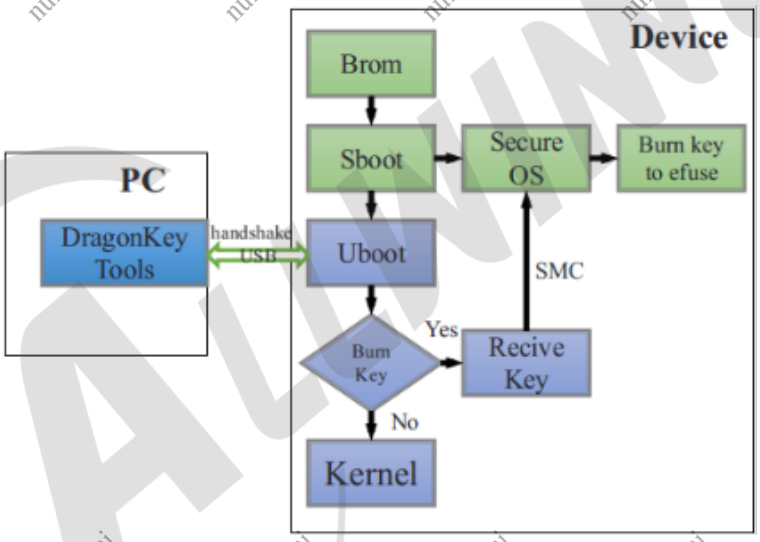

图 8: DragonSN 烧写 efuse 流程

3.4.1.2 DragonSN 烧写 rotpk 步骤

DragonSN 烧 rotpk 具体步骤如下：

- 设置 burn_key 属性为 1。设置 burn_key 的属性为 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 tina/target/allwinner/<board>/configs/sys_config.fex 文件中 [target] 项下，如下图所示。如果未显式配置，按照 burn_key=0 处理。

```
[target]
boot_clock      = 1008
storage_type    = 5
burn_key        = 1
```

图 9: burnkey 配置

- 重新打包安全固件并烧录。
- 在 PC 端配置 DragonSN 工具后（烧写 rotpk 关键配置如下图所示），将设备通过 usb 与 pc 连接，重启设备。注意烧录的 rotpk 必须与安全固件签名的 key 对应。

图 10: rotpk 烧写配置

- DragonSN 显示设备已连接后，开始烧录。为了保证不会烧录错误的 rotpk，在烧录过程中设备会做额外的处理，即将 PC 端下发的 rotpk 与当前固件中根证书公钥的 hash 值进行对比，匹配后才烧录该 rotpk。

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** 有开发支持。

注：要支持此功能，需要在 uboot 中 configs/sun*iw*p1*_defconfig 文件中打开如下宏：CONFIG_SUNXI_BURN_ROTPK_ON_SPRITE=y

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。

3.4.3 方法三

方法三是在 Linux 用户空间烧写 rotpk 与 secure enable bit。由于 rotpk 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 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：量产时比较方便，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。

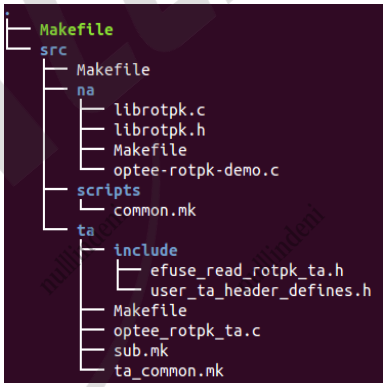

图 11: 烧写 rotpk 的 API 源码

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);

/**
 * write_seucre_enable_bit() - write secure enable bit in efuse.
 *
 * return value: zero, write success; non-zero, write failed.
 */
int write_secure_enable_bit(void);
```

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.
   s write enable secure bit in 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。
```"rotpk_na s"```，烧写 efuse 中的 secure enable bit。
```

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
  Security --->
    OPTEE --->
      -*- optee-client..... optee-client
      -*- optee-os-dev-kit..... optee-os-dev-kit
      <*> optee-rotpk..... OPTEE rotpk read write
```

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。

3.5.1 uboot 校验 squashfs 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：当前该方法仅支持 **squashfs** 类型的 **rootfs**，同时仅 **R328** 有开发支持。

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 target/allwinner/<board>/configs/env-x.x.cfg 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//image/rootfs.fex 进行采样，获取文件 out//image/rootfs-extract.fex。使用秘钥 SCPFirmwareContentCertPK 对该 rootfs-extract.fex 进行签名，生成证书 out//image/toc1/cert/rootfs.der。然后使用工具 update_squashfs 将该 rootfs.der 证书附着在 out//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/sun8iw18p1*_defconfig 中开启了 CONFIG_SUNXI_PART_VERIFY=y 的配置。

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 技术，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。

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下图所示。

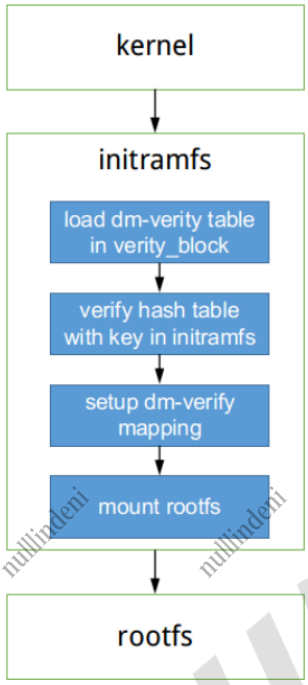

图 12: dm-verity 验证 rootfs 流程

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** #cmd to config target/allwinner/cowbell-perf1/defconfig_ramfs

注：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。为节省空间，对于不需要的应用请尽可能关闭。

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 --- busybox-init-base-files
        └—> [*] Customize busybox init base files options
        └—> (busybox-init-base-files_ramfs) PATH for busybox base files
└—> Security --->
    └—> <*> cryptsetup
        └—> Device Mapper --->
            └—> use crypt lib (use libopenssl) --->
└—> Utilities --->
    └—> <*> openssl-util
```

4. **make_ramfs [-jN]**

注：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

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
└—> Device Drivers --->
    └—> [*] Multiple devices driver support (RAID and LVM) --->
        └—> <*> Device mapper support
            └—> <*> Verity target support
```

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

生成一组 rsa key，保存在 out/cowbell-perf1/verity/keys 中，同时将公钥 **rsa.pk** 复制到 **out/cowbell-perf1/compile_dir/target/rootfs_ramfs**，命名为 **verity_key** 中。

注 **1**：另一个 rsa_key.pair 是非常重要的隐私数据，用以对 dm-verity table 进行签名，请妥善保存。

注 **2**：此脚本仅需执行一次。如希望更换一次 key，再运行一次。注意公钥 rsa.pk 是否复制到了 out/cowbell-perf1/compile_dir/target/rootfs_ramfs 目录。

6. **make -j**

注:在 **pack** 之前请确认 **target/allwinner/cowbell-perf1/configs/**下包含 **sys_partition_secure.fex** 文件，此文件相对于 **sys_partition.fex** 文件的差异就是新增了一个 **verity_block** 分区：

```
--- sys_partition.fex 2018-09-19 09:25:42.690203172 +0800
+++ sys_partition_secure.fex 2018-09-22 15:44:55.895133052 +0800
@@ -82,5 +82,11 @@
        user_type = 0x8000

 [partition]
+ name = verity_block
+ size = 2048
+ downloadfile = "verity_block.fex"
+ user_type = 0x8000
+
+[partition]
        name = UDISK
        user_type = 0x8100
```

7. **pack -s [-d]**

-s 表示制作安全固件。-d 参数使生成的固件包串口信息转到 tf 卡座输出。

通过以上步骤后，就可以生成一个支持 dm-verity 的安全固件。

3.5.2.3 dm-verity 测试

常见的测试方法如下：

- 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)
```

- 方法三：更换 rootfs.img 进行验证

在 scripts/pack_img.sh 文件中新增如下行，即在构建好 verity_block，用一个新的 rootfs-new.img 来替换与 verity_block 匹配的 rootfs.img。

```
grep "CONFIG_USE_DM_VERITY=y" ${PACK_TOPDIR}/.config > /dev/null
if [ $? -eq 0 ]; then
        ${PACK_TOPDIR}/scripts/dm-verity-block.sh ${ROOT_DIR}/rootfs.img ${ROOT_DIR}/verity/verity_block

        if [ -f ${ROOT_DIR}/verity/verity_block ]; then
                ln -s ${ROOT_DIR}/verity/verity_block verity_block.fex
        fi
        cp -rf ${ROOT_DIR}/rootfs-new.img ${ROOT_DIR}/rootfs.img
fi
```

Tina 启动过后，挂载根文件系统时会提示：

```
[2.651314] device-mapper: verity: 179:5: data block 11 is corrupted
[2.651359] EXT4-fs error (device dm-0): __ext4_get_inode_loc:4117: inode #2: block 11: comm mount: unable to read itable block
[2.651417] Buffer I/O error on dev rootfs, logical block 0, lost sync page write
[2.651435] device-mapper: verity: 179:5: data block 31 is corrupted
[2.651457] EXT4-fs (dm-0): get root inode failed
[2.651461] EXT4-fs (dm-0): mount failed
```

- 方法四：修改 rootfs 内容进行验证

将 rootfs 的格式选成 ext4。在 initramfs 新增如下内容，即在使用 dm-verity 之前，将其先挂载成可写，并写入一个文件，然后将其卸载，再进行 dm-verity。

```
# test dm-verity
mount -t ext4 -o rw $1 /mnt
touch /mnt/test
umount $1

# create rootfs
veritysetup create rootfs --hash-offset=${hash_offset} $1 $2 $root_hash -s $salt
```

Tina 启动过后，挂载根文件系统时会提示：

```
[ 735.369396] device-mapper: verity: 179:5: data block 0 is corrupted
[ 735.369396] EXT4-fs (dm-0): unable to read superblock
mount: mounting /dev/mapper/rootfs on /mnt/ failed: Invalid argument
```

3.5.2.4 dm-verity 影响

- 占用 flash 空间需要新增 dm-verity 分区。同时会用到 initramfs，导致内核镜像增大。
- OTA 升级如果对 rootfs.img 升级，verity_block 也需要一起升级，否则将挂载根文件系统会出错。

注：当前仅 AB 分区公用一个 verity_block。如果 AB 分区分别对应一个 verity_block，需要建立两个 verity_block 分区，并修改 tina/package/busybox-init-base-files/busybox-init-base-files_ramfs/init_sunxi，然后重新编译 initramfs 以及 rootfs。

- 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，会延长降低启动时间，降低对 rootfs 分区的读取速度。

3.6 安全启动带来的影响

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 频率等因素的不同而上下波动。

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，因此在做 ota 升级时，请确保选中正确的文件。

- 升级 optee/uboot/dts/sys_config，需要使用 tina/out/<TARGET_BOARD>/image/toc1.fex 文件；
- 升级 sboot，需要使用 tina/out/<TARGET_BOARD>/image/toc0.fex 文件；
- 升级 linux kernel，需要使用 tina/out/<TARGET_BOARD>/image/boot.fex 文件；
- 升级 rootfs，需要使用 tina/out/<TARGET_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 SOC 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：

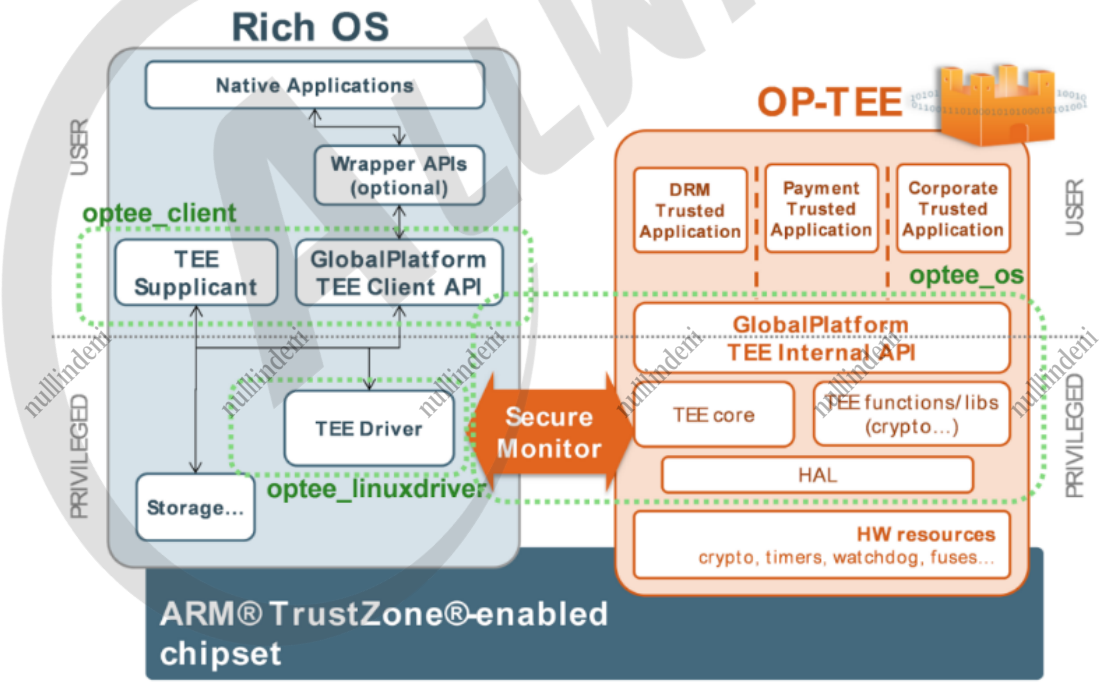

图 13: optee 总体架构

4.2 如何开启 Secure OS

4.2.1 Secure OS 镜像

Tina 安全固件在打包时加入 -s 选项，就会自动把 Secure OS 镜像打包到安全固件中。Secure OS 镜像位于 tina/target/allwinner/<platform>-common/bin/optee_*.bin。

TEE 环境使用的内存有 3 个部分，各部分大小需要在 Secure OS 编译时指定。各部分作用如下：

- 共享内存。用于 REE 与 TEE 的数据交换，TrustZone 技术中 REE 与 TEE 的交互通过 smc 进行，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。共享内存 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。

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 也预留了 3M 的内存。

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

图 14: 内核预留内存

注：配置预留内存是与 **Secure** 环境使用的内存对应的，需要按照 **optee_os** 编译时指定的大小来设置。

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，提供 CA 所需的 tee-supplicant 以及 libteec 库
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 secure 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,OEM_SEC 区域的 TA/CA demo 程序。

上面 1 与 2 是开发 TA/CA 所需的环境，3、4、5、6 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。

注：要使用 TA/CA 开发环境，前提是要支持 Secure boot 以及 Secure OS。

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，需先开启 1、2 两个 packages，执行 make menuconfig，开启如下选项：

```
Tina Configuration
└—> Security --->
    └—> OPTEE --->
        └—> < > optee-base64..... OPTEE base64 demo
        └—> <*> optee-client..... optee-client
        └—> < > optee-efuse-read..... OPTEE efuse read demo
```

```
└─> < > optee-helloworld..... OPTEE Hello World
└─> <*> optee-os-dev-kit..... optee-os-dev-kit
└─> < > optee-secure-storage..... Secure Storage in OP-TEE
```

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 15: 编译选项设置

5.3 TA 签名

Tina 上支持更换 TA 签名 key。如果未进行特殊设置，TA 签名会使用默认的 key，该 key 路径位于 package/security/optee-os-dev-kit/dev_kit/arm-plat-sun×*iw*p1/export-ta_arm32/keys/default_ta.pem。

如果需要进行更换，可使用 openssl genrsa -out default_ta.pem 2048 命令重新生成一个，对默认 key 进行替换。

在编译 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。该脚本使用说明如下：

```
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。

当前，Tina 支持在 **R328** 方案上将编译出 TA 二进制加密后再签名，其他方案暂未开发。

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 with efuse ssk key
```

tina/package/security/optee-os-dev-kit/dev_kit/arm-plat-sun8iw18p1/export-ta_arm32/keys/ta_aes_key.bin 文件为加密密钥。

注：默认使用 ssk 作为加密秘钥，因此开启 TA 加密功能之前，需要烧写 efuse 上的 ssk。加密密钥也可以自定义，此时需要对 optee 进行开发。

5.5 keybox

由于 efuse 空间受限，Tina 上支持了 keybox 功能。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 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。

安全方案默认均开启了 keybox 功能。

5.5.1 keybox 烧写及读取流程

keybox 烧写流程如下图所示。可以看到 keybox 也是通过 DragonSN 工具来进行烧写。

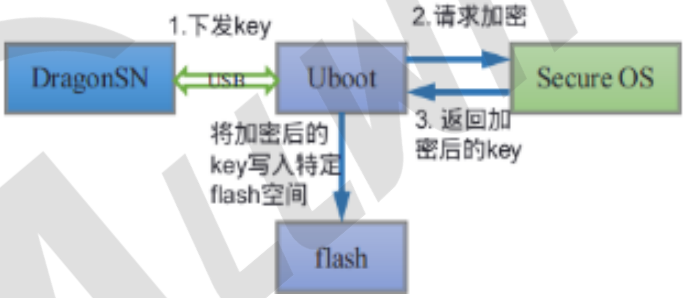

图 16: keybox 烧写流程

keybox 读取流程如下图所示。在系统启动时 uboot 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。

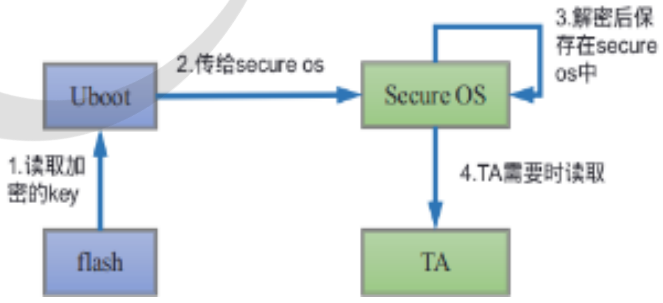

图 17: keybox 读取流程

5.5.2 烧写 efuse 与 keybox 时 DragonSN 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 18: 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、ssk 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。

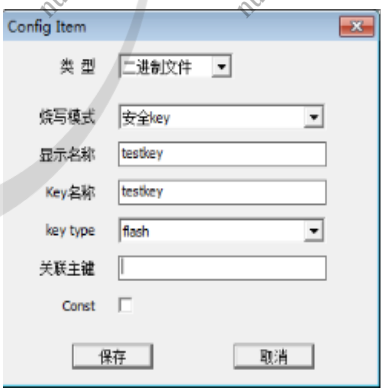

图 19: keybox key 烧写参考配置

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，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。

5.5.3 keybox 列表

这里也分为两种情况。对于非 R328 方案，启动时，所有加密的 key 都会别加载至 secure os 中进行解密。

对于 R328 可以通过环境变量 keybox_list 来选择加载至 secure os 中的 key。keybox_list 环境变量位于 target/allwinner/<board>/configs/env-<kernel_version>.cfg 中，使用逗号分隔各 key。比如下面的例子中，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
```

5.6 安全应用 demo

5.6.1 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
```

5.6.2 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。

注：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
```

5.6.3 optee-base64 效果

该 demo 在 TA 端软件实现了 base64 算法。

Base64 使用说明："-e" 后为需要编码的字节串，两个字符对应一个字节；"-d" 后为需要解码的字符串。

```
root@TinaLinux:/# tee-supplicant
root@TinaLinux:/# base64_demo_na -e 123456
input bytes:
0x12 0x34 0x56
NA:open session
TA:creatyentry!
TA:open session!
NA:allocate memoryTA:rec cmd 0x221
input size:3
encode result:
EjRW
NA:finish with 0
root@TinaLinux:/# base64_demo_na -d EjRW
NA:open session
TA:creatyentry!
TA:open session!
NA:allocate memoryTA:rec cmd 0x222
input size:4
decode result:
0x12 0x34 0x56
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 OP-TEE 中的 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）中。此功能与具体的设备绑定，充分保证了数据的私密性与完整性。

注 **1**：**Secure Storage** 依赖 **Secure OS**，因此只有安全固件中才包含 **OP-TEE Secure Storage** 功能。

注 **2**：当前仅 **R18**、**R328** 完全支持 **Optee Secure Storage** 功能，具体原因详见 **6.3.1** 小节。

6.1 OP-TEE Secure Storage 功能框架

OP-TEE Secure Storage 的软件架构如下图所示。

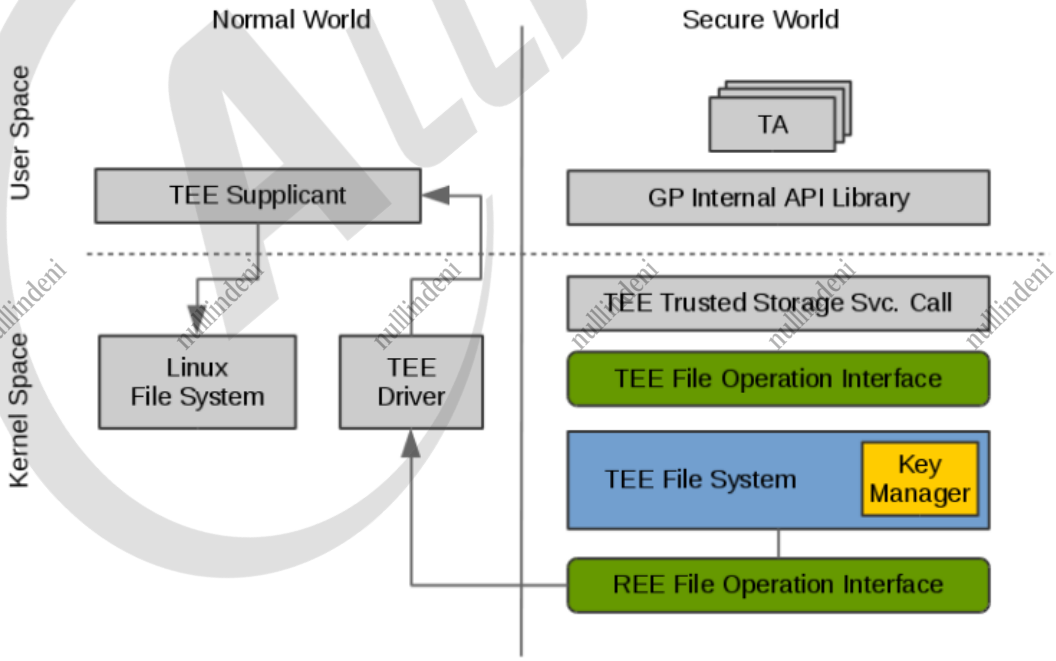

图 20: OP-TEE Secure Storage 软件架构

6.2 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。

6.3 安全存储 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)。

6.3.1 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, Chip ID || ``static string")

其中 HUK 为 Hardware Unique Key.

注：这里的 **HUK** 是通过 **tee_otp_get_hw_unique_key** 函数获取的。对于 **R328** 来说，该函数会读取 **efuse** 中 **chipid** 的内容并进行派生；对于 **R18** 来说，该函数会获取 **efuse** 中 **huk** 内容的前 **128bit**；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**。

6.3.2 TA Storage Key - TSK

TSK 是一个 per-Trusted Application key，用来对 FEK 进行加解密。SSK 由如下公式计算得出：

$TSK = HMAC_{SHA256}$ (SSK, TA_UUID)

6.3.3 File Encryption Key - FEK

当创建一个 TEE 文件时，key manager 会通过 PRNG(pesudo random number generator) 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 文件信行加解密。

6.3.4 Meta Data 加密流程

Meta Data 加密流程如下图所示。

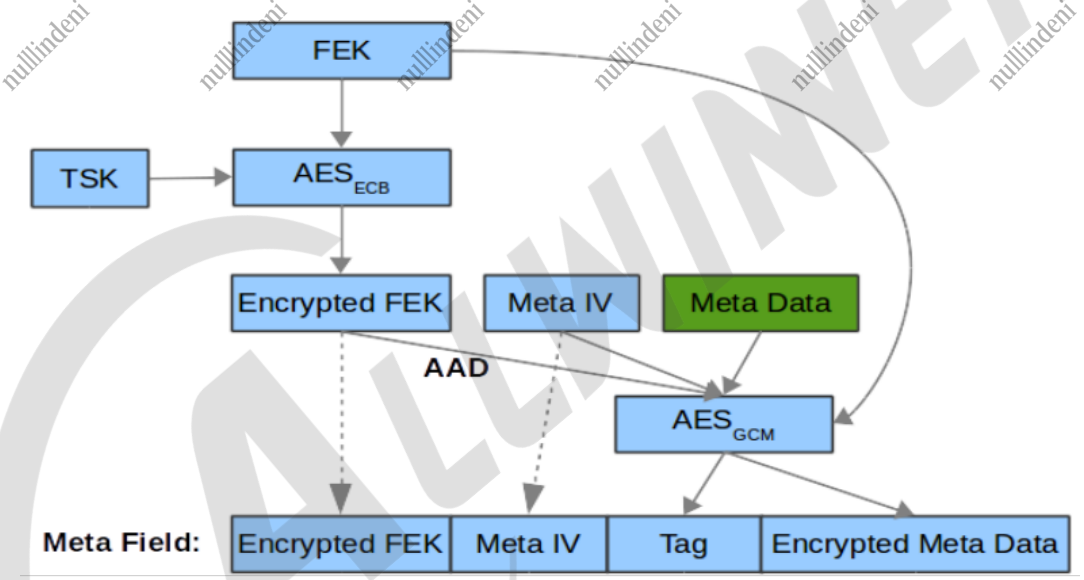

图 21: Meta Data 加密流程

6.3.5 Block data 加密流程

Block data 加密流程如下图所示。

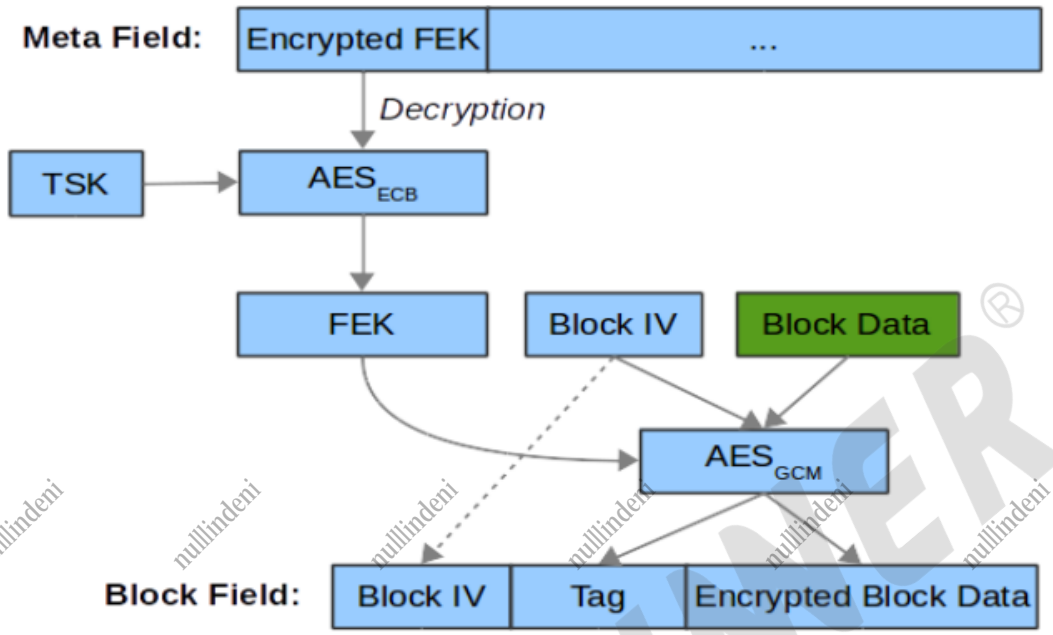

图 22: Block Data 加密流程

6.4 Tina OP-TEE 安全存储 demo

Tina 的安全存储是基于 OP-TEE 的 Secure Storage 技术来实现的。根据 Secure Storage 的软件架构，我们在 Tina 上加入了三个方面内容。

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
```

6.4.1 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。

6.4.2 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。

6.4.2.1 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 端，最后一种表示存储在 mmc RPMB 分区，目前尚未支持，所以这个配置为 TEE_STORAGE_PRIVATE_REE 即可。

6.4.2.2 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

  前面两种支持文件加密存储在 REE 端 data/tee 端，最后一种表示存储在 mmc RPMB 分区，目前尚未支持，所以这个配置为 TEE_STORAGE_PRIVATE_REE 即可。

6.4.2.3 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

6.4.2.4 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

6.4.2.5 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

6.4.3 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 [options] [file name]
    [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
```

当运行"ss_demo -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。

6.5 Tina OP-TEE 安全存储使用

6.5.1 开启 Secure Storage 支持

6.5.1.1 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
    Security --->
        OPTEE --->
            -*- optee-os-dev-kit
            -*- optee-client
            <*> optee-secure-storage
```

6.5.1.2 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
```

6.5.1.3 设置 dts

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

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

图 23: 设置 dts

6.5.2 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。

6.6 OP-TEE 安全存储使用效果

```
root@tulip-mozart:/# tee-supplicant &
root@tulip-mozart:/# ss_demo
usage: ss_demo [options] [file name]
  [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
root@tulip-mozart:/# ss_demo -c test.file
root@tulip-mozart:/# ls /data/tee/
0 dirf.db
root@tulip-mozart:/# ss_demo -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@tulip-mozart:/# ss_demo -w test.file
---- Write file:test.file with 256 Bytes data: ----
26 4e b4 c6 37 94 7d 98 35 10 c6 78 24 40 92 0e
ec 46 fc 43 b0 3f 15 17 1a a3 04 22 36 88 53 aa
6e be 63 b7 49 d0 3e 07 90 18 ae b5 5b 93 b6 43
33 bf 6a 96 4a 6b 79 5a a7 cf 30 71 50 4d af 16
eb 02 47 f3 20 b7 bf f9 76 ac 54 56 4e b8 11 b3
1b 2a 42 52 e4 82 7c 08 09 94 2d f8 c6 50 e2 ef
fb 2c 5d 11 ed cd 24 ca 8f ec 44 13 b7 18 aa d4
ea 64 c8 cc c0 43 56 83 28 e8 af 22 5d 4b 87 34
cf d6 45 d8 ce aa 32 3d 7a b3 34 a4 36 66 59 43
82 6c aa dc 6e 4c 01 ec 26 d9 52 a1 b8 d0 d5 f6
fc 19 67 7b 60 b8 52 87 bb f9 a5 59 39 fd 18 41
95 07 7c 48 5f cd 09 a5 e1 a7 cb 40 b0 be 84 3b
dd c0 02 4d 1f b9 72 7d 34 66 e9 48 ce 8f 70 6f
9c b4 58 e6 c7 fd 88 f9 84 21 9e 09 de 42 64 62
0b 28 f6 10 47 0f bd de fa 97 2e 77 c9 93 4a 36
5c fe 9f f1 74 45 a5 f6 6e 11 49 de 58 a8 f4 f2

---- Write file:test.file end! ----
root@tulip-mozart:/# ss_demo -r test.file
---- Read file:test.file 256 Bytes data: ----
26 4e b4 c6 37 94 7d 98 35 10 c6 78 24 40 92 0e
ec 46 fc 43 b0 3f 15 17 1a a3 04 22 36 88 53 aa
6e be 63 b7 49 d0 3e 07 90 18 ae b5 5b 93 b6 43
33 bf 6a 96 4a 6b 79 5a a7 cf 30 71 50 4d af 16
eb 02 47 f3 20 b7 bf f9 76 ac 54 56 4e b8 11 b3
1b 2a 42 52 e4 82 7c 08 09 94 2d f8 c6 50 e2 ef
fb 2c 5d 11 ed cd 24 ca 8f ec 44 13 b7 18 aa d4
ea 64 c8 cc c0 43 56 83 28 e8 af 22 5d 4b 87 34
cf d6 45 d8 ce aa 32 3d 7a b3 34 a4 36 66 59 43
82 6c aa dc 6e 4c 01 ec 26 d9 52 a1 b8 d0 d5 f6
fc 19 67 7b 60 b8 52 87 bb f9 a5 59 39 fd 18 41
95 07 7c 48 5f cd 09 a5 e1 a7 cb 40 b0 be 84 3b
dd c0 02 4d 1f b9 72 7d 34 66 e9 48 ce 8f 70 6f
9c b4 58 e6 c7 fd 88 f9 84 21 9e 09 de 42 64 62
0b 28 f6 10 47 0f bd de fa 97 2e 77 c9 93 4a 36
5c fe 9f f1 74 45 a5 f6 6e 11 49 de 58 a8 f4 f2

---- Read file:test.file end! ----
root@tulip-mozart:/# ss_demo -d test.file
Delete file:test.file !
root@tulip-mozart:/# ls /data/tee/
dirf.db
```

```
root@tulip-mozart:/# ss_demo -r test.file
Failed to OP-TEE_fs_open: test.file, ret = 0xffff0008
```

7. 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。

7.1 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

7.2 安全固件版本管理

安全固件打包时会解析 target/allwinner/<platform>-common/version/version_base.mk 文件或者 target/allwinner/generic/version/version_base.mk 文件决定，前者优先级更高，并基于其中的内容生成对应版本的固件。

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。

7.3 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。

7.4 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。

7.5 关闭 jtag

将 tina/allwinner//configs/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. 参考资料

8.1 TrustZone

[1] PRD29-GENC-009492C_trustzone_security_whitepaper.pdf

8.2 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

8.3 OP-TEE

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

8.4 Dm-verity

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

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

9. Declaration

This document is the original work and copyrighted property of Allwinner Technology (“Allwinner”). Reproduction in whole or in part must obtain the written approval of Allwinner and give clear acknowledgement to the copyright owner. The information furnished by Allwinner is believed to be accurate and reliable. Allwinner reserves the right to make changes in circuit design and/or specifications at any time without notice. Allwinner does not assume any responsibility and liability for its use. Nor for any infringements of patents or other rights of the third parties which may result from its use. No license is granted by implication or otherwise under any patent or patent rights of Allwinner. This datasheet neither states nor implies warranty of any kind, including fitness for any particular application.tates nor implies warranty of any kind, including fitness for any particular application.