Rockchip Linux Secure Boot 开发指南

文件标识：RK-KF-YF-379

发布版本：V3.0.0

日期：2022-4-30

文件密级：□绝密 □秘密 □内部资料 ■公开

免责声明

本文档按“现状”提供，瑞芯微电子股份有限公司（“本公司”，下同）不对本文档的任何陈述、信息和内容的准确性、可靠性、完整性、适销性、特定目的性和非侵权性提供任何明示或暗示的声明或保证。本文档仅作为使用指导的参考。

由于产品版本升级或其他原因，本文档将可能在未经任何通知的情况下，不定期进行更新或修改。

商标声明

“Rockchip”、“瑞芯微”、“瑞芯”均为本公司的注册商标，归本公司所有。

本文档可能提及的其他所有注册商标或商标，由其各自拥有者所有。

瑞芯微电子股份有限公司

Rockchip Electronics Co., Ltd.

地址： 福建省福州市铜盘路软件园A区18号

网址： www.rock-chips.com

客户服务电话： +86-4007-700-590

客户服务传真： +86-591-83951833

客户服务邮箱： fae@rock-chips.com

前言

概述

本文档主要介绍 RK Linux 平台下，Secure Boot 的使用步骤和注意事项，方便客户在此基础上进行二次开发。安全启动功能旨在保护设备使用正确有效的固件，非签名固件或无效固件将无法启动。

产品版本

芯片名称	Kernel 检验方式	内核版本
RK3308/RK3399/RK3328/RK3326/PX30	AVB	4.4
RK3588	FIT	5.10

读者对象

本文档（本指南）主要适用于以下工程师：

技术支持工程师

软件开发工程师

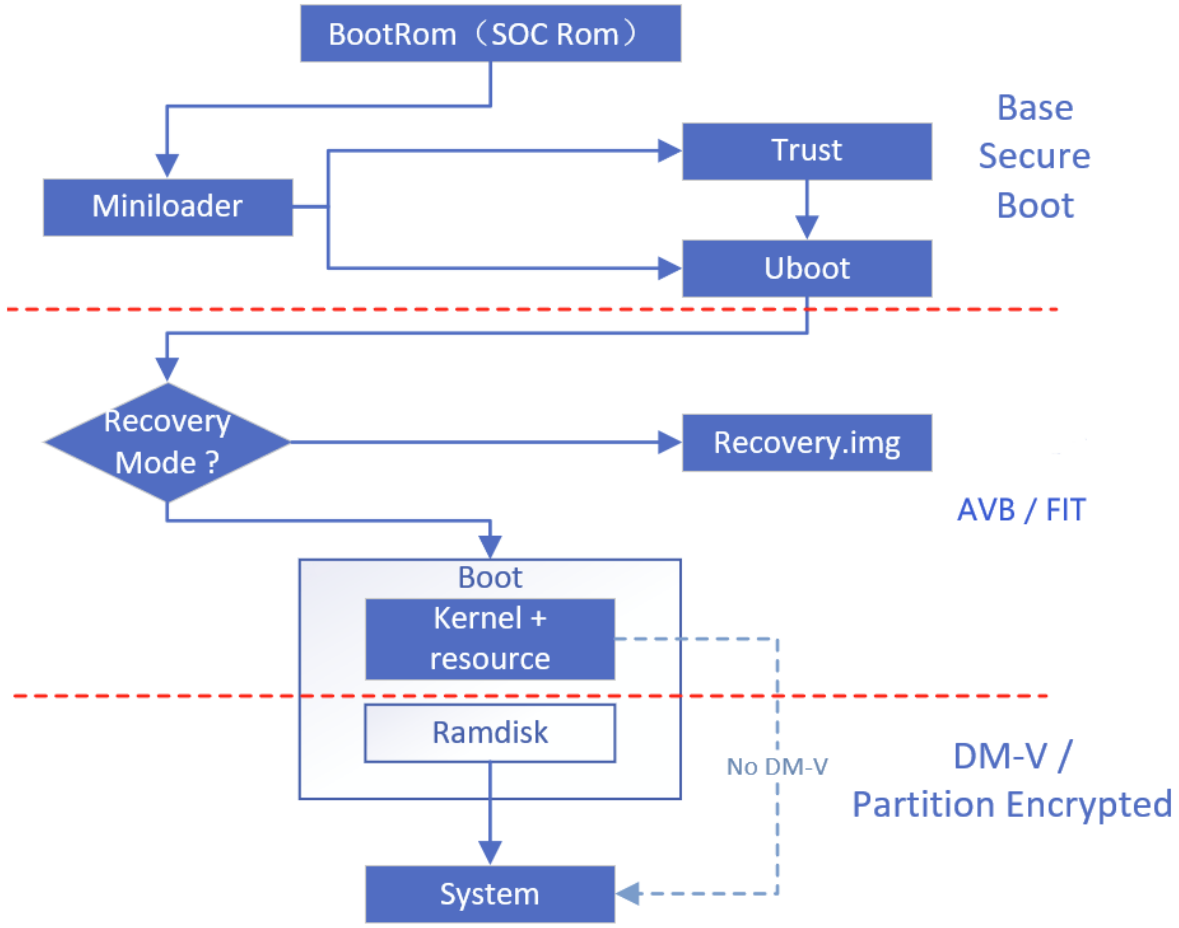
修订记录

版本号	作者	修改日期	修改说明
V1.0.0	WZZ	2018-10-31	初始版本
V1.0.1	WZZ	2018-12-17	修改笔误 vbmeta->security
V2.0.0	WZZ	2019-06-03	Sign_Tool 兼容 AVB boot.img, 修改 device-mapper 相关使用说明
V2.0.1	Ruby Zhang	2020-08-10	调整格式，更新公司名称
V3.0.0	WZZ	2022-04-30	添加RK3588 FIT支持 修改AVB Key存储描述 修改参考文献目录 更新工具链接 升级AVB描述

目录

1. Secure Boot 介绍

1.1 Secure Boot 流程



如上图所示，Linux 平台下，Secure Boot 从 BootRom 开始逐级建立了一个可靠的安全校验方案，并有序划分为三个部分，客户可以自行选择校验内容，以适配自己的需求。

Base Secure Boot: 由 BootRom 开始，逐步校验 Miniloader/Trust/Uboot 三级。

AVB / FIT: 由 Uboot 开始，校验 Boot 和 Recovery（可无）。

DM-V: 由打包在 Boot 中的 Ramdisk 工具完成校验或解密 System 分区。

Note: 以上流程和 Android 平台最大的不同是在 DM-V 阶段，Android 上借助 fs_mgr 机制，实现了 kernel 下的 DMV 校验；而这一部分，Linux 则是借助了 Ramdisk 达到校验的作用。

1.2 Secure Boot 安全存储

Linux 平台下由以下几个安全存储区域:

存储区域	说明
OTP / eFuse	位于 SOC 上，都是熔断机制的不可逆烧写。 OTP 可由 Miniloader 烧写，eFuse 只能通过 PC 工具烧写 详见 2.2 节 安全信息烧写 不同的 SOC 采用不同的介质，目前 Linux 平台主要有： eFuse： RK3399 / RK3288 OTP： RK3308 / RK3326 / PX30 / RK3328 详见1.3 节 相关资源Rockchip-Secure-Boot-Application-Note-V1.9.pdf
RPMB	位于 eMMC 上的一块物理分区，文件系统上不可见， 需要 SOC 签权访问（即只能由 TEE 访问）， 一般认为是安全区域。
Security Partition	位于存储介质上的逻辑分区，是为弥补 Flash 介质上无 RPMB 而加入的临时分区。分区内容加密存放， 无法挂载，但可能被强制擦除。同样只能由 TEE 访问 （强制擦除，TEE 访问报错，Secure Boot 无法正常启动）。

Note: 由于 OTP（eFuse）主要由 Rockchip 内部使用，客户安全信息请优先考虑 RPMB/Security 等其他区域。如有硬性需求，请向业务申请对应资料。

各阶段中，安全信息及其存储位置：

安全信息	存储位置
Base Secure Boot	Public Key Hash 存在 OTP/eFuse
AVB	OTP 设备中：permanent_attributes.bin hash eFuse 设备中： permanent_attributes.bin 存在 RPMB/Security Partition permanent_attributes_cer.bin 存在 RPMB/Security Partition （permanent_attributes.bin 由 Base Secure Boot Key 校验安全）
DM-V	Root Hash 存在 Boot 的 Ramdisk 中，由 AVB 校验 Boot 内容，保证无误

1.3 相关资源

参考文档：

Rockchip_Developer_Guide_UBoot_Nextdev_CN.pdf

Rockchip-Secure-Boot-Application-Note-V1.9.pdf

Rockchip-Secure-Boot2.0.pdf

SDK/tools/linux/Linux_SecurityAVB/Readme.md

SDK/kernel/ Documentation/device-mapper/

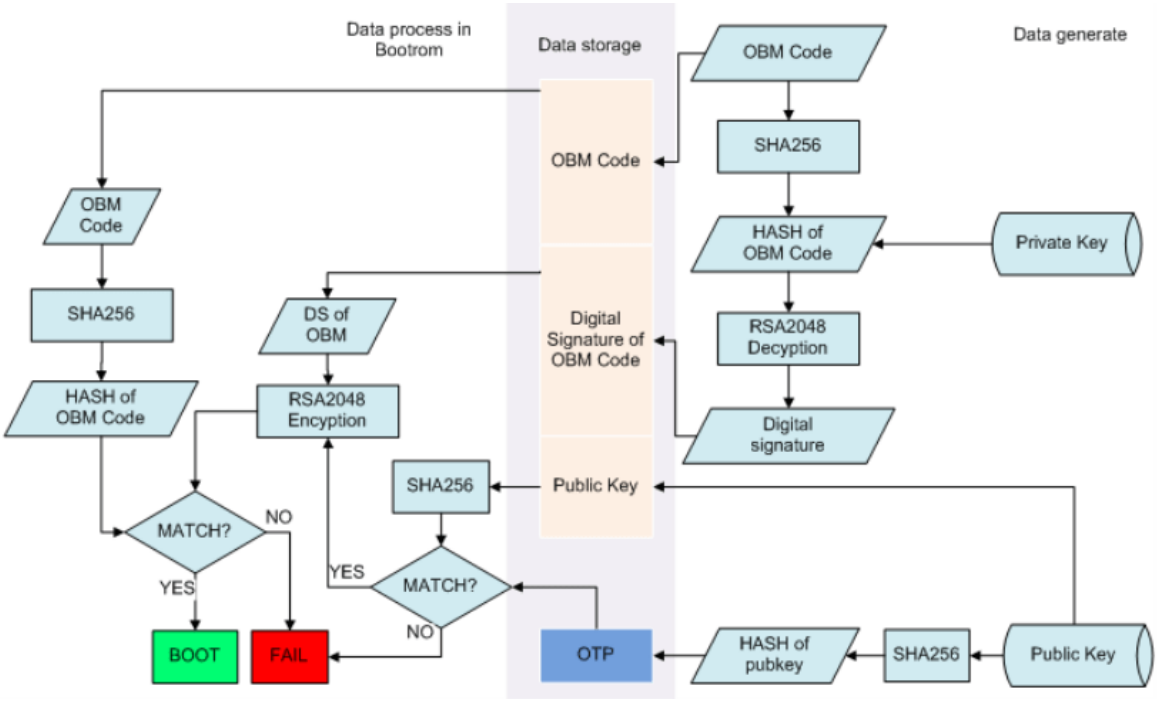
https://android.googlesource.com/platform/external/avb/+/master/README.md

https://source.android.google.cn/security/verifiedboot/dm-verity

2. Base Secure Boot

Base Secure Boot 提供基础的安全保障到 U-boot（loader/trust/uboot）

启动流程如图:



简略说，一个签名固件包括 Firmware(OBM Code) + Digital Signature + Public key

其中 Digital Signature + Public Key 都由签名工具添加。

存储上，签名固件放在 eMMC 或 Flash 上，Public Key Hash 放在芯片的 OTP(eFuse)上。

启动的时候，通过 OTP 中的 Hash 校验固件尾端的公钥，再用公钥校验数字签名的方式，达到芯片与签名代码绑定的效果。

详见 Rockchip-Secure-Boot-Application-Note-V1.9.pdf

如果芯片平台使用的是FIT方案，可以跳过Base Secure Boot操作，FIT已经集成了Base Secure Boot，不需要额外再操作。

2.1 签名工具

2.1.1 UI工具(Windows)

SDK/tools/windows/SecureBootTool_v1.94 或见企业网盘，详见 1.3节 相关资源。

1. 修改配置

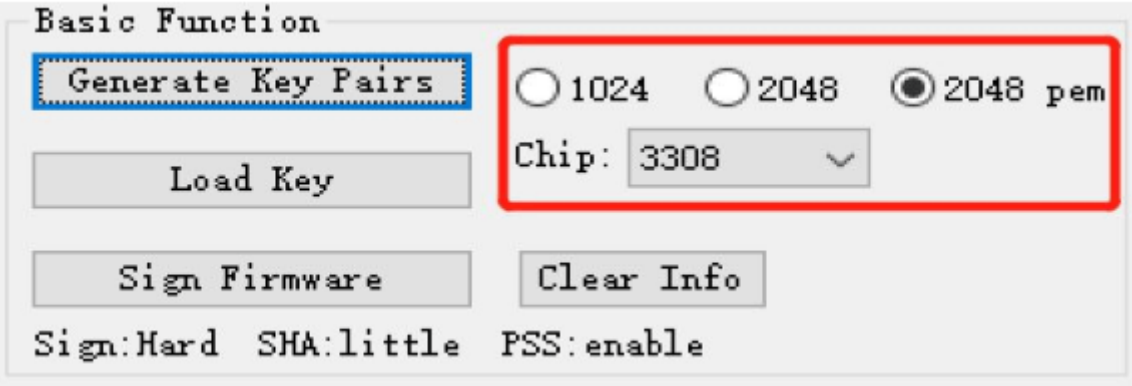
打开工具中的 setting.ini:

如果需要用到 AVB，则修改 exclude_boot_sign = True。

如果芯片使用 OTP 启用 Secure Boot 功能，将其中的 sign_flag=0x20。（bit 5: loader OTP write enabled, 已经写过 OTP 的板子上，或 eFuse 芯片，该 flag 置空。）

2. 生成公私钥

选定 Chip 和 Key 格式（pem 为通用格式），点击 Generate Key Pairs，生成 PrivteKey.pem 和 PublicKey.pem。**(密钥随机生成，请妥善保存这两个密钥，在安全功能启用后，如果丢失了这两个密钥，机子将无法刷机)**



3. 载入密钥

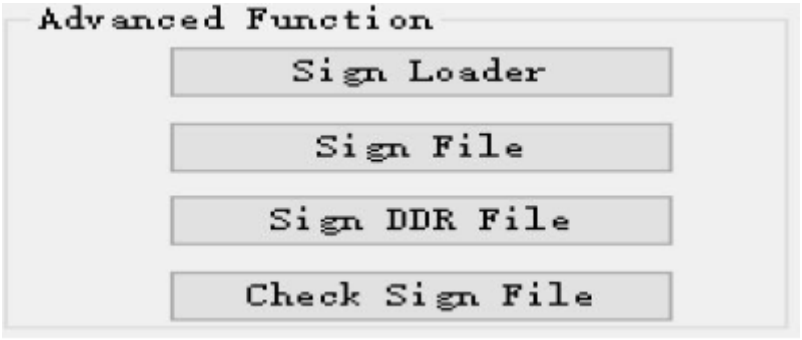
选择 Load Key，根据提示将公私钥载入。

4. 签名

签名有两种方式：只签 update.img 以及独立签名。

如果已经打包好了 update.img，那可以直接使用 Sign Firmware 对 update.img 进行签名。

独立签名，需要先按 CTRL + R + K 打开 Advanced Function



使用 Sign Loader 给 Miniloader.bin 签名，

使用 Sign File 给 trust.img 和 uboot.img 签名。

（实际上对 update.img 签名，也是将 update.img 解包，再对各个分立固件签名后，总体再打包，然后针对整体再签一次名）

2.1.2 命令行工具

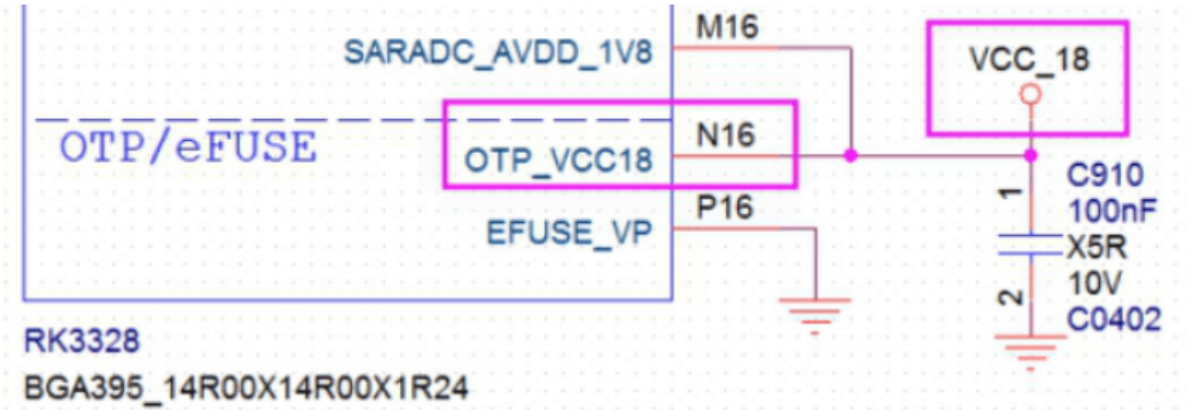
```
# step1: 产生rsa公私钥 （如果已经有了key，跳过这一步）
 ./rk_sign_tool kk --out .
# step2: 加载公私钥,只需进行一次，路径自动保存到setting.ini
./rk_sign_tool lk --key privateKey.pem --pubkey publicKey.pem
# step3: 选择芯片来决定签名方案
./rk_sign_tool cc --chip 3326
# stpe4: 打开setting.ini，将 sign_flag = 0x20。如果平台使用OTP存储安全信息，将 sign_flag = 0x20，使能RKloader OTP 写功能，空板必须开启；否则该项清空。如果需要用到AVB，则修改 exclude_boot_sign = True。
# stpe5: 整体签名，独立签名跳过该步骤。
./rk_sign_tool sf --firmware update.img
```

```
# stpe6: 签名loader，整体签名，跳过6-8步骤。
./rk_sign_tool sl --loader  rk3326loader.bin
# stpe7: 签名uboot，v1.3之前的版本，RK3326/RK3308需要带--pss；否则不带
./rk_sign_tool si --img uboot.img
# stpe8: 签名trust，v1.3之前的版本，RK3326/RK3308需要带--pss；否则不带
./rk_sign_tool si --img trust.img
```

2.2 安全信息烧写

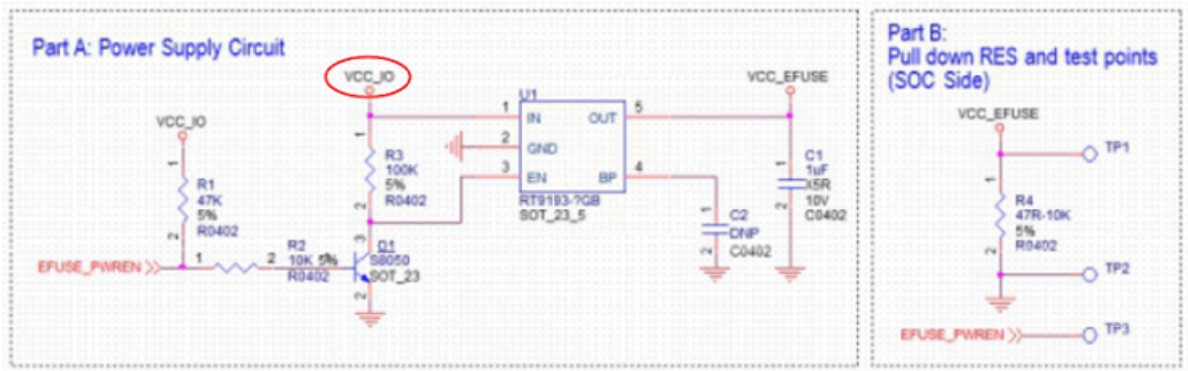
2.2.1 OTP

如果芯片使用 OTP 启用 Secure Boot 功能，保证芯片的 OTP 引脚在 Loader 阶段有供电。直接通过 AndroidTool(Windows) / upgrade_tool(Linux)把固件下载下去，第一次重启，Loader 会负责将 Key 的 Hash 写入 OTP，激活 Secure Boot。再次重启，固件就处于保护中了。



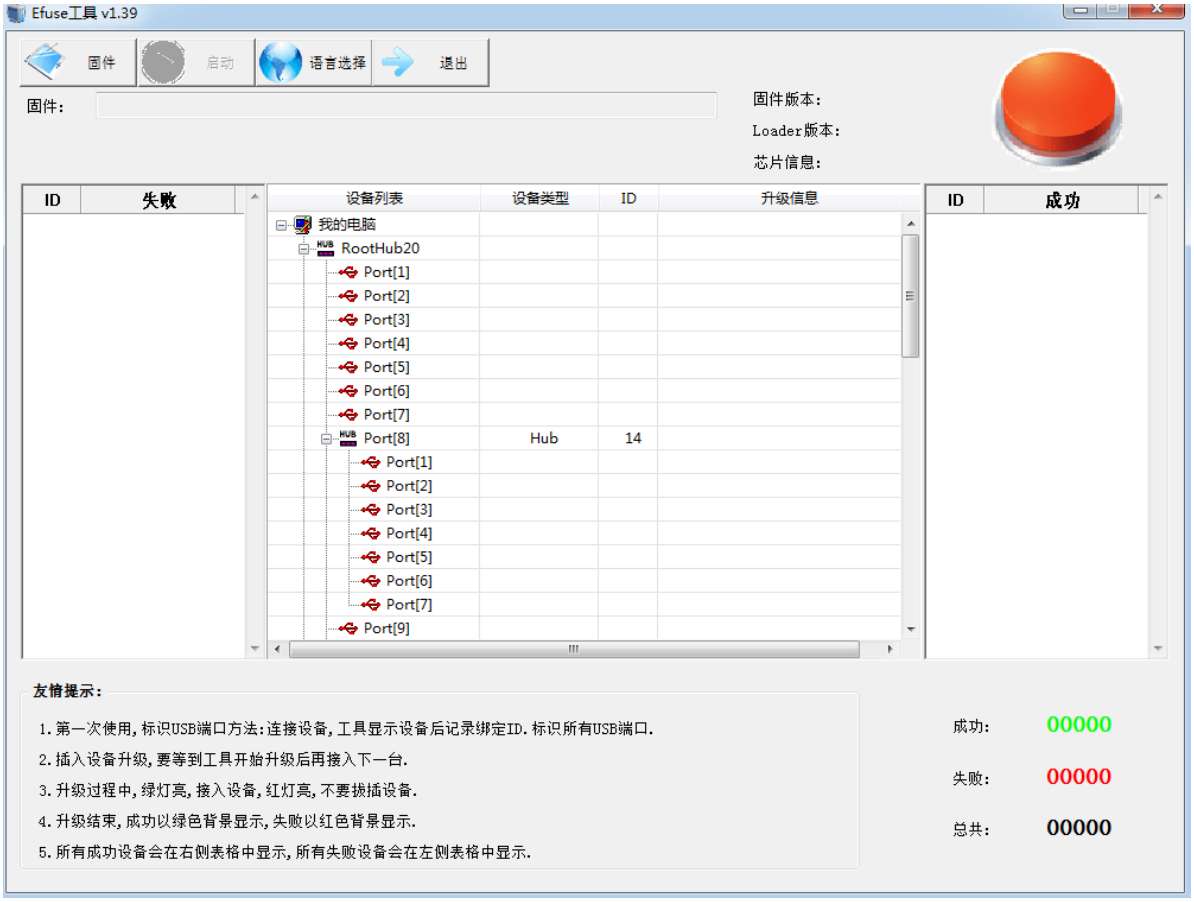
2.2.2 eFuse

如果芯片使用 eFuse 启用 Secure Boot 功能，请保证硬件连接没有问题，因为 eFuse 烧写时，Kernel 尚未启动，所以，请保证 VCC_IO 在 MaskRom 状态下有电才能用。



使用 tools/windows/eFusetool_vXX.zip，板子进入 MaskRom 状态。

点击"固件"，选择签名的 update.img，或者 Miniloader.bin，点击运行"启动"，开始烧写 eFuse。



eFuse 烧写成功后，再断电重启，进入 MaskRom，使用 AndroidTool，将其他签名固件下载到板子上。

2.3 验证

安全启动生效后，有类似如下 Log 在 Loader 阶段输出。

```
SecureMode = 1
Secure read PBA: 0x4
SecureInit ret = 0, SecureMode = 1
```

3. Kernel检验方案

3.1 AVB

AVB 需要 U-boot 配合使用，Linux 上 AVB 用来保证 uboot.img 下一级的完整性（包括 boot.img 和 recovery.img）

对应的工具在 tools/linux/Linux_SecurityAVB

具体使用，参考 tools/linux/Linux_SecurityAVB/Readme.md

（如果说明冲突，以 Linux_SecurityAVB/Readme.md 为准）

3.1.1 注意事项

关于 Device Lock & Unlock：

当设备处于 Unlock 状态，程序还是会校验整个 boot.img，如果固件有错误，程序会报具体是什么错误，**但设备正常启动**。而如果设备处于 Lock 状态，程序会校验整个 boot.img，如果固件有误，则不会启动下一级固件。所以调试阶段设置 Device 处于 Unlock 状态，方便调试。

3.1.2 固件配置

3.1.2.1 Trust

进入 rkbin/RKTRUST，以 RK3308 为例，找到 RK3308TRUST.ini，修改

```
[BL32_OPTION]
SEC=0
```

改为：

```
[BL32_OPTION]
SEC=1
```

TOS 格式的trust，默认已经开启了`BL32_OPTION`，比如`RK3288TOS.ini`

3.1.2.2 U-boot

U-boot 需要以下特性支持：

```
# OPTEE support
CONFIG_OPTEE_CLIENT=y
CONFIG_OPTEE_V1=y           #RK312x/RK322x/RK3288/RK3228H/RK3368/RK3399 与V2互斥
CONFIG_OPTEE_V2=y           #RK3308/RK3326 与V1互斥

# CRYPTO support
CONFIG_DM_CRYPTO=y          # eFuse 设备必须项
CONFIG_ROCKCHIP_CRYPTO_V1=y # eFuse 设备, 比如 RK3399/RK3288,  按芯片选择
CONFIG_ROCKCHIP_CRYPTO_V2=y # eFuse 设备, 比如 RK1808, 按芯片选择

# AVB support
CONFIG_AVB_LIBAVB=y
CONFIG_AVB_LIBAVB_AB=y
CONFIG_AVB_LIBAVB_ATX=y
CONFIG_AVB_LIBAVB_USER=y
CONFIG_RK_AVB_LIBAVB_USER=y
CONFIG_AVB_VBMETA_PUBLIC_KEY_VALIDATE=y
CONFIG_ANDROID_AVB=y
CONFIG_OPTEE_ALWAYS_USE_SECURITY_PARTITION=y        # 非eMMC设备打开，必须存在
security分区
CONFIG_ROCKCHIP_PRELOADER_PUB_KEY=y                 # 只适用于eFuse设备
CONFIG_RK_AVB_LIBAVB_ENABLE_ATH_UNLOCK=y
```

```
#fastboot support
CONFIG_FASTBOOT=y
CONFIG_FASTBOOT_BUF_ADDR=0x2000000                    # 必要时，可修改
CONFIG_FASTBOOT_BUF_SIZE=0x08000000                   # 必要时，可修改
CONFIG_FASTBOOT_FLASH=y
CONFIG_FASTBOOT_FLASH_MMC_DEV=0
```

使用`./make.sh xxxx`，生成 uboot.img, trust.img 和 loader.bin。

3.1.2.3 Parameter

vbmeta分区和system分区是分区表中必有的项目，security分区需要按硬件情况可选添加：

- vbmeta分区是存放目标分区的签名信息的文件，大小1M
- system分区就是buildroot系统中的rootfs分区，两者本质上是一样的，只是AVB中，需要兼容不同系统，所以统一叫做system。如果你是buildroot系统，需要把rootfs分区重命名为system
- security分区，一般是作为RPMB的替代品，如果在非eMMC设备上，没有RPMB，就需要加入这个分区，大小4M。

因此，一般分区表可以这样分配：

```
# AVB parameter:
0x00002000@0x00004000(uboot),0x00002000@0x00006000(trust),0x00002000@0x00008000(
misc),0x00010000@0x0000a000(boot),0x00010000@0x0001a000(recovery),0x00010000@0x0
002a000(backup),0x00020000@0x0003a000(oem),0x00300000@0x0005a000(system),0x00000
800@0x0035a000(vbmeta),0x00002000@0x0035a800(security),-
@0x0035c800(userdata:grow)
uuid:system=614e0000-0000-4b53-8000-1d28000054a9

# AVB and A/B parameter:
0x00002000@0x00004000(uboot),0x00002000@0x00006000(trust),0x00004000@0x00008000(
misc),0x00010000@0x0000c000(boot_a),0x00010000@0x0001c000(boot_b),0x00010000@0x0
002c000(backup),0x00020000@0x0003c000(oem),0x00300000@0x0005c000(system_a),0x003
00000@0x0035c000(system_b),0x00000800@0x0065c000(vbmeta_a),0x00000800@0x0065c800
(vbmeta_b),0x00002000@0x0065d000(security),-@0x0065f00(userdata:grow)
```

下载的时候，工具上的名称要同步修改，修改后，重载 parameter.txt。

3.1.3 AVB Key

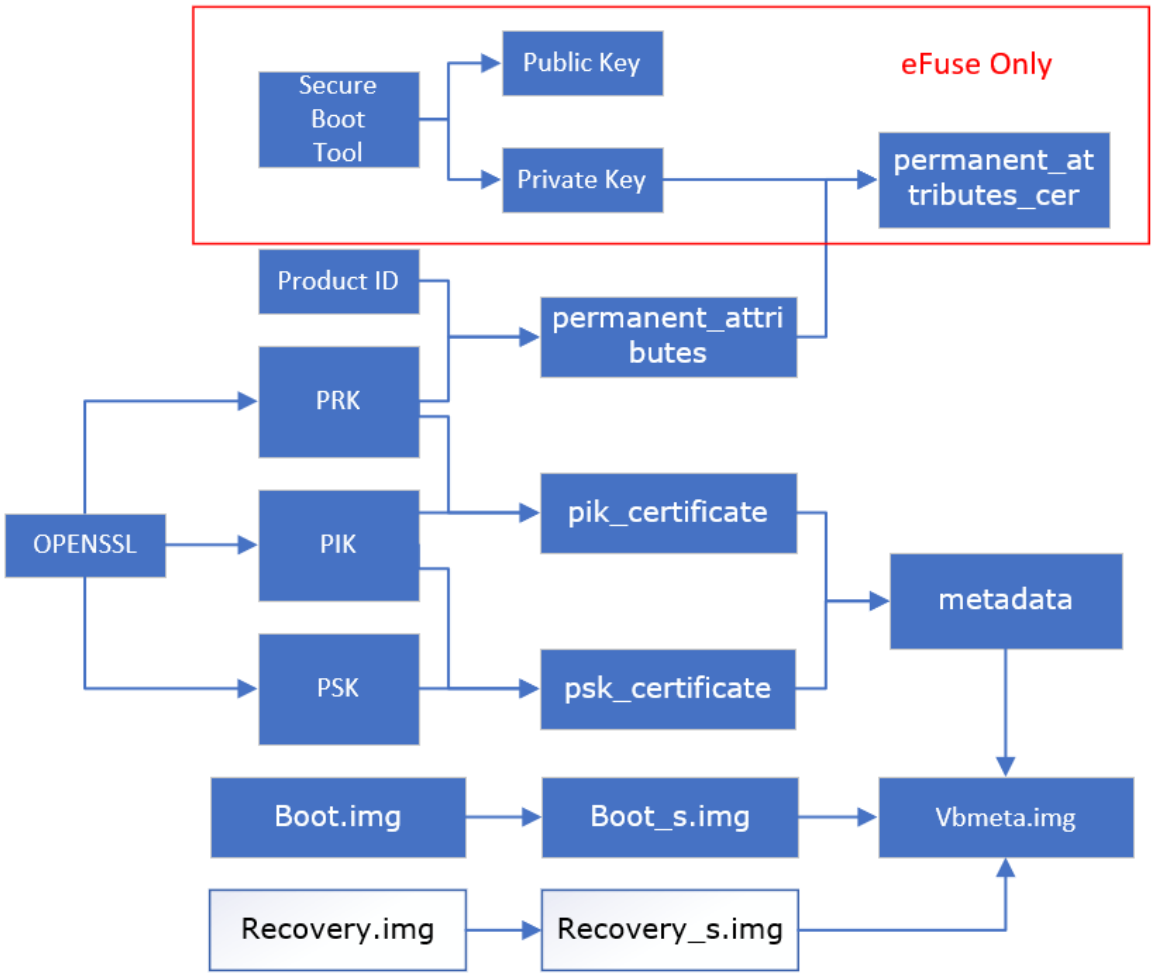
AVB 中主要信息包含以下 4 把 Key：

Product RootKey (PRK)：AVB 的 Root Key，eFuse 设备中，由 Base Secure Boot 的 Key 校验相关信息。OTP 设备中，直接读取预先存放于 OTP 中的 PRK-Hash 信息校验；

ProductIntermediate Key (PIK)：中间 Key，中介作用；

ProductSigning Key (PSK)：用于签名固件的 Key；

ProductUnlock Key (PUK)：用于解锁设备。



AVB 根据这 4 把 Key 派生出一系列的文件，如图所示，具体信息参 Google AVB 开源https://android.googlesource.com/platform/external/avb/+/master/README.md

和原版 AVB 相比，Linux 下主要截取了 AVB 最主要的固件校验功能，其中为适配 RK 平台，在 eFuse 上，额外产生了 permanent_attributes_cer.bin，这使得我们可以不必将 permanent_attributes.bin 存入 eFuse 内，直接由 Base Secure Boot Key 校验 permanent_attributes.bin 信息，达到节省 eFuse 空间的作用。

该目录下已经有一套测试的证书和 Key，如果需要新的 Key 和证书，可以按下面步骤自行生成。

```
./avb_user_tool.sh -n <Product ID> #The size of Product ID is 16 bytes.
```

如果是eFuse设备，还需要额外生成permanent_attributes_cer.bin （OTP 设备可以跳过此步骤）

```
./avb_user_tool.sh -f < /Path/to/PrivateKey.pem >
```

生成permanent_attributes_cer.bin的同时，会默认生成.setting配置文件，并在其中标记该项目使用的设备为efuse设备，**如果需要更换设备，务必记得修改此处设备类型。**

请妥善保管生成的文件，否则加锁之后将无法解锁，机子将无法刷机。

3.1.4 签名固件

开启AVB后，必须使用带签名的固件，才能正常启动。

```
./avb_user_tool.sh -s -b < /path/to/boot.img > -r < /path/to/recovery.img >
# Remove -r option if no recovery.img
```

签名固件会生成在`out`目录下。

该步骤完成了对boot和recovery的签名，并生成vbmeta.bin。固件签名需要借助 2.1 签名工具。

一般有2种签名方法：

- 使用签名工具对，loader.bin / uboot.img / trust.img 单独签名，之后再使用打包工具，打包成update.img。
- 使用`out`目录下的boot.img / recovery.img / vbmeta.bin 先打包成update.img，再使用签名工具对update.img直接签名。（该步骤需要设置exclude_boot_sign=True）。

> 打包的时候注意，security分区不用放任何固件。该分区是由trust.img初始化并使用的，不需要初始数据。

3.1.5 烧写流程

Linux下使用upgrade_tool工具下载固件，Windows下使用RKDevTool下载。

可以直接使用打包后的update.img直接更新固件，也可以按分区逐个烧录。

按分区烧录，使用AVB工具后，需要对固件进行一些替换：

- boot.img 和 recovery.img 需要使用`out`中的固件替换。
- vbmeta 需要下载。
- MiniloaderAll.bin / uboot.img / trust.img 都需要使用签名过的固件。
- parameter.txt 需要使用修改过的parameter，参考3.1.2.3 Parameter。

请在工具中添加 vbmeta 分区，地址不填。然后重新加载 parameter，工具会自行更新地址。

下载之后，设备默认处于 Unlock 状态，此时固件还是会校验，但是不会阻拦系统启动，只会报错。因为此时设备中还没正常下载AVB Key，固件无法正常检验。

3.1.6 AVB Lock & Unlock

AVB 提供 Lock 和 Unlock 两种状态：

- Lock： 检验uboot.img下级固件，boot.img或recovery.img，如果检验没有通过，会打印错误，同时停止启动。
- Unlock：检验uboot.img下级固件，boot.img或recovery.img，如果检验没有通过，会打印错误，但会继续启动。

因此，如果项目还处于调试阶段，建议使用Unlock状态。

AVB 下，所有的操作，都是通过fastboot写入的。进入fastboot有以下方法：

- 如果板子上有预留fastboot按键的，可以在启动的时候，长按fastboot按键进入fastboot模式。
- 在正常命令行下，使用`reboot fastboot`。
- 在U-boot命令行中，使用`fastboot usb 0`。

如果使用Linux操作fastboot，可以使用封装好的脚本，直接下载AVB Key，将sudo密码输入给脚本，以便脚本自动操作：

```
# 保存用户密码
./avb_user_tool.sh --su_pswd < /user/password >
# 下载 AVB Keys
./avb_user_tool.sh -d
# Lock 设备
./avb_user_tool.sh -l # reboot device after finishing lock.
# Unlock 设备
./avb_user_tool.sh -u # reboot device after finishing unlock.
```

如果没用使用脚本，或者使用windows工具，可以使用一下命令，手动烧写

PUB Key 烧写：

```
sudo ./fastboot stage permanent_attributes.bin
sudo ./fastboot oem fuse at-perm-attr
#permanent_attributes.bin存放到RPMB/security分区，在OTP设备上，还会计算
permanent_attributes.bin的Hash，并烧入到OTP

# eFuse only，  skip this step if used OTP
sudo ./fastboot stage permanent_attributes_cer.bin
sudo ./fastboot oem fuse at-rsa-perm-attr
#下载permanent_attributes_cer.bin到RPMB/security，这样在eFuse设备中，可以使用Base
Secure Boot Root Key校验permanent_attributes.bin
```

Lock 流程：

```
sudo ./fastboot oem at-lock-vboot
sudo ./fastboot reboot
```

Unlock 流程：

```
sudo ./fastboot oem at-get-vboot-unlock-challenge
sudo ./fastboot get_staged raw_unlock_challenge.bin
./make_unlock.sh
sudo ./fastboot stage unlock_credential.bin
sudo ./fastboot oem at-unlock-vboot
```

最终的 Lock Log:

```
ANDROID: reboot reason: "(none)"
Could not find security partition
read_is_device_unlocked() ops returned that device is LOCKED
```

3.2 FIT

FIT（flattened image tree）是U-Boot支持的一种新固件类型的引导方案，支持任意多个image打包和校验。FIT 使用 its (image source file) 文件描述image信息，最后通过mkimage工具生成 itb (flattened image tree blob) 镜像。its文件使用 DTS 的语法规则，非常灵活，可以直接使用libfdt 库和相关工具。同时自带一套全新的安全检验方式。

FIT与AVB主要区别：

- FIT是签名编译一起完成。AVB是先编译，后签名。
- FIT OTP/eFuse中只要存放一把Key Hash。AVB要存放两把。
- FIT签名后，签名信息是带在目标固件自身上。AVB签名信息是集中存放在vbmeta.bin中。
- FIT格式的固件，没有trust.img固件，合并进uboot.img。AVB没有对trust.img做规定。

FIT方案支持以下features：

- sha256 + rsa2048 + pkcs-v2.1(pss) padding
- 固件防回滚
- 固件重签名(远程签名)
- Crypto硬件加速

目前默认只支持 sha256+rsa2048+pkcs-v2.1(pss) padding 的安全校验模式。

更多FIT细节信息，参考Rockchip_Developer_Guide_UBoot_Nextdev文档第十二章。

3.2.1 Keys 生成

FIT Keys和Base Secure Boot Keys是同一套密钥，只是他集成到编译中，实际也是调用的rk_sign_tool生成Key以及对固件签名。

U-Boot工程下执行如下三条命令可以生成签名用的RSA密钥对。通常情况下只需要生成一次，此后都用这对密钥签名和验证固件，请妥善保管。

```
# 1. 放key的目录：keys
mkdir -p keys

# 2. 使用RK的"rk_sign_tool"工具生成RSA2048的私钥privateKey.pem和publicKey.pem，分别更名存放为：keys/dev.key和keys/dev.pubkey。命令为：
../rkbin/tools/rk_sign_tool kk --bits 2048 --out .

# 3. 使用-x509和私钥生成一个自签名证书：keys/dev.crt （效果本质等同于公钥）
openssl req -batch -new -x509 -key keys/dev.key -out keys/dev.crt
```

> 如果报错用户目录下没有.rnd文件：
>
> Can't load /home4/cjh//.rnd into RNG
> 140522933268928:error:2406F079:random number generator:RAND_load_file:Cannot open file:../crypto/rand/randfile.c:88:Filename=/home4/cjh//.rnd
>
> 请先手动创建：touch ~/.rnd

`ls keys/`查看结果：

```
dev.crt  dev.key  dev.pubkey
```

> 注意：上述的"keys"、"dev.key"、"dev.crt" 、"dev.pubkey"名字都不可变。因为这些名字已经在its文件中静态定义，如果改变则会打包失败。

3.2.2 配置

U-Boot的defconfig打开如下配置：

```
# 必选。
CONFIG_FIT_SIGNATURE=y
CONFIG_SPL_FIT_SIGNATURE=y

# 可选。
CONFIG_FIT_ROLLBACK_PROTECT=y        # boot.img防回滚
CONFIG_SPL_FIT_ROLLBACK_PROTECT=y    # uboot.img防回滚
```

> 建议通过make menuconfig的方式选中配置后，再通过make savedefconfig更新原本的defconfig文件。这样可以避免因为强加defconfig配置而导致依赖关系不对，进而导致编译失败的情况。

3.2.3 U-Boot编译及固件签名

U-Boot编译的时候，会同时对boot.img / recovery.img / loader.bin / uboot.img进行签名。因此在编译签，需要确定要签名的boot.img和recovery.img的位置。

（1）基础命令（不防回滚）：

```
./make.sh rv1126 --spl-new --boot_img boot.img --recovery_img recovery.img
```

编译结果：

```
# ......
# 编译完成后，生成已签名的uboot.img,boot.img和recovery.img。
start to sign rv1126_spl_loader_v1.00.100.bin
# ......
sign loader ok.
# ......
Image(signed, version=0):  uboot.img (FIT with uboot, trust...) is ready
Image(signed, version=0):  recovery.img (FIT with kernel, fdt, resource...) is
ready
Image(signed, version=0):  boot.img (FIT with kernel, fdt, resource...) is ready
Image(signed):  rv1126_spl_loader_v1.05.106.bin (with spl, ddr, usbplug) is ready
pack uboot.img okay! Input: /home4/cjh/rkbin/RKTRUST/RV1126TOS.ini

Platform RV1126 is build OK, with new .config(make rv1126-secure_defconfig)
```

（2）扩展命令1：

如果开启防回滚，必须对上述（1）追加rollback参数。例如：

```
// 指定 uboot.img,boot.img和recovery.img的最小版本号分别为10、12、12.
./make.sh rv1126 --spl-new --boot_img boot.img --recovery_img recovery.img --
rollback-index-uboot 10 --rollback-index-boot 12 --rollback-index-recovery 12
```

编译结果：

```
......

// 编译完成后，生成已签名的uboot.img,boot.img和recovery.img，且包含防回滚版本号。
start to sign rv1126_spl_loader_v1.00.100.bin
......
sign loader ok.
......
Image(signed, version=0, rollback-index=10):  uboot.img (FIT with uboot, trust)
is ready
Image(signed, version=0, rollback-index=12):  recovery.img (FIT with kernel, fdt,
resource...) is ready
Image(signed, version=0, rollback-index=12):  boot.img (FIT with kernel, fdt,
resource...) is ready
Image(signed):  rv1126_spl_loader_v1.00.100.bin (with spl, ddr, usbplug) is ready
```

（3）扩展命令2：

如果要把公钥hash烧写到OTP/eFUSE，必须对上述（1）或（2）追加参数`--burn-key-hash`。例如：

```
# 指定uboot.img,boot.img和recovery.img的最小版本号分别为10、12、12.
# 要求SPL阶段把公钥hash烧写到OTP/eFUSE中。
./make.sh rv1126 --spl-new --boot_img boot.img --recovery_img recovery.img --
rollback-index-uboot 10 --rollback-index-boot 12 --rollback-index-recovery 12 --
burn-key-hash
```

> 开启回滚后，更新的版本号不得低于前一版本（可以相等），否正系统会不让启动

编译结果：

```
# ......
# 使能 burn-key-hash
### spl/u-boot-spl.dtb: burn-key-hash=1

# 编译完成后，生成已签名的uboot.img,boot.img和recovery.img，且包含防回滚版本号。
start to sign rv1126_spl_loader_v1.00.100.bin
# ......
sign loader ok.
# ......
Image(signed, version=0, rollback-index=10):  uboot.img (FIT with uboot, trust)
is ready
Image(signed, version=0, rollback-index=12):  recovery.img (FIT with kernel, fdt,
resource...) is ready
Image(signed, version=0, rollback-index=12):  boot.img (FIT with kernel, fdt,
resource...) is ready
Image(signed):  rv1126_spl_loader_v1.00.100.bin (with spl, ddr, usbplug) is ready
```

> 上电开机时能看到SPL打印：RSA: Write key hash successfully。

（4）注意事项：

- `--boot_img`：可选。指定待签名的boot.img。
- `--recovery_img`：可选。指定待签名的recovery.img。
- `--rollback-index-uboot`、`--rollback-index-boot`、`--rollback-index-recovery`：可选。指定防回滚版本号。

- `--spl-new`：如果编译命令不带此参数，则默认使用rkbin中的spl文件打包生成loader；否则使用当前编译的spl文件打包loader。

因为u-boot-spl.dtb中需要被打包进RSA公钥（来自于用户），所以RK发布的SDK不会在rkbin仓库提交支持安全启动的spl文件。因此，用户编译时要指定该参数。但是用户也可以把自己的spl版本提交到rkbin工程，此后编译固件时就可以不再指定此参数，每次都使用这个稳定版的spl文件。
编译后会生成三个固件：loader、uboot.img、boot.img，只要RSA Key 没有更换，就允许单独更新其中的任意固件。

3.2.4 启动Log

```
BW=32 Col=10 Bk=8 CS0 Row=15 CS=1 Die BW=16 Size=1024MB
out
U-Boot SPL board init
U-Boot SPL 2017.09-gacb99c5-200408-dirty #cjh (Apr 09 2020 - 20:51:21)
unrecognized JEDEC id bytes: 00, 00, 00

Trying to boot from MMC1
// SPL完成签名校验
sha256,rsa2048:dev+
// 防回滚检测：当前uboot.img固件版本号是10，本机的最小版本号是9
rollback index: 10 >= 9, OK
// SPL完成各子镜像的hash校验
### Checking optee ... sha256+ OK
### Checking uboot ... sha256+ OK
### Checking fdt ... sha256+ OK

Jumping to U-Boot via OP-TEE
I/TC:
E/TC:0 0 plat_rockchip_pmu_init:2003 0
E/TC:0 0 plat_rockchip_pmu_init:2006 cpu off
E/TC:0 0 plat_rockchip_pmusram_prepare:1945 pmu sram prepare 0x14b10000 0x8400880
0x1c
E/TC:0 0 plat_rockchip_pmu_init:2020 pmu sram prepare
E/TC:0 0 plat_rockchip_pmu_init:2056 remap
I/TC: OP-TEE version: 3.6.0-233-g35ecf936 #1 Tue Mar 31 08:46:13 UTC 2020 arm
I/TC: Next entry point address: 0x00400000
I/TC: Initialized


U-Boot 2017.09-gacb99c5-200408-dirty #cjh (Apr 09 2020 - 20:51:21 +0800)

Model: Rockchip RV1126 Evaluation Board
PreSerial: 2
DRAM:  1023.5 MiB
Sysmem: init
Relocation Offset: 00000000, fdt: 3df404e0
Using default environment

dwmmc@ffc50000: 0
Bootdev(atags): mmc 0
MMC0: HS200, 200Mhz
PartType: EFI
```

```
boot mode: normal
conf: sha256,rsa2048:dev+
resource: sha256+
DTB: rk-kernel.dtb
FIT: signed, conf required
HASH(c): OK

I2c0 speed: 400000Hz
PMIC:  RK8090 (on=0x10, off=0x00)
vdd_logic 800000 uV
vdd_arm 800000 uV
vdd_npu init 800000 uV
vdd_vepu init 800000 uV
......

Hit key to stop autoboot('CTRL+C'):  0
### Booting FIT Image at 0x3d8122c0 with size 0x0052b200
Fdt Ramdisk skip relocation
### Loading kernel from FIT Image at 3d8122c0 ...
   Using 'conf' configuration
   // uboot完成签名校验
   Verifying Hash Integrity ... sha256,rsa2048:dev+ OK
   // 防回滚检测: 当前boot.img固件版本号是22, 本机的最小版本号是21
   Verifying Rollback-index ... 22 >= 21, OK
   Trying 'kernel' kernel subimage
     Description:  Kernel for arm
     Type:         Kernel Image
     Compression:  uncompressed
     Data Start:   0x3d8234c0
     Data Size:    5349248 Bytes = 5.1 MiB
     Architecture: ARM
     OS:           Linux
     Load Address: 0x02008000
     Entry Point:  0x02008000
     Hash algo:    sha256
     Hash value:
64b4a0333f7862967be052a67ee3858884fcefebf4565db5c3828a941a15f34a
   Verifying Hash Integrity ... sha256+ OK  // 完成kernel的hash校验
### Loading ramdisk from FIT Image at 3d8122c0 ...
   Using 'conf' configuration
   Trying 'ramdisk' ramdisk subimage
     Description:  Ramdisk for arm
     Type:         RAMDisk Image
     Compression:  uncompressed
     Data Start:   0x3dd3d4c0
     Data Size:    0 Bytes = 0 Bytes
     Architecture: ARM
     OS:           Linux
     Load Address: 0x0a200000
     Entry Point:  unavailable
     Hash algo:    sha256
     Hash value:
e3b0c44298fc1c149afbf4c8996fb92427ae41e4649b934ca495991b7852b855
   Verifying Hash Integrity ... sha256+ OK  // 完成ramdisk的hash校验
   Loading ramdisk from 0x3dd3d4c0 to 0x0a200000
```

```
### Loading fdt from FIT Image at 3d8122c0 ...
   Using 'conf' configuration
   Trying 'fdt' fdt subimage
     Description:  Device tree blob for arm
     Type:         Flat Device Tree
     Compression:  uncompressed
     Data Start:   0x3d812ec0
     Data Size:    66974 Bytes = 65.4 KiB
     Architecture: ARM
     Load Address: 0x08300000
     Hash algo:    sha256
     Hash value:
8fb1f170766270ed4f37cce4b082a51614cb346c223f96ddfe3526fafc5729d7
   Verifying Hash Integrity ... sha256+ OK // 完成fdt的hash校验
   Loading fdt from 0x3d812ec0 to 0x08300000
   Booting using the fdt blob at 0x8300000
   Loading Kernel Image from 0x3d8234c0 to 0x02008000 ... OK
   Using Device Tree in place at 08300000, end 0831359d
Adding bank: 0x00000000 - 0x08400000 (size: 0x08400000)
Adding bank: 0x0848a000 - 0x40000000 (size: 0x37b76000)
Total: 236.327 ms

Starting kernel ...

[    0.000000] Booting Linux on physical CPU 0xf00
[    0.000000] Linux version 4.19.111 (cjh@ubuntu) (gcc version 6.3.1 20170404
(Linaro GCC 6.3-2017.05)) #28 SMP PREEMPT Wed Mar 25 16:03:27 CST 2020
[    0.000000] CPU: ARMv7 Processor [410fc075] revision 5 (ARMv7), cr=10c5387d
```

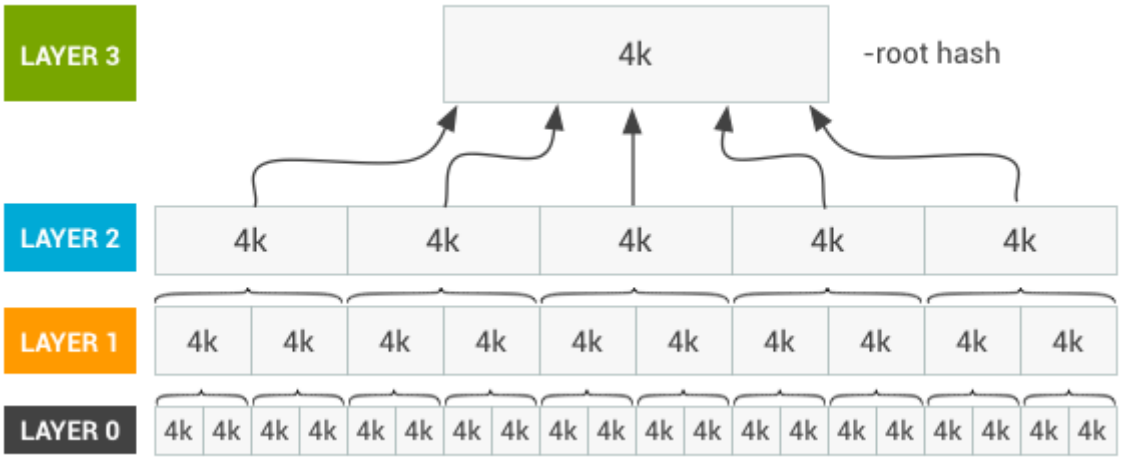
4. DM-V

使用 DM-V 原理的一个前提是 boot.img 必须是安全的，该方案会在 boot.img 中打包一个 Ramdisk，Ramdisk 安全由 AVB 保证，Ramdisk 中利用 veritysetup 工具验证挂载后级固件。

DM-V 的优点在于，校验速度快，固件越大，效果越明显。

而缺点是，DM-V 只能作用于只读文件系统，Boot 和 System 固件体积会变大。

基本原理是 Device-Mapper-Verity 技术，该机制会对校验固件进行 4K 切分，并对每个 4K 数据片进行 hash 计算，迭代多层，并生成对应 Hash-Map（30M 以内） 和 Root-Hash。创建基于 DM-V 的虚拟分区时，会对 Hash-Map 进行校验，保证 Hash-Map 无误。



分区被挂载后，有数据被访问时，会对数据所在的 4K 分区单独进行 Hash 校验，当校验出错时，返回 I/O 错误，无法使用对应位置，与文件系统损坏一样。

具体可参考 Kernel 底下 Documentation/device-mapper/

或者 https://source.android.google.cn/security/verifiedboot/dm-verity。

4.1 固件签名

DM-V 功能需要 Kernel 打开相关资源才能使用，编译 Kernel 请注意打开 CONFIG_DM_VERITY。

工具上，在SDK下 `<SDK>/tools/linux/Linux_SecurityDM_v1_01.tar.gz`。

上述压缩文件请在 Linux 环境下解压，里面包含软连接，在 Windows 环境下，会被展开成原文件大小，造成文件变大。

将文件解压，得到：

```
|── config
|── mkbootimg
|── mkdm.sh
└── ramdisk.tar.gz
```

首先配置 config，请根据实际填写对应信息

```
ROOT_DEV=    # 实际root固件在flash中的分区位置，比如/dev/mmcblk2p8
INIT=        # 实际root运行的第一个脚本，一般为/init 或 /sbin/init
ROOTFS_PATH=     #需要签名的rootfs 固件
KERNEL_PATH=         #Kernel Image位置，一般kernel/arch/arm(64)/boot/Image
RESOURCE_PATH=  #kernel resource.img 位置，一般 kernel/resource.img
```

然后运行 ./mkdm.sh -m dmv -c config (--debug)，脚本会自动将 Root-Hash 打包进 Ramdisk，并和 kernel，resource 打包成 boot.img。Hash-Map 附在 rootfs.img 后面。

output 目录下，得到 boot.img 和 rootfs_dmv.img。

将这两个固件代替原来的 boot.img 和 rootfs.img 下载到板上，即可。

4.1.1 版本升级改动

新版本下，为使Ramdisk对芯片有更好的兼容性，Ramdisk改为编译生成，不再使用固定版本的Ramdisk。

```
$COMMON_DIR/mk-ramdisk.sh ramboot.img $RK_CFG_RAMBOOT
```

- $COMMON_DIR 一般指`<SDK>/device/rockchip/common/`
- $RK_CFG_RAMBOOT 为SDK预定义的编译配置。目前仅支持RK3588。
- 因为Ramdisk需要部分系统信息，所以先编译系统，后编译Ramdisk，系统配置，mk-ramdisk.sh会从`<SDK>/device/rockchip/.BoardConfig.mk`中读取。
- 加密同样使用的是这个命令，加密和检验是靠`<SDK>/device/rockchip/.BoardConfig.mk`中的RK_SYSTEM_CHECK_METHOD变量区分的，DM-V是校验，DM-E是加密。

5. 分区加密

分区加密同样基于 device-mapper 技术，区别在于对各个分区块的处理方式不同。参考第4章DM-V。

优点，安全性高，文件系统自由，可读可写。

缺点，加密分区时无法压缩；读写数据都必须通过加解密计算，一定程度上影响读写效率。

5.1 rootfs加密

同 DM-V 相同，分区加密同样需要 Kernel 打开相关资源才能使用:

```
CONFIG_BLK_DEV_DM
CONFIG_DM_CRYPT
CONFIG_BLK_DEV_CRYPTOLOOP
```

工具上，和 DM-V 共用一套工具（Linux_SecurityDM_v1_01.tar.gz）

需要在 config 文件中配置以下内容:

```
ROOT_DEV=           # 实际root固件在flash中的分区位置，比如/dev/mmcblk2p8
INIT=               # 实际root运行的第一个脚本，一般为/init 或 /sbin/init
KERNEL_PATH=        # Kernel Image位置，一般kernel/arch/arm(64)/boot/Image
RESOURCE_PATH=      # Kernel resource.img 位置，一般 kernel/resource.img
inputimg=           # 需要加密的固件
cipher=             # 默认使用aes-cbc-plain
key=                # 请注意格式大小，使用与cipher匹配的key
```

使用`./mkdm.sh -m fde-s -c config`

会在 output 目录下，得到 boot.img 和 encrypted.img。

将这两个固件代替原来的 boot.img 和 rootfs.img 下载到板上，即可。

该系统加密方案，仅为一个演示代码，密钥是直接存储于`init`脚本中，因为涉及客户密钥，客户需要自行对密钥做处理。如果没有密钥处理方案，可以参考Keybox

5.1.1 版本升级改动

同4.1.1 版本升级改动，其中`RK_SYSTEM_CHECK_METHOD=DM-E`

5.2 非系统固件加密

固件加解密有很多开源工具可选，这里使用 dmsetup（与 5.1 工具一致），使用该工具需要打开以下配置：

```
# Kernel:
CONFIG_BLK_DEV_DM
CONFIG_DM_CRYPT
CONFIG_BLK_DEV_CRYPTOLOOP

# Buildroot:
BR2_PACKAGE_LUKSMETA
```

工具上，和 DM-V 共用一套工具（Linux_SecurityDM_v1_01.tar.gz）
需要在 config 文件中配置以下内容：

```
inputimg=   # 需要加密的固件，或者使用inputfile加密文件夹
cipher=     # 默认使用aes-cbc-plain
key=        # 请注意格式大小，使用与cipher匹配的key
```

使用`./mkdm.sh -m fde -c config`

会在 output 目录下，得到 encrypted.img 和 encrypted_info.

encrypted.img 即为加密过的文件，可将该文件用 dmsetup 虚拟成分区设备，挂载使用。其中 encrypted_info 为加密信息，如：

```
# dmsetup create encfs-4284779680572201071 --table "0 550912 crypt aes-cbc-plain
000102030405060708090a0b0c0d0e0f101112131415161718191a1b1c1d1e1f 0
TARGET_PARTITION 0 1 allow_discards"
sectors=550912
cipher=aes-cbc-plain
key=000102030405060708090a0b0c0d0e0f101112131415161718191a1b1c1d1e1f
```

挂载加密文件到/mnt 方法：

```
source encrypted_info
loopdevice=`losetup -f`
losetup ${loopdevice} encrypted.img
dmsetup create encrypt-file --table "0 $sectors crypt $cipher $key 0 $loopdevice
0 1 allow_discards"
mount /dev/mapper/encrypt-file /mnt
```

卸载方法：

```
umount /mnt
dmsetup remove encrypt-file
losetup -d ${loopdevice}
```

6. Keybox

在使用分区或系统加密的时候，需要一个密钥存储的位置，通用做法是通过OPTEE，将密钥存储在RPMB或者Security分区。由于这部分代码设计是和厂家私钥直接接触，原则上是需要厂家自行设计。考虑到部分客户对OPTEE不了解，SDK中添加了一个Keybox的实例代码，并且在7. Security Demo中进行了实际使用，供客户参考。**（推荐该部分自己重构代码）**

OPTEE相关代码，位于`<SDK>/external/security/`下，包括bin和rk_tee_user两个仓库：

- bin存放预编译的部分非公开代码和头文件，是OPTEE运行的基础库。
- rk_tee_user是编译用户自己的CA/TA的工程，里面自带部分demo test工程。

OPTEE如何使用，参考**Rockchip_Developer_Guide_TEE_SDK_CN.pdf**文档，里面有详细的介绍，这里就不在赘述，这个章节主要介绍OPTEE在buildroot中如何使用，其他系统，也可以参考buildroot中的编译规则进行编译。

rk_tee_user下预留了一个编译子项目`extra_app`，如果这个编译目录存在，则会进行编译`extra_app`，否则跳过该项编译。因此，buildroot在编译rk_tee_user的时候，会将Keybox的代码添加到rk_tee_user中，再进行编译，得到keybox_app和ta程序。

buildroot相关代码，位于`<SDK>/buildroot/package/rockchip/tee-user-app`

```
# tree buildroot/package/rockchip/tee-user-app/
buildroot/package/rockchip/tee-user-app/
├── Config.in
├── extra_app
│   ├── host
│   │   ├── main.c
│   │   └── Makefile
│   └── ta
│       ├── include
│       │   ├── ta_keybox.h
│       │   └── user_ta_header_defines.h
│       ├── keybox.c
│       ├── Makefile
│       └── sub.mk
└── tee-user-app.mk
```

> keybox_app可以在任何情况下，进行写Key操作，但是只能在Ramdisk下，且PID=1的程序中读Key。由于Ramdisk是打包在boot.img中，因此，只要boot.img是经过检验无误的，那么keybox也能安全读取Key。这要求客户，在使用这套代码时，一定要搭配FIT安全启动或者AVB安全启动。

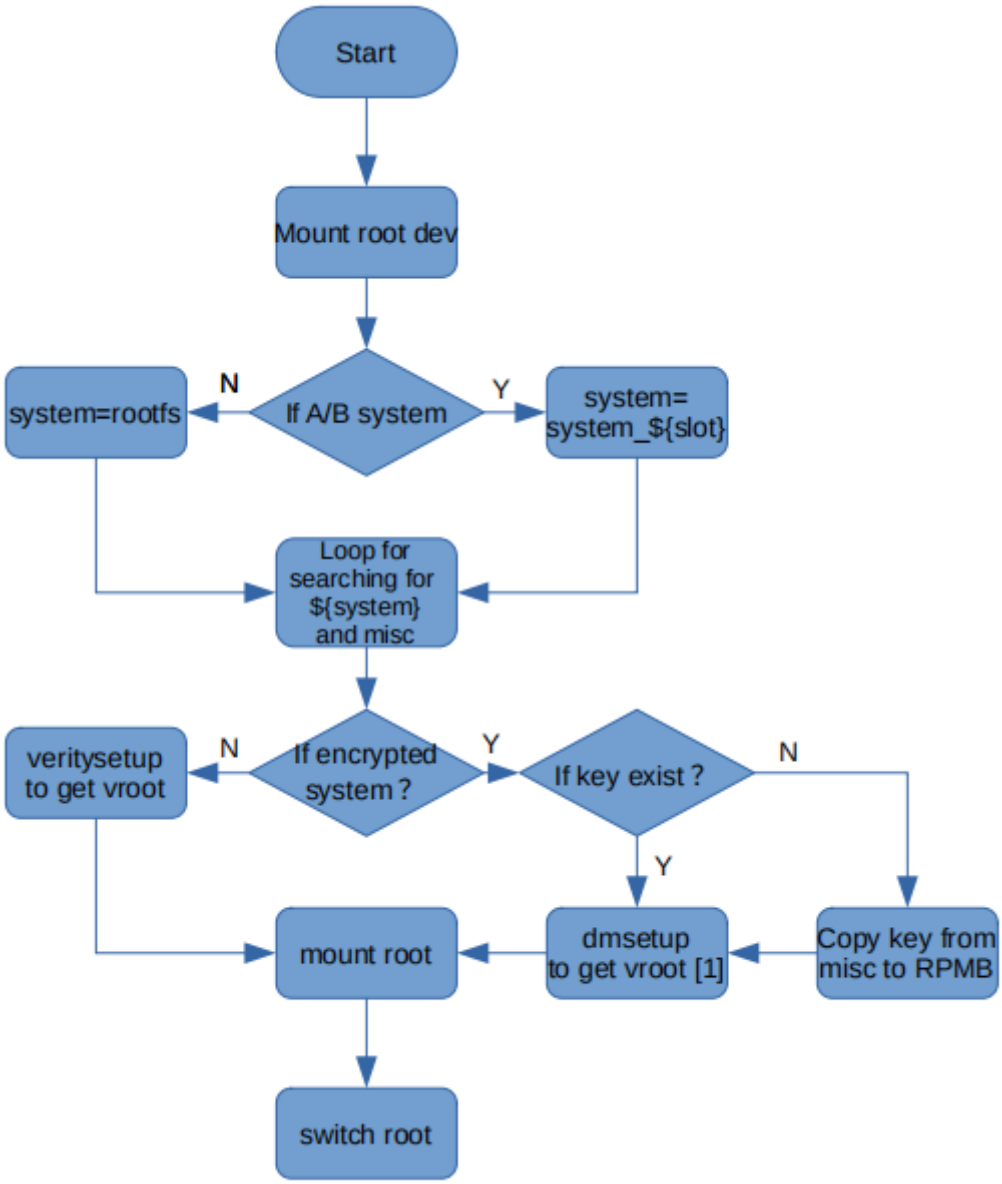
7. Security Demo

Linux SDK推出了一套基于自身编译脚本./build.sh的快速编译环境，通过几个简单的配置，实现系统加密或者系统校验的目的。

（目前该套编译环境仅适用于FIT环境，具体支持列表，参考产品版本）

7.1 Ramdisk启动流程

这套Demo使用FIT + Ramdisk，由Ramdisk引导系统安全启动。而Ramdisk中，最为关键的部分，就是他的init脚本。Ramdisk中，根据系统打包的类型，对系统进行解密或者验签，并统一成vroot设备。大致流程如下图，兼顾了A/B系统，系统校验和系统加密。



> [1] 加密系统下，会根据系统分区的实际大小对system文件系统进行扩容。

7.2 系统配置

在支持Security Demo的`<SDK>/device/rockchip/.target_product`目录下，都会存在BoardConfig-security-base.mk和BoardConfig-ab-base.mk配置

```
# cat BoardConfig-security-base.mk
```

```
export RK_LOADER_UPDATE_SPL=true
# ramboot config
export RK_CFG_RAMBOOT=rockchip_rk3588_ramboot
# Set ramboot image type
export RK_RAMBOOT_TYPE=CPIO
# Define enable RK SECUREBOOT
export RK_RAMDISK_SECURITY_BOOTUP=true
export RK_SECURITY_OTP_DEBUG=true                    # 默认开，打开的时候OTP不会真正写入。
export RK_SYSTEM_CHECK_METHOD=DM-V                   # 系统检验时DM-V，系统加密是DM-E

# cat BoardConfig-ab-base.mk
export RK_PACKAGE_FILE_AB=rk3588-package-file-ab
export RK_MISC=blank-misc.img
export RK_PARAMETER=parameter-ab.txt
export RK_CFG_RECOVERY=                              # a/b系统中，不存在recovery
```

> **以上选项请特别注意RK_SECURITY_OTP_DEBUG，默认状态都是打开的，打开时，不会真正将Key Hash写入OTP，也就是默认loader.bin是安全的。正式生产产品的时候，请关闭该选项**

需要开启的时候，需要在`<SDK>/device/rockchip/.BoardConfig.mk`默认追加BoardConfig-security-base.mk，比如

```
diff --git a/rk3588/BoardConfig-rk3588-evb1-lp4-v10.mk b/rk3588/BoardConfig-rk3588-evb1-lp4-v10.mk
index 5fa455d..753bb6c 100644
--- a/rk3588/BoardConfig-rk3588-evb1-lp4-v10.mk
+++ b/rk3588/BoardConfig-rk3588-evb1-lp4-v10.mk
@@ -59,3 +59,5 @@ export RK_DISTRO_MODULE=
 export RK_BOARD_PRE_BUILD_SCRIPT=app-build.sh
 # Define package-file
 export RK_PACKAGE_FILE=rk3588-package-file
+REALDIR=`dirname $(readlink -f ${BASH_SOURCE[0]})`
+source ${REALDIR}/BoardConfig-security-base.mk
```

> 需要A/B系统时，还需要添加BoardConfig-ab-base.mk。

以上配置结束后，可以直接使用`./build.sh`进行编译。过程中会出现报错，根据提示，排除错误即可。或者依照文档，继续进行修改配置。

7.3 详细配置

- 配置FIT Key

  ```
  ./build.sh createkeys
  ```

- 如果使用的是系统加密，将带root权限的用户密码存储在U-Boot中

  ```
  echo "Pass work for sudo" > u-boot/keys/root_passwd
  ```

  > 需要编译加密系统时，会使用的系统的loopback系统，这需要用户对编译的电脑有root的操作权限，这意味着无法该系统无法在服务器环境下编译（除非你的用户有root权限）。

- 修改Kernel配置，添加以下配置

```
CONFIG_BLK_DEV_DM=y
CONFIG_DM_CRYPT=y
CONFIG_BLK_DEV_CRYPTOLOOP=y
CONFIG_DM_VERITY=y

CONFIG_TEE=y                    # 加密系统必须选
CONFIG_OPTEE=y                  # 加密系统必须选
```

- 如果使用的是系统加密，修改Kernel dts文件，添加OPTEE节点

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

- 修改U-Boot配置，添加以下配置

```
CONFIG_FIT_SIGNATURE=y
CONFIG_SPL_FIT_SIGNATURE=y
CONFIG_ANDROID_AB=y         # 只有A/B系统可选，其他系统不要选
```

- ROOTFS需要添加以下配置

```
BR2_ROOTFS_OVERLAY="board/rockchip/common/security-system-overlay"
BR2_PACKAGE_RECOVERY=y                      # 加密系统必须
BR2_PACKAGE_RECOVERY_UPDATEENGINEBIN=y      # 加密系统必选
BR2_PACKAGE_RECOVERY_BOOTCONTROL=y          # A/B系统系统必须，依赖于前两项
```

编译无误，正确下载后，系统可以正常启动，并处于保护中。任意一个固件被非法替换，系统都无法正常启动。

7.4 调试方法

如果init无法正常跳转系统，可以打开打印，并且使用`DEBUG`函数添加一些私有打印，按照7.1 Ramdisk启动流程确定他的运行状态。

```
diff --git a/board/rockchip/common/security-ramdisk-overlay/init.in
b/board/rockchip/common/security-ramdisk-overlay/init.in
index 756d0b5375..c3267e28b1 100755
--- a/board/rockchip/common/security-ramdisk-overlay/init.in
+++ b/board/rockchip/common/security-ramdisk-overlay/init.in
@@ -16,7 +16,7 @@ BLOCK_TYPE_SUPPORTED="
 mmcblk
 flash"

-MSG_OUTPUT=/dev/null
+MSG_OUTPUT=/dev/kmsg
 DEBUG() {
        echo $1 > $MSG_OUTPUT
 }
```