Rockchip

Secureboot 使用说明

发布版本:**1.01**

日期:**2018.12**

免责声明

本文档按“现状”提供，福州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。

版权所有 © 2018 福州瑞芯微电子股份有限公司

福州瑞芯微电子股份有限公司
Fuzhou 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： +86-591-83991906
客户服务传真： +86-591-83951833
客户服务邮箱： service@rock-chips.com

前言

产品版本

芯片名称	内核版本
RK3308/RK3399/RK3326	4.4

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

日期	版本	作者	修改说明
2018-10-31	V1.00	王征增	
2018-12-17	V1.01	王征增	修改笔误 vbmeta->security

目录

1 Secureboot

1.1 概述

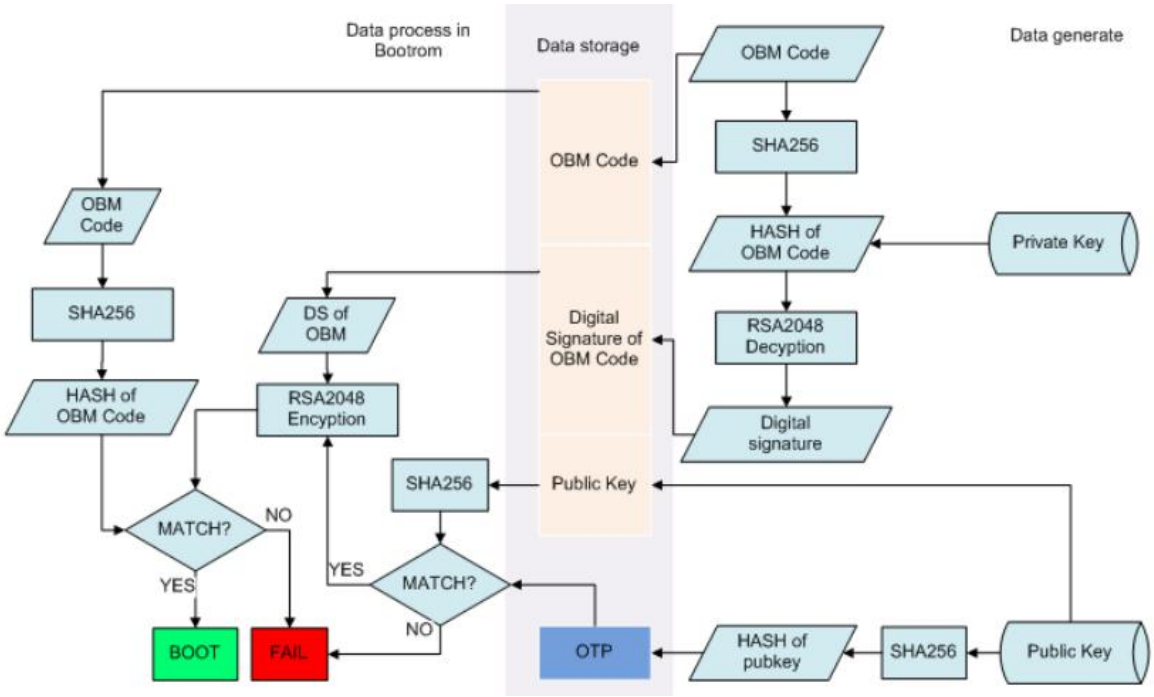
本文档主要介绍 RK linux 平台下，secureboot 的使用步骤和注意事项，方便客户在此基础上进行二次开发。安全启动功能旨在保护设备使用正确有效的固件，非签名固件或无效固件将无法启动。

相关工具： 链接：https://eyun.baidu.com/s/3qZwY9FQ 密码：OubV

1.2 Secureboot 存储区域

根据芯片不同，将使用不同区域作为 secureboot 存储区。
Efuse： rk3399/rk3288
Otp： rk3308 / rk3326 / rk3328

1.3 Loader/trust/uboot 校验



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

1.3.1 签名工具

1.3.1.1 UI 工具(windows)：tools/windows/SecureBootTool_v1.89

1. 修改配置

如果芯片使用 otp 启用 secureboot 功能，请修改工具根目录下的 config.ini 文件，将其中的 sign_flag=20。

2. 生成公私钥

选定 chip 和 key 格式（pem 为通用格式），点击 Generate Key Pairs，生成生成 PrivteKey.pem 和 PublicKey.pem。(密钥随机生成，两次生成的密钥一定不同，所以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 2 中方式：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）

1.3.1.2 命令行工具

网盘地址（见概述）下，rk_sign_tool_v1.0_win.zip/rk_sign_tool_v1.0_linux.zip

1 ./rk_sign_tool kk --out . //产生 rsa 公私钥 （如果已经有了 key，跳过这一步）

2 ./rk_sign_tool lk --key privateKey.pem --pubkey publicKey.pem //加载公私钥,只需进行一次，路径自动保存到 setting.ini

3 ./rk_sign_tool cc --chip 3326 //选择芯片来决定签名方案

4 打开 setting.ini 将 sign_flag = 0x20 //如果平台使用 otp 存储安全信息，使能 RKloader OTP 写功能，空板必须开启；否则该项清空。

5 ./rk_sign_tool sl --loader rk3326loader.bin //签名 loader

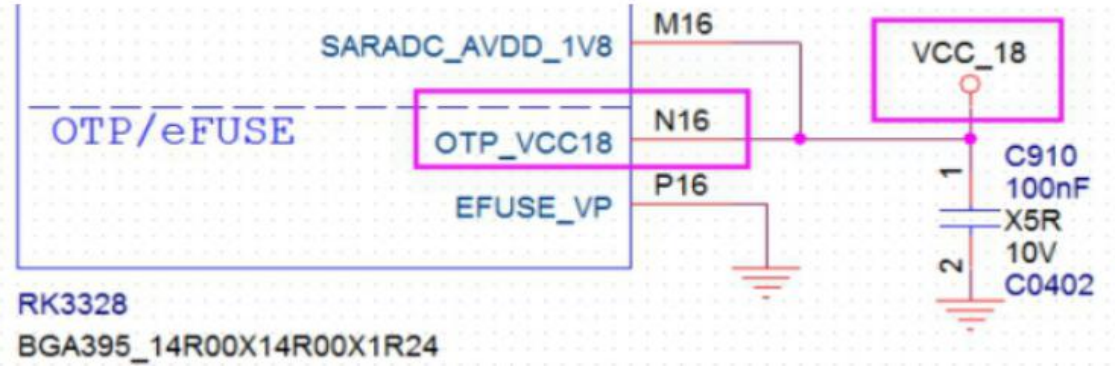
6 ./rk_sign_tool si --img uboot.img --pss //签名 uboot， rk3326/3308 需要带--pss；否则不带

7 ./rk_sign_tool si --img trust.img --pss //签名 trust，rk3326/3308 需要带--pss；否则不带

1.3.2 下载

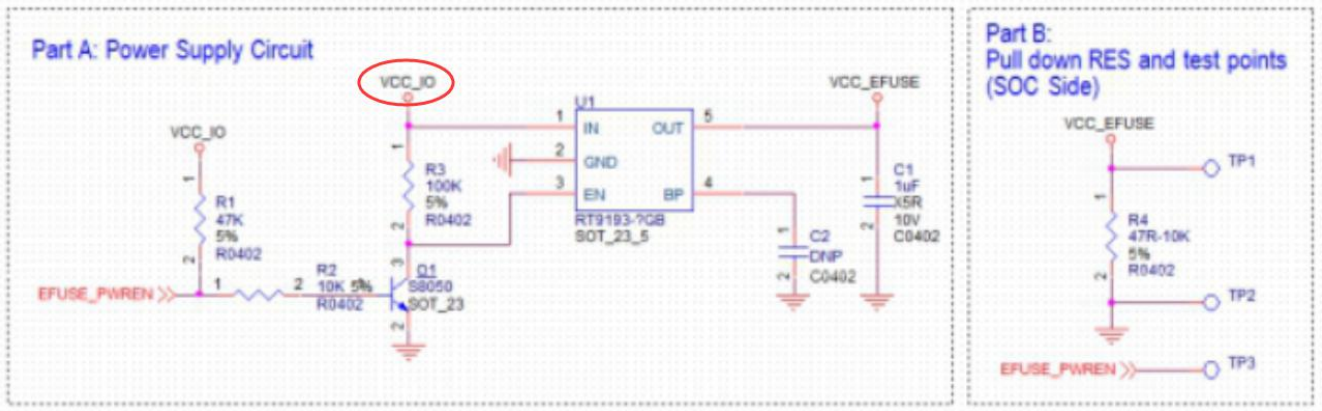
1. OPT

如果芯片使用 otp 启用 secureboot 功能，保证芯片的 OTP 引脚有供电。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 secureboot。再次重启，固件就处于保护中了。



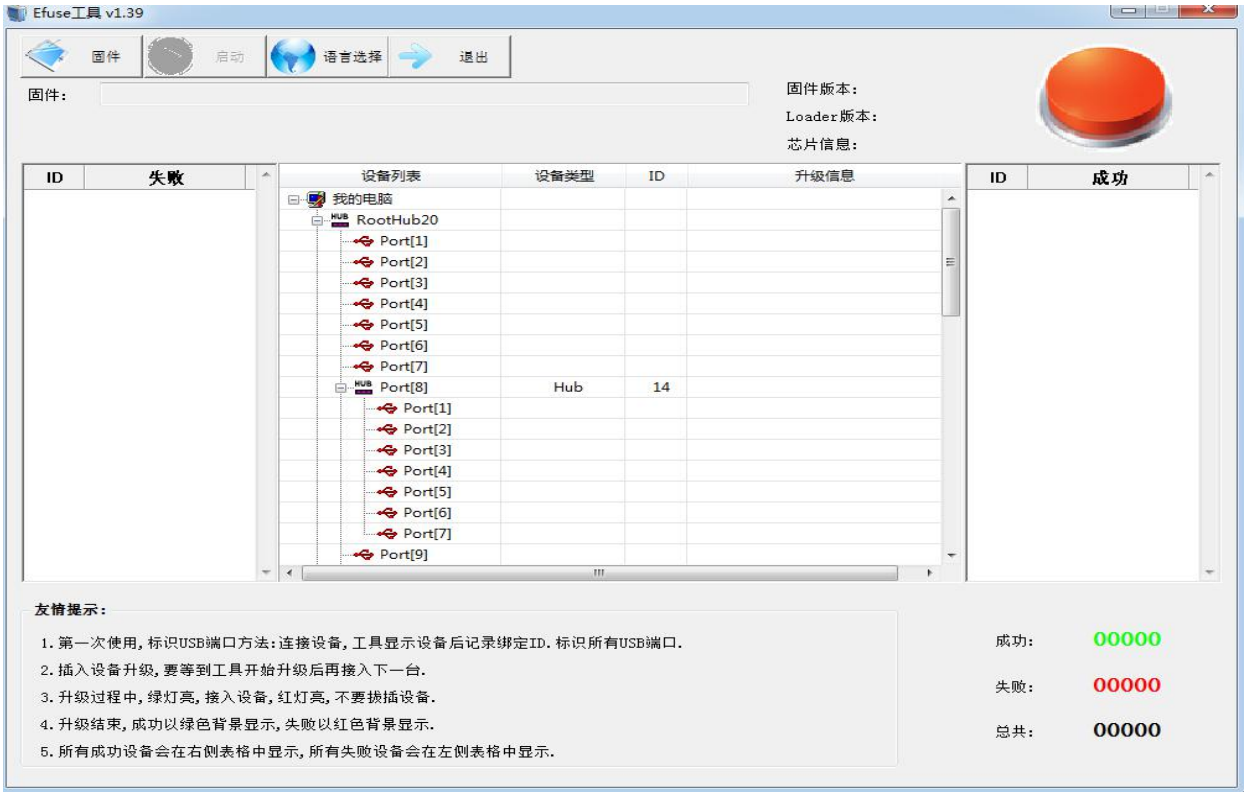
2. Efuse

如果芯片使用 efuse 启用 secureboot 功能，请保证硬件连接没有问题，因为 efuse 的烧写不在 kernel 阶段，需要保证图中 VCC_IO 默认是有电的。



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，或则 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。

1.3.3 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
```

1.4 Boot 校验（AVB）

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

对应的工具在 tools/linux/Linux_SecurityAVB.

1.4.1 修改 parameter

AVB 需要添加 vbmeta 分区，用来存放固件签名信息。内容加密存放。大小 1M，位置无关。

AVB 需要 system 分区，在 buildroot 上，即 rootfs 分区，需要将 rootfs 改名为 system，如果使用了 uuid，同时修改 uuid 分区名。

如果存储介质使用 flash，还需要另外添加 security 分区，用来存放操作信息。内容同样加密存放。大小 4M，位置无关。（emmc 无需添加该分区，emmc 操作信息存放在物理 rpmb 分区）

Note：rk3399 无论使用什么存储，统一使用 security 分区。

以下是 avb parameter 例子：

0x00002000@0x00004000(uboot),0x00002000@0x00006000(trust),0x00002000@0x00008000(misc),0x00010000@0x0000a000(boot),0x00010000@0x0001a000(recovery),0x00010000@0x0002a000(backup),0x00020000@0x0003a000(oem),0x00300000@0x0005a000(system),0x00000800@0x0035a000(vbmeta),0x00002000@0x0035a800(security),-@0x0035c800(userdata:grow)

uuid:system=614e0000-0000-4b53-8000-1d28000054a9

1.4.2 配置 u-boot

U-boot 中需要打开 avb 对应开关：

```
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_ANDROID_AVB=y
```

还有配合使用的其他开关：

OPTEE：

```
CONFIG_OPTEE_CLIENT=y
    CONFIG_OPTEE_V1=y  #rk312x/rk322x/rk3288/rk3228H/rk3368/rk3399
    CONFIG_OPTEE_V2=y  #rk3326/rk3308
    V1/V2 选择一项打开即可
CONFIG_OPTEE_ALWAYS_USE_SECURITY_PARTITION  #rpmb 无法使用是打开，默认不开
```

FASTBOOT：

```
CONFIG_FASTBOOT=y
CONFIG_FASTBOOT_BUF_ADDR=0x800800
CONFIG_FASTBOOT_BUF_SIZE=0x04000000
CONFIG_FASTBOOT_FLASH=y
CONFIG_FASTBOOT_FLASH_MMC_DEV=0
Note：以上 FASTBOOT 数值为参考值，各平台有有所不同。
```

打开公钥安全校验：

```
lib/avb/libavb_user/avb_ops_user.c 中定义
#define AVB_VBMETA_PUBLIC_KEY_VALIDATE
```

打开 rpmb 支持，在 include/configs 找到对应的板子，比如 evb_rk3399.h，在其中定义：

```
#define CONFIG_SUPPORT_EMMC_RPMB
```

RK3399：

因为 avb 加入，使得 u-boot 大小过大，需要额外裁剪掉一些内容，腾出空间，建议裁剪显示部分，和网卡部分

1.4.3 配置 RKBIN

以 rk3308 为例，

进入 rkbin/RKTRUST，找到 RK3308TRUST.ini，修改

```
[BL32_OPTION]
```

SEC=0

改为

[BL32_OPTION]

SEC=1

以上配置完成后，就能编译 u-boot,生成 loader/uboot/trust

1.4.4 配置 AVB KEY

tools/linux/Linux_SecurityAVB 下已经包含一套签名用的密钥和相关配置文件（仅供测试）

对应文件包含

```
├── atx_metadata.bin
├── atx_permanent_attributes.bin
├── atx_pik_certificate.bin
├── atx_psk_certificate.bin
├── atx_puk_certificate.bin
├── testkey_atx_pik.pem
├── testkey_atx_prk.pem
├── testkey_atx_psk.pem
└── testkey_atx_puk.pem
```

下面介绍如何自己生成这些文件。

```
openssl genpkey -algorithm RSA -pkeyopt rsa_keygen_bits:4096 -outform PEM -out testkey_atx_prk.pem

openssl genpkey -algorithm RSA -pkeyopt rsa_keygen_bits:4096 -outform PEM -out testkey_atx_psk.pem

openssl genpkey -algorithm RSA -pkeyopt rsa_keygen_bits:4096 -outform PEM -out testkey_atx_pik.pem

openssl genpkey -algorithm RSA -pkeyopt rsa_keygen_bits:4096 -outform PEM -out testkey_atx_puk.pem

touch temp.bin
python avbtool make_atx_certificate --output=atx_pik_certificate.bin --subject=temp.bin --subject_key=testkey_atx_pik.pem --subject_is_intermediate_authority --subject_key_version 42 --authority_key=testkey_atx_prk.pem

echo 123456789ABCDEF > atx_product_id.bin ##其中 atx_product_id.bin 需要自己定义，占 16 字节，可作为产品 ID 定义
python avbtool make_atx_certificate --output=atx_psk_certificate.bin --subject=atx_product_id.bin --subject_key=testkey_atx_psk.pem --subject_key_version 42 --authority_key=testkey_atx_pik.pem

python avbtool make_atx_metadata --output=atx_metadata.bin --intermediate_key_certificate=atx_pik_certificate.bin --product_key_certificate=atx_psk_certificate.bin
```

python avbtool make_atx_permanent_attributes --output=atx_permanent_attributes.bin --product_id=atx_product_id.bin --root_authority_key=testkey_atx_prk.pem

atx_unlock_credential.bin 为需要下载到设备解锁的证书，其生成过程如下：

python avbtool make_atx_certificate --output=atx_puk_certificate.bin --subject=atx_product_id.bin --subject_key=testkey_atx_puk.pem --usage=com.google.android.things.vboot.unlock --subject_key_version 42 --authority_key=testkey_atx_pik.pem

python avbtool make_atx_unlock_credential --output=atx_unlock_credential.bin --intermediate_key_certificate=atx_pik_certificate.bin --unlock_key_certificate=atx_puk_certificate.bin --challenge=atx_unlock_challenge.bin --unlock_key=testkey_atx_puk.pem

1.4.5 签名 boot/recovery

Avb 签名使用 avbtool，基本格式：

python avbtool add_hash_footer --image <IMG> --partition_size <SIZE> --partition_name <PARTITION> --key testkey_atx_psk.pem --algorithm SHA512_RSA4096

其中，partition_size 至少比原文件大 64K，且不超过 parameter 中定义的分区大小，大小必须 4K 对齐。PARTITION = boot / recovery。

签名完成后，用签名过的文件生成 vbmeta.img

基本格式：

python avbtool make_vbmeta_image --public_key_metadata atx_metadata.bin --include_descriptors_from_image <IMG> SHA256_RSA4096 --rollback_index 0 --key testkey_atx_psk.pem --output vbmeta.img

--include_descriptors_from_image <IMG> 该字段可以多次使用，即有多少个加密过的文件，就添加多少个 --include_descriptors_from_image。

例如：

python avbtool make_vbmeta_image --public_key_metadata atx_metadata.bin --include_descriptors_from_image boot.img --include_descriptors_from_image recovery.img--algorithm SHA256_RSA4096 --rollback_index 0 --key testkey_atx_psk.pem --output vbmeta.img

tools/linux/Linux_SecurityAVB 默认有个 make_vbmeta.sh 脚本，可以直接对该脚本进行修改，运行生成对应的 vbmeta.img

Note：生成的 vbmeta.img 中是包含固件的签名信息，这个时候，只要使用对应的固件，即使未签名，也是可以起来的。

1.4.6 加解锁

以上步骤成功实现后，会在 uboot 打印中出现以下字段：

read_is_device_unlocked() ops returned that device is UNLOCKED

说明 AVB 已经成功启用，但是当前设备处于 unlocked 状态，该状态下，会进行 boot/recovery 校验，但是不会阻止签名异常固件启动，只会报 hash 不匹配的 log，如：

bootfilesh of data does not match digest in descriptor.

加锁：

在 u-boot 命令行中执行 fastboot usb 0 进入 fastboot 模式，此时 u-boot 命令行无法继续输入。

或在系统中执行 reboot fastboot，系统自动进 u-boot，并停在 fastboot 命令下。

之后 PC 在 tools/linux/Linux_SecurityAVB 中执行：

```
sudo ./fastboot stage atx_permanent_attributes.bin
sudo ./fastboot oem fuse at-perm-attr
sudo ./fastboot oem at-lock-vboot
sudo ./fastboot reboot
```

这样 avb 就进入锁定状态，uboot 打印

```
read_is_device_unlocked() ops returned that device is LOCKED
```

解锁：

```
sudo ./fastboot oem at-get-vboot-unlock-challenge
sudo ./fastboot get_staged raw_atx_unlock_challenge.bin
./make_unlock.sh
sudo ./fastboot stage atx_unlock_credential.bin
sudo ./fastboot oem at-unlock-vboot
sudo ./fastboot reboot
```

1.5 Rootfs 校验（dm-v）

需要网盘（见概述）文件 ramdisk_dmv.zip

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。

使用 dmv 前注意：dmv 只能校验只读文件系统

kernel 中需要打开 dmv 支持，使用 menuconfig 配置 CONFIG_DM_VERITY

先将压缩文件 ramdisk_dmv.zip 解压到 external 下面，然后运行

```
./create_input.sh
```

创建 input 文件夹，由于各芯片平台不同，如果该脚本运行出错，请自行提取相关文件到 input 文件夹。

Kernel：

ARM64： kernel/arch/arm64/boot/Image

ARM：kernel/arch/arm/boot/Image

根据平台选择一个

Resource：kernel/resource.img

Rootfs：

buildroot/output/rockchip_rkxxxx/image/rootfs.xx

在 input 中创建 config 文件（根据上述提取文件）：

```
echo "KERNEL=Image" > ${PWD}/input/config
echo "RESOURCE=resource.img" >> ${PWD}/input/config
echo "ROOTFS=rootfs.xx" >> ${PWD}/input/config
```

生成 dm-v 需要的 root_hash 和 hash_mapper。

```
./mkdmv.sh /dev/sda8 /dev/sda9 input
```

其中/dev/sda8 /dev/sda9 请输入 PC 中未存在的分区（工作时，即为虚拟分区），脚本执行结束，

自动删除。

生成带校验信息的 boot.img

/mkfirmware_dmv.sh input/ ramdisk/ /dev/mmcblk1p10 /dev/mmcblk1p8

其中/dev/mmcblk1p10 为 arm 机器中尚未存在的分区，改分区工作中将扩展为虚拟分区，存放 hash_mapper.

/dev/mmcblk1p8 为 rootfs 挂载点。

之后将 input 中生成的 boot.img 和 rootfs 下载到设备中即可。

此时，rootfs 校验信息已经加入到了 boot 当中，更换 rootfs，设备将无法进入 rootfs。

1.6 全盘加密

Linux 上已经有开源的全盘加密工具，该工具已经加到 buildroot 当中。

Config 需要打开：

Buildroot：BR2_PACKAGE_LUKSMETA

Kernel: CONFIG_DM_CRYPT

加解密磁盘可以在任意设备上创建使用，只要设备有 cryptsetup 工具。

创建加密 img：

1. dd if=/dev/zero of=/dev/virtual_dev bs=1M count=100 #创建一个虚拟 100M 的磁盘，如果要对已有磁盘加密，跳过该步骤。
2. cryptsetup luksFormat /dev/virtual_dev #创建加密设备
3. cryptsetup luksOpen /dev/virtual_dev cryption_dev #打开加密节点，需要输入密码
4. mkfs.ext2 /dev/mapper/cryption_dev #格式化加密磁盘
5. mount /dev/mapper/cryption_dev /mnt #挂载加密磁盘
6. 添加文件到加密盘/mnt
7. umount /mnt #卸载加密盘。
8. dd if=/dev/virtual_dev of=cryption.img #生成加密固件，大小为步骤 1 中创建的大小

注意：

1. 创建加密磁盘时，需要格式化磁盘（步骤 4）
2. 目标设备上，可以直接运行 3/5 步骤打开加密磁盘。