

Android 11.0 Secureboot 使用指南

文档版本 V1.0
发布日期 2020-08-12

版权所有 © 紫光展锐科技有限公司。保留一切权利。

本文件所含数据和信息都属于紫光展锐所有的机密信息，紫光展锐保留所有相关权利。本文件仅为信息参考之目的提供，不包含任何明示或默示的知识产权许可，也不表示有任何明示或默示的保证，包括但不限于满足任何特殊目的、不侵权或性能。当您接受这份文件时，即表示您同意本文件中内容和信息属于紫光展锐机密信息，且同意在未获得紫光展锐书面同意前，不使用或复制本文件的整体或部分，也不向任何其他方披露本文件内容。紫光展锐有权在未经事先通知的情况下，在任何时候对本文件做任何修改。紫光展锐对本文件所含数据和信息不做任何保证，在任何情况下，紫光展锐均不负任何与本文件相关的直接或间接的、任何伤害或损失。

请参照交付物中说明文档对紫光展锐交付物进行使用，任何人对紫光展锐交付物的修改、定制化或违反说明文档的指引对紫光展锐交付物进行使用造成的任何损失由其自行承担。紫光展锐交付物中的性能指标、测试结果和参数等，均为在紫光展锐内部研发和测试系统中获得的，仅供参考，若任何人需要对交付物进行商用或量产，需要结合自身的软硬件测试环境进行全面的测试和调试。非经本公司书面许可，任何单位和个人不得擅自摘抄、复制本文档内容的部分或全部，并不得以任何形式传播。

紫光展锐科技有限公司



前言

概述

本文主要介绍基于 ARM 架构的 Android 11.0 平台的 Secureboot 方案，主要介绍了功能设计实现、配置及调试指导。

读者对象


本文主要针对 Android 11.0 平台 Secureboot 方案的软件开发者。

缩略语

缩略语	英文全名	中文解释
REE	Rich Execute Environment	功能丰富的可执行环境，如 Linux+Android。
TEE	Trusted Execute Environment	可信执行环境，如 Trusty。
SPL	Second Primary Bootloader	第二阶段的引导程序
TOS	Trusted Operating System	可信操作系统

符号约定

在本文中可能出现下列标志，它所代表的含义如下。

符号	说明
 说明	用于突出重要/关键信息、补充信息和小窍门等。 “说明”不是安全警示信息，不涉及人身、设备及环境伤害。

变更信息

文档版本	发布日期	修改说明
V1.0	2020-08-12	第一次正式发布。

关键字

Secureboot、Android 11.0

目 录

1 安全启动介绍.....	1
1.1 密码学原理.....	1
1.2 TEE 介绍.....	2
2 镜像签名.....	4
2.1 镜像签名介绍.....	4
2.2 安全相关配置.....	4
2.3 生成签名密钥对.....	4
2.3.1 BSP 签名密钥.....	5
2.3.2 Avb2.0 签名密钥.....	5
2.4 签名方案.....	7
2.4.1 BSP 签名方案.....	7
2.4.2 Avb2.0 签名方案.....	7
2.5 OTA 编译时的 Avb 签名流程.....	10
3 安全启动过程.....	12
3.1 AP 安全启动过程.....	12
3.2 Modem 类子系统的安全启动.....	13
3.3 安全更新.....	13
3.3.1 工具下载.....	13
3.3.2 Fast boot 烧录.....	14
3.3.3 OTA 升级.....	15
3.4 防版本回退.....	16
3.5 安全调试.....	16
3.6 安全产线部署.....	18

图目录

图 1-1 数字签名和验证过程.....	2
图 1-2 Trusty 概览图	3
图 2-1 签名脚本	5
图 2-2 密钥对文件	6
图 2-3 key 的对应配置	6
图 2-4 可信固件签名后的格式	7
图 2-5 Avb2.0 签名后的格式	8
图 3-1 Secureboot load TEE and REE images process	12
图 3-2 Modem 启动流程	13
图 3-3 工具下载模式启动过程	14
图 3-4 fastboot 模式启动过程	15
图 3-5 生成 SC9863A 调试证书	17

表目录

表 3-1 产线 eFuse 分区内部表	18
----------------------------	----

1 安全启动介绍

Android 采用业界领先的安全功能，并与开发者和设备实现人员密切合作，确保 Android 平台和生态系统的安全。

验证启动会尽力确保所有已执行的代码均来自可信的来源（通常是设备的 OEM），以防受到攻击或者损坏。它建立了一个完整的信任链，该信任链从硬件保护的信任根开始，延伸到引导加载程序，再延伸到启动分区及其它验证分区。

除确保设备运行的是安全的 Android 版本外，验证启动还会检查 Android 版本是否存在内置回滚保护。回滚保护可确保设备只会更新到更高的 Android 版本，避免可能的漏洞持续存在。

OEM 厂商确保用户设备不应该用于和设计不同的目的，不能运行未经允许的软件，保护用户的资产价值；终端用户需要确保自己的设备没有被篡改，是值得被信赖的。

1.1 密码学原理

信息安全需要解决以下三个问题：

- 保密性（Confidentiality）：信息在传输时不被泄露
- 完整性（Integrity）：信息在传输时不被篡改
- 有效性（Availability）：信息的使用者是合法的

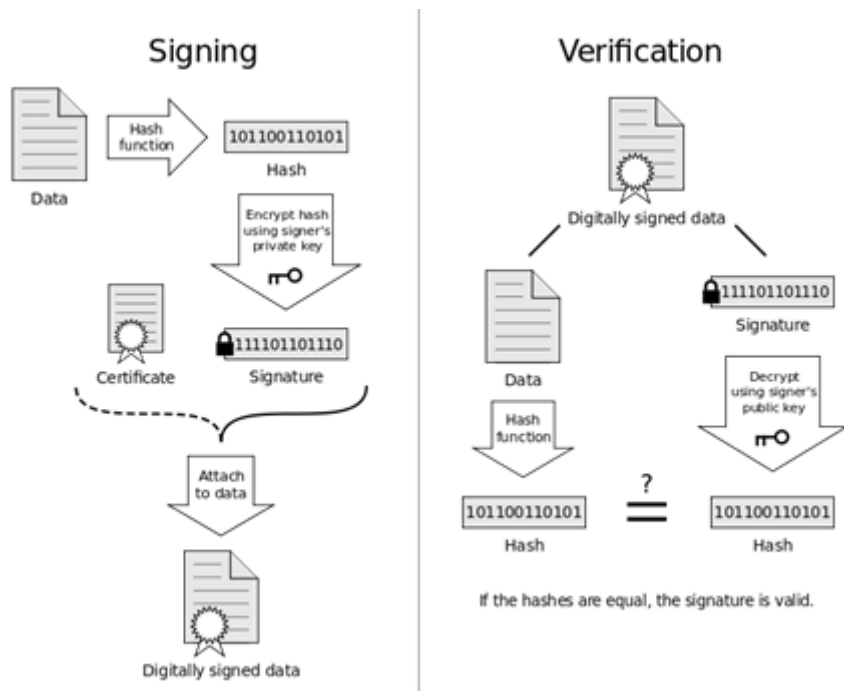
保密性、完整性和有效性统称为 CIA Triad。公钥密码解决保密性问题，数字签名解决完整性问题和有效性问题。

数字签名是通过一些密码算法对数据进行签名，以保护源数据的做法。

典型的数字签名方案包括以下三个算法：

- 密钥生成算法：用来输出公钥和私钥。
- 签名算法：用私钥对给定数据进行加密来生成签名。
- 签名验证算法：用公钥对加密过的消息进行解密验证。

图1-1 数字签名和验证过程



Secureboot 中数字签名由哈希算法和 RSA 算法组成。PC 端生成、部署密钥并对镜像进行签名，终端在启动时对镜像进行签名验证。

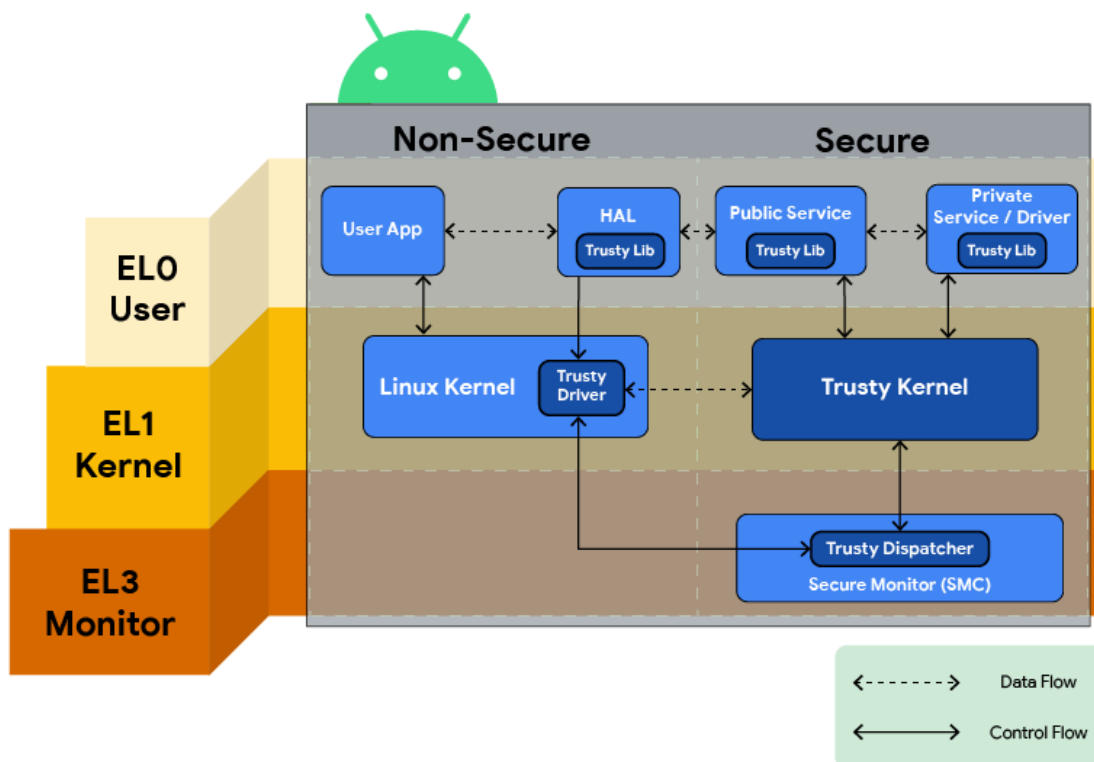
RSA 私钥是 Secureboot 的保障，需要小心保存。

1.2 TEE 介绍

Trusty 是一种安全的操作系统 (OS)，可为 Android 提供可信执行环境 (TEE)。Trusty 操作系统与 Android 操作系统在同一处理器上运行，但 Trusty 通过硬件和软件与系统的其余组件隔离开来。Trusty 与 Android 彼此并行运行。Trusty 可以访问设备主处理器和内存的全部功能，但完全隔离。隔离可以保护 Trusty 免受用户安装的恶意应用以及可能在 Android 中发现的潜在漏洞的侵害。

Trusty 与 ARM 和 Intel 处理器兼容。在 ARM 系统中，Trusty 使用 ARM 的 Trustzone™ 虚拟化主处理器，并创建安全的可信执行环境。

图1-2 Trusty 概览图



Trusty 包含以下组件:

- Little Kernel 衍生的小型操作系统内核 TOS
- Linux 内核驱动程序，用于在安全环境和 Android 之间传输数据。
- Android 用户空间库，用于通过内核驱动程序与可信应用（即安全任务/服务）通信。

2

镜像签名

针对需要保护的镜像的不同特点，采取不同的签名和验证方案。不同签名和验证方案的原理基本相似，区别在于签名算法及验证签名的元数据的组织方式存在不同。

2.1 镜像签名介绍

在引导阶段加载启动的镜像，采用展锐自己的签名验签方案。这些镜像有 fdl1.bin、fdl2.bin、u-boot-spl-16k.bin、sml.bin、tos.bin、u-boot.bin、teecfg.bin。这些镜像的执行处于系统早期的引导阶段，一般都运行在安全级别较高的模式。

Uboot 启动后，会逐步加载和校验内核及 Android 系统和 modem 系统镜像，这些系统镜像均采用 Avb2.0 方案进行签名和验证。对仅读取一次的小分区（例如 boot、dtbo、recovery 和 modem bins）通常是通过将整个内容计算哈希并签名；对于内存装不下的较大分区（如文件系统 system、vendor、product、socko 及 odmko 分区）可以使用哈希树的方式签名；启动时，验证流程会在将数据加载到内存时持续进行。

2.2 安全相关配置

安全启动默认开启，MD 组态编译系统可以根据启动芯片让它跑起来或其它产品需要，使能或禁用安全相关的配置，如果想关闭安全启动，以 UMS512 为例，需作以下修改：

在 device/sprd/sharkl5Pro/ums512_1h10/product/var.mk 中增加

```
$(call md-set, BOARD_SECUREBOOT_CONFIG, false) //false will close
```

在 BSP 板级配置中同步导出如下配置：

```
Bsp/device/sharkl5Pro/androidr/ums512_1h10/ums512_1h10_base/common.cfg
//secureboot
export BSP_PRODUCT_SECURE_BOOT="SPRD" // "NONE" for close
export BSP_PRODUCT_VBOOT="V2" // "" for close
//firewall
export BSP_CONFIG_TEE_FIREWALL="true"
export BSP_BOARD_TEE_CONFIG="trusty"
export BSP_BOARD_ATF_CONFIG="true"
export BSP_BOARD_ATF_BOOT_TOS_CONFIG="true"
```

2.3 生成签名密钥对

安全启动方案需要配置如下的密钥来完成所有系统镜像的签名和校验。

2.3.1 BSP 签名密钥

BSP 相关镜像签名密钥文件的路径如下：

bsp/build/packimage_scripts/configs/

主要有以下几对：

- rsa2048_0.pem & rsa2048_0_pub.pem
- rsa2048_1.pem & rsa2048_1_pub.pem
- rsa4096_vbmeta.pem & Rsa4096_vbmeta.pem

下述命令生成 RSA 密钥对：

生成 RSA 私钥

```
$ openssl genrsa -out rsa2048 0.pem 2048
Generating RSA private key, 2048 bit long modulus
.....+++
.....+++
e is 65537 (0x10001)
```

生成 RSA 公钥

```
$ openssl rsa -in rsa2048 0.pem -pubout -out rsa2048 0 pub.pem
writing RSA key
$ls -al *.pem
-rw-r--r-- 1 user group 1679 Feb 26 20:46 rsa2048 0.pem
-rw-r--r-- 1 user group 451 Feb 26 20:48 rsa2048_0_pub.pem
```

BSP 签名是在 chipram、bootloader 及 trusty 等目标的编译过程中自动进行。如图 2-1 所示，签名脚本将会在目标编译成功后触发。

图2-1 签名脚本

```
if [ $ret -eq 0 ] ; then
    if [ ! "$ONE_SHOT_MAKEFILE" ]; then
        build_tool_and_sign_images "$@"
    fi
fi

local HOST_OUT_EXE=$(get_build_var HOST_OUT_EXECUTABLES)
. $(gettop)/$HOST_OUT_EXE/packimage.sh "$@"
. $(gettop)/$HOST_OUT_EXE/make_vbmeta_gsi.sh
```

其中 rsa2048_0 公钥对用于 fdl1.bin 和 u-boot-spl-16k.bin 的签名；rsa2048_1*.pem 用于 sml.bin、tos.bin 和 u-boot.bin 的签名。

2.3.2 Avb2.0 签名密钥

Avb2.0 签名方案所需密钥对文件部署在以下目录：

vendor/sprd/proprieties-source/packiamge_scripts/signimage/sprd/config/

该目录中的密钥对文件如图 2-2 所示。

图2-2 密钥对文件

rsa4096_boot.pem	H A D	27-Dec-2019	3.2 KiB	52	51
rsa4096_boot_pub.bin	H A D	27-Dec-2019	1 KiB		
rsa4096_modem.pem	H A D	27-Dec-2019	3.2 KiB	52	51
rsa4096_modem_pub.bin	H A D	27-Dec-2019	1 KiB		
rsa4096_odmko.pem	H A D	27-Dec-2019	3.2 KiB	52	51
rsa4096_odmko_pub.bin	H A D	27-Dec-2019	1 KiB		
rsa4096_product.pem	H A D	27-Dec-2019	3.2 KiB	52	51
rsa4096_product_pub.bin	H A D	27-Dec-2019	1 KiB		
rsa4096_recovery.pem	H A D	27-Dec-2019	3.2 KiB	52	51
rsa4096_recovery_pub.bin	H A D	27-Dec-2019	1 KiB		
rsa4096_socko.pem	H A D	27-Dec-2019	3.2 KiB	52	51
rsa4096_socko_pub.bin	H A D	27-Dec-2019	1 KiB		
rsa4096_system.pem	H A D	27-Dec-2019	3.2 KiB	52	51
rsa4096_system_ext.pem	H A D	05-May-2020	3.2 KiB	52	51
rsa4096_system_ext_pub.bin	H A D	05-May-2020	1 KiB		
rsa4096_system_pub.bin	H A D	27-Dec-2019	1 KiB		
rsa4096_vbmeta.pem	H A D	27-Dec-2019	3.2 KiB	52	51
rsa4096_vbmeta_pub.bin	H A D	27-Dec-2019	1 KiB		
rsa4096_vendor.pem	H A D	27-Dec-2019	3.2 KiB	52	51
rsa4096_vendor_pub.bin	H A D	27-Dec-2019	1 KiB		

使用 genkey.sh 脚本可生成新的密钥对，以 oem 分区密钥对为例：

```
sprd/config$ ./genkey.sh oem
Generating RSA private key, 4096 bit long modulus
.....++
.....
.....++
e is 65537 (0x10001)
$ls -al *oem*.
-rw-r--r-- 1 user group 3247 Feb 26 21:05 rsa4096_oem.pem
-rw-r--r-- 1 user group 1032 Feb 26 21:05 rsa4096_oem_pub.bin
```

在 device/sprd/sharkle/common/security_feature.mk 文件中可以找到 key 的对应配置，如图 2-3 所示。

图2-3 key 的对应配置

```
#config key&version for boot
BOARD_AVB_BOOT_KEY_PATH:=$(CONFIG_PATH)/rsa4096_boot.pem
BOARD_AVB_BOOT_ALGORITHM:=SHA256_RSA4096
BOARD_AVB_BOOT_ROLLBACK_INDEX:=$(shell sed -n '/avb_version_boot/p'
BOARD_AVB_BOOT_ROLLBACK_INDEX_LOCATION:=1
```

BOARD_AVB_BOOT_KEY_PATH 定义了 BOOT 镜像签名使用的私钥名称

BOARD_AVB_BOOT_ALGOTITH 定义的签名算法为 SHA256_4096

完成 boot image 的编译时，会通过以下脚本对 boot image 进行签名：

扩展参数后如下：

```
./external/avb/avbtool add hash footer -image out/target/product/xxxx/boot.img -partition size 36700160 --algorithm SHA256 RSA4096 --rollback index 0 --prop com.android.build.boot.os version:10 -key vendor/sprd/proprieties-source/packimage_scripts/signimage/sprd/config/rsa4096_boot.pem
```

其它分区以此类推。

2.4 签名方案

2.4.1 BSP 签名方案

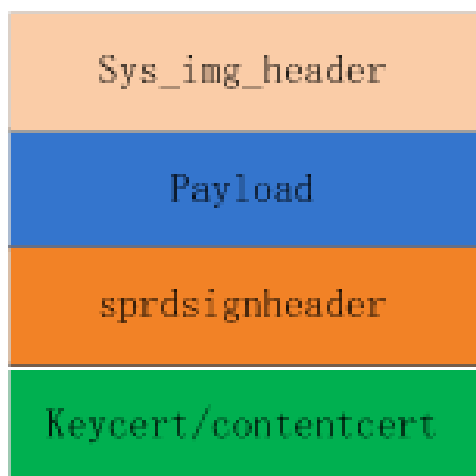
BSP 方案签名工具主要有 packimage.sh、sprd_sign 及 imageheaderinsert 三个工具；工具在成功编译后将其安装在 “out/host/linux-x86/bin” 目录中，会在编译过程中自动参与系统镜像的签名。

- Packimage.sh：起始签名脚本，包含了工程配置信息、需要签名的可信固件、签名方式等，会去调用 imageheaderinsert 工具对镜像头部插入用来校验的元数据和尾部插入签名证书。
- Imageheaderinsert：执行将检验需要的元数据插入到镜像头文件的工具。
- Sprd_sign：使用密钥对镜像进行签名的工具。

BSP 签名方案签名的镜像包括：SPL、fdl1、uboot、fdl2、sml、TOS、vbmeta。

可信固件签名后的格式如图 2-4 所示。

图2-4 可信固件签名后的格式



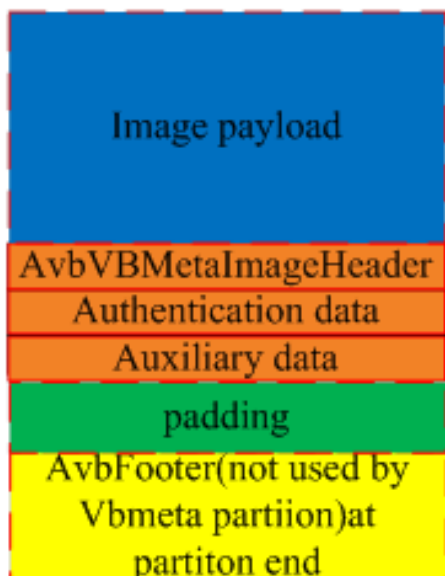
2.4.2 Avb2.0 签名方案

使用 Avb2.0 签名方案签名的文件有：

dtbo/boot/recovery/system/vendor/product/socko/odmko/modem bins

Avb2.0 签名后的格式如错误!未找到引用源。所示。

图2-5 Avb2.0 签名后的格式



镜像头部没有校验用的元数据，Avb 签名方案使用镜像尾部 64 字节的 AvbFooter 来定位校验数据位置。Android 10.0 引入了 super 分区概念，从而使得位于 super 分区的 system、vendor 和 product 分区的 footer 按照上述设计也无法访问，目前的方案是将校验 system、vendor 和 product 分区的元数据复制保存到独立的分区：vbmeta_system 和 vbmeta_vendor。

对应关系如下：

- System 和 product 分区的 metadata 保存到 vbmeta_system 分区
- Vendor 分区的 metadata 保存到 vbmeta_vendor 分区

注意 super 分区中包含的动态分区的元数据一定要和 vbmeta_system 和 vbmeta_vendor 分区的元数据保持一致。

super image 中动态分区的 metadata 和 vbmeta_system/vendor image 中是否一致，可以通过以下流程来确认：

1. 将 super image 从 sparse 格式转化为 raw data 格式，为下一步解开 super image 做准备。

```
$/out/host/linux-x86/bin/simg2img out/target/product/s9863a1h10/super.img
out/target/product/s9863a1h10/super-raw.img
```

2. 使用 lpunpack 从 super image 中提取出动态分区的镜像。

```
./out/host/linux-x86/bin/lpunpack out/target/product/s9863a1h10/unpack/super-raw.img
out/target/product/s9863a1h10/unpack/
```

完成后会在 unpack 目录解压出 super 分区里包含分区的镜像。

3. 使用 avbtool 分区查看动态分区的 metadata 和 vbmeta_xxx 分区的 metadata 是否一致。

```
$ ./external/avb/avbtool info_image --image ../unpack/system.img
Footer version:      1.0
Image size:          1345548288 bytes
Original image size: 1324228608 bytes
VBMeta offset:       1345212416
VBMeta size:         704 bytes
--
Minimum libavb version: 1.0
Header Block:         256 bytes
Authentication Block:  0 bytes
Auxiliary Block:       448 bytes
Algorithm:             NONE
Rollback Index:        0
Flags:                 0
Release String:        'avbtool 1.1.0'
Descriptors:
  Hashtree descriptor:
    Version of dm-verity: 1
    Image Size:           1324228608 bytes
    Tree Offset:          1324228608
    Tree Size:            10432512 bytes
    Data Block Size:      4096 bytes
    Hash Block Size:      4096 bytes
    FEC num roots:        2
    FEC offset:           1334661120
    FEC size:             10551296 bytes
    Hash Algorithm:       sha1
    Partition Name:       system
    Salt:                 ed49162cf97df4672cbf6793955dbbb7061956e3
    Root Digest:          6d65697a8156574d5e433ca5e585322e22fcdadb
    Flags:                0
  Prop: com.android.build.system.os version -> '10'
  Prop: com.android.build.system.security patch -> '2020-02-05'

$ ./external/avb/avbtool info_image --image ../unpack/vbmeta system.img
Minimum libavb version: 1.0
Header Block:         256 bytes
Authentication Block:  0 bytes
Auxiliary Block:       832 bytes
Algorithm:             NONE
Rollback Index:        0
Flags:                 0
Release String:        'avbtool 1.1.0'
Descriptors:
  Prop: com.android.build.system.os version -> '10'
  Prop: com.android.build.system.security patch -> '2020-02-05'
  Prop: com.android.build.product.os version -> '10'
  Prop: com.android.build.product.security patch -> '2020-02-05'
  Hashtree descriptor:
    Version of dm-verity: 1
    Image Size:           447315968 bytes
    Tree Offset:          447315968
    Tree Size:            3530752 bytes
    Data Block Size:      4096 bytes
    Hash Block Size:      4096 bytes
```



```

FEC num roots:      2
FEC offset:         450846720
FEC size:           3571712 bytes
Hash Algorithm:      sha1
Partition Name:      product
Salt:               743ca7308a1574360168362a53cc53f511242232
Root Digest:        e6a8dda20f28a9905b36c0154fe79158221471f4
Flags:              0
Hashtree descriptor:
Version of dm-verity: 1
Image Size:         1324228608 bytes
Tree Offset:        1324228608
Tree Size:          10432512 bytes
Data Block Size:    4096 bytes
Hash Block Size:    4096 bytes
FEC num roots:      2
FEC offset:         1334661120
FEC size:           10551296 bytes
Hash Algorithm:      sha1
Partition Name:      system
Salt:               ed49162cf97df4672cbf6793955dbbb7061956e3
Root Digest:        6d65697a8156574d5e433ca5e585322e22fcdadb
Flags:              0

```

比较 **system** 分区的 **Salt** 和 **hashTree** 的根哈希是否一样。如果不一样，系统会在内核挂载系统分区时出现校验错误，引起系统重启。

2.5 OTA 编译时的 Avb 签名流程

编译 OTA 时，OTA 包编译系统使用 **target** 目录的内容生成新的 **system**、**vendor** 和 **product** 分区镜像，这样为了保持 OTA 包中的版本和将来从 **PRODUCT_OUT** 目录打包到 **pac** 包中的版本保持一致，需要用 OTA 编译生成的新的镜像替换 **PRODUCT_OUT** 下的 **system**、**vendor**、**product**、**vbmeta_system** 和 **vbmeta_vendor** 镜像。

相关脚本位于 **vendor/sprd/build/tasks/sprdbuildota.mk**

```

$(hide) -$(ACP) $(SPRD_BUILT_TARGET_FILES_PACKAGE)/IMAGES/vbmeta_system.img
$(PRODUCT_OUT)/vbmeta_system.img -rfv
$(hide) -$(ACP) $(SPRD_BUILT_TARGET_FILES_PACKAGE)/IMAGES/vbmeta_vendor.img
$(PRODUCT_OUT)/vbmeta_vendor.img -rfv

```

system vendor 和 **product** 镜像则是通过使用 **target** 目录的动态分区镜像重新编译生成 **PRODUCT_OUT** 下的 **super.img** 完成同步。

编译命令如下：

```

lpmake --metadata-size 65536 --super-name super --metadata-slots 2 --device super:4299161600 --group
group unisoc:4299161600 --partition system:readonly:1345548288:group unisoc --image
system=out/target/product/s9863alh10/obj/PACKAGING/target_files_intermediates/s9863alh10 Natv-
target_files-eng.wenquan.zhang/IMAGES/system.img --partition vendor:readonly:405712896:
group unisoc --image
vendor=out/target/product/s9863alh10/obj/PACKAGING/target_files_intermediates/s9863alh10 Natv-
target_files-eng.wenquan.zhang/IMAGES/vendor.img --partition
product:readonly:454574080:group_unisoc --image

```

```
product=out/target/product/s9863alh10/obj/PACKAGING/target_files_intermediates/s9863alh10_Natv-  
target_files-eng.wenquan.zhang/IMAGES/product.img --sparse --output  
out/target/product/s9863alh10/super.img "
```

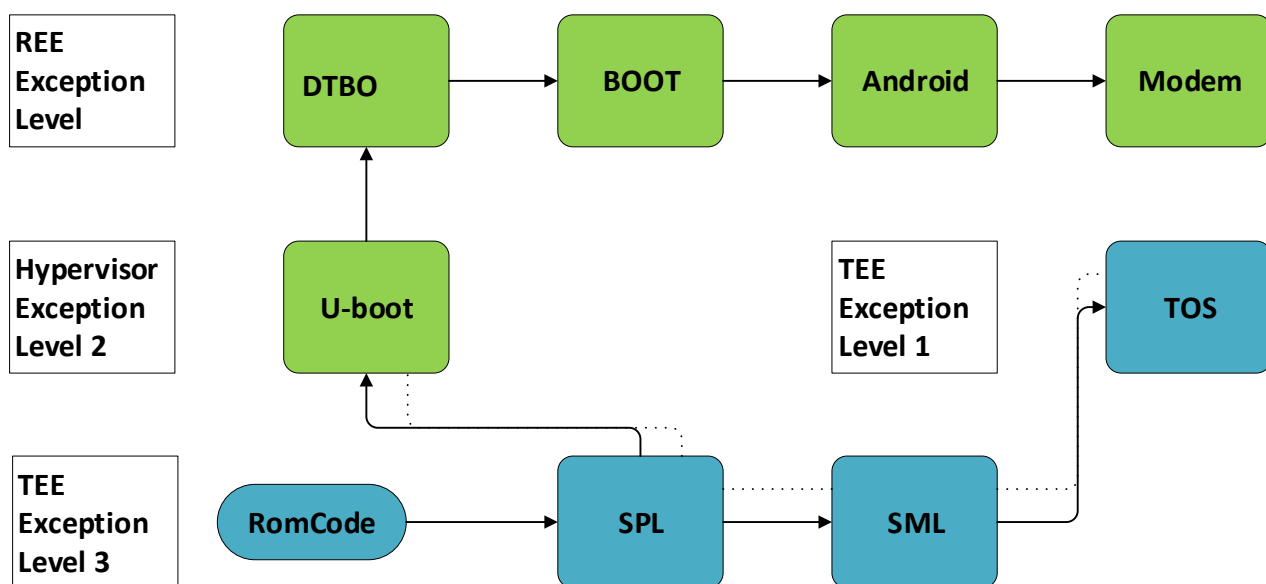
所以完成 OTA 包编译后，PRODUCT_OUT 目录的 system、vendor 和 product 分区的内容和 super.img unpack 出来的 system、vendor 和 product 镜像应该是不一致的，只需要确认 super image unpack 出来的 system、vendor 和 product 镜像的 metadata 和 PRODUCT_OUT 目录下的 vbmeta_system&vendor 镜像的 metadata 一致即可。

3 安全启动过程

3.1 AP 安全启动过程

AP 从 Romcode 开始，启动时运行在安全模式，逐级加载并校验安全镜像，再引导正常的系统镜像，直到内核和安卓系统启动完毕，如图 3-1 所示。

图3-1 Secureboot load TEE and REE images process



- 安全部署的设备会在 AP Soc 的 ROTPK efuse 区域写入用于验证 SPL 镜像签名的第一级公钥（rsa2048_0_pub.bin）的哈希。
- Romcode 作为信任根的一部分，是不能被重写（篡改）的，是安全启动的基础。上电后 Romcode 加载 SPL 镜像，从 SPL 分区读取第一级公钥，并计算其哈希和 ROTPK 记录的公钥哈希比较，一致才能使用这个公钥来验证 SPL 镜像的数字签名。
- SPL 镜像作为 Secureboot loader 运行在 Secure IRAM，负责初始化 DRAM 并将 sml(Arm Trusted firmware), TOS (Trusty OS) 和 uboot (REE bootloader) 加载到 DRAM，同时负责校验这三个系统镜像的合法性；校验通过后将执行 sml。
- Sml 运行 TOS，TOS 运行内置的 TA，然后 TOS 返回到 sml，sml 返回到 uboot 执行。
- Uboot 运行在 Hypervisor 模式，负责检查设备状态，加载和校验 dtbo 和 boot/recovery 镜像的内容，并准备验证启动 Android 系统的内核参数。
- Uboot 引导内核启动后，内核将负责验证 Android 系统分区，验证通过后将挂接为只读的系统分区。
- Android 启动后，modem_control 服务负责校验和加载 modem 子系统的固件。

3.2 Modem 类子系统的安全启动

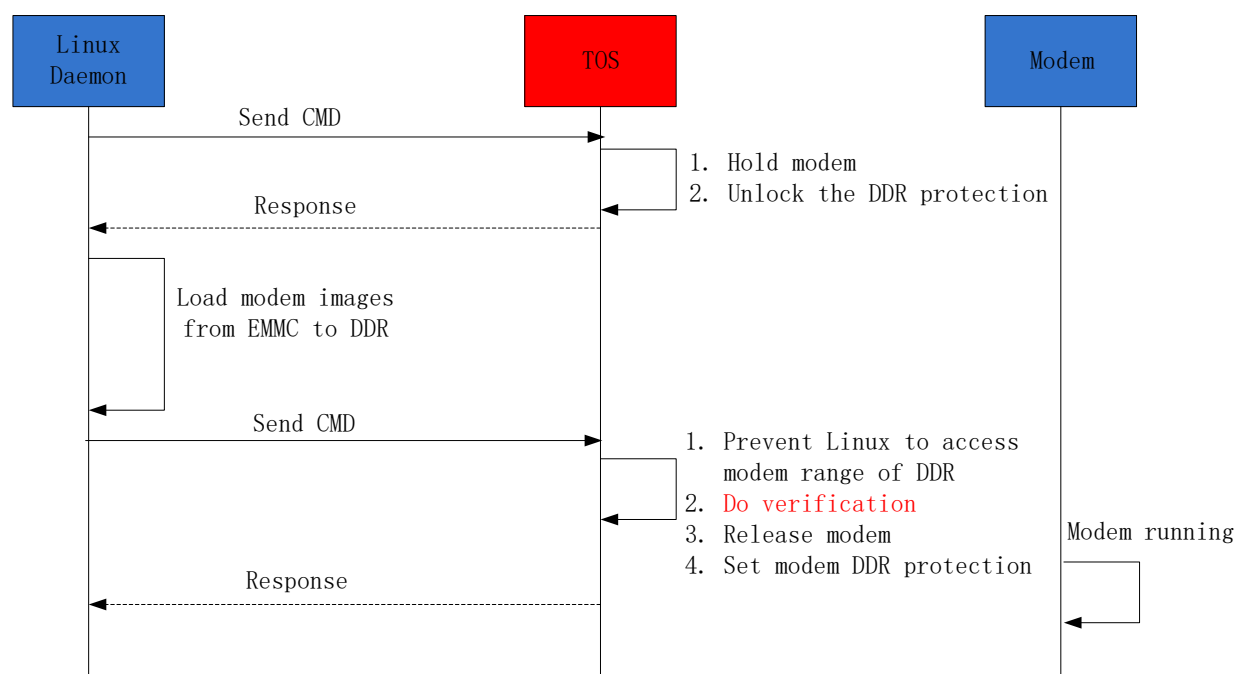
Modem、Audio、VDSP、GPU firmware、SCP 等不含 ROM 的子系统，通常需要通过主控配合来完成子系统的安全启动流程。

设计思路如下：

- 通过 Register Firewall 限制子系统的启动。
- 通过 DDR Firewall 来限制子系统的地址访问范围，通常仅访问自己的地址空间。
- 镜像先通过 AP 子系统加载到 DDR，然后限制这段空间的访问，验签成功后启动相应的子系统。避免验签和运行的是不同的子系统。

Modem 子系统启动流程如图 3-2 所示。

图3-2 Modem 启动流程

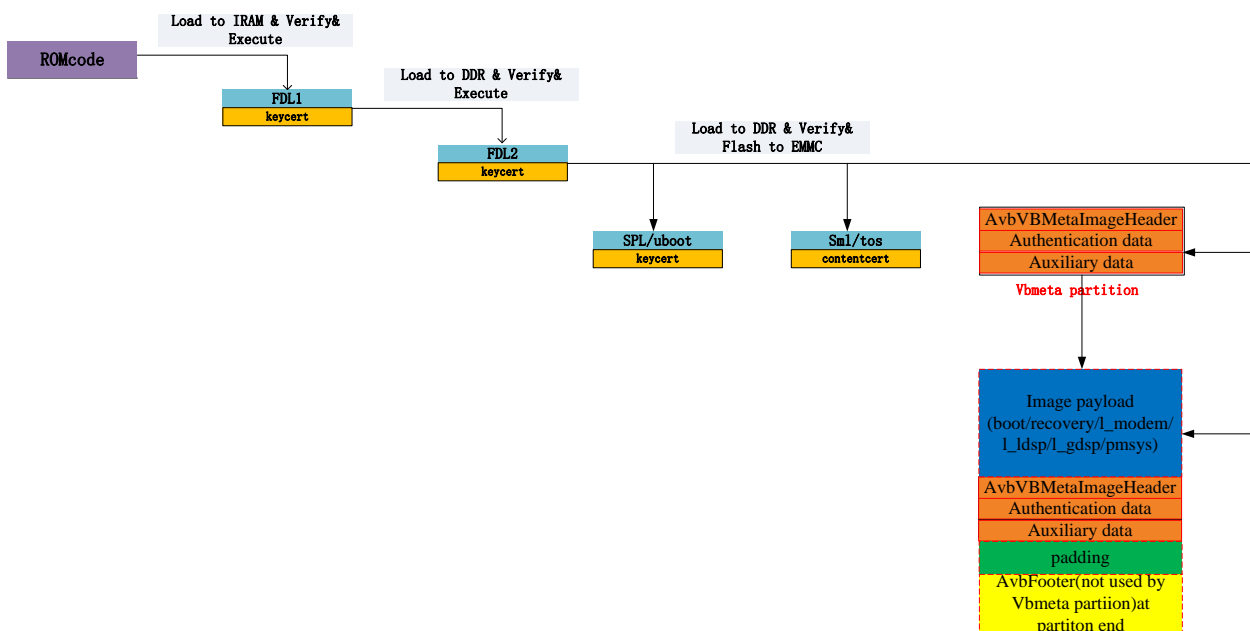


3.3 安全更新

3.3.1 工具下载

下载模式在 fdl1.bin 作用和正常启动模式的 SPL 镜像相同，在安全部署的机器上同样会使用 romcode 校验 fdl1.bin 的数字签名，通过后 fdl1 初始化 DRAM 并加载和校验 fdl2.bin；fdl2 负责和 PC 端交互，接收和校验 PC 端下载的数据，并在验证通过后将其更新到 EMMC/NAND 等静态存储。

图3-3 工具下载模式启动过程



在调试 **super** 分区内容时，如果使用 **pac** 包下载了 **super** 分区内容，必须同时下载 **super** 镜像对应的 **vbmeta_system** 镜像和 **vbmeta_vendor** 镜像。

3.3.2 Fast boot 烧录

fastboot 模式烧录系统分区时，设备应处于解锁状态。

设备状态用于指明能够以多大的自由度将软件刷写到设备上，以及是否强制执行验证。设备状态为 **LOCKED** 和 **UNLOCKED**。**LOCKED** 状态的设备禁止刷写软件，**UNLOCKED** 状态的设备允许刷写软件。

使用 **fastboot flashing [unlock_bootloader | lock_bootloader] signature.bin** 命令可更改设备状态。设备状态发生变化，都会先擦除数据分区中的数据，为保护用户数据会在删除数据之前要求用户确认。

解锁 **bootloader**

Boot-debug.img 是内核加上一个 **debug** 版本的 **ramdisk**，让 **user** 版本的 **VTS&STS** 版本可以拥有 **ROOT** 权限；设计上不需要 **OEM** 签名，只在解锁 **bootloader** 时允许 **verification error** 的设备上使用。

如何解锁 **bootloader**，请参考 **AndroidQ** 一键解锁 **bootloader** 的相关文档。

解锁后，系统启动过程处于 **bootloader** 阶段时机器左上角显示如下提示：

```
INFO: LOCK FLAG IS : UNLOCK!!!
```

内核启动参数中也会指示设备状态：

```
androidboot.verifiedbootstate=orange androidboot.flash.locked=0
```

解锁之后，如果需要 **remount**，执行下述命令：

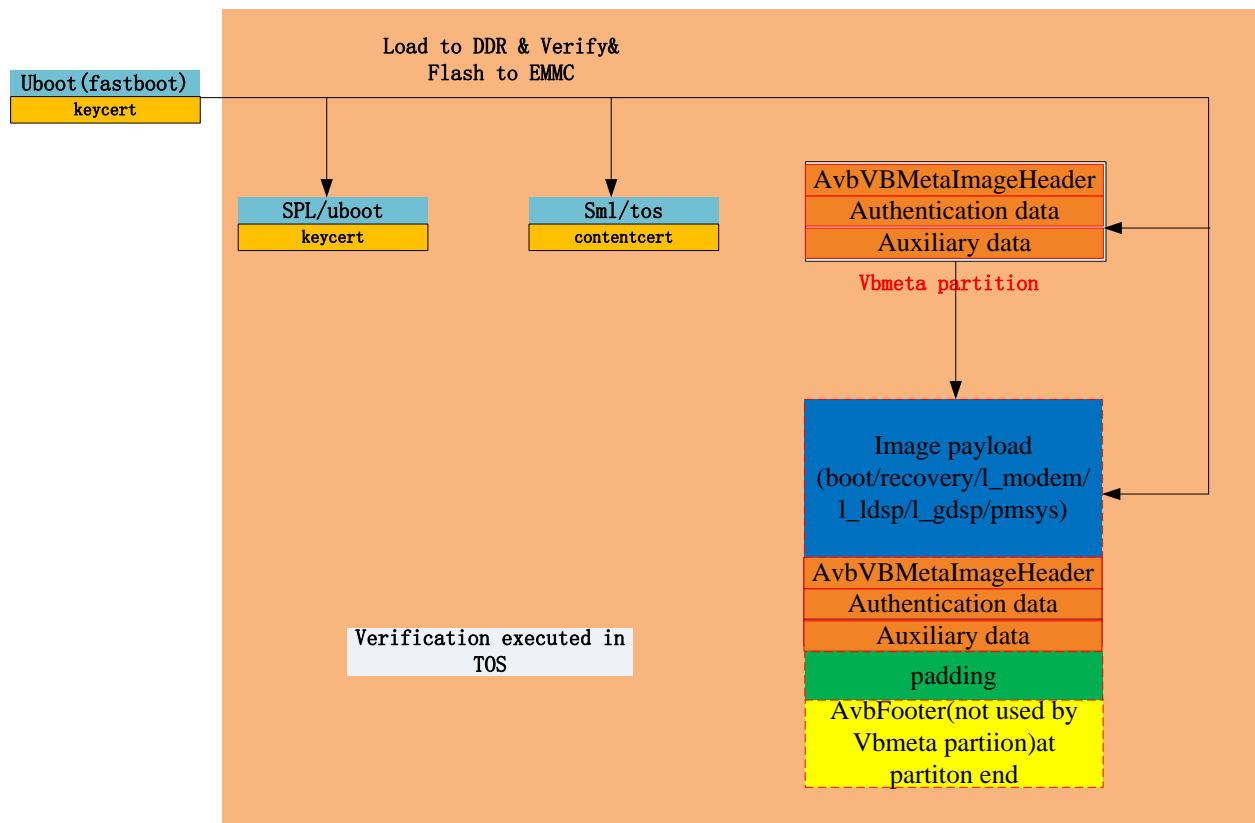
```
$adb root
$adb disable-verify
```

```
$adb reboot
$adb wait-for-device
$adb root
$adb remounts
```

执行完以上指令，系统 remount 成功。

fastboot 模式安全启动过程如图 3-4 所示，它的镜像数字签名验证在 TOS 中进行，具有更高的安全性。

图3-4 fastboot 模式启动过程



在 fastbootd 模式下，动态分区使用 fastboot 命令进行更新。super 分区需要解锁设备才可以更新。

fastbootd 调试请参考 fastbootd 相关的文档。

说明

fastbootd 模式更新 super/system/vendor/product 镜像时，需要同时更新 vbmeta_system/vbmeta_vendor 分区。

3.3.3 OTA 升级

终端用户通过安全更新来获取新的发布版本，新版本除了配置防回退的版本外，也都需要合法的签名，OTA 更新包也会做整体的签名和验证，保证用户更新版本的合法性和完整性。

3.4 防版本回退

即使更新流程完全安全，攻击者仍可能会利用非永久性系统漏洞来手动安装更易受攻击的旧版系统，重新启动进入易受攻击的版本，然后通过该版本来安装永久性漏洞。在这种情况下，攻击者可通过这种漏洞永久拥有相应设备，并可以执行任何操作（包括停用更新）。

防范这类攻击的保护措施称为“回滚保护”。“回滚保护”通常通过以下方式实现：使用防篡改的存储空间来记录最新的版本，并在版本低于记录的版本时拒绝启动系统。系统通常会针对每个分区来跟踪版本。

展锐支持对每一个分区进行防版本回退的配置，防回退的版本号可以在以下配置文件中配置：

- Bsp/build/packimage_scripts/config/version.cfg
- Vendor/sprd/proprieties-source/packimage_scripts/signimage/sprd/config/version.cfg

防回退版本会编译到安全启动的元数据中，在安全启动过程中，系统会在各个启动阶段比较当前启动的系统镜像的版本号是否大于等于记录在 **efuse**（**trusty firmware**）和 **rpmb** 分区（**Avb2.0** 签名镜像）的版本号，只有版本号满足条件，才会允许启动，且在系统完全启动后，如果时新的版本号，则更新到一次性可写区域。

说明

防回退版本只在将来量产版本中配置，不能在研发调试阶段配置。

3.5 安全调试

安全调试之前，需要生成调试证书，复制 **fdl1-sign.bin** 和 **u-boot-spl-16k-sign.bin** 镜像文件并使用生成的调试证书对这两个复制文件进行签名。

具体操作步骤如下：

步骤 1 将 **fdl1-sign.bin** 和 **u-boot-spl-16k-sign.bin** 复制到以下目录：

```
$/vendor/sprd/proprieties-source/packimage_scripts /signimage/sprd/mkdbimg/bin
```

步骤 2 获取 **socid** 号(每颗芯片都有唯一的 **socid** 号)

fastboot 模式获取 **socid** 的命令如下：

```
sudo./fastboot getsocid socid
```

获取的 **socid** 如下：

```
socid is:
939ff32ca2d078bbc04a045bdfbac721
8fdb2603bd2adaaa3a6f4fa780d797f8
```

步骤 3 生成调试证书

调试证书包括初级证书与二级证书，如果不更改 **devkey**，则可以忽略步骤 **a**。

a.（可选）将 **rsa2048_devkey_pub.pem** 放入如下路径：

```
$/vendor/sprd/proprieties-source/packimage_scripts/signimage/sprd/config
```

将 **rsa2048_devkey_pub.pem** 和 **rsa2048_devkey.pem** 放入如下路径：

```
$vendor/sprd/proprieties-source/packimage_scripts/signimage/sprd/mkdbimg/config
```

b. 在以下目录执行如下命令

```
$vendor/sprd/proprieties-source/packimage_scripts/signimage/sprd/mkdbimg/script
$./sprd_mkdbimg.sh 0xffffffffffffffff
```

0xffffffffffffffff 是调试掩码，输入的调试掩码是给初级证书使用，可以根据需要设置。

终端将显示：

```
Note: Only 9863a/7731e/9832e device series padding is pkcs15, the other is pss.
enter your device padding type [ 1:pss 2:pkcs15 ]
```

以 SC9863A 系列工程为例，输入 2

终端将显示：

```
next enter your device socid and debug mask:
pls input parameter like: 0xfacd...de 0xffff
eg.0x939ff32ca2d078bbc04a045bdfbac7218fdb2603bd2adaaa3a6f4fa780d797f8 0xffffffffffffffff
```

按提示输入 socid 和 debugmask 参数

socid 参数是步骤 2 中获取的 soc 的唯一标识。

debugmask 参数是 64bit 调试掩码。可根据调试需要开启，如果要启用所有调试位，需要输入“0xffffffffffffffff”。

生成 SC9863A 工程的调试证书示例如图 3-5 所示。

图3-5 生成 SC9863A 调试证书

```
pengao.liu1@bjand08:~/sprdroidr_trunk/vendor/sprd/proprieties-source/packimage_scripts/signimage/sprd/mkdbimg/script$ ./s
prd_mkdbimg.sh 0xffffffffffffffff
Note:Only 9863a/7731e/9832e device series padding is pkcs15, The other is pss .
enter your device padding type [ 1:pss 2:pkcs15 ] 2
your device padding is pkcs15
input mask is: 0xffffffffffffffff
create primary_debug.cert sucessfull!
next enter your device socid and debug mask :
pls input parameter like: 0xfacd...de 0xffff eg. fda424214371504f465721d8a86fa794219405493a4e41403fce5c2b7b22791c 0xffff
ffffffffffff
use pkcs1 format
mkdbimg fdli-sign.bin ok!
use pkcs1 format
mkdbimg spl-sign.bin ok!
pengao.liu1@bjand08:~/sprdroidr_trunk/vendor/sprd/proprieties-source/packimage_scripts/signimage/sprd/mkdbimg/script$
```

步骤 4 使用生成的调试证书对步骤 1 复制的两个镜像文件进行签名。

----结束

说明

- 输入设备证书填充类型[1: pss 2: pkcs15]，证书填充类型根据芯片系列区分不同的填充方式，9863a/7731e/9832e/8541e 系列及其衍生芯片系列采用 pkcs15 方式，其余如 ums312/ums512/9230 等采用 pss 填充方式。可通过 source/lunch 的工程名来判断。
- 如果安全调试证书无法启用 JTAG，按如下命令操作：

```
cd vendor/sprd/proprieties-source/packimage source
mma
```


- 如果运行 `sprd_mkprimarycert.sh` 或 `sprd_mkdbimg.sh` 时出现以下警告：“加载共享库时出错：libc++so：无法打开共享库文件：没有这样的文件或目录”，可以将 `vendor/sprd/proprietary-source/packimage_scripts/signimage/sprd/mkdbimg/bin` 中的 `libc++.so` 文件复制到 `/lib/x86_64-linux-gnu/`。

3.6 安全产线部署

安全启动相关的数据在产线 eFuse 分区需要写入的内容如表 3-1。

表3-1 产线 eFuse 分区内部表

名称	用途	长度	位置	说明
ROTPK	Root of trust Public key Hash	256	private efuse 区 bit[512]~bit[767]	根可信公钥的消息摘要，用于验证签名证书的真实性。由 TAM 维护方进行发布，一般是 OEM 厂商确定。
Secure OS 最小版本号	anti-rollback counter	32	private efuse 区 bit[768]~bit[799]	存放第三方 Secure OS 厂家软件版本号
Android 最小 版本号	anti-rollback counter	224	private efuse 区 bit[800]~bit[1023]	存放 OEM/ODM 厂家 Android 软件版本号

说明

上述信息在 efuse 中位置必须与展锐约定一致。