



Unisoc Confidential For kxdwww

# Android 13 Secure Boot 使用指南


文档版本                      V1.2  
发布日期                      2023-08-23

## 版权所有 © 紫光展锐（上海）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（上海）科技有限公司（以下简称紫光展锐）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。

## 商标声明

**紫光展锐**、**UNISOC**、、展讯、Spreadtrum、SPRD、锐迪科、RDA 及其他紫光展锐的商标均为紫光展锐（上海）科技有限公司及/或其子公司、关联公司所有。

本文档提及的其他所有商标或注册商标，由各自的所有人拥有。

## 免责声明

本文档可能包含第三方内容，包括但不限于第三方信息、软件、组件、数据等。紫光展锐不控制且不对第三方内容承担任何责任，包括但不限于准确性、兼容性、可靠性、可用性、合法性、适当性、性能、不侵权、更新状态等，除非本文档另有明确说明。在本文档中提及或引用任何第三方内容不代表紫光展锐对第三方内容的认可、承诺或保证。

用户有义务结合自身情况，检查上述第三方内容的可用性。若需要第三方许可，应通过合法途径获取第三方许可，除非本文档另有明确说明。

# 紫光展锐（上海）科技有限公司



# 前 言

## 概述

本文主要介绍展锐 Android 13 平台的 Secure Boot 方案，主要包括 Secure Boot 签名、启动过程及调试指导说明。

## 读者对象




本文适用于使用展锐 Android 13 平台 Secure Boot 方案的软件开发人员。

## 缩略语

缩略语	英文全名	中文解释
LK	Little Kernel	小内核
REE	Rich Execution Environment	富可执行环境
SPL	Second Primary Bootloader	第二阶段的引导程序
TEE	Trusted Execution Environment	可信执行环境
TOS	Trusted Operating System	可信操作系统

## 符号约定

在本文中可能出现下列符号，每种符号的说明如下。

符号	说明
 <b>说明</b>	用于突出重要或关键信息、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。
 <b>注意</b>	用于突出容易出错的操作。 “注意”不是安全警示信息，不涉及人身、设备及环境伤害。
 <b>警告</b>	用于可能无法恢复的失误操作。 “警告”不是危险警示信息，不涉及人身及环境伤害。

## 变更信息

文档版本	发布日期	修改说明
V1.2	2023-08-23	增加加 UMS9158 平台信息。
V1.1	2022-11-30	<ul style="list-style-type: none"><li>• 新增密钥生成命令</li><li>• 新增 Kernel5.15 中新增分区的所需密钥</li><li>• 新增启动过程中设备状态</li><li>• 新增 UMS9621 防回滚双 slot 说明</li></ul>
V1.0	2022-01-14	第一次正式发布

## 关键字

Secure Boot/安全启动、Avb2.0、防版本回退、安全更新

Unisoc Confidential For kxdwww

# 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 Secure Boot 签名.....	4
2.1 安全启动配置.....	4
2.2 镜像签名.....	4
2.3 签名密钥对.....	5
2.3.1 BSP 签名密钥对.....	5
2.3.2 Avb2.0 签名密钥对.....	8
2.4 签名方案.....	10
2.4.1 BSP 签名方案.....	10
2.4.2 Avb2.0 签名方案.....	11
2.5 OTA 编译时的 Avb 签名流程.....	14
3 Secure Boot 启动过程.....	16
3.1 AP 安全启动过程.....	16
3.1.1 启动流程概述.....	16
3.1.2 启动过程中设备状态.....	17
3.2 Modem 类子系统的安全启动.....	17
3.2.1 modem 启动介绍.....	17
3.2.2 wcn 和 gnss 启动介绍.....	18
3.3 安全更新.....	18
3.3.1 工具下载.....	18
3.3.2 fastboot 烧录.....	19
3.3.3 OTA 升级.....	22
3.4 防版本回退.....	22
3.4.1 版本号配置.....	22
3.4.2 版本号更新策略.....	23
3.5 安全调试.....	25
3.6 安全产线部署.....	26
4 sprd_sign 签名工具.....	27
4.1 生成签名工具.....	27
4.2 工具使用说明.....	27
5 genkey.sh 和 version.cfg.....	31
5.1 genkey.sh 脚本.....	31
5.2 version.cfg 文件.....	31

6 常见问题答疑.....	33
6.1 如何回读分区 .....	33
6.2 如何关闭 secureboot（配置 nasec 版本） .....	33
6.3 如何确定 bsp 镜像签名是否正常.....	33

Unisoc Confidential For kxdwww

## 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 BSP 签名后可信镜像的格式 .....	11
图 2-2 Avb2.0 签名后镜像的格式 .....	12
图 3-1 安全启动镜像加载过程 .....	16
图 3-2 Modem 启动流程 .....	18
图 3-3 工具下载模式启动过程 .....	19
图 3-4 fastboot 模式安全启动过程 .....	22
图 4-1 sprd_sign 工具使用说明 .....	28

Unisoc Confidential For kxdwww

# 表目录

表 2-1 平台芯片支持的安全算法 .....	5
表 2-2 BSP 签名方案镜像及密钥对间的映射关系 .....	6
表 2-3 镜像、密钥对及安全调试证书间的映射关系 .....	7
表 2-4 动态 ta 签名密钥及加密密钥 .....	7
表 2-5 私钥生成命令 .....	8
表 2-6 公钥生成命令 .....	8
表 2-7 Avb2.0 签名方案镜像及密钥对间的映射关系 .....	8
表 3-1 防回退版本号的配置说明 .....	23
表 3-2 SPL 双 slot 支持信息 .....	24
表 3-3 镜像版本号检查 .....	24
表 3-4 产线 efuse 分区写入内容 .....	26
表 4-1 签名参数说明 .....	29
表 4-2 签名算法选择和签名填充参数与适用芯片的对应关系 .....	29



# 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
- 可用性（Availability）：信息的使用者是合法的

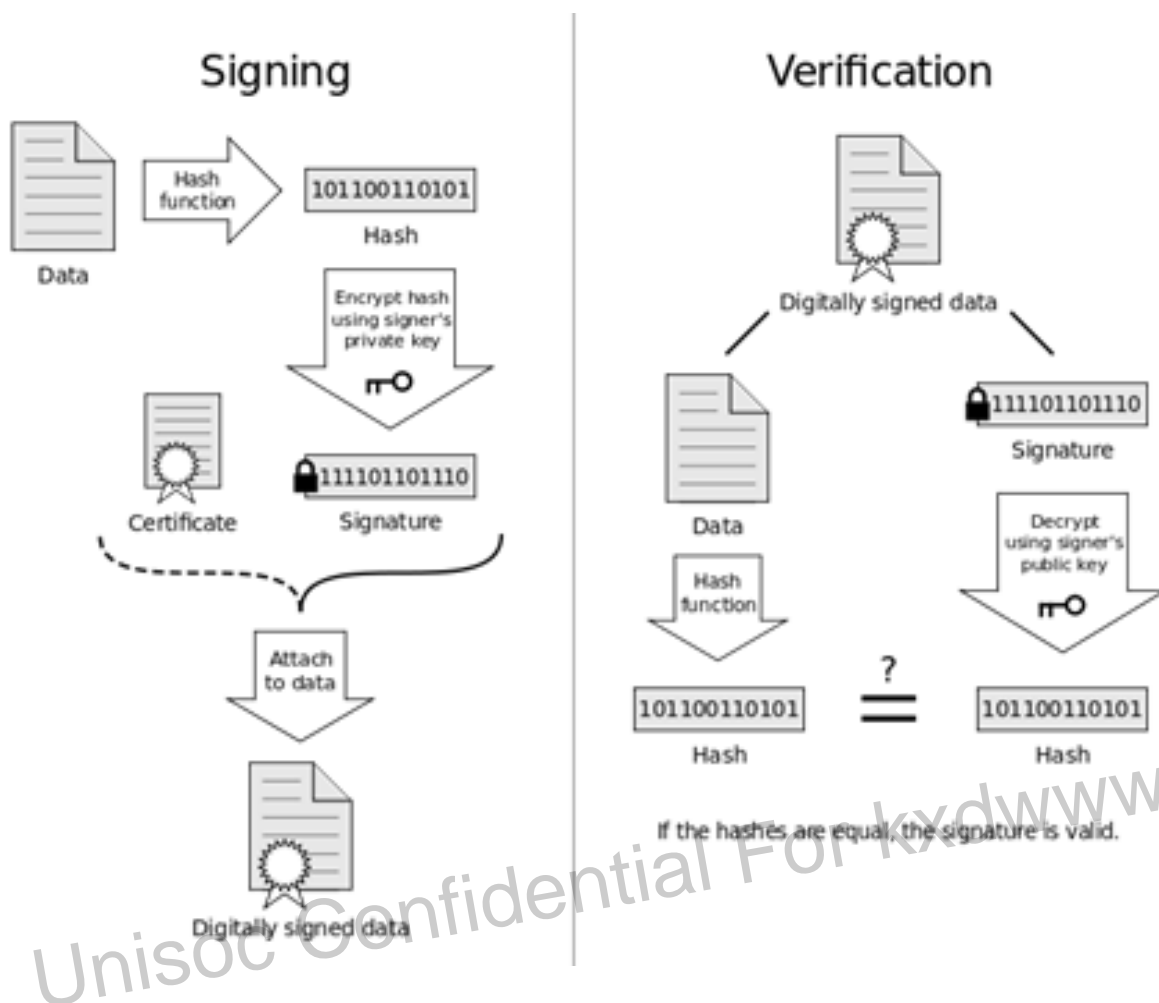
保密性、完整性和可用性统称为 CIA Triad。公钥密码解决保密性问题，数字签名解决完整性问题和可用性问题。

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



Secure Boot 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，设备在启动时对镜像进行签名验证。

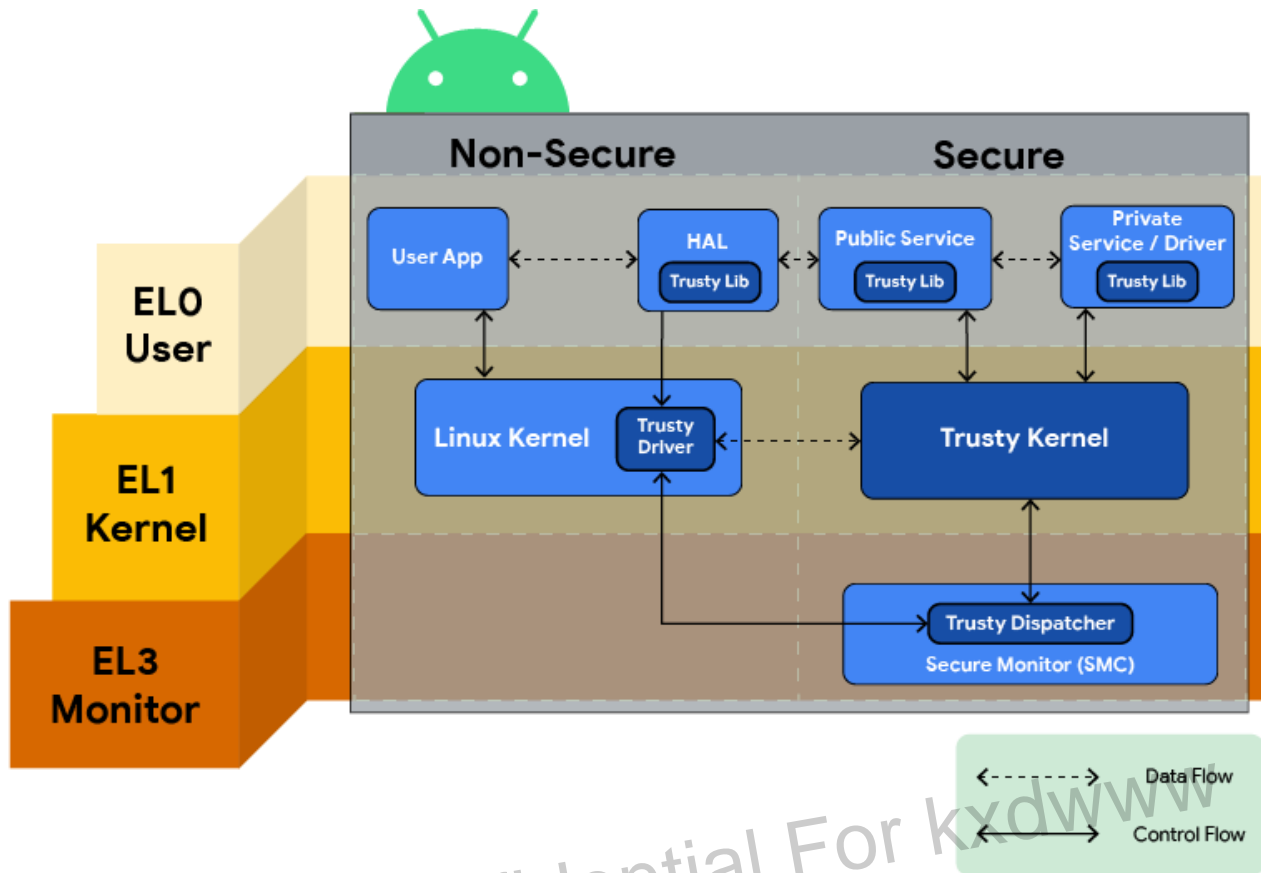
RSA 私钥是 Secure Boot 的保障，需要小心保存。

## 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：

- LK（Little Kernel，小内核）衍生的小型操作系统内核 TOS（Trusty Operating System，可信操作系统）。
- 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（安全任务/服务）通信。

# 2 Secure Boot 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 安全启动配置

安全启动默认开启。配置 BOARD\_SECBOOT\_CONFIG、BSP\_PRODUCT\_SECURE\_BOOT 和 BSP\_PRODUCT\_VBOOT 三个宏变量可开启或关闭安全启动。

以 UMS9230 为例，开启安全启动的参考配置如下。

- 在 device/sprd/qogirl6/ums9230\_1h10/product/ums9230\_1h10\_Natv/var.mk 中增加以下配置。  
`$(call md-set, BOARD_SECBOOT_CONFIG, true) //false for disable secureboot`
- 在 bsp/device/qogirl6/androidt/ums9230\_1h10/ums9230\_1h10\_base/common.cfg 板级配置中同步导出以下配置。

```
//Secure Boot
export BSP_PRODUCT_SECURE_BOOT="SPRD"//“NONE” for disable secureboot
export BSP_PRODUCT_VBOOT= “V2” //“” for disable secureboot
//firewall
```

### 说明

- 安全启动是否开启，取决于 BOARD\_SECBOOT\_CONFIG、BSP\_PRODUCT\_SECURE\_BOOT、BSP\_PRODUCT\_VBOOT 的配置，请根据需求配置。
- 配置文件路径仅为参考，请根据对应工程检索宏变量的实际位置进行修改。

## 2.2 镜像签名

按签名验签方式划分，安全启动镜像分为以下两类。

- 使用 BSP 签名方案的镜像  
包括引导阶段加载和启动的镜像、wcn、gnss 及 vdsp 镜像，这类镜像使 [2.4.1 BSP 签名方案](#) 进行签名及验证。引导阶段加载和启动的镜像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。引导阶段加载和启动的镜像如下：
  - fdl1.bin
  - lk-fdl2.bin
  - u-boot-spl-16k.bin
  - sml.bin
  - tos.bin

- lk.bin
- teecfg.bin
- 使用 Avb2.0 签名方案的镜像
 

镜像包括内核镜像、Android 系统镜像和 modem 系统镜像。这类镜像使用 [2.4.2 Avb2.0 签名方案](#) 进行签名及验签，LK 启动后会逐步加载和校验这类镜像。

  - 仅读取一次的小分区（如 boot、dtbo、recovery 和 modem bins），通常将整个分区的内容用于计算哈希并签名。
  - 较大分区（如 system、vendor、product、socko 及 odmko 等分区）使用哈希树方式签名。
  - 在加载镜像到内存的过程中验证流程会持续进行。

平台 BSP 签名方案和 Avb2.0 签名方案使用的安全算法与平台芯片间的对应关系如[表 2-1](#) 所示。

表2-1 平台芯片支持的安全算法

芯片名称	BSP 签名方案支持的安全算法	Avb2.0 签名方案支持的安全算法
<ul style="list-style-type: none"> <li>SC9832E</li> <li>SC7731E</li> <li>SC9863A</li> <li>UMS512(T)</li> <li>UMS312</li> </ul>	sha256 & rsa2048	sha256 & rsa4096
<ul style="list-style-type: none"> <li>UMS9230(T)</li> <li>UMS9620/UIS7870/UMS9157</li> <li>UMS9621/UMS9158</li> </ul>	<ul style="list-style-type: none"> <li>sha256 &amp; rsa4096（默认）</li> <li>sha256 &amp; ecc384（可选）</li> <li>sha256 &amp; rsa3072（可选）</li> </ul>	

## 2.3 签名密钥对

系统镜像进行签名及验签时需要相应的签名密钥对。

签名密钥对存放路径：bsp/tools/secureboot\_key/config/。

### 注意

基线版本的 bsp/tools 路径下没有 secureboot\_key 目录和 secureboot\_key/config 子目录，用户需自行创建。

### 2.3.1 BSP 签名密钥对

BSP 签名方案的密钥生成算法有以下四种：

- rsa2048
- rsa3072
- rsa4096
- ecc384

使用 BSP 签名方案的镜像与密钥对之间的映射关系参见[表 2-2](#)、[表 2-3](#)、[表 2-4](#)。

BSP 密钥对命名一般与密钥生成算法的名称相关。平台默认使用 rsa2048 或者 rsa4096 密钥生成算法，默认使用的 BSP 签名密钥对名称及生成命令参见表 2-2、表 2-3。

表 2-2 BSP 签名方案镜像及密钥对的映射关系

镜像名称	密钥对名称	生成命令
u-boot-spl-16k.bin、fdl1.bin	rsa4096_0.pem rsa4096_0_pub.pem	openssl genrsa -out rsa4096_0.pem 4096 openssl rsa -in rsa4096_0.pem -pubout -out rsa4096_0_pub.pem
	rsa3072_0.pem rsa3072_0_pub.pem	openssl genrsa -out rsa3072_0.pem 3072 openssl rsa -in rsa3072_0.pem -pubout -out rsa3072_0_pub.pem
	rsa2048_0.pem rsa2048_0_pub.pem	openssl genrsa -out rsa2048_0.pem 2048 openssl rsa -in rsa2048_0.pem -pubout -out rsa2048_0_pub.pem
	ecc384_0.pem ecc384_0_pub.pem	openssl ecparam -out ecc384_0.pem -name secp384r1 -genkey openssl ec -in ecc384_0.pem -pubout -out ecc384_0_pub.pem
lk.bin、lk-fdl2.bin、teecfg.bin、tos.bin、sml.bin	rsa4096_1.pem rsa4096_1_pub.pem	openssl genrsa -out rsa4096_1.pem 4096 openssl rsa -in rsa4096_1.pem -pubout -out rsa4096_1_pub.pem
	rsa3072_1.pem rsa3072_1_pub.pem	openssl genrsa -out rsa3072_1.pem 3072 openssl rsa -in rsa3072_1.pem -pubout -out rsa3072_1_pub.pem
	rsa2048_1.pem rsa2048_1_pub.pem	openssl genrsa -out rsa2048_1.pem 2048 openssl rsa -in rsa2048_1.pem -pubout -out rsa2048_1_pub.pem
	ecc384_1.pem ecc384_1_pub.pem	openssl ecparam -out ecc384_1.pem -name secp384r1 -genkey openssl ec -in ecc384_1.pem -pubout -out ecc384_1_pub.pem
wcn	wcn_2048_private.pem wcn_2048_public.pem	openssl genrsa -out wcn_2048_private.pem 2048 openssl rsa -in wcn_2048_private.pem -pubout -out wcn_2048_public.pem
	wcn_4096_private.pem wcn_4096_public.pem	openssl genrsa -out wcn_4096_private.pem 4096 openssl rsa -in wcn_4096_private.pem -pubout -out wcn_4096_public.pem
gps	gps_2048_private.pem gps_2048_public.pem	openssl genrsa -out gps_2048_private.pem 2048 openssl rsa -in gps_2048_private.pem -pubout -out gps_2048_public.pem
	gps_4096_private.pem gps_4096_public.pem	openssl genrsa -out gps_4096_private.pem 4096 openssl rsa -in gps_4096_private.pem -pubout -out gps_4096_public.pem
vdsp	rsa2048_vdsp.pem rsa2048_vdsp_pub.pem	openssl genrsa -out rsa2048_vdsp.pem 2048 openssl rsa -in rsa2048_vdsp.pem -pubout -out rsa2048_vdsp_pub.pem
	rsa4096_vdsp.pem rsa4096_vdsp_pub.pem	openssl genrsa -out rsa4096_vdsp.pem 4096 openssl rsa -in rsa4096_vdsp.pem -pubout -out rsa4096_vdsp_pub.pem

## 说明

生成 u-boot-spl-16k.bin、fdl1.bin、lk.bin、lk-fdl2.bin、teecfg.bin、tos.bin、sml.bin 等镜像的签名密钥对如果需使用 ecc384、rsa3072 算法，请联系展锐 FAE 协助修改。

表2-3 镜像、密钥对及安全调试证书间的映射关系

镜像名称	密钥对名称	生成命令	安全调试证书
u-boot-spl-16k-sign.bin、 fdl1-sign.bin	rsa4096_1.pem rsa4096_1_pub.pem	复用表 2-2 生成的密钥	主证书 (Primary cert)
	rsa3072_1.pem rsa3072_1_pub.pem		
	rsa2048_1.pem rsa2048_1_pub.pem		
	ecc384_1.pem ecc384_1_pub.pem		
	rsa4096_devkey.pem rsa4096_devkey_pub.pem	openssl genrsa -out rsa4096_devkey.pem 4096 openssl rsa -in rsa4096_devkey.pem -pubout -out rsa4096_devkey_pub.pem	副证书 (Developer cert)
	rsa3072_devkey.pem rsa3072_devkey_pub.pem	openssl genrsa -out rsa3072_devkey.pem 3072 openssl rsa -in rsa3072_devkey.pem -pubout -out rsa3072_devkey_pub.pem	
	rsa2048_devkey.pem rsa2048_devkey_pub.pem	openssl genrsa -out rsa2048_devkey.pem 2048 openssl rsa -in rsa2048_devkey.pem -pubout -out rsa2048_devkey_pub.pem	
	ecc384_devkey.pem ecc384_devkey_pub.pem	openssl ecparam -out ecc384_devkey.pem -name secp384r1 -genkey openssl ec -in ecc384_devkey.pem -pubout -out ecc384_devkey_pub.pem	

## 说明

- 生成安全调试证书（主证书和副证书）前，需提供相应的密钥对。只需生成副证书需要的 devkey 密钥对，主证书需要的密钥对则复用 lk.bin、lk-fdl2.bin、teecfg.bin、tos.bin、sml.bin 等镜像签名使用的密钥对。
- 表 2-2、表 2-3 包含了 bsp 签名所有需要的密钥对，编译前请按照表中命令生成密钥对。

表2-4 动态 ta 签名密钥及加密密钥

镜像名称	签名密钥名称	签名密钥生成命令	加密密钥名称
dynamic ta	dynamic_ta_privatekey.pem	openssl genrsa -out dynamic_ta_privatekey.pem 2048	aeskey

## 说明

平台默认 ta 只签名不加密，一般不需要生成 aeskey。

私钥生成命令如表 2-5 所示，公钥生成命令如表 2-6 所示。参见表 2-5、表 2-6 中命令，替换密钥名称可生成表 2-2、表 2-3、表 2-4 中的所有密钥。替换密钥名称时需要注意公钥和私钥名称的对应关系。

表2-5 私钥生成命令

密钥生成算法	命令示例
rsa2048	\$ openssl genrsa -out rsa2048_0.pem 2048
rsa3072	\$ openssl genrsa -out rsa3072_0.pem 3072
rsa4096	\$ openssl genrsa -out rsa4096_0.pem 4096
ecc384	\$ openssl ecparam -out ecc384_0.pem -name secp384r1 -genkey

表2-6 公钥生成命令

密钥生成算法	命令示例
rsa2048	\$ openssl rsa -in rsa2048_0.pem -pubout -out rsa2048_0_pub.pem
rsa3072	\$ openssl rsa -in rsa3072_0.pem -pubout -out rsa3072_0_pub.pem
rsa4096	\$ openssl rsa -in rsa4096_0.pem -pubout -out rsa4096_0_pub.pem
ecc384	\$ openssl ec -in ecc384_0.pem -pubout -out ecc384_0_pub.pem

## 2.3.2 Avb2.0 签名密钥对

使用 Avb2.0 签名方案的镜像及签名密钥对之间的映射关系参见表 2-7。平台 Avb2.0 签名方案默认使用的算法为 rsa4096-sha256。

表2-7 Avb2.0 签名方案镜像及密钥对间的映射关系

镜像名称	密钥对名称	生成命令
boot.img、dtbo.img	rsa4096_boot.pem rsa4096_boot_pub.bin	./genkey.sh boot
vbmeta.img	rsa4096_vbmeta.pem rsa4096_vbmeta_pub.bin	./genkey.sh vbmeta
vbmeta_system.img	rsa4096_system.pem rsa4096_system_pub.bin	./genkey.sh system
vbmeta_system_ext.img	rsa4096_system_ext.pem rsa4096_system_ext_pub.bin	./genkey.sh system_ext



镜像名称	密钥对名称	生成命令
vbmeta_vendor.img	rsa4096_vendor.pem rsa4096_vendor_pub.bin	./genkey.sh vendor
vbmeta_product.img	rsa4096_product.pem rsa4096_product_pub.bin	./genkey.sh product
vbmeta_odm.img	rsa4096_odm.pem rsa4096_odm_pub.bin	./genkey.sh odm
socko.img	rsa4096_socko.pem rsa4096_socko_pub.bin	./genkey.sh socko
recovery.img	rsa4096_recovery.pem rsa4096_recovery_pub.bin	./genkey.sh recovery
odmko.img	rsa4096_odmko.pem rsa4096_odmko_pub.bin	./genkey.sh odmko
l_modem、nr_modem、 l_ldsp、l_gdsp、l_agdsp、 l_cdsp、pn_sys	rsa4096_modem.pem rsa4096_modem_pub.bin	./genkey.sh modem
vendor_boot.img	rsa4096_vendor_boot.pem rsa4096_vendor_boot_pub.bin	./genkey.sh vendor_boot
init_boot.img	rsa4096_init_boot.pem rsa4096_init_boot_pub.bin	./genkey.sh init_boot

## 说明

表 2-7 列出了需要 Avb 签名的所有可能镜像，实际的镜像类型和数量取决于具体芯片类型与工程。请在编译之前生成这些密钥对，避免因缺少密钥对而导致编译报错。

在 bsp/tools/secureboot\_key/config/路径，执行./genkey.sh xxx 脚本命令生成 xxx（分区名）分区密钥对。以 oem 分区为例，生成 oem 分区密钥对时脚本命令执行过程如下。

```
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
```

生成其他分区密钥对时，替换其他分区名再执行 genkey.sh 脚本命令即可。genkey.sh 脚本内容详见 [5.1 genkey.sh 脚本](#)。

## 2.4 签名方案

### 2.4.1 BSP 签名方案

BSP 签名方案支持四种签名算法：rsa2048、rsa4096、rsa3072 和 ecc384。修改 BSP 与 Android 下的相关配置文件选择其中一种算法，BSP 与 Android 下签名算法的选择须保持一致。以 UMS9230 为例，其 BSP 配置文件路径和 Android 配置文件路径如下。

- BSP 配置文件路径：  
sprdroidr\_trunk/bsp/device/qogirl6/androidt/ums9230\_1h10/ums9230\_1h10\_base/common.cfg
- Android 配置文件路径：  
sprdroidr\_trunk/device/sprd/mpool/module/security/msoc/qogirl6/qogirl6.mk

选择签名算法的配置如下：

- 选择 rsa4096 签名算法  
SPRD\_SECUREBOOT\_ALGORITHM := rsa4096
- 选择 rsa3072 签名算法  
SPRD\_SECUREBOOT\_ALGORITHM := rsa3072
- 选择 ecc384 签名算法  
SPRD\_SECUREBOOT\_ALGORITHM := ecc384

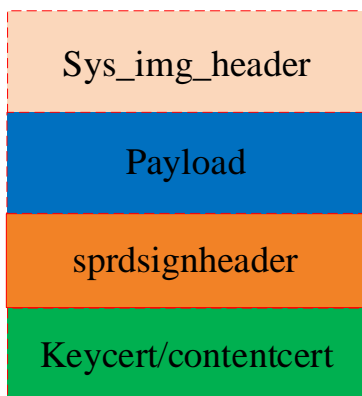
chipram、bootloader 及 trusty 等目标在编译时自动完成 BSP 签名。如下所示，签名脚本将在目标编译成功后触发。

```
if [[ `echo "$signed_images" grep -w $cmd` ]]; then
    build tool and sign images "$cmd"
    if [[ $? -ne 0 ]]; then
        return 1
    fi
fi

if [[ $is_packing -eq 0 ]]; then
    is_packing=1
    . $BSP_SIGN_DIR/packimage.sh "$@"
```

使用 BSP 签名方案签名后可信镜像的格式如图 2-1 所示。

图2-1 BSP 签名后可信镜像的格式



BSP 签名方案使用的签名工具介绍参见 [4 sprd\\_sign 签名工具](#)。

## 2.4.2 Avb2.0 签名方案

使用 Avb2.0 签名方案签名的镜像文件：

- dtbo
- boot
- recovery
- system
- system\_ext
- vendor
- vendor.boot
- odm
- product
- vendor\_dlkm
- socko
- odmko
- modem bins

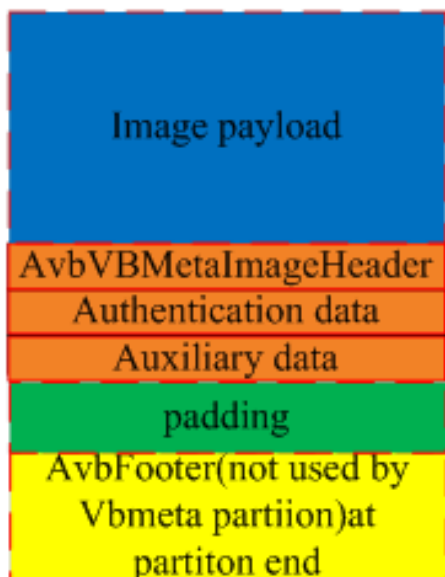
以 UMS9230 boot 镜像为例，device/sprd/mpool/module/security/msoc/qogirl6 文件中其密钥配置信息如下。

```
#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 boot 镜像签名使用的私钥名称
- BOARD\_AVB\_BOOT\_ALGORITHM 定义 boot 镜像签名使用的算法名称

使用 Avb2.0 签名方案完成签名后的镜像格式如图 2-2 所示，镜像头部没有校验用的元数据，镜像尾部 64 字节的 AvbFooter 用于定位校验数据的位置。

图2-2 Avb2.0 签名后镜像的格式



按上述设计，super 分区中 system、vendor 和 product 分区的 footer 将无法访问，目前的解决方案是将校验 system、vendor 和 product 分区的元数据复制到独立的 vbmeta\_system 和 vbmeta\_vendor 等分区。

分区之间复制元数据时的对应关系：

- system 分区，system\_ext 分区的元数据复制到 vbmeta\_system 分区
- product 分区的元数据复制到 vbmeta\_product 分区
- vendor 的元数据保存到 vbmeta\_vendor 分区
- vendor\_dlkm 分区和 system\_dlkm 分区的元数据复制到 vbmeta\_system\_ext 分区
- odm 分区的元数据复制到 vbmeta\_odm 分区

#### 注意

- super 分区中动态分区的元数据一定要和 vbmeta\_system 和 vbmeta\_vendor 等分区的元数据保持一致。
- system\_dlkm 分区为 kernel5.15 上的新增分区，若项目搭载 kernel5.4，vbmeta\_system\_ext 分区仅保存 vendor\_dlkm 分区元数据。

确认 super 分区中动态分区的元数据与 vbmeta\_system 和 vbmeta\_vendor 中元数据是否一致的过程：

步骤 1 将 super 分区镜像文件从 sparse 格式转化为 raw data 格式。

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

步骤 2 从 super 分区镜像中提取 system、product 和 vendor 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/
```

提取完成后在 unpack 目录得到 super 分区中相关动态分区的镜像。

步骤 3 使用 avbtool 确认步骤 2 中提取的 system 和 product 动态分区的元数据与 vbmeta\_system 分区的

元数据是否一致，vendor 动态分区的元数据与 vbmeta\_vendor 分区的元数据是否一致。

获取分区元数据信息的 avbtool 命令及输出如下：

```
$ ./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'
  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。

----结束

## 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，为保证 OTA 包中的分区镜像版本和将来从 PRODUCT\_OUT 目录打包到 pac 包中的分区镜像版本保持一致，需要将 PRODUCT\_OUT 目录下的 system、vendor、product、vbmeta\_system 和 vbmeta\_vendor 等镜像替换为 OTA 编译生成的新镜像。

## 替换 vbmeta\_system 和 vbmeta\_vendor 镜像

使用脚本 vendor/sprd/build/tasks/sprdbuildota.mk 中的以下脚本，完成 vbmeta\_system 和 vbmeta\_vendor 镜像的替换。

```
$(hide) -$(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
```

## 替换 system、vendor 和 product 镜像

system、vendor 和 product 镜像是通过使用 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/s9863a1h10/obj/PACKAGING/target_files_intermediates/s9863a1h10_Natv-  
target_files-eng.wenquan.zhang/IMAGES/system.img --partition vendor:readonly:405712896: group_unisoc --  
image vendor=out/target/product/s9863a1h10/obj/PACKAGING/target_files_intermediates/s9863a1h10_Natv-  
target_files-eng.wenquan.zhang/IMAGES/vendor.img --partition product:readonly:454574080:group_unisoc --  
image product=out/target/product/s9863a1h10/obj/PACKAGING/target_f  
iles_intermediates/s9863a1h10_Natv-target_files-eng.wenquan.zhang/IMAGES/product.img --sparse --output  
out/target/product/s9863a1h10/super.img"
```

OTA 包编译完成后，PRODUCT\_OUT 目录的 system、vendor 和 product 分区的内容和 super.img 提取出来的 system、vendor 和 product 镜像是否一致，只需确认 super.img 提取出来的 system、vendor 和 product 镜像的元数据和 PRODUCT\_OUT 目录下的 vbmeta\_system 及 vbmeta\_vendor 镜像的元数据是否一致。

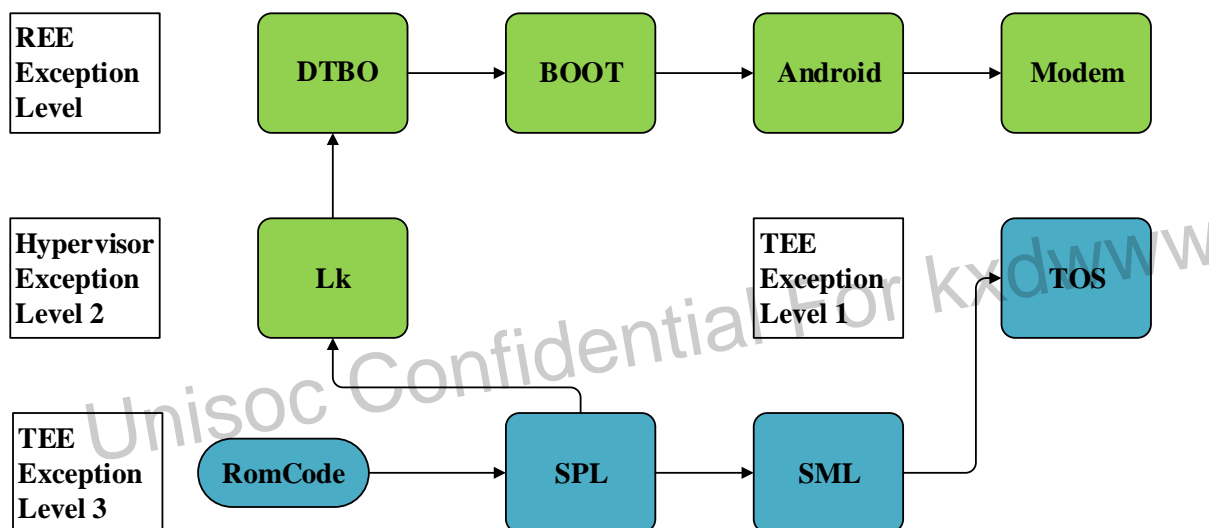
# 3 Secure Boot 启动过程

## 3.1 AP 安全启动过程

### 3.1.1 启动流程概述

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 Android 系统启动完毕，如图 3-1 所示。

图3-1 安全启动镜像加载过程



- 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（\*\_0\_pub.bin）的哈希值。
- RomCode 作为信任根的一部分，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，并计算其哈希值和 ROTPK 记录的公钥哈希值比较，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 Secure Boot Loader 运行在 Secure IRAM，初始化 DRAM，加载 SML (Arm® Trusted firmware)、TOS (Trusty OS) 和 LK (REE bootloader) 到 DRAM，校验这三个镜像的合法性；校验通过将执行 SML。
- SML 运行 TOS，TOS 运行内置的 TA，然后 TOS 返回到 SML，SML 返回到 LK 执行。
- LK 运行在 Hypervisor 模式，负责检查设备状态，加载并校验 DTBO、BOOT 及 Recovery 镜像的内容，并准备验证启动 Android 系统的内核参数。
- LK 引导内核启动后，内核将负责验证 Android 系统分区，验证通过后将其挂载为只读的系统分区。
- Android 启动后，modem\_control 服务负责校验和加载 Modem 子系统的固件。



### 3.1.2 启动过程中设备状态

芯片平台在启动过程中增加了传达启动时的验证状态逻辑，主要包含：已解锁设备的状态显示逻辑、dm-verity 损坏时显示逻辑、找不到有效操作系统时的显示逻辑。该部分的详细描述参见：

<https://source.android.com/docs/security/verifiedboot/boot-flow>。

- 已解锁设备的状态显示逻辑

如果设备处于已解锁状态，则会在每次启动时显示“INFO:LOCK FLAG IS UNLOCK!!!”。10 秒钟后，设备会继续启动。如果用户按下电源按钮，“INFO: press power button to pause.” 文字便会更改为“INFO: press power button to continue.”，且屏幕保持常亮状态。直到再次按下按钮，手机才会继续启动。

- dm-verity 损坏时显示逻辑

如果找到有效的 Android 版本，且设备当前处于 EIO 模式，则会显示“you device is corrupt, It can't be trusted and may not work properly”。用户需要按电源按钮才能继续。如果用户未在 30 秒内确认警告屏幕，设备将关机。

- 找不到有效操作系统时的显示逻辑

如果找不到有效的 Android 版本，屏幕会显示“WARNING: NO VALID OS FOUND!!!”。设备无法继续启动。如果用户未在 30 秒内确认警告屏幕，设备将关机。

#### 说明

芯片平台简化了屏幕显示，在开机过程中，紫光展锐界面左上角以字符串的形式显示，厂商可根据需求定义显示的 Logo。Logo 定义详情参见 <https://source.android.com/docs/security/verifiedboot/boot-flow>。

## 3.2 Modem 类子系统的安全启动

### 3.2.1 modem 启动介绍

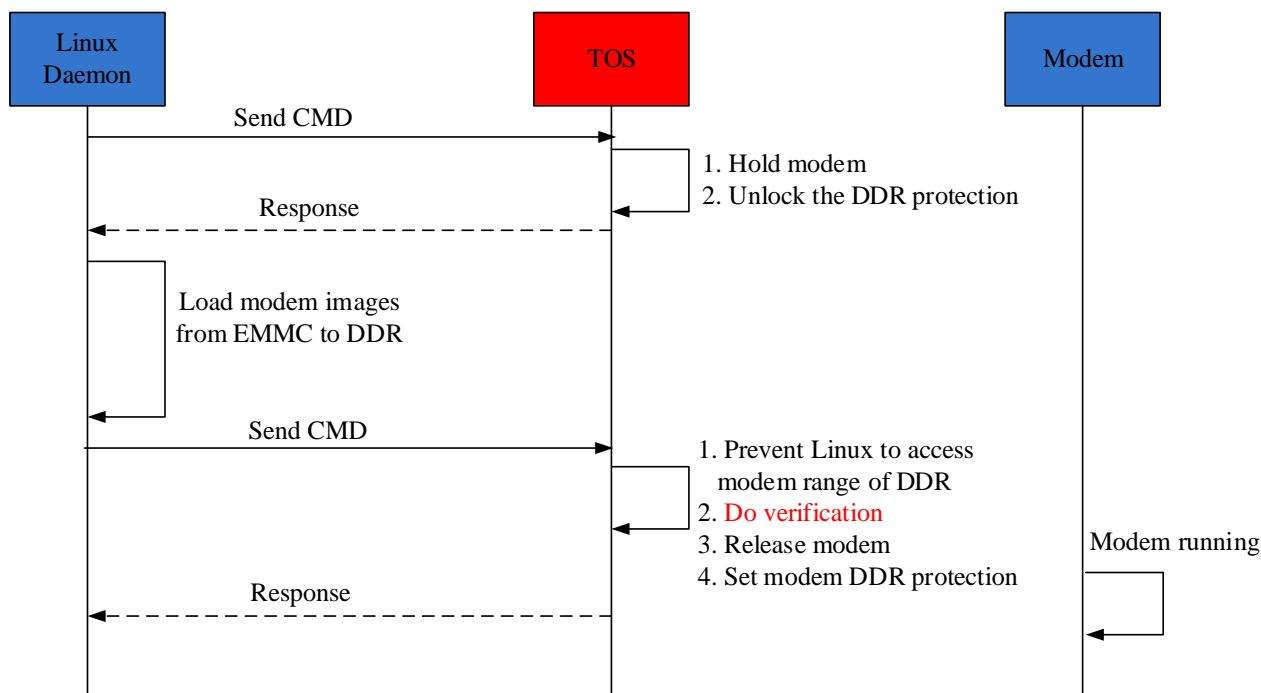
Modem、Audio、VDSP、GPU Firmware、SCP 等不含 ROM 的子系统，通常需要通过主控配合来完成子系统的安全启动流程。

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



### 3.2.2 wcn 和 gnss 启动介绍

wcn 和 gnss 镜像签名使用 [2.4.1 BSP 签名方案](#)，使用 [2.3.1 BSP 签名密钥对](#)，签名算法为 rsa4096，签名填充（padding）为 RSA\_PKCS1\_PSS\_PADDING（参见[表 4-1](#)）。

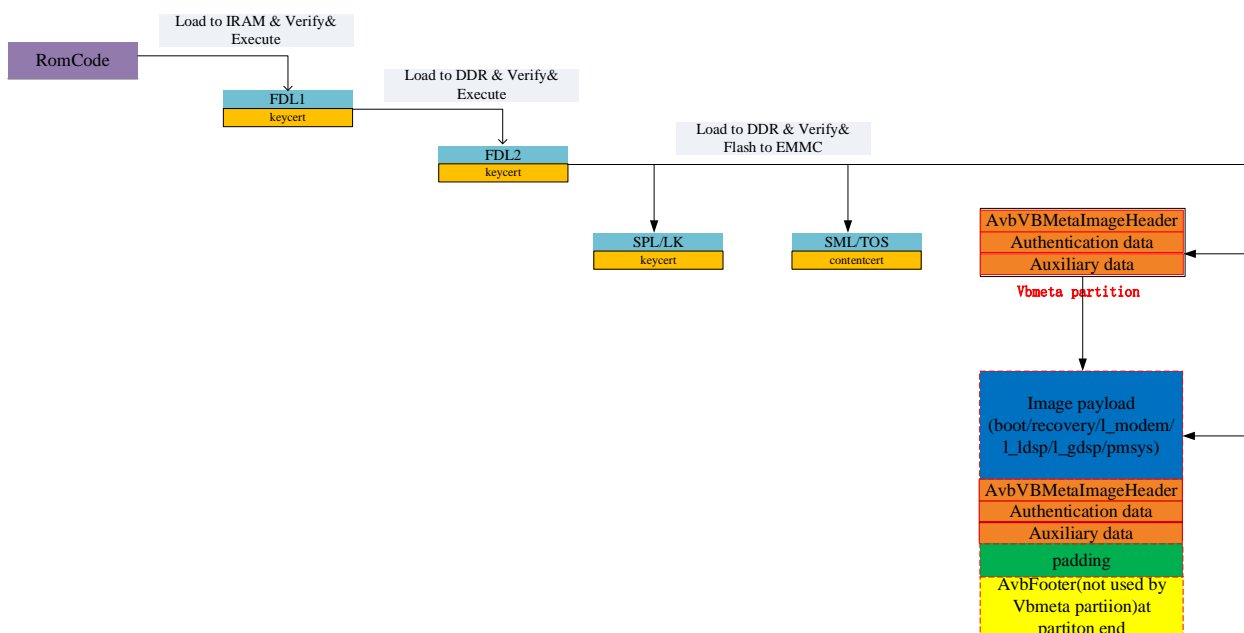
支持 rsa4096 算法的芯片都支持 wcn 镜像和 gnss 镜像的安全校验。

## 3.3 安全更新

### 3.3.1 工具下载

下载模式在 fd11.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 fd11.bin 的数字签名，通过后 fd11 初始化 DRAM 并加载和校验 lk-fd12.bin；lk-fd12 负责和 PC 端交互，接收和校验 PC 端下载的数据，并在验证通过后将更新到 eMMC/NAND 等静态存储。

图3-3 工具下载模式启动过程



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（即 vbmeta\_\*）。

### 3.3.2 fastboot 烧录

fastboot 模式烧录系统分区时，设备应处于解锁状态。

#### 设备锁状态

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。

使用如下命令更改设备状态。

```
[fastboot flashing unlock_bootloader signature.bin] //unlock
```

```
[ fastboot flashing lock_bootloader] //lock
```

设备状态发生变化，都会先擦除数据分区中的数据（userdata 分区和 metadata 分区）。为保护用户数据，删除数据前会需要用户确认。

解锁设备：

步骤 1 获取设备 Product SN 序列号。

```
adb reboot bootloader
fastboot oem get_identifier_token
//以下是执行结果
(bootloader) Identifier token:
```

```
(bootloader) 30313233343536373839414243444546
343339
OKAY [0.077s]
Finished. Total time: 0.084s
```

## 步骤 2 生成解锁凭证。

Linux 环境下使用完整 SN 序列号生成证书（certificate.bin）。

- 生成证书的脚本路径如下：  
vendor/sprd/tools/packimage\_scripts/signidentifier\_unlockbootloader.sh
- 证书需要使用签名的密钥及路径如下：
  - 密钥：rsa4096\_vbmeta.pem
  - 路径：bsp/tools/secureboot\_key/config/

生成证书的命令如下：

```
./signidentifier_unlockbootloader.sh
30313233343536373839414243444546343339 ../../../../bsp/tools/secureboot_key/config/rsa4096_vbmeta
.pem certificate.bin
```

## 步骤 3 PC 上执行如下的解锁指令，需要在设备侧按音量下键确认。

```
./fastboot flashing unlock_bootloader certificate.bin
Sending 'unlock_message' (0 KB) OKAY [0.071s]
unlocking bootloader OKAY [59.165s]
```

## ----结束

## 锁定设备：

Fastboot 模式通过以下方式锁定设备，设备锁定后将开启验证启动。

```
./fastboot flashing lock
OKAY [0.089s]
Finished. Total time: 0.094s
```

判断设备锁状态的两种方式：

- 系统启动过程处于 bootloader 阶段时机器左上角显示如下提示，即为 UNLOCK，否则为 LOCK 状态。

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

- 内核启动参数中也会指示设备状态，如下为 UNLOCK，否则为 LOCK 状态。

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

## 分区烧录

物理分区烧录操作如下：

### 步骤 1 设备解锁，锁状态为 UNLOCK。

### 步骤 2 进入 fastboot 模式。

```
$adb reboot bootloader
```

步骤 3 烧录镜像到物理分区。

```
$fastboot flash [分区名] [镜像名] //eg: fastboot flash boot_a boot.img
```

----结束

逻辑分区烧录操作如下：

fastbootd 模式更新 super 中的 system、vendor 和 product 等镜像。

步骤 1 设备解锁，锁状态为 UNLOCK。

步骤 2 进入 fastbootd 模式。

```
$adb reboot fastboot
```

步骤 3 烧录镜像到逻辑分区。

```
$fastboot flash [分区名] [镜像名] //eg: fastboot flash system_a system.img
```

----结束

## remount 功能

执行 adb remount 可以将只读分区重新挂载为可读写模式，方便研发进行替换测试。这里只提供 remount 的方法。remount 前提是设备锁状态为 UNLOCK。

```
$adb root
```

```
$adb disable-verity
```

```
$adb reboot
```

```
$adb wait-for-device
```

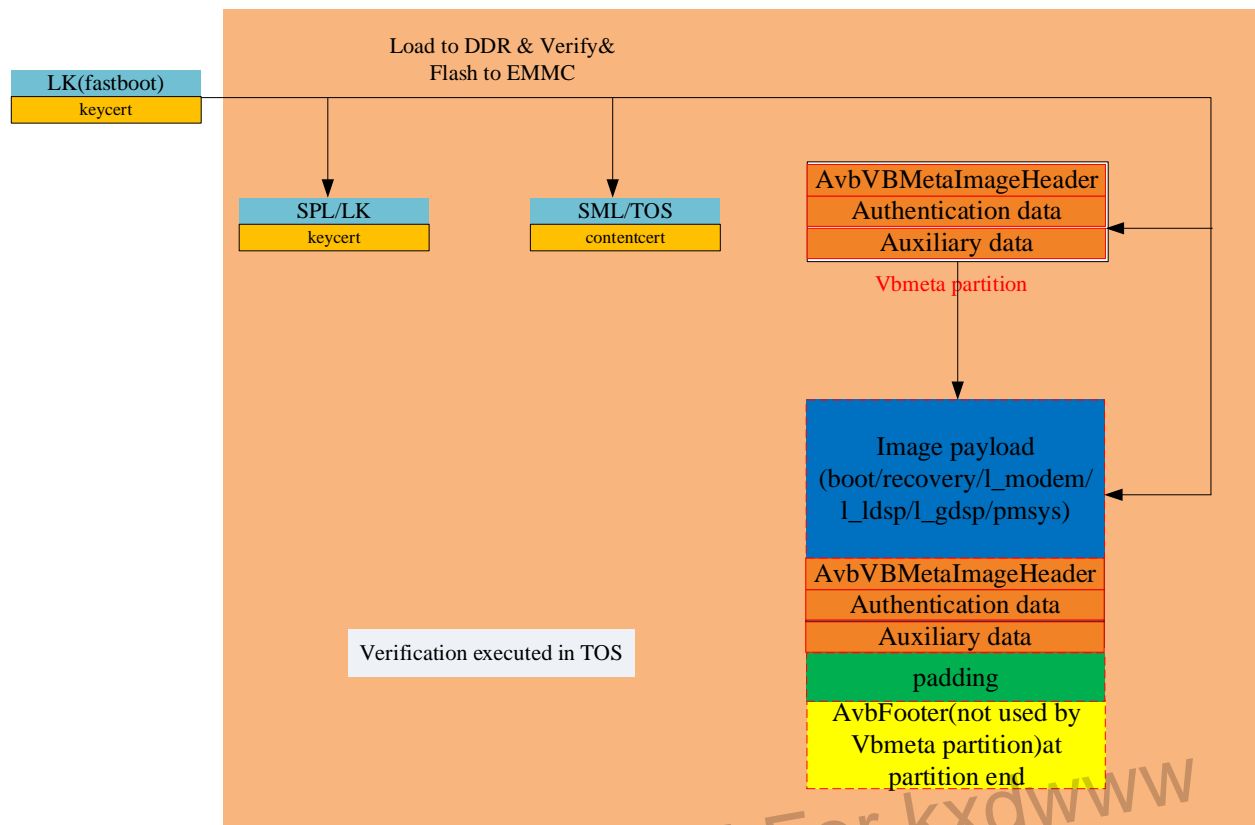
```
$adb root
```

```
$adb remount
```

## fastboot 模式启动介绍

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，更具安全性。

图3-4 fastboot 模式安全启动过程



### 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。

### 3.4.1 版本号配置

展锐支持对每一个分区进行防版本回退的配置，防回退的版本号在 bsp/tools/secureboot\_key/config/version.cfg 文件中配置。version.cfg 文件内容参见 [5.2 version.cfg 文件](#)。

version.cfg 文件中各个配置说明见 [表 3-1](#)。

表3-1 防回退版本号的配置说明

配置项	配置值	保存区域	相关镜像
trusted_version	[0-32]或[0-64]	efuse	u-boot-spl-16k.bin、fdl1.bin、lk.bin、lk-fdl2.bin、teecfg.bin、tos.bin、sml.bin
avb_version_vbmeta	uint32_t 类型	rpmb	vbmeta-sign.img
avb_version_wcn			wcn 镜像
avb_version_gps			gps 镜像
avb_version_boot			boot.img、dtbo.img
avb_version_system			vbmeta_system.img、system.img
avb_version_system_ext			vbmeta_system_ext.img、system_ext.img
avb_version_vendor			vbmeta_vendor.img、vendor.img
avb_version_product			vbmeta_product.img、product.img
avb_version_socko			socko.img
avb_version_odmko			odmko.img
avb_version_modem			cp 侧镜像
avb_version_vendor_boot			vendor_boot.img
avb_version_odm			odm.img
avb_version_init_boot			init_boot.img

## 说明

- 防回退版本的配置需要验证，请研发调试阶段配置后做好充足的验证，包括且不限于：1、回刷低版本测试，2、ota 掉电回滚测试等。
- 将 **5 genkey.sh** 和 **version.cfg** 中的文件同步到 bsp/tools/secureboot\_key/config/路径。
- 表 3-1** 包括所有固件的配置，有些固件部分芯片不支持，允许 version.cfg 配置上所有的配置项。
- trusted\_version 取值范围，目前仅 UMS9620 支持[0,64]，其他都是[0,32]。

防版本回退会编译到安全启动的元数据中。在安全下载及启动过程中，系统会在各个下载及启动阶段比较启动的当前系统镜像版本号是否大于等于记录在 efuse（trusty firmware）和 rpmb 分区的版本号。版本号满足大于等于条件，才允许启动，在系统完全启动后，如果是新的版本号，则更新到一次性可写区域。

## 3.4.2 版本号更新策略

- 使用 v-ab 分区方案时
  - rpmb 空间划分为 a/b 两块区域储存版本号，rpmb 储存的版本号在第一次开机升级对应区域（a/b）的版本号。

- 在不支持 SPL 双 slot 的芯片中，efuse 只有一块区域储存版本号，efuse 储存的版本号在第二次开机时升级。在支持 SPL 双 slot 的芯片中（如 UMS9621），efuse 有 a/b 两块区域用于存储版本号，在第一次开机流程中更新对应区域（a/b）的版本号。

平台芯片对 SPL 双 slot 是否支持情况如所表 3-2 示。

表3-2 SPL 双 slot 支持信息

芯片名称	是否支持 SPL 双 slot
SC9832E、SC7731E、SC9863A、UMS512、UMS312、UMS9230、UMS9620、UIS7870	不支持
UMS9621、UMS9158	支持

- 未使用 v-ab 分区方案时

第一次开机就升级版本号。

在下载阶段和启动阶段的镜像版本号检查情况参见表 3-3。

表3-3 镜像版本号检查

镜像名称	下载阶段	启动阶段
u-boot-spl-16k.bin、fdl1.bin、lk.bin、lk-fdl2.bin、teecfg.bin、tos.bin、sml.bin	检查	检查
vbmeta-sign.img	检查	检查
wcn 镜像、gps 镜像	非物理分区，不检查	检查（未签名则不检查）
boot.img、vendor_boot.bin	不检查	检查（设备 lock 状态）
		不检查（设备 unlock 状态）
dtbo.img	检查	检查
init_boot	检查	检查
vbmeta_system.img、vbmeta_odm.img、vbmeta_system_ext.img、vbmeta_vendor.img、vbmeta_product.img	检查	检查（设备 lock 状态）
		不检查（设备 unlock 状态）
socko.img、odmko.img	不检查	检查（设备 lock 状态）
		不检查（设备 unlock 状态）
cp 侧镜像	检查	检查



## 3.5 安全调试

安全调试需要制作调试证书，使用 fdl1-sign.bin 和 u-boot-spl-16k-emmc-sign.bin（emmc 类型的 spl）两个签名镜像并按以下操作生成调试证书。

### 说明

- spl 分为 emmc 和 ufs 两种类型，制作调试证书时需要选择与手机硬件对应类型的 spl。
- 检查 out/host/linux-x86/bin 目录下是否已有 sprd\_sign 签名工具，若有可省略下述步骤 1。
- 以下使用 UMS9230 为例，若使用其他类型的芯片，--algorithm 与 --rsa\_padding 参数请参照表 4-1 修改。

步骤 1 编译生成签名工具 sprd\_sign。

具体生成过程作参见 4.1 生成签名工具。

步骤 2 将 fdl1-sign.bin 和 u-boot-spl-16k-emmc-sign.bin 复制到 \$ out/host/linux-x86/bin/ 目录。

步骤 3 获取 socid 号（每颗芯片都有唯一的 socid 号）。

fastboot 模式下获取 socid 的命令如下：

```
sudo./fastboot getvar socid
```

获取的 socid 示例：

socid is:

```
939ff32ca2d078bbc04a045bdfbac721
```

```
8fdb2603bd2adaaa3a6f4fa780d797f8
```

步骤 4 生成调试证书。

完整调试证书包括主证书与副证书，如果不更改 devkey，可以省略子步骤 a。

- 将 \*\_devkey\_pub.pem 和 \*\_devkey.pem 放入 bsp/tools/secureboot\_key/config/。
- 在 bsp/tools/secureboot\_key/config/ 目录执行以下命令：

```
./sprd_sign make_debug_certificate --image fdl1-sign.bin --soc_id  
0x939ff32ca2d078bbc04a045bdfbac7218fdb2603bd2adaaa3a6f4fa780d797f8 --mask  
0xFFFFFFFFFFFFFFFF --config_dir ../.././bsp/tools/secureboot_key/config/ --algorithm rsa4096  
--rsa_padding 6
```

```
./sprd_sign make_debug_certificate --image u-boot-spl-16k-emmc-sign.bin --soc_id  
0x939ff32ca2d078bbc04a045bdfbac7218fdb2603bd2adaaa3a6f4fa780d797f8 --mask  
0xFFFFFFFFFFFFFFFF --config_dir ../.././bsp/tools/secureboot_key/config/ --algorithm rsa4096  
--rsa_padding 6
```

生成的 fdl1-sign.bin，u-boot-spl-16k-emmc-sign.bin 替换到手机中即可。

make\_debug\_certificate 命令参数参见表 4-1。

### 说明

输入参数 mask 的具体配置，请咨询 FAE。

----结束

## 3.6 安全产线部署

安全启动相关数据在产线 efuse 分区中需写入的内容参见表 3-4。

表3-4 产线 efuse 分区写入内容

名称	用途	长度	位置	说明
ROTPK	Root of trust Public key Hash	256 bit	private efuse 区	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 bit / 64 bit	private efuse 区	存放第三方 Secure OS 厂家软件版本号。

### 说明

- 上述信息在 efuse 中位置必须与展锐约定一致。
- UMS9620 芯片 anti-rollback 的长度为 64 bit，其他芯片 anti-rollback 的长度为 32 bit。

Unisoc Confidential For kxdwww

# 4

## sprd\_sign 签名工具

本章介绍 BSP 签名方案使用的 sprd\_sign 签名工具。该签名工具对 AP 引导启动镜像进行签名，对 nasec 版本插入哈希头部信息以及制作 debug 证书等功能。

签名对象包括 chipram 编译生成的 spl 和 fdl1，bootloader 编译生成的 lk.bin 和 lk-fdl2，以及 sml、tos、teecfg、wcn 和 gnss 等镜像。签名功能嵌入到 Android/BSP 编译流程，在成功完成目标编译后触发签名操作。

### 4.1 生成签名工具

签名工具的源码路径：vendor/sprd/tools/packimage\_source。

生成 sprd\_sign 签名工具：

步骤 1 根目录下执行 source build/envsetup.sh。

步骤 2 执行 lunch 命令。

步骤 3 任意选择一个工程名中不包含“nosec”字样的工程。

步骤 4 进入 vendor/sprd/tools/packimage\_source，输入命令 mm，开始编译。

步骤 5 在 out/host/linux-x86/bin/目录下生成 sprd\_sign 签名工具，复制 sprd\_sign 签名工具到 bsp/tools/android/packimage\_scripts/路径下。

编译版本时会自动调用 bsp/tools/android/packimage\_scripts/路径下的签名工具进行签名。

----结束

### 4.2 工具使用说明

sprd\_sign 签名工具命令格式及参数说明如图 4-1 所示。

图4-1 sprd\_sign 工具使用说明

```
./sprd_sign (null)
no valid command for sprdsign please enter as follow:
{insert_image_header,sign_image,make_debug_certificate,verify_image,erase_signature}
-----samples-----
usage: signtool insert_image_header [--image IMAGE]
arguments:
  --image IMAGE          Image to insert image header

usage: signtool sign_image          [--image IMAGE]
                                [--config_dir PATH]
                                [--algorithm SIGNATURE_ALGORITHMS]
                                [--rsa_padding RSA_PADDING]
arguments:
  --image IMAGE          Image to insert image header
  --config_dir PATH      Path include key pairs and version config file
  --algorithm SIGNATURE_ALGORITHMS Signature algorithm [ecc384|rsa3072|rsa4096]
  --rsa_padding RSA_PADDING The padding argument must be one of the RSA_*_PADDING values.
                        1 RSA_PKCS1_PADDING When used with signing, this is RSASSA-PKCS1-v1_5
                        6 RSA_PKCS1_PSS_PADDING This denotes the RSASSA-PSS signature scheme.

usage: signtool make_debug_certificate [--image IMAGE]
                                [--config_dir PATH]
                                [--algorithm SIGNATURE_ALGORITHMS]
                                [--soc_id SOCID]
                                [--mask MASK_BITS]
                                [--rsa_padding RSA_PADDING]
arguments:
  --image IMAGE          Image to insert image header
  --config_dir PATH      Path include key pairs and version config file
  --algorithm SIGNATURE_ALGORITHMS Signature algorithm [ecc384|rsa3072|rsa4096]
  --rsa_padding RSA_PADDING The padding argument must be one of the RSA_*_PADDING values.
                        1 RSA_PKCS1_PADDING When used with signing, this is RSASSA-PKCS1-v1_5
                        6 RSA_PKCS1_PSS_PADDING This denotes the RSASSA-PSS signature scheme.
  --mask                Debug port config maskbit
  --soc_id              Soc unique id to enable secure debug function

usage: signtool verify_image          [--image IMAGE]
arguments:
  --image IMAGE          Image to insert image header

usage: signtool erase_signature          [--image IMAGE]
arguments:
  --image IMAGE          Image to insert image header

usage: signtool [-h]
  -h, --help            show this help message and exit
=====
```

表4-1 签名参数说明

参数	描述说明
--image	输入的镜像名称。
--config_dir	BSP 签名密钥对的放置路径，参见 <a href="#">2.3 签名密钥对</a> 。
--algorithm	签名算法选择[rsa2048 ecc384 rsa3072 rsa4096]。
--rsa_padding	签名填充（padding）。 <ul style="list-style-type: none"> <li>1 (RSA_PKCS1_PADDING)</li> <li>6 (RSA_PKCS1_PSS_PADDING)</li> </ul>
--mask	0xffffffffffffffff 是调试掩码，输入调试掩码给主证书使用，可根据需要设置。默认为 0xffffffffffffffff。
--soc_id	Socid 号。

签名算法和签名填充参数与适用芯片的对应关系如 [表 4-2](#) 所示。

表4-2 签名算法选择和签名填充参数与适用芯片的对应关系

参数名称	参数说明	适用芯片名称
--algorithm	rsa2048	SC9832E、SC7731E、SC9863A、UMS512(T)、UMS312
	rsa4096（默认）、ecc384、rsa3072	UMS9230(T)、UMS9620、UMS9157
--rsa_padding	1 (RSA_PKCS1_PADDING)	SC9832E、SC7731E、SC9863A
	6 (RSA_PKCS1_PSS_PADDING)	UMS512、UMS312、UMS9230(T)、UMS9620、UIS7870、UMS9157

sprd\_sign 签名工具命令示例：

- 插入签头（insert\_image\_header）：适用于 nasec 版本  

```
./sprd_sign insert_image_header --image imagename
```
- 镜像签名（sign\_image）：适用于单个镜像签名调试  

```
./sprd_sign sign_image --image fdl1.bin --config_dir config/ --algorithm rsa4096 --rsa_padding 6
```
- 制作调试证书（make\_debug\_certificate）：详见 [3.5 安全调试](#)  

```
./sprd_sign make_debug_certificate --image fdl1-sign.bin --config_dir config/ --algorithm rsa4096 --soc_id 0x939ff32ca2d078bbc04a045bdfbac7218fdb2603bd2adaaa3a6f4fa780d797f8 --mask 0xFFFFFFFFFFFFFFFF --rsa_padding 6
```

```
./sprd_sign make_debug_certificate --image u-boot-spl-16k-sign.bin --config_dir config/ --algorithm rsa4096 --soc_id 0x939ff32ca2d078bbc04a045bdfbac7218fdb2603bd2adaaa3a6f4fa780d797f8 --mask 0xFFFFFFFFFFFFFFFF --rsa_padding 6
```
- 擦除签名（erase\_signature）：

```
./sprd_sign erase_signature --image fdl1-sign.bin
```

单个 bsp 镜像签名也可以使用以下方法。下面以 UMS9230 的 tos.bin 签名为例来介绍这种方法。

步骤 1 将 tos.bin 放到根目录的 out/target/product/ums9230\*\*/路径下。

步骤 2 根目录执行：source build/envsetup.sh。

步骤 3 根目录执行：lunch。

步骤 4 选择编译工程编号。

步骤 5 根目录执行：build\_tool\_and\_sign\_images。

out/target/product/ums9230\*\*/下生成 tos-sign.bin。

----结束

Unisoc Confidential For kxdwww

# 5

## genkey.sh 和 version.cfg

在 bsp/tools/secureboot\_key/config/路径下新建 genkey.sh 脚本及 version.cfg 文件，文件具体内容参见 [5.1 genkey.sh 脚本](#)和 [5.2 version.cfg 文件](#)。

### 5.1 genkey.sh 脚本

```
#!/bin/bash

AVB_TOOL=../../../../../external/avb/avbtool

if [ $# -ne 1 ]; then
    echo "Usage: genkey.sh <partition_name>:boot/recovery/system/vendor/..."
    exit 1
fi

if [ ! -f "$AVB_TOOL" ]; then
    echo "avbtool not found! please check!"
    exit
fi

partition=$1
privatekey=rsa4096_"$partition".pem
publickey=rsa4096_"$partition"_pub.bin

openssl genrsa -out $privatekey 4096
$AVB_TOOL extract_public_key --key $privatekey --output $publickey
```

### 5.2 version.cfg 文件

```
trusted_version=0
avb_version_wcn=0
avb_version_gps=0
avb_version_vbmeta=0
```

```
avb_version_boot=0
avb_version_system=0
avb_version_system_ext=0
avb_version_vendor=0
avb_version_product=0
avb_version_socko=0
avb_version_odmko=0
avb_version_modem=0
avb_version_vendor_boot=0
avb_version_odm=0
avb_version_init_boot=0
```

Unisoc Confidential For kxdwww



# 6

## 常见问题答疑

### 6.1 如何回读分区

以 sml 分区为例，回读操作如下：

步骤 1 使用 ResearchDownload 加载原始 pac 包。

步骤 2 main page 页只勾选 fd11、lk-fd12。

步骤 3 “Flash operation” 页勾选 “Active read flash”，并勾选列表中的 sml 分区。

步骤 4 确认后开始下载。

步骤 5 回读的镜像保存在\Bin\ReadFlash\目录。

----结束

### 6.2 如何关闭 secureboot（配置 nsec 版本）

以 UMS9230 为例，关闭安全启动的参考配置如下。

- 在 device/sprd/qogirl6/ums9230\_1h10/product/ums9230\_1h10\_Natv/var.mk 中增加以下配置。

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

- 在 bsp 板级配置中同步导出以下配置。

bsp 板级配置文件：bsp/device/qogirl6/androidt/ums9230\_1h10/ums9230\_1h10\_base/common.cfg

```
//Secure Boot
```

```
export BSP_PRODUCT_SECURE_BOOT= "NONE" //"NONE" for close
```

```
export BSP_PRODUCT_VBOOT= "" //" " for close
```

```
//firewall
```

修改配置之后，涉及打包问题，可联系展锐 FAE 协助。

### 6.3 如何确定 bsp 镜像签名是否正常

- 通过刷机查看 log 的方式，检查镜像签名是否正确。
- 使用 UltraEdit 等文本编辑工具查看，签完名的镜像是有对应的 magic 的，头部有 DHTB 的字符串标识。尾部有 SIMGHDR 的字符串标识。