# Android 11.0 Secureboot User Guide

Document Version **V1.0**
release date **2020-08-12**

Unisoc Technology Co., Ltd.

Unisoc Technology Co., Ltd.

Document version V1.0 (2020-08-12)

# Preface

## Overview

This paper mainly introducesThe Secureboot solution for the Android 11.0 platform based on ARM architecture mainly introduces the functional design implementation, configuration and debugging guidance.

## Target Audience

This article mainly focuses onSoftware developer of Secureboot solution for Android 11.0 platform.

## Abbreviations

| Abbreviations | English full name | Chinese explanation |
|---|---|---|
| REE | Rich Execute Environment | Feature-rich executable environment, such asLinux+Android. |
| TEE | Trusty Execute Environment | Trusted execution environment, such asTrusty. |
| SPL | Second Primary Bootloader | Second stage bootloader |
| TOS | Trusty Operating System | Trusted Operating System |

## Symbol Conventions

The following symbols may appear in this document and have the following meanings.

| symbol | illustrate |
|---|---|
| 说明 | Used to highlight important/Key information, additional information, tips and more.<br>NOTE is not a safety warning and does not involve personal injury, equipment or environmental damage. |

## Change Information

| Document Version | release date | Modification Notes |
|---|---|---|
| V1.0 | 2020-08-12 | First official release. |

# Keywords

Secureboot, Android 11.0

# Table of contents



# Figure Catalog



# Table of Contents

# 1 Introduction to Secure Boot

Android uses industry-leading security features and works closely with developers and device implementers to ensure the security of the Android platform and ecosystem.

Verified Boot strives to ensure that all executed code comes from a trusted source (usually the device's OEM) to prevent attack or corruption. It establishes a complete chain of trust that starts with a hardware-protected root of trust, extends to the bootloader, and then to the boot partition and other verified partitions.

In addition to ensuring that the equipment is operating safelyIn addition to the Android version, Verified Boot also checks the Android version for built-in rollback protection. Rollback protection ensures that the device is only updated to a higher Android version, preventing possible vulnerabilities from persisting.

OEM manufacturers ensure that user devices should not be used for purposes different from those for which they were designed, and cannot run unauthorized software, thereby protecting the value of users' assets; end users need to ensure that their devices have not been tampered with and are trustworthy.

## 1.1 Principles of Cryptography

Information security needs to solve the following three problems:

- Confidentiality (Confidentiality): Information is not leaked during transmission
- Integrityrity): Information is not tampered with during transmission
- AvailabilityIllicitability): The user of the information is legitimate

Confidentiality, integrity, and validity are collectively referred to as the CIA Triad. Public key cryptography solves the confidentiality problem, and digital signatures solve the integrity and validity problems.

Digital signature is the practice of signing data through some

cryptographic algorithms to protect the source data. Typical

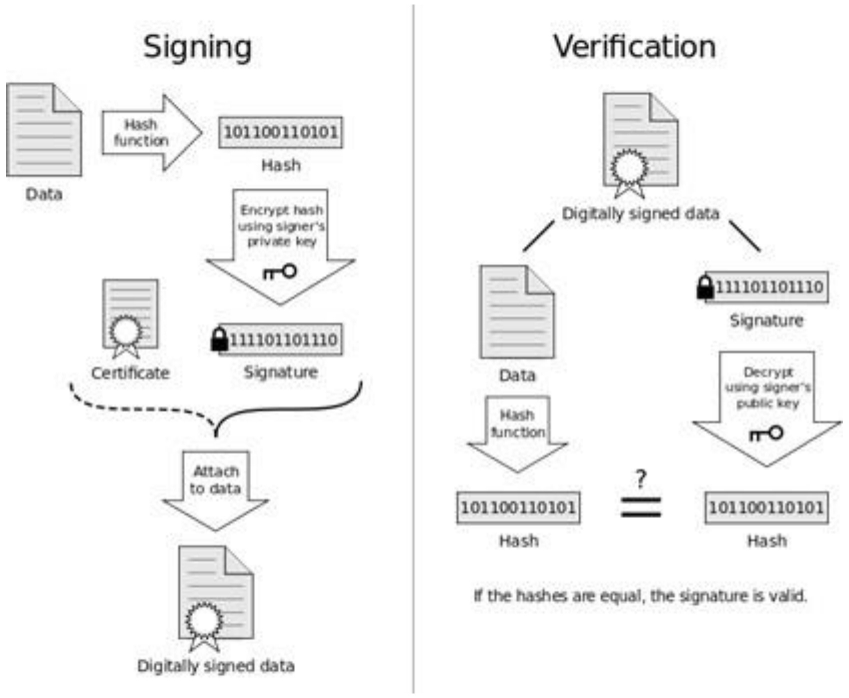
digital signature schemes include the following three

algorithms:

- Key generation algorithm: used to output public and private keys.
- Signature algorithm: Encrypt the given data with a private key to generate a signature.
- Signature verification algorithm: Use the public key to decrypt and verify the encrypted message.

Figure 1-1 Digital signature and verification process




The digital signature in Secureboot consists of a hash algorithm and an RSA algorithm. The PC generates and deploys the key and signs the image, and the terminal verifies the signature of the image when it starts.

The RSA private key is the guarantee of Secureboot and needs to be kept carefully.

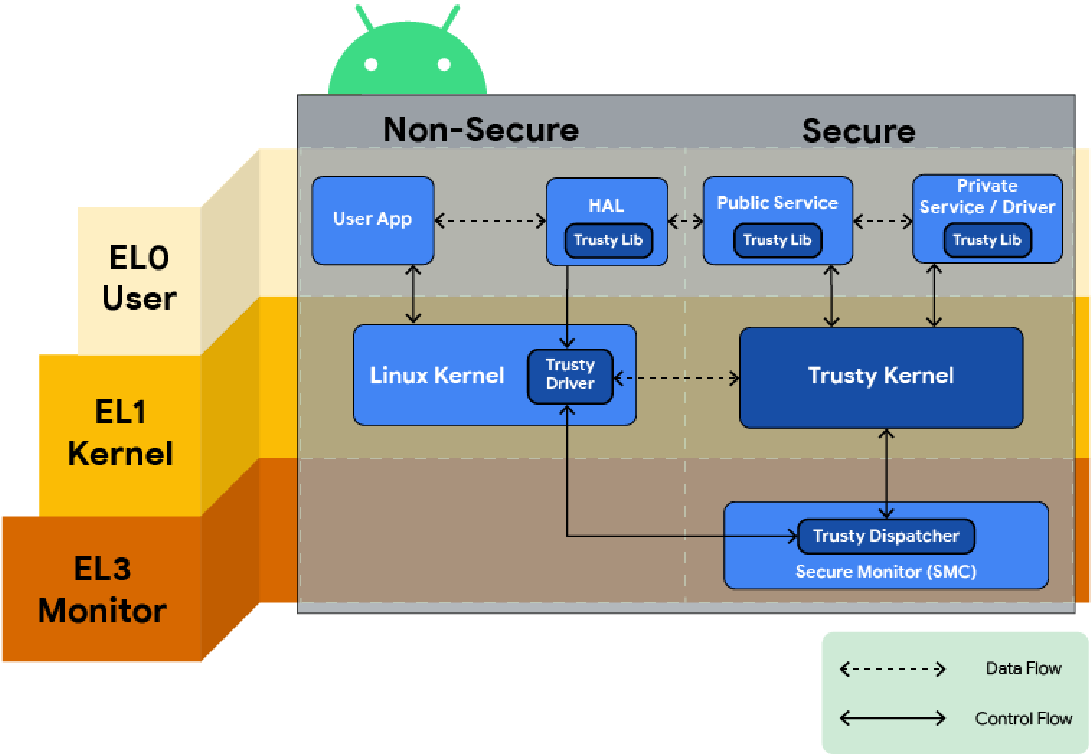
## 1.2 Introduction to TEE

Trusty is a secure operating system (OS) that provides a trusted execution environment (TEE) for Android.The Android operating system runs on the same processor, but Trusty is isolated from the rest of the system through hardware and software. Trusty and Android run in parallel with each other. Trusty has access to the full functionality of the device's main processor and memory, but is completely isolated. Isolation protects Trusty from malicious user-installed apps and potential vulnerabilities that may be found in Android.

Trusty is compatible with ARM and Intel processors. On ARM systems, Trusty uses ARM's Trustzone™ to virtualize the main processor and create a secure trusted execution environment.

Figure 1-2 Trusty overview




Trusty consists of the following components:

- Little Kernel derived small operating system kernel TOS
- Linux kernel driver used to transfer data between the secure environment and Android.
- Android userspace library used to communicate with trusted applications (i.e., secure tasks/services) through kernel drivers.

# 2 Image Signature

Different signature and verification schemes are adopted according to the different characteristics of the images that need to be protected. The principles of different signature and verification schemes are basically similar, and the difference lies in the different organization methods of the signature algorithm and the metadata for verifying the signature.

## 2.1 Image signature introduction

The image loaded during the boot phase uses Spreadtrum's own signature verification scheme. These images have fdl1.bin, fdl2.bin, u-boot-spl-16k.bin, sml.bin, tos.bin, u-boot.bin, teecfg.bin. These images are executed in the early boot stage of the system and generally run in a mode with a higher security level.

After Uboot is started, it will gradually load and verify the kernel, Android system and modem system images. These system images are all based on
Avb2.0 scheme is used to sign and verify. For small partitions that are only read once (such as boot, dtbo, recovery, and modem bins), the entire content is usually hashed and signed; for larger partitions that cannot fit in memory (such as file systems, vendor,
product, socko, and odmko partitions) can be signed using a hash tree approach; at boot time, the verification process continues as the data is loaded into memory.

## 2.2 Security related configuration

Secure boot is enabled by default. The MD configuration compilation system can enable or disable security-related configurations according to the boot chip to make it run or other product needs. If you want to turn off secure boot, taking UMS512 as an example, you need to make the following changes:

Add in device/sprd/sharkl5Pro/ums512_1h10/product/var.mk

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

In BSPThe following configurations are exported synchronously in the board-level configuration:

```
Bsp/device/sharkl5Pro/androidr/ums512_1h10/ums512_1h10_base/common.cfg
//secureboot
export BSP_PRODUCT_SECURE_BOOT=”SPRD”//”NONE”
forclose 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”
```

## 2.3 Generate a signing key pair

The secure boot solution requires the configuration of the following keys to complete the signing and verification of all system images.

## 2.3.1 BSP Signing Key

The paths to the BSP-related image signing key files are as follows:

bsp/build/packimage_scripts/configs/

There are mainly the following pairs:

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

The following command generatesRSA key pair:

### Generate RSA private key

```
$ openssl genrsa -out rsa2048_0.pem 2048
Generating RSA private key, 2048 bit long modulus
.................. +++
.....+++
e is 65537 (0x10001)
```

### Generate RSA public key

```
$ 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 signing is done automatically during the compilation of targets such as chipram, bootloader and trusty.picture2-1 As shown, the signature script will be triggered after the target is compiled successfully.

Figure 2-1 Signature script

```
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
```

inThe rsa2048_0 public key pair is used to sign fdl1.bin and u-boot-spl-16k.bin; rsa2048_1*.pem is used to sign sml.bin, tos.bin, and u-boot.bin.

## 2.3.2 Avb2.0 Signing Key

The key pair files required for the Avb2.0 signature scheme are deployed in the following directory:

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

The key pair files in this directory are as followsFigure 2-2shown.

Figure 2-2 Key pair file

| | | | | | |
|---|---|---|---|---|---|
| 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 | | |

Use the genkey.sh script to generate a new key pair. Take the OEM partition key pair as an example:

```
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
```

existThe corresponding configuration of the key can be found in the device/sprd/sharkle/common/security_feature.mk file, such aspicture2-3shown.

Figure 2-3 Corresponding configuration of key

```
#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 defines the private key name used to sign the BOOT image

The signature algorithm defined by BOARD_AVB_BOOT_ALGOTITH is SHA256_4096

FinishWhen compiling the boot image, the boot image will be signed by the

following script: After the parameters are extended, it is as follows:

```
./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/proprietories-source/packimage scripts/signimage/sprd/config/rsa4096 boot.pem
```

The same goes for other partitions.

# 2.4 Signature Scheme

## 2.4.1 BSP Signature Scheme

The BSP solution signing tools mainly include packimage.sh, sprd_sign and imageheaderinsert. After the tools are successfully compiled, they are installed in the "out/host/linux-x86/bin" directory and will automatically participate in the signing of the system image during the compilation process.

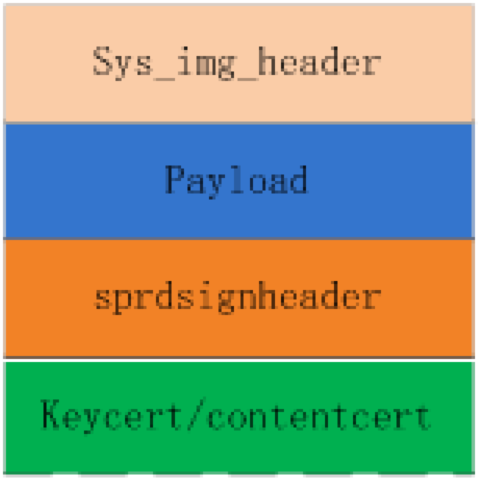
- Packimage.sh: The initial signature script contains the project configuration information, the trusted firmware to be signed, the signature method, etc., and will callThe imageheaderinsert tool inserts verification metadata into the image header and the signature certificate into the tail.
- Imageheaderinsert: A tool that inserts metadata required for verification into the image header file.
- Sprd_sign: A tool for signing images using a key.

The images signed by the BSP signature scheme include: SPL, fdl1, uboot, fdl2, sml,

TOS, vbmeta. The format of the trusted firmware after signing is as follows:picture2-

4shown.

Figure 2-4 Format of the trusted firmware after signing




## 2.4.2 Avb2.0 signature scheme

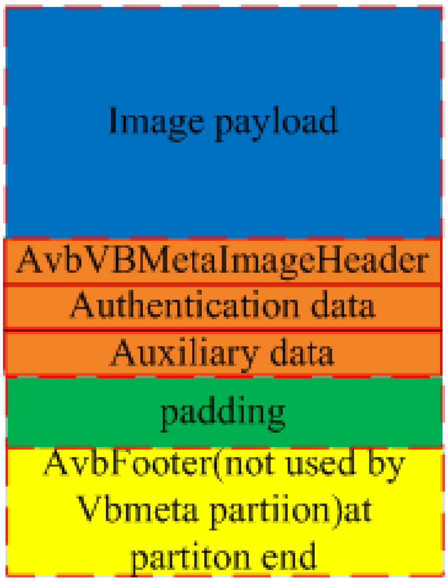
  

Using Avb2.0The files signed by the signature scheme are:

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

Avb2.0 signed format is incorrect!Reference source not found.

Figure 2-5 Format after Avb2.0 signature




There is no metadata for verification in the image header. The Avb signature scheme uses the 64-byte AvbFooter at the end of the image to locate the verification data. Android 10.0 introduced the concept of super partition, which makes the footer of the system, vendor, and product partitions in the super partition inaccessible according to the above design. The current solution is to copy and save the metadata for verifying the system, vendor, and product partitions to independent partitions: vbmeta_system and vbmeta_vendor.

The corresponding relationship is as follows:

- The metadata of the system and product partitions is saved to the vbmeta_system partition
- The metadata of the vendor partition is saved to the vbmeta_vendor partition

NoticeThe metadata of the dynamic partitions contained in the super partition must be consistent with the metadata of the vbmeta_system and vbmeta_vendor partitions.

Whether the metadata of the dynamic partition in the super image is consistent with that in the vbmeta_system/vendor image can be confirmed by the following process:

1. WillThe super image is converted from sparse format to raw data format to prepare for the next step of unlocking the super image.

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

2. uselpunpack extracts the dynamic partition image from the super image.

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

After completion, the super will be unpacked in the unpack directory. A partition contains an image of a partition.

3. useUse avbtool partition to check whether the metadata of the dynamic partition is consistent with the metadata of the vbmeta_xxx partition.

```
$ ./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:            704bytes
--
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:          1324228608bytes
     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:            10551296bytes
     Hash Algorithm:      sha1
     Partition Name:      system
     Salt:
                         ed49162cf97df4672cbf6793955dbbb7061956e3R
     oot 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:          447315968bytes
     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:            3571712bytes
    Hash Algorithm:       sha1
    Partition Name:       product
    Salt:               743ca7308a1574360168362a53cc53f511242232
    Root Digest:         e6a8dda20f28a9905b36c0154fe79158221471f4
    Flags:              0
  Hashtree descriptor:
    Version of dm-verity: 1
    Image Size:          1324228608bytes
    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:            10551296bytes
    Hash Algorithm:       sha1
    Partition Name:       system
    Salt:
                        ed49162cf97df4672cbf6793955dbbb7061956e3R
```

CompareCheck whether the Salt of the system partition is the same as the root hash of the hashTree. If they are not the same, a verification error will occur when the kernel mounts the system partition, causing the system to restart.

## 2.5 Avb signature process during OTA compilation

CompileWhen OTA is running, the OTA package compilation system uses the contents of the target directory to generate new system, vendor, and product partition images. In order to keep the version in the OTA package consistent with the version packaged from the PRODUCT_OUT directory to the pac package in the future, it is necessary to replace the system, vendor, product, vbmeta_system, and vbmeta_vendor images under PRODUCT_OUT with the new images generated by OTA compilation.

The relevant script is located in 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
```

The system vendor and product images are recompiled to generate PRODUCT_OUT using the dynamic partition image in the target directory.

The super.img under is synchronized. The compilation command is as follows:

```
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.zh
                              ang/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     "
```

So after the OTA package is compiled, the contents of the system, vendor and product partitions in the PRODUCT_OUT directory should be inconsistent with the system, vendor and product images unpacked by super.img. You only need to confirm that the super image unpackThe metadata of the system, vendor, and product images should be consistent with the metadata of the vbmeta_system&vendor images in the PRODUCT_OUT directory.

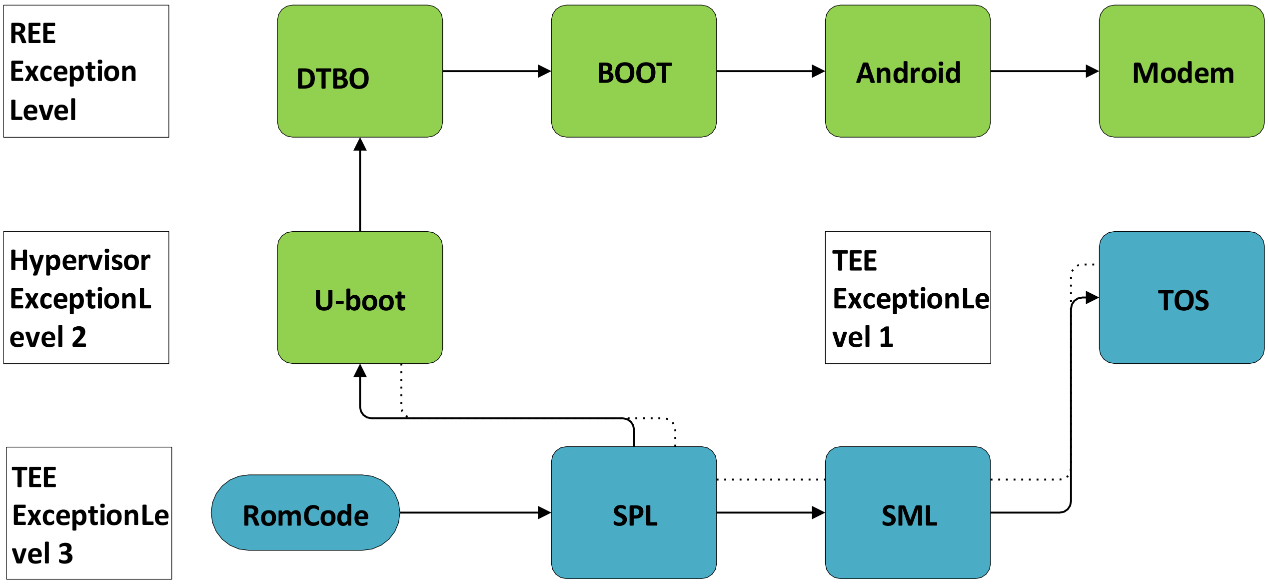
  

# 3 Secure Boot Process

## 3.1 AP secure boot process

The AP starts from Romcode and runs in safe mode when it starts. It loads and verifies the safe image step by step, and then boots the normal system image until the kernel and Android system are started.picture3-1shown.

Figure 3-1 Secureboot load TEE and REE images process




- Securely deployed devices will beThe first-level public key used to verify the SPL image signature is written into the ROTPK efuse area of the AP Soc
  The hash of (rsa2048_0_pub.bin).
- As part of the root of trust, Romcode cannot be rewritten (tampered with) and is the basis of secure boot. After power-on, Romcode loads the SPL image, reads the first-level public key from the SPL partition, and calculates its hash and compares it with the public key hash recorded in ROTPK. Only when they are consistent can this public key be used to verify the digital signature of the SPL image.
- The SPL image runs as a Secureboot loader in Secure IRAM, responsible for initializing DRAM and loading sml (Arm Trusted firmware), TOS (Trusty OS) and uboot (REE bootloader) into DRAM, and is also responsible for verifying the legitimacy of the three system images; sml will be executed after the verification passes.
- Sml runs TOS, TOS runs the built-in TA, then TOS returns to sml, and sml returns to uboot for execution.
- Uboot runs in Hypervisor mode and is responsible for checking the device status, loading and verifying the contents of dtbo and boot/recovery images, and preparing to verify the kernel parameters for booting the Android system.
- After the Uboot kernel is started, the kernel will be responsible for verifying the Android system partition. After verification, it will be mounted as a read-only system partition.

- After Android starts, the modem_control service is responsible for verifying and loading the firmware of the modem subsystem.

# 3.2 Secure boot of modem subsystem

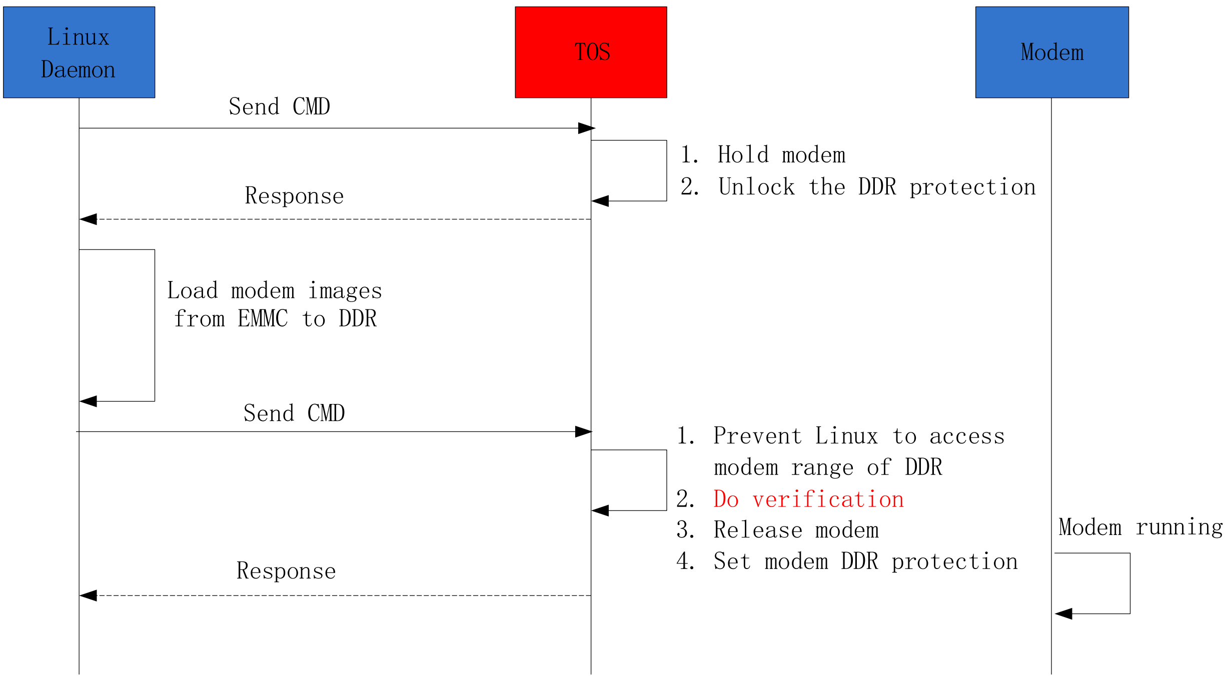
Modem, Audio, VDSP, GPU firmware, SCP and other subsystems that do not contain ROM usually need the cooperation of the main control to complete the subsystem's secure boot process.

The design ideas are as follows:

- passRegister Firewall restricts the startup of subsystems.
- passDDR Firewall is used to limit the address access range of a subsystem, usually accessing only its own address space.
- Mirror image firstThe AP subsystem is loaded into DDR, and then the access to this space is restricted. After the signature verification is successful, the corresponding subsystem is started to avoid the signature verification and running of different subsystems.

Modem subsystem startup process is as followsFigure 3-2shown.

Figure 3-2 Modem startup process



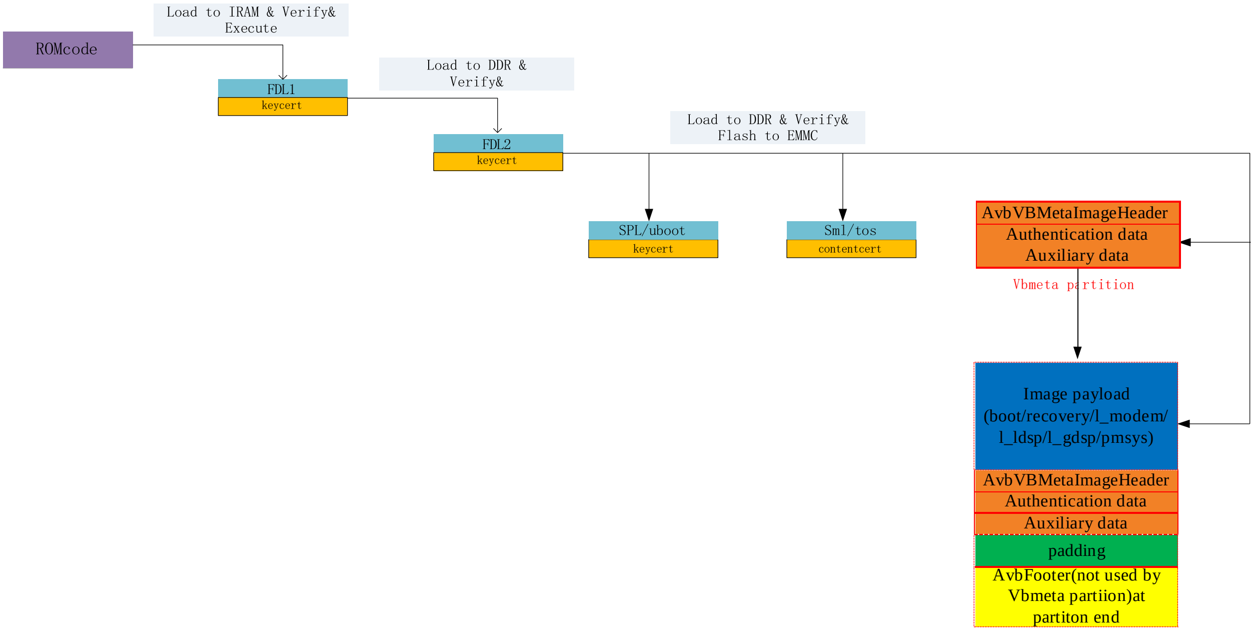

# 3.3 Security Updates

## 3.3.1 Tool Download

Download mode inThe function of fdl1.bin is the same as that of the SPL image in normal boot mode. On a securely deployed machine, romcode is also used to verify the digital signature of fdl1.bin. After passing the verification, fdl1 initializes DRAM and loads and verifies fdl2.bin. fdl2 is responsible for interacting with the PC, receiving and verifying the data downloaded by the PC, and updating it to static storage such as EMMC/NAND after verification.

Figure 3-3 Tool download mode startup process




When debugging the super partition content, if you use the pac package to download the super partition content, you must also download the superMirror image
The vbmeta_system image and the vbmeta_vendor image.

## 3.3.2 Fast boot burning

When burning the system partition in fastboot mode, the device should be unlocked.

The device state indicates how freely software can be flashed to the device and whether verification is enforced.
LOCKED and UNLOCKED. Devices in the LOCKED state are prohibited from flashing software, while devices in the UNLOCKED state are allowed to flash software.

useThe fastboot flashing [unlock_bootloader | lock_bootloader] signature.bin command can change the device status. When the device status changes, the data in the data partition will be erased first. To protect user data, the user will be asked to confirm before deleting the data.

Unlocking the bootloader

Boot-debug.img is the kernel plus a debug version of the ramdisk, so that the user version of VTS & STS can have ROOT permission; OEM signature is not required by design, and it is only used on devices that allow

verification error when unlocking the bootloader. For how to unlock the bootloader, please refer to the relevant

documentation of AndroidQ one-click unlocking bootloader.

After unlocking, the following prompt is displayed in the upper left corner of the machine during the bootloader stage of the system startup process:

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

The device status is also indicated in the kernel boot parameters:

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

After unlocking, if you need to remount, execute the following command:

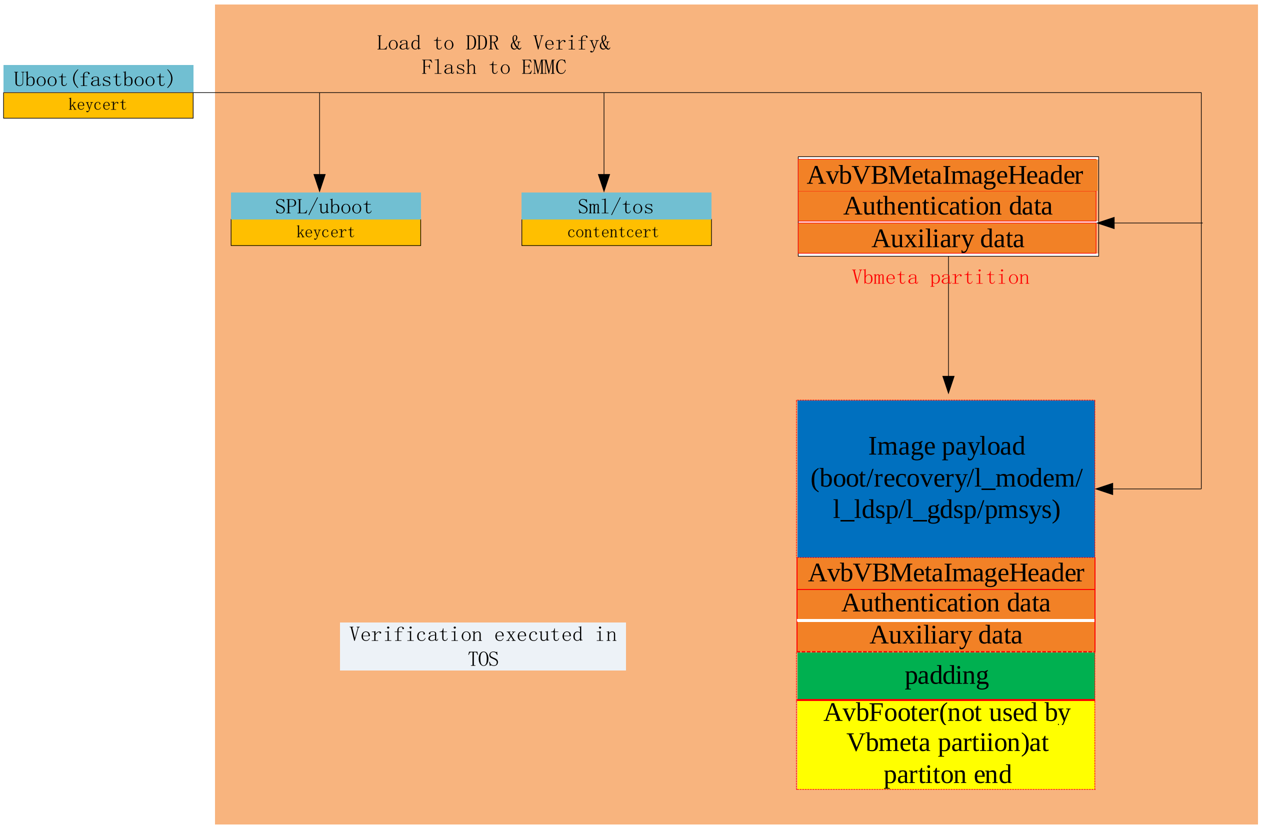
```
$adb root
$ adb disable-verify
```

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

After executing the above instructions, the system remounted successfully.

fastbootMode safe boot process such asFigure 3-4As shown, its mirror digital signature verification is performed in TOS, which has higher security.

Figure 3-4 Fastboot mode startup process




existIn fastbootd mode, dynamic partitions are updated using fastboot commands. The super partition can only be updated after the device is unlocked.

For fastbootd debugging, please refer to the fastbootd related documents.

**说明**

When fastbootd mode updates the super/system/vendor/product image, the vbmeta_system/vbmeta_vendor partition needs to be updated at the same time.

### 3.3.3 OTA Updates

End users obtain new releases through security updates. In addition to configuring anti-rollback versions, new versions also require legal signatures.
The OTA update package will also be signed and verified as a whole to ensure the legitimacy and integrity of the user's updated version.

## 3.4 Prevent version rollback

Even if the update process is completely secure, an attacker could potentially exploit a non-persistent system vulnerability to manually install an older, more vulnerable version of the system, reboot into the vulnerable version, and then install the persistent vulnerability from that version. In this case, the vulnerability would allow the attacker to permanently own the device and take any action, including disabling updates.

The protection against this type of attack is called "rollback protection". "Rollback protection" is usually implemented by using tamper-proof storage to record the latest version and refusing to boot the system if the version is lower than the recorded version. The system usually tracks the version for each partition.

Unisoc supports anti-version rollback configuration for each partition. The anti-rollback version number can be configured in the following configuration files:

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

The anti-rollback version will be compiled into the metadata of the secure boot. During the secure boot process, the system will compare the version number of the currently booted system image at each boot stage to see if it is greater than or equal to the version number recorded inThe version number of efuse (trusty firmware) and rpmb partition (Avb2.0 signature image) is allowed to start only if the version number meets the conditions. After the system is fully started, if it is the latest version number, it is updated to the one-time writable area.

**说明**

The anti-rollback version is only configured in future mass production versions and cannot be configured during the R&D and debugging phase.

## 3.5 Security Debugging

Before secure debugging, you need to generate a debugging certificate and copyfdl1-sign.bin and u-boot-spl-16k-sign.bin image files and sign these two copies with the generated debug certificate.

The specific steps are as follows:

step1 WillCopy fdl1-sign.bin and u-boot-spl-16k-sign.bin to the following directory:

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

step2 GetSocid number (each chip has a unique socid number)

The command to get socid in fastboot mode is as follows:

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

The obtained socid is as follows:

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

step3 Generate a debug certificate

The debugging certificate includes a primary certificate and a secondary certificate. If you do not change the devkey, you can ignore step a.

a. (Optional) Place rsa2048_devkey_pub.pem in the following path:

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

Put rsa2048_devkey_pub.pem and rsa2048_devkey.pem into the following path:

```
$vendor/sprd/proprietories-source/packimage scripts/signimage/sprd/mkdbimg/config
```

b. Execute the following command in the following directory

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

0xffffffffffffffff is the debug mask,The debug mask entered is for the primary certificate and can be set as needed. The terminal will display:

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

Take the SC9863A series project as an example, input 2

The terminal will display:

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

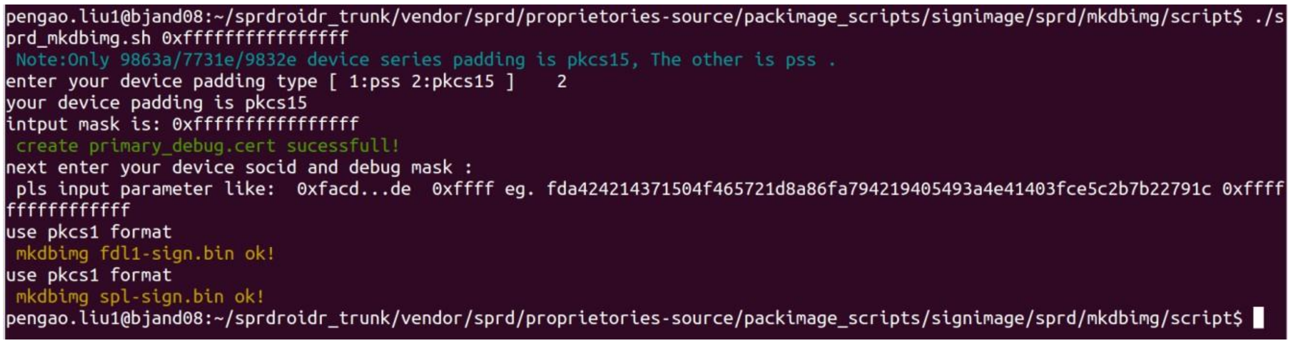
Enter the socid and debugmask parameters as prompted

The socid parameter is the unique identifier of the soc obtained in step 2.

The debugmask parameter is a 64-bit debug mask. It can be turned on as needed. If you want to enable all debug bits, you need to enter
"0xffffffffffffffff".

generateAn example of a commissioning certificate for a SC9863A project is:picture3-5shown.

Figure 3-5 Generate SC9863A debugging certificate



```
pengao.liu1@bjand08:~/sprdroidr_trunk/vendor/sprd/proprietories-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
intput 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 fdl1-sign.bin ok!
use pkcs1 format
 mkdbimg spl-sign.bin ok!
pengao.liu1@bjand08:~/sprdroidr_trunk/vendor/sprd/proprietories-source/packimage_scripts/signimage/sprd/mkdbimg/script$
```

step4 Use the generated debug certificate to step1 Sign the two copied image files.

**----**Finish

**说明**

- Enter the device certificate filling type [1: pss 2: pkcs15]. The certificate filling type has different filling methods according to the chip series. The 9863a/7731e/9832e/8541e series and their derivative chip series use the pkcs15 method, and the others such as ums312/ums512/9230 use the pss filling method. You can judge by the project name of source/lunch.
- If the secure debug certificate does not enable JTAG, follow these commands:

```
cd vendor/sprd/proprietories-source/packimage_source
mma
```

- If the following warning appears when running sprd_mkprimarycert.sh or sprd_mkdbimg.sh: "Error loading shared libraries: libc++so: cannot open shared object file: No such file or directory", you canCopies the libc++.so files in vendor/sprd/proprietories- source/packimage_scripts/signimage/sprd/mkdbimg/bin to /lib/x86_64-linux-gnu/.

## 3.6 Safe production line deployment

The data related to secure boot needs to be written into the eFuse partition of the production line. Table 3-1.

Table 3-1 Production line eFuse partition internal table

| name | use | length | Location | illustrate |
|---|---|---|---|---|
| ROTPK | Root of trust Public key Hash | 256 | private efusedistrict bit[512]~bit[767] | Message digest of the root trusted public key, To verify the signatureCertificate of authenticity.TAMThe maintainer releases it, usuallyOEMManufacturer confirmed. |
| Secure OS Minimum version number | anti-rollback counter | 32 | private efusedistrict bit[768]~bit[799] | Third party storageSecure OSManufacturer software version number |
| AndroidMinimum version number | anti-rollback counter | 224 | private efusedistrict bit[800]~bit[1023] | StorageOEM/ODMfactoryAndroidSoftware version number |

📖 说明

The above information isThe position in efuse must be consistent with the agreement of Spreadtrum.