紫光展锐
U N I S O C

Android9.0 Secure Boot Integration Guide

版本： V1.0

日期：2018-12-11

目 录

版本历史

版本	日期	作者	备注
V1.0	2018.12.11		初始版本

声明

1. Introduction

This document is to present secure boot design for Spreadtrum platform. It will cover the signature, Image download&update over the PC tools; the system secure boot process; etc. Copy some sections from other document for better understanding the secure boot. EMMC solution all support secure boot.

1.1. Request & Purpose

The secure boot in Spreadtrum platform should have the following features. If you want to find more, please find pc tool documents and bootloader documents and others.

1. Image signature
2. Secure Boot Process
3. Image download, update

1.2. Definitions & Abbreviations

Name	Description
CA	Certificate authority
PUK	Public key
NV	Non-Volatile
SD	Secure Debug
TEE	Trusted Execution Environment
LCS	Lifecycle States
OEM	Original Equipment Manufacture
OTP	One-Time-Programmable Memory
SB	Secure Boot
PrvKB0	Boot Private Key 0
PrvKB1	Boot Private Key 1

1.3. Reference

[1] http://en.wikipedia.org/wiki/Digital_signature

2. Images signature

2.1. Digital signature

A digital signature scheme typically consists of three algorithms:

1. A key generation algorithm that selects a private key uniformly at random from a set of possible private keys. The algorithm outputs the private key and a corresponding public key.
2. A signing algorithm that, given a message and a private key, produces a signature.
3. A signature verifying algorithm that, given a message, public key and a signature, either accepts or rejects the message's claim to authenticity

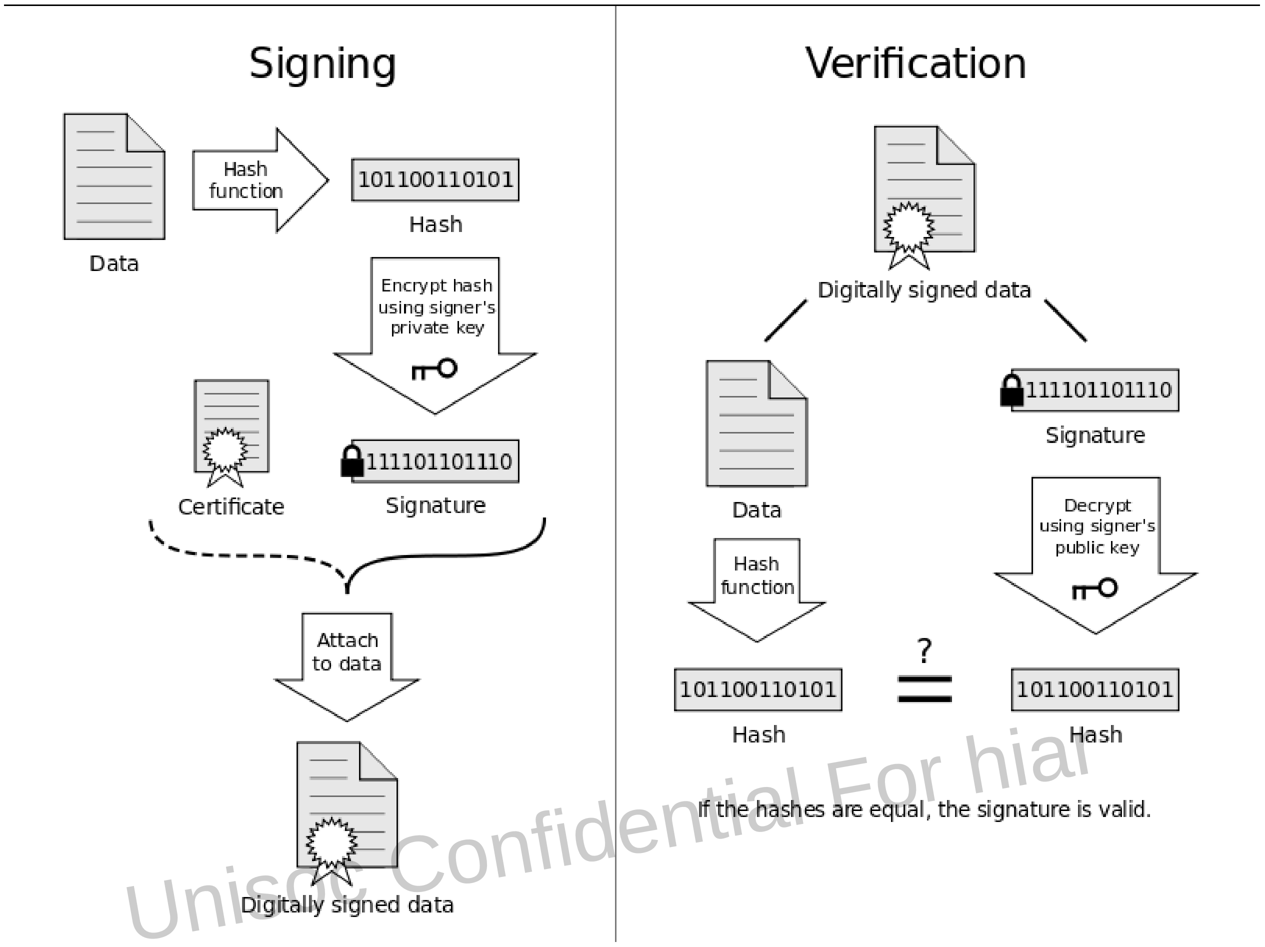

Figure 1 digital signature scheme

The diagram explain the signing and verify process. We use SHA2 for hash and RSA for encryption. Signature tool in the branch of code warehouse,it will sign images automatic after compilation, the phone software do the verification.

2.2. Images signature

Signature tool signs the images with the key which generated manual .
The signature modules are: packimge.sh,imgheaderinsert,sprd_sign,avbtool.py. These modules will be involved in the whole system compilation,then installed in the "out/host/linux-x86/bin" directory after the successful compilation.

- Packimage.sh: entry for sprd trusted firmware,call the imgheaderinsert tool to insert imgheader and sprd_sign
  Imgheaderinsert:binary file which executes insert image header

Sprd_sign :   binary file which signs the images with the key

sprd trusted firmware:spl/fdl1,uboot/fdl2, sml,tos,vbmeta

- Avbtool.py: sign tool of android avb 2.0 for none-trusted firmware

  None-trusted firmware:boot/recovery/system/vendor/l_modem/l_ldsp/l_gdsp/pm_sys.

1. For sprd trusted firmware , Figure 2 shows the layout of the image after signature, different images will select the certificate from keycert and contentcert ,which can only be one of these two certificates.

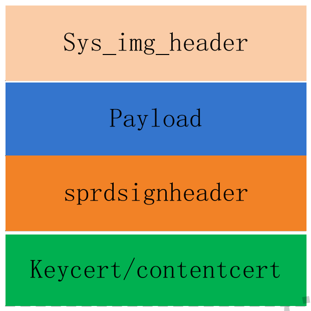

Figure 2 signed image layout for sprd trusted firmware

By default when the chip boots it should be in the debug disabled mode aka green mode,probemode is disabled in green, in order to enable probemode a developer needs to use the debug certificate.

Figure 3 show the layout of image with debug certificate.

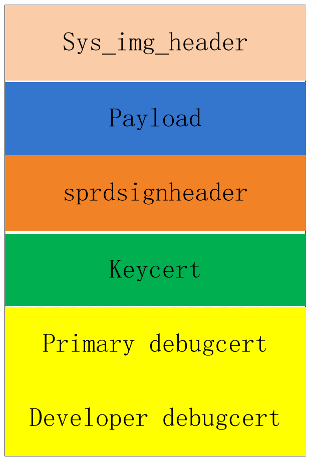

Figure 3 image with debug certificate

Please note: only fdl1.bin and spl.bin need debug certificate if probemode is enabled.

2. For None-trusted firmware, the format after signing by avbtool is as following:

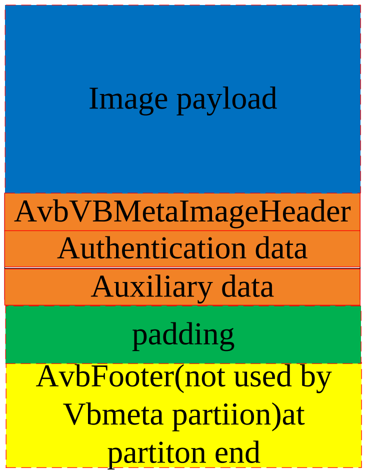

Figure 4 signed image layout for none-trusted firmware

3. Secure Boot Process

Secure boot is defined as verification of all the software in the boot chain from the reset vector to launching the Android system partition, Also the assumption is that if any devices are being initialized with device firmware,the firmware has to be either verified by the trusted world(TEE) or by trusted software(already verified in the boot process) or ROM on the device itself. If the device gets reset after OS is running and the firmware has to be re-loaded,the firmware should get verified by the TEE or by the ROM on the device or by non-OS trusted SW verified during secure boot.

The root of trust (Rot) in the platform is a 256 bit Rot public key hash that is stored in the fuses,this key is programmed by the OEM.This key forms the root of trust for all software except the BaseIA microcode,the BaseIA microcode is signed by a private key with the corresponding public in BaseIA uROM.

3.1. Secure Boot Flow

The boot process is as below:

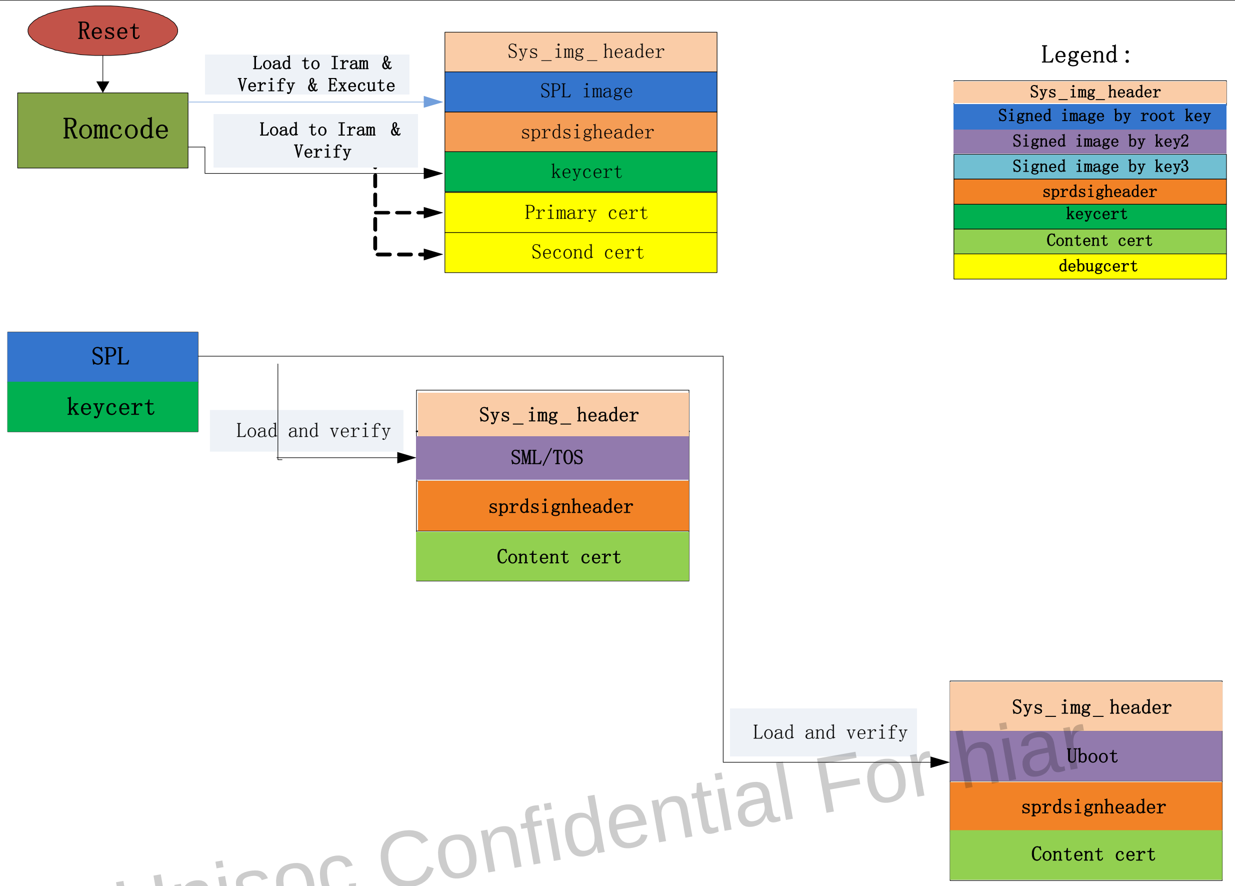

Figure 5 The first phase of boot process

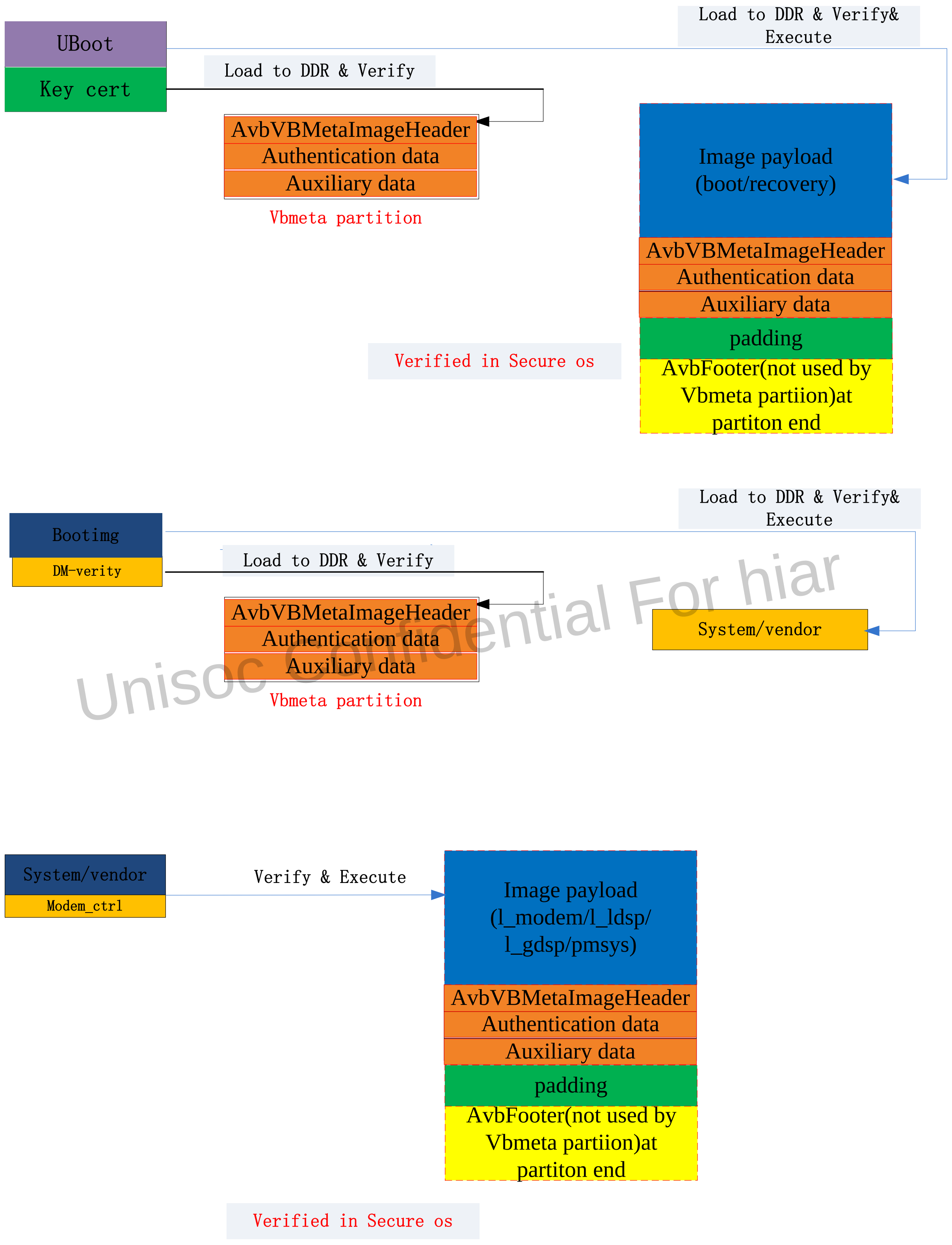

Figure 6 The second phase of boot process

note:the verification process of boot.img and modem is in the secure os.

3.2. Secure Download

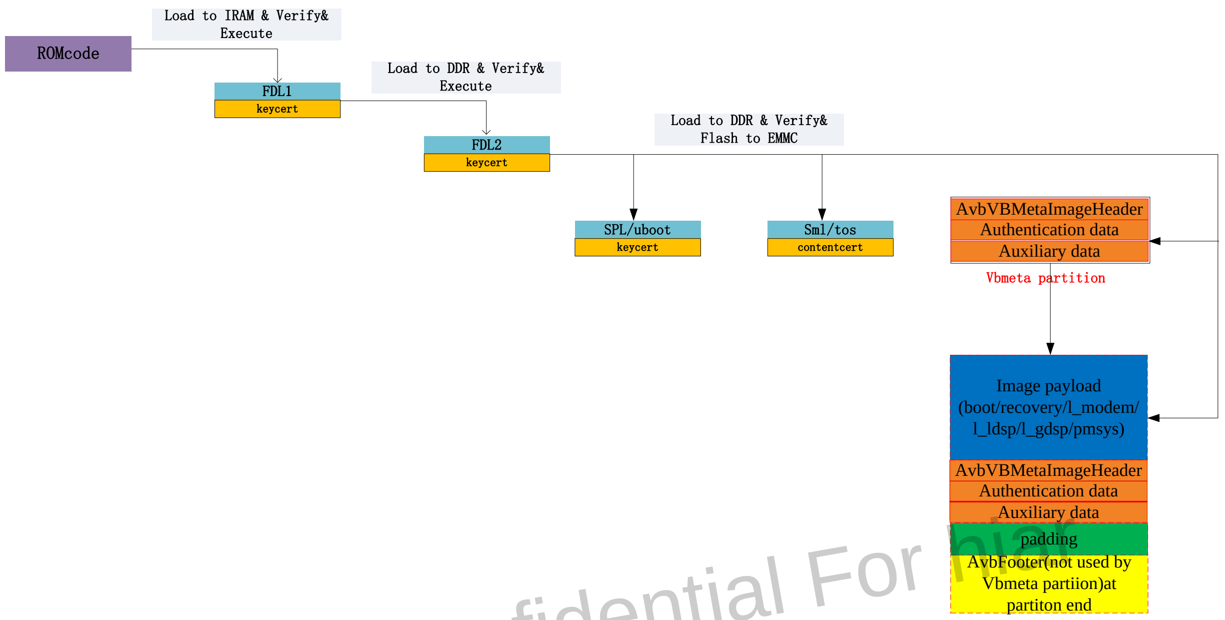
Load to IRAM & Verify& Execute
ROMcode
FDL1
keycert
Load to DDR & Verify& Execute
FDL2
keycert
Load to DDR & Verify& Flash to EMMC
SPL/uboot
keycert
Sml/tos
contentcert
AvbVBMetaImageHeader
Authentication data
Auxiliary data
Vbmeta partition
Image payload (boot/recovery/l_modem/ l_ldsp/l_gdsp/pmsys)
AvbVBMetaImageHeader
Authentication data
Auxiliary data
padding
AvbFooter(not used by Vbmeta partiion)at partiton end

Figure 7 Secure Download flow

3.3. Secure Fastboot

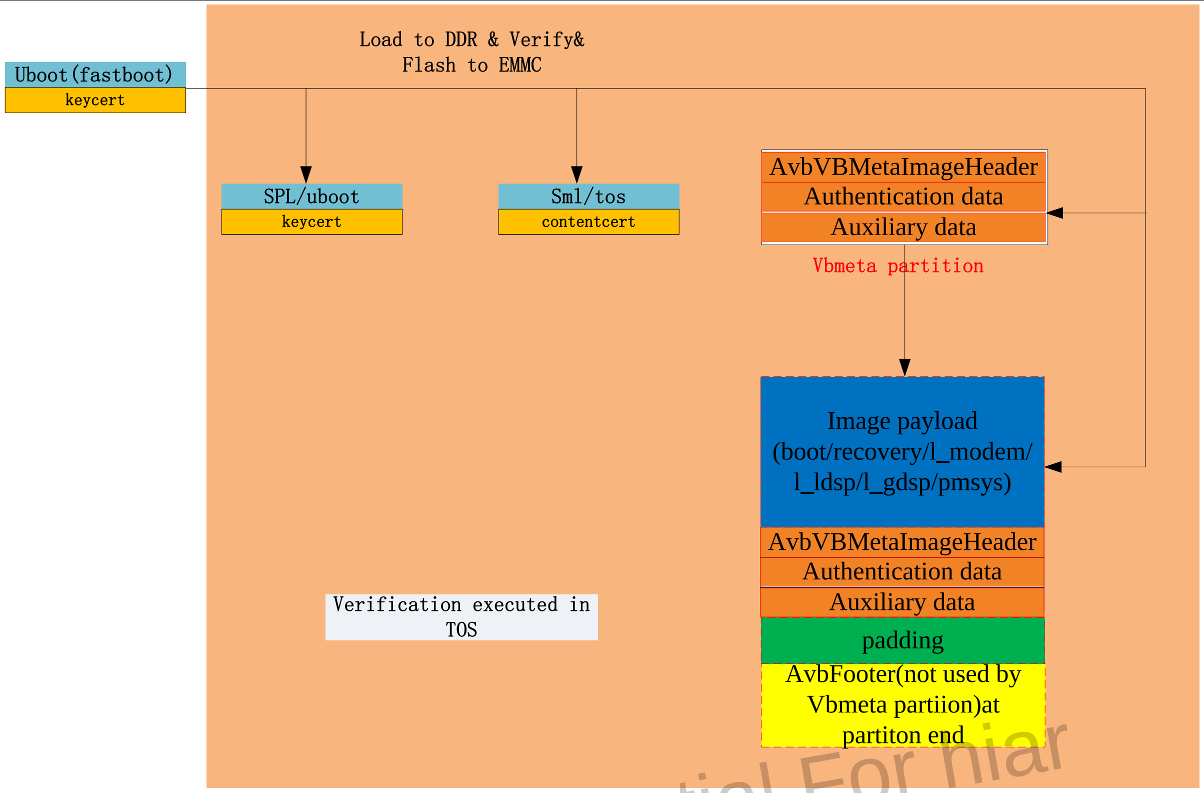

Figure 8 Secure Fastboot flow

3.4. Secure debug

Secure debug is available to OEMs through registers exposed by one of the IPs in the south.In the Intel terminology,this is called as Orange Unlock and will unlock the BaseIA TAP and enable PROBEMODE. This will enabled the OEM to use Lauterbach to step through the OS,BIOS and read any control registers,program counters,debug registers,MSRs etc.This capability is only given to the OEM and not to the end-user.So the OEMs should ensure that the bits that are used for enabling secure debug are restricted to the OEMs.

If the device has been enabled secure boot, the device debug interface will be closed,trace will not be connected to the device successfully. In this case, if there is a need to debug,then secure debug certification is needed.

The debug certification contains the uid 、debug mask、 pubkey hash and signature,and attached to the image which with keycert and contentcert. (please note:only spl and fdl1 need the debug certification ,because spl and fdl1 locates in first step of boot and download). During the download process,the romcode verify fdl1 with the key certification 、content certification and debug certification ,if the uid value in the debug certification is same as the device uid and the signature is right,then romcode will set relate registers as debug mask value,open the debug interface. The boot process is same as download process,the only difference is change fdl1 to spl.

The follow figure shows process of debugging certificate generation.

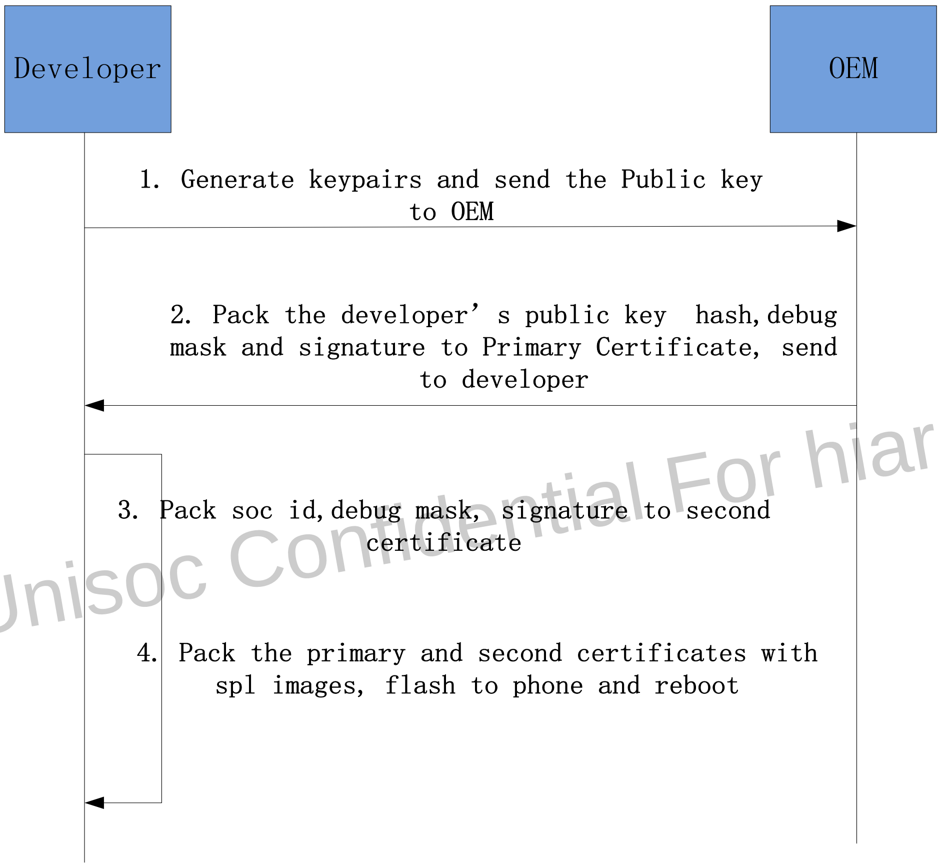
Developer
OEM
1. Generate keypairs and send the Public key to OEM
2. Pack the developer’s public key hash, debug mask and signature to Primary Certificate, send to developer
3. Pack soc id, debug mask, signature to second certificate
4. Pack the primary and second certificates with spl images, flash to phone and reboot

Figure 9 process of debugging certificate generation

3.4.1. process flow

The debug_mask from primary_debugcert gives the permission of debug control, bit value 1 means debug control of the corresponding bit is permitted. While debug_mask from developer_debugcert means the request of debug control, bit value 1 means request to enable this debug control bit. The request is valid only if the corresponding bit in debug_mask of primary_debugcert gives permission.

For example: if mask in primary_debugcert is 0x1234 , conversion become binary is 0001 0010 0011 0100, so the bit2 bit4 bit5 bit9 bit12 has permission to set. Then get the mask and take the value of bit2 bit4 bit5 bit9 bit12 from developer_debugcert , value 1 means to enable this control bit.(Notice: how to enable this control bit should be defined by specific register, maybe 1 means enable or 0 means enable) Summarize,the mask from primary_debugcert decide which bit can be setted. According to the bit which can be setted to get the control bit from developer_debugcert.

4. Anti-rollback

The images version has been kept in efuse,during the boot and download process ,the vesion will be checked,if the image version is less than the value which keep in efuse,the image will not be loaded and the flow will be halted. The images version protected by certificate.

There are 32 bits secure efuse for secure images, 1 bit as overflow. For normal images including boot,recovery, agdsp,modem etc,we must use rpmb to store the version due to no efuse block reserved for this.

The image version will be updated only if verification passed during boot. When flash images by tools ,such as pc, fastboot, version will be checked, but not updated. If the version check failed,the flash process will be halted.

It updates the version efuse bit only after verify all that type images.

If flash the lower version image,download will fail, the lower image will not be flashed to device,the device will boot successfully with higher verison image which located in emmc .

Uboot need to support rpmb read for normal image version.

TOS need to support for rpmb write.

5. Life cycle status

There are 4 life cycle status: CM, DM, Secure Enable.

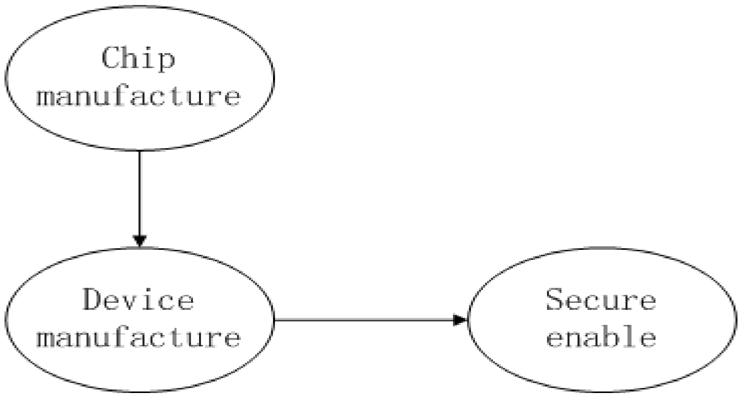

Figure 10 The Life cycle status

For the CM, DM, Secure enable reply on the lock bits. If all the secure fuse bits are zero, the chip is in CM status.

If the HUK was blew and the HUK lock bits were set, the chip is in the DM status. The software only need to check the lock bits to check whether it is in the DM status.

If the ROTPK or another ROTPK was blew and the corresponding lock bits were set, the system is in the Secure enable status. The software only need to check the lock bits to make sure whether it is secure enable status or not. The ROTPK was selected by the life cycle status.

6. Images download

6.1. PC Download/Update

If secure boot enable, some download images should be signed, the spl,,fdl1,fdl2,u-boot should also enable secure feature.

The first time (in factory for example), Romcode don’t verify spl because OTP hasn’t been programmed. If Efuse has been activated, do verification every time when download or boot.

The signed image includes fdl1, fdl2, spl,u-boot,bootimage,recoveryimage, dsp,modem,wcnmodem etc.

When download the images, it will verify first. If verify fail, it can not be downloaded.

Download process

1. Bootrom loads FDLl and verify the public key in keycert with hash in the OTP.
2. Bootrom using the public key in FDL1 to verify FDL1 payload.
3. Fdl1 uses the hash of public key in keycert to verify the public key in keycert which is added to FDL2,if the verification is successful,Fdl1 uses the public key to verify Fdl2 payload.
4. Fdl2 uses the hash of public key to verify the public key in contencert which is added to other images,if the verification is successful,Fdl2 uses the public key to verify the other images.
5. If the load images is spl and u-boot, the verify process is same with boot process.

The public keys in stage 2/3/4 may be different for different controlling management

6.2. Fastboot

Fastboot verify the signed image first when phone secure feature activated.

Fastboot splits the signed images and save to the right position just same like the fdl2.

Because fastboot send the whole image from PC, if the image size is larger than the RAM size, the original fastboot code should be changed to support split images download.

It support 4 signed images: spl.bin,u-boot,boot.image,recovery.image.

6.3. Recovery

When updated with recovery, Follow the original program of android, does the default verification.

7. CP Verification

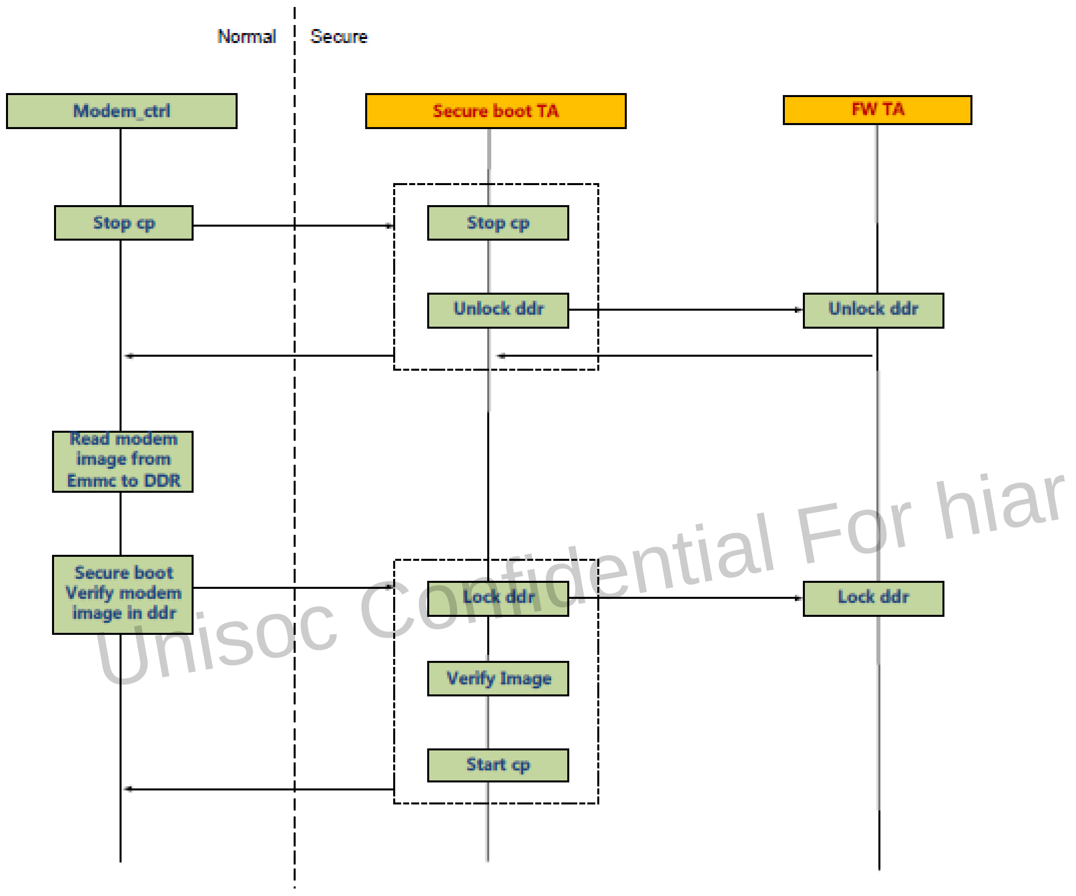

Figure 11 The CP Verification Process

8. Sign tool-Packimage

8.1. Sign process

To understand image signature process, please refer to chapter 2.2.
Different images have different levels of certificates.

spl/fdl1 signed by key0.
uboot/fdl2 signed by key1.
Boot/dtb/dtbo signed by key2.
Modem signed by key3.
Recovery signed by key4.
System signed by key5.
Vbmata sigend by key6.
Vendor signed by key7.
Product signed by key8.

8.2. Sign tool

Packimage is a top directory which contains all the security tools.It`s for integrating imgheader insert tool,SPRD signature tool into Android. The following is the packimage directory tree.
The location of packimage in Android:
$(TOPDIR)/vendor/sprd/proprietories-source/packimage_scripts

Signature configuration:

```
packimage_scripts/
├── packimage.sh    <pack image>
├── signidentifier_unlockbootloader.sh <generate signature for identifier_token>
└── signimage
└── sprd
  ├── config
  |   ├── genkey.sh    <generate boot/recovery/moded etc keypair>
  |   ├── rsa2048_0.pem   <key0>
  |   ├── rsa2048_0_pub.pem
  |   ├── rsa2048_1.pem   <key1>
  |   ├── rsa2048_1_pub.pem
  |   ├── rsa2048_devkey_pub.pem <debug cert key>
  |   ├── rsa4096_boot.pem <key2>
  |   ├── rsa4096_boot_pub.bin
  |   ├── rsa4096_modem.pem <key3>
  |   ├── rsa4096_modem_pub.bin
  |   ├── rsa4096_recovery.pem <key4>
  |   ├── rsa4096_recovery_pub.bin
  |   ├── rsa4096_system.pem <key5>
  |   ├── rsa4096_system_pub.bin
  |   ├── rsa4096_vbmeta.pem <key6>
  |   ├── rsa4096_vbmeta_pub.bin
  |   ├── rsa4096_vendor.pem <key7>
  |   ├── rsa4096_vendor_pub.bin
  |   ├── rsa4096_product.pem <key8>
  |   ├── rsa4096_product_pub.bin
  |   └── version.cfg  <config version value. Use for anti-rollback>
  ├── mkdbimg<debug cert>
  |   ├── config
  |        ├── createdevkey.sh    <generate debug cert keypair>
  |        ├──rsa2048_devkey.pem <debug cert key>
  |        └──rsa2048_devkey_pub.pem
  └── mkprimarycert        <primary cert>
```

Secure debug:

If require to enable secure debug, please firstly prepare debug certificate, fdl1-sign.bin and u-boot-spl-16k-sign.bin debug. The following step will be executed::

1．Copy fdl1-sign.bin and u-boot-spl-16k-sign.bin to
$(TOPDIR)/vendor/sprd/proprietories-source/
packimage_scripts/signimage/sprd/mkdbimg/bin

2．Get socid. Each chip have it’ s own id.
 Use fastboot to get socid. sudo ./fastboot getsocid socid
The right socid should as follow:
socid is:
939ff32ca2d078bbc04a045bdfbac721
8fdb2603bd2adaaa3a6f4fa780d797f8

3．Generate primary debug cert
The default primary_debug.cert is put in:
$(TOPDIR)/vendor/sprd/proprietories-source/
packimage_scripts/signimage/sprd/mkdbimg/bin
If not change devkey, you can ignore first step.
First:
you should put rsa2048_devkey_pub.pem in
$(TOPDIR)/vendor/sprd/proprietories-source/ packimage_scripts/signimage/sprd/config
Put rsa2048_devkey_pub.pem and rsa2048_devkey.pem in
vendor/sprd/proprietories-source/ packimage_scripts/signimage/sprd/mkdbimg/config

Second:
Enter
vendor/sprd/proprietories-
ource/packimage_scripts/signimage/sprd/mkprimarycert/script
Run ./sprd_mkprimarycert.sh 0xffffffffffffffff
The 0xffffffffffffffff is debugmask, it can set as your wanted.

4．Generate debug cert:
Enter
vendor/sprd/proprietories-source/packimage_scripts/signimage/sprd/mkdbimg/script
Run ./sprd_mkdbimg.sh
The terminal will display:

Enter you device serial number type [1:socid 2:uid]

Please enter "1"

The terminal will display:

Pls input parameter like: 0xfacd...de 0xffff eg.

Please enter "socid debugmask"

The socid is got on step2

The debug mask is 64bit . the value is depended by customer. To enable all debug bit need input 0xffffffffffffffff

For example:

"0x939ff32ca2d078bbc04a045bdfbac7218fdb2603bd2adaaa3a6f4fa780d797f8 0xffffffffffffffff"

The fdl1&spl binary copied on step1 will be signed with debug cert.

Remark :

(1) If secure debug cert cannot enable JTAG ,please do the following:

    cd vendor/sprd/proprietories-source/packimage_source

    mma

    Repeat 1/2/3/4 steps

(2) If there is warning: "error while loading shared libraries: libc++.so: cannot open shared object file: No such file or directory" while running sprd_mkprimarycert.sh or sprd_mkdbimg.sh, you can get the file "libc++.so" from the directory: "vendor/sprd/proprietories-source/packimage_scripts/signimage/sprd/mkdbimg/bin" and put it into the directory: "/lib/x86_64-linux-gnu/"

9. Customer Guideline

9.1. Configure keys

You shouldn’ t use the default keys.

9.1.1. Generate keys

1. Use openssl command to generate 2 key pairs

openssl genrsa -out rsa2048_0.pem 2048
openssl rsa -in rsa2048_0.pem -pubout -out rsa2048_0_pub.pem

openssl genrsa -out rsa2048_1.pem 2048
openssl rsa -in rsa2048_1.pem -pubout -out rsa2048_1_pub.pem

2. Run genkey.sh to generate keypair for Boot/Recovery/System/Modem/Vbmeta/Vendor

If there isn’ t out/host/linux-x86/bin/avbtool, please compile it first. You can configure the build environment, enter external/avb, then execute mma command.

Enter
vendor/sprd/proprietories-source/packimage_scripts/signimage/sprd/config
Run ./genkey.sh boot （The boot is partition name）
Run ./genkey.sh recovery
Run ./genkey.sh system
Run ./genkey.sh vbmeta
Run ./genkey.sh vendor

Run ./genkey.sh product
Run ./genkey.sh modem

to generate keypairs.

3. If you want to change keys again:
Replace the key in config folder with generated key .
Run sign tool again. The binary will signed by your generated key.
Be careful if the ROTPK had been written. Should not change the key 0 anymore.

9.1.2. Configure Version

The preferred original version is zero.

Configure version (anti–rollback version):
Just modify version.cfg
1. trusted_version = 1: trusted_version mean trusted firmware version (spl/fdl1/uboot/fdl2/tos/sml) , "1" mean version number. It should be 1,2... , The max value for trusted_version is 32.
2. avb_version_vbmeta = 1: avb_version_meta mean vbmeta image version, "1" mean version number. It should be 1,2...
3. avb_version_boot = 1: avb_version_boot mean boot(recovery/dtbo/dtb) image version, "1" mean version number. It should be 1,2...
4. avb_version_system = 1: avb_version_system mean system image version, "1" mean version number. It should be 1,2...
5. avb_version_vendor = 1: avb_version_vendor mean vendor image version, "1" mean version number. It should be 1,2...
6. avb_version_product = 1: avb_version_product mean product image version, "1" mean version number. It should be 1,2...
7. avb_version_modem = 1: avb_version_modem mean modem image version, "1" mean version number. It should be 1,2...

9.1.3. Secure debug

The JTAG function was disable in board factory.
If you want to re-enable JTAG function, you have to generate secure debug certificate.

Secure debug key:
The customer can change devkey.
Enter
vendor/sprd/proprietories-source/packimage_scripts/signimage/sprd/mkdbimg/config
Run ./createdevkey.sh
The above script will generate rsa2048_devkey.pem and rsa2048_devkey_pub.pem and copy rsa2048_devkey_pub.pem to
vendor/sprd/proprietories-source/packimage_scripts/signimage/sprd/config directory.

9.2. CP image signature

The l_modem/l_ldsp/l_gdsp/pm_sys image is signed through avb_sign_modem_xxxxx.sh on build server, to get this script, please contanct to PROJECT BM.

On the other hand, we also may provide sign tool for the cp images in windows platform, but not recommend to use.

9.3. Enable/disable secure boot

If require to enable secureboot for specific board(for example: sp9832e_1h10_native-userdebug):
please change board config file
device/sprd/sharkle/sp9832e_1h10/sp9832e_1h10_native.mk
Add :
BOARD_SECBOOT_CONFIG := true

Also you also can disable secureboot in following step:
please change board config file device/sprd/sharkle/sp9832e_1h10/sp9832e_1h10_native.mk
Change:
BOARD_SECBOOT_CONFIG := true =====>BOARD_SECBOOT_CONFIG := false

Note: The operation to enable/disable secureboot on other board is similar to the above method in the other project.