Wireless Module Expert

Quectel Wireless Solutions Co.,Ltd.

SC20_Secboot_签名操作流程说明

目录

本文的目录建立可以参考文档 “源码&编译&烧写方式”文档中的章节--Win7 上创建工作目录

1.编译生成 image

参考--”源码&编译&烧写方式”文档，该文档有介绍 android linux 和 SBL 部分的编译。

1.1 按照“1.4. Win7 上创建工作目录”章节建立对应工作目录

1.2 android 部分源码，参考“2.6 Android 编译”章节，编译 android 部分源码后，按照“1.4.Win7 上创建工作目录”章节，将 android 部分编译后的文件拷贝到 Win7 工作目录中。

1.3 如果有 SBL 部分源码，参考“3.SBL 源码和编译”编译 SBL

1.3 其他部分例如 rpm ， modem，TZ 会以编译后生成的文件形式交付给客户。

Img name	路径
SBL	boot_images\build\ms\bin\8909\emmc\sbl1.mbn
aboot	LINUX\android\out\target\product\msm8909\emmc_appsboot.mbn
mba	modem_proc\build\ms\bin\8909.genns.prod\mba.mbn
modem	modem_proc\build\ms\bin\8909.genns.prod\qdsp6sw.mbn
tz	trustzone_images\build\ms\bin\MAZAANAA\tz.mbn
rpm	rpm_proc\build\ms\bin\8909\pm8909\rpm.mbn

NON-HLOS.bin 需要在工作目录中执行 build.dat 生成，生成后的文件路径 common\build\bin\asic\NON-HLOS.bin

NON-HLOS.bin 包含了 mba.mbn，qdsp6sw.mbn，cmnlib.mbn，widerine.mbn，keymaster.mbn，wcnss.mbn。如果要使用私有 key 签名的 NON-HLOS.bin，需要先签名 mba.mbn，qdsp6sw.mbn，cmnlib.mbn，widerine.mbn，keymaster.mbn，wcnss.mbn，在执行 build.dat 生成 NON-HLOS.bin。

2.Image 签名

签名原理示意图

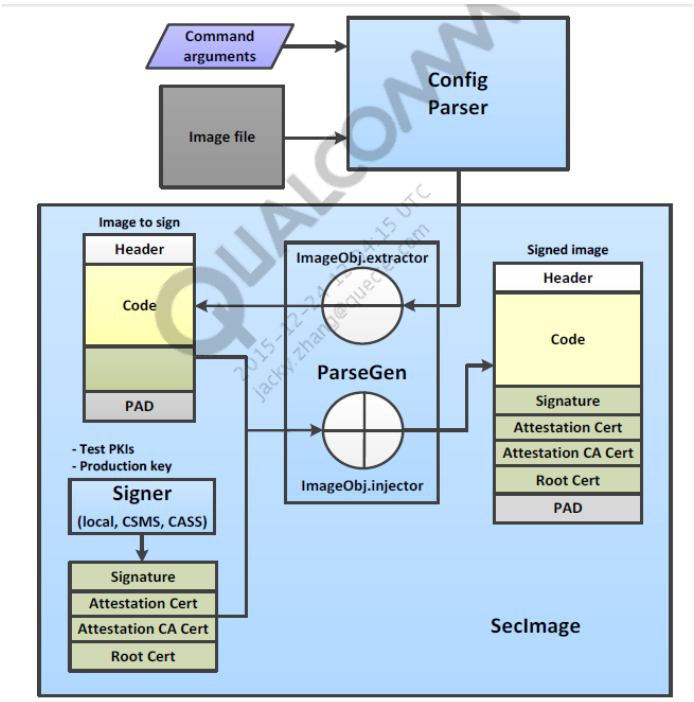


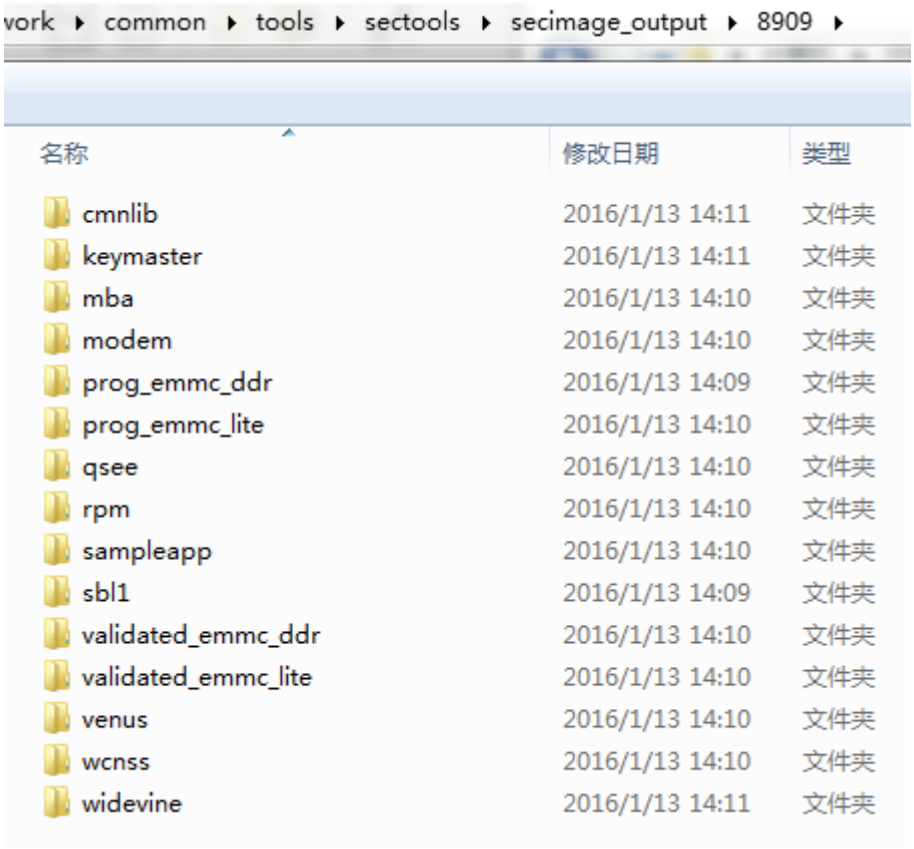
Figure 2-1 Image signing with SecImage

2.1 签名工作目录下所有 img

在 window 中 cmd 进入工作目录
执行下面命令
env.bat
再执行
python common\tools\sectools\sectools.py secimage -m . -p 8909 -sa
注意执行命令时的命令路径。

```
E:\win7_work_package\work>python common\tools\sectools\sectools.py secimage -m .
 -p 8909  -sa
          半:
```

在 common\tools\sectools\secimages_output\8909\下生成对应的签名文件。



2.2 手动签名 image

进入代码根目录 work 下，执行
env.bat

sectools.py secimage 的使用说明如下。
python common\tools\sectools\sectools.py secimage –m <meta_build_path> -g <sign_id> –p <platform> -i <file_path> -o <output_dir> -c <config_file_path> - <operation_short>

<meta_build> the meta build location 即work根目录
<sign_id> identifies the image format and signing attributes， 下面章节介绍签名不同image时会使用不同的sign_id，也可以自己查看8909_secimage.xml文件，该文件有介绍不同image的sign_id。
<platform> the platform for the set of config files to be used 在SC20中即8909。
<operation> can be one of the following

- Integrity check (t)– add hash table segment
- Sign (s)– Process the image for signing
- Encryption (e)– Process the image for encryption
- Validate (a)– Validate signed image

可以使用 sectools.py secimage 方法可以签名多种 image，例如 sbl,rpm, aboot, mba。

2.2.1 签名 sbl1

执行下面命令签名 sbl1

python common\tools\sectools\sectools.py secimage -i boot_images\build\ms\bin\8909\emmc\sbl1.mbn -c common\tools\sectools\config\8909\8909_secimage.xml -g sbl1 –sa

```
E:\win7_work_package\work>python common\tools\sectools\sectools.py  secimage  -i
 boot_images\build\ms\bin\8909\emmc\unsigned\sbl1.mbn -c common\tools\sectools\c
onfig\8909\8909_secimage.xml -g sbl1 -sa
```

在 common\tools\sectools\secimage_output\8909\sbl1 下生成 sbl1.mbn

名称	输入输出	路径
sectools.py secimage	签名命令	common\tools\sectools\
sbl1.mb	输入 待签名的 sbl	-i boot_images\build\ms\bin\8909\emmc \sbl1.mbn
8909_secimage.xml	输入 签名 key 配置文件	-c common\tools\sectools\config\8909\8909 _secimage.xml
sbl1	输入 签名文件类型	-g sbl1
-sa	指明该命令签名 img 同时验证签名后的 img	
	输出文件	默认输出路径 common\tools\sectools\secimage_output\8909\sbl1

2.2.2 签名 rpm

执行下面命令签名 sbl1

python common\tools\sectools\sectools.py secimage -i rpm_proc\build\ms\bin\8909\pm8909\rpm.mbn -c common\tools\sectools\config\8909\8909_secimage.xml -g rpm –sa

```
E:\win7_work_package\work>python common\tools\sectools\sectools.py  secimage  -i
 rpm_proc\build\ms\bin\8909\pm8909\rpm.mbn -c common\tools\sectools\config\8909\
8909_secimage.xml -g rpm -sa
```

在 common\tools\sectools\secimage_output\8909\rpm 下生成 rpm.mbn

名称	输入输出	路径
sectools.py secimage	签名命令	common\tools\sectools\
rpm.mbn	输入 待签名的 img	-i rpm_proc\build\ms\bin\8909\pm8909\rpm .mbn
8909_secimage.xml	输入 签名 key 配置文件	-c common\tools\sectools\config\8909\8909 _secimage.xml
sbl1	输入 签名文件类型	-g rpm
-sa	指明该命令签名 img 同时验证签名后的 img	
	输出文件	默认输出路径 common\tools\sectools\secimage_output\8

		909\rpm

2.2.3 签名 aboot

执行下面命令签名 sbl1

python common\tools\sectools\sectools.py secimage -i LINUX\android\out\target\product\msm8909\emmc_appsboot.mbn -c common\tools\sectools\config\8909\8909_secimage.xml -g appsbl -sa

```
E:\win7_work_package\work>python common\tools\sectools\sectools.py  secimage  -i
 rpm_proc\build\ms\bin\8909\pm8909\rpm.mbn -c common\tools\sectools\config\8909\
8909_secimage.xml -g rpm -sa
```

在 common\tools\sectools\secimage_output\8909\appsbl 下生成 emmc_appsboot.mbn

名称	输入输出	路径
sectools.py secimage	签名命令	common\tools\sectools\
emmc_appsboot.mbn	输入 待签名的 img	-i LINUX\android\out\target\product\msm8909\emmc_appsboot.mbn
8909_secimage.xml	输入 签名 key 配置文件	-c common\tools\sectools\config\8909\8909_secimage.xml
appsbl	输入 签名文件类型	-g appsbl
-sa	指明该命令签名 img 同时验证签名后的 img	
	输出文件	默认输出路径 common\tools\sectools\secimage_output\8909\appsbl

2.2.4 签名 MBA

执行下面命令签名 MBA

python common\tools\sectools\sectools.py secimage -i modem_proc\build\ms\bin\8909.genns.prod\mba.mbn -c common\tools\sectools\config\8909\8909_secimage.xml -g mba -sa

```
E:\win7_work_package\work>python common\tools\sectools\sectools.py  secimage  -i
 modem_proc\build\ms\bin\8909.genns.prod\mba.mbn -c common\tools\sectools\config
\8909\8909_secimage.xml -g mba -sa
```

在 common\tools\sectools\secimage_output\8909\mba 下生 mba.mbn

名称	输入输出	路径
sectools.py secimage	签名命令	common\tools\sectools\
mba.mbn	输入 待签名的 img	-i modem_proc\build\ms\bin\8909.genns.prod\mba.mbn
8909_secimage.xml	输入 签名 key 配置文件	-c common\tools\sectools\config\8909\8909_secimage.xml
mba	输入 签名文件类型	-g mba

-sa	指明该命令签名 img 同时验证签名后的 img	
	输出文件	默认输出路径 common\tools\sectools\secimage_output\8909\mba

2.2.4 签名 Modem

执行下面命令签名 modem

python common\tools\sectools\sectools.py secimage -i modem_proc\build\ms\bin\8909.genns.prod\qdsp6sw.mbn -c common\tools\sectools\config\8909\8909_secimage.xml -g modem -sa

```
E:\win7_work_package\work>python common\tools\sectools\sectools.py  secimage  -i
 modem_proc\build\ms\bin\8909.genns.prod\mba.mbn -c common\tools\sectools\config
\8909\8909_secimage.xml -g mba -sa
```

在 common\tools\sectools\secimage_output\8909\modem 下生 modem.mbn

名称	输入输出	路径
sectools.py secimage	签名命令	common\tools\sectools\
qdsp6sw.mbn	输入 待签名的 img	-i modem_proc\build\ms\bin\8909.genns.prod\ qdsp6sw.mbn
8909_secimage.xml	输入 签名 key 配置文件	-c common\tools\sectools\config\8909\8909_secimage.xml
modem	输入 签名文件类型	-g modem
-sa	指明该命令签名 img 同时验证签名后的 img	
	输出文件	默认输出路径 common\tools\sectools\secimage_output\8909\ modem

2.2.5 签名 linux boot

本节需要在 ubuntu 环境的 android 源码下操作，完整编译 android 源码，进入源码根目录。

默认的签名 boot.img recovery.img 的 key 在 build/target/product/verity.mk 定义。

PRODUCT_VERITY_SIGNING_KEY := build/target/product/security/verity

即使用的签名文件在 build/target/product/security/下。

公钥证书部分保存在 aboot 的头文件中，通过编译 aboot，编译到系统中。aboot 检验 boot.img。

2.2.5.1 确认 aboot 验证功能开启

在 build/target/product/verity.mk 设置 PRODUCT_SUPPORTS_VERITY := true，默认情况已经设置为 true。

2.2.5.2 签名 boot

执行如下命令

out/host/linux-x86/bin/boot_signer /boot out/target/product/msm8909/boot.img build/target/product/security/verity.pk8 build/target/product/security/verity.x509.pem

out/target/product/msm8909/boot_signed.img

out/host/linux-x86/bin/boot_signer	签名工具
/boot	
out/target/product/msm8909/boot.img	需要签名的文件
build/target/product/security/verity.pk8	私钥
build/target/product/security/verity.x509.pem	签名证书
out/target/product/msm8909/boot_signed.img	签名后的生成文件

编译源码后的 out/target/product/msm8909/boot.img，默认情况下，已经使用 build/target/product/security/verity*签名过了。

2.2.5.3 aboot 中的 public key

aboot 会去判断签名后的 boot.img 时候合法，所有 aboot.img 中会存储公钥部分。

aboot 中的公钥存在于 bootable/bootloader/lk/platform/msm_shared/include/oem_keystore.h 头文件中，const unsigned char OEM_KEYSTORE[] = {…}

2.2.5.4 oem_keystore.h 的制作

参考 “android linux 证书生成”生成 keystore.img

执行命令

xxd –i keystore.img,

```
barret@barret-pc:/data/sc20/LINUX/android$ xxd -i keystore.img
unsigned char keystore_img[] = {
  0x30, 0x82, 0x06, 0x4f, 0x02, 0x01, 0x00, 0x30, 0x82, 0x01, 0x1f, 0x30,
  0x82, 0x01, 0x1b, 0x30, 0x0b, 0x06, 0x09, 0x2a, 0x86, 0x48, 0x86, 0xf7,
  0x0d, 0x01, 0x01, 0x0b, 0x30, 0x82, 0x01, 0x0a, 0x02, 0x82, 0x01, 0x01,
```

新建 oem_keystore.h 头文件

添加内容如下

```
#ifndef __OEM_KEYSTORE_H
#define __OEM_KEYSTORE_H
const unsigned char OEM_KEYSTORE[] = {

#endif
```

将上一步操作中的十六进制数据复制到 OEM_KEYSTORE[]中，

用这个 oem_keystore.h 替换 bootable/bootloader/lk/platform/msm_shared/include/oem_keystore.h 文件。

aboot 根据 OEM_KEYSTORE[]中 key 的信息验证 boot.img 是否是合法签名。

2.2.5.6 重新编译 aboot

make aboot –j8

2.2.6 签名 recovery.img

执行如下命令

out/host/linux-x86/bin/boot_signer /recovery out/target/product/msm8909/recovery.img build/target/product/security/verity.pk8 build/target/product/security/verity.x509.pem out/target/product/msm8909/recovery_ signed.img

out/host/linux-x86/bin/boot_signer	完整编译源码后，会在该路径生成签名工具

/ recovery	
out/target/product/msm8909/ recovery.img	需要签名的文件
build/target/product/security/verity.pk8	签名私钥
build/target/product/security/verity.x509.pem	签名证书
out/target/product/msm8909/ recovery _signed.img	签名后的生成文件

编译源码后的 out/target/product/msm8909/ recovery.img，默认情况下，已经使用 build/target/product/security/verity*签名过了。

3.升级 image

3.1 Fastboot 升级签名 iamge

执行 fastboot flash <partition> <filepath>

partition	File path
Sbl1	fastboot flash sbl1 <sbl1.mbn file path>
rpm	fastboot flash rpm <rpm.mbn file path>
aboot	fastboot flash aboot < emmc_appsbl.img file path>
boot	fastboot flash boot <boot.img file path>
recovery	fastboot flash recovery < recovery.img file path>
modem	fastboot flash modem < NON-HLOS.bin file path>
tz	fastboot flash tz <tz file path>

3.2 QFIL 升级签名 iamge

3.2.1 替换签名后的文件

将签名后的 image 覆盖 编译生成目录下的 image。

按照签名 image 的方法，获得签名后的文件，替换原始文件。

签名后文件路径	源文件路径
common\tools\sectools\secimage_output\8909\cmnlib\cmnlib.mbn	trustzone_images/build/ms/bin/MAZAANAA/cmnlib.mbn
common\tools\sectools\secimage_output\8909\ widevine\ widevine.mbn	trustzone_images/build/ms/bin/MAZAANAA/widevine.mbn
common\tools\sectools\secimage_output\8909\ keymaster\ keymaster.mbn	trustzone_images/build/ms/bin/MAZAANAA/keymaster.mbn
common\tools\sectools\secimage_output\8909\ wcnss\wcnss.mbn	wcnss_proc/build/ms/bin/SCAQMAZ/reloc\wcnss.mbn
common\tools\sectools\secimage_output\8909\ mba \ mba.mbn	modem_proc/build/ms/bin/8909.genns.prod\mba.mbn
common\tools\sectools\secimage_output\8909\modem\modem.mbn	modem_proc/build/ms/bin/8909.genns.prod\qdsp6sw.mbn
common\tools\sectools\secimage_output\8909\ venus \venus.*	LINUX/android/vendor/qcom/proprietary/prebuilt_HY11/target/product/msm8909/system/etc/firmware/venus.*
common\tools\sectools\secimage_output\8909\appsbl\ emmc_appsboot.mbn	LINUX\android\out\target\product\msm8909\emmc_appsboot.mbn
common\tools\sectools\secimage_output\8909\qsee\ tz.mbn	trustzone_images\build\ms\bin\MAZAANAA\tz.mbn
common\tools\sectools\secimage_output\8909\rpm\rpm.mbn	rpm_proc\build\ms\bin\8909\pm8909\rpm.mbn

common\tools\sectools\secimage_output\8909\sbl1\ sbl1.mbn	boot_images\build\ms\bin\8909\emmc\sbl1.mbn 注意检查 Qfil 实际使用的文件，我这边从 log 中看到 flatmeta build 打包实际使用的是 boot_images\build\ms\bin\8909\emmc\unsigned\sbl1.mbn
common\tools\sectools\secimage_output\8909\prog_emmc_ddr\prog_emmc_firehose_8909_ddr.mbn	boot_images\build\ms\bin\8909\emmc\unsigned\prog_emmc_firehose_8909_ddr.mbn

3.2.2 QFIL 下载包和烧写

参考“”源码&编译&烧写方式””中关于 QFIL 下载包制作和 QFILE 烧写 image。

3.3 OTA 升级签名 Image 包

复制签名后的文件 emmc_appsboot.mbn NON-HLOS.bin rpm.mbn sbl1.mbn tz.mbn 到 device/qcom/msm8909/radio/下。

具体方法参考“SC10_升级包制作”文档中 Modem 等非 HLOS 加入升级包的方法

4.FUSE 烧写

制作 sec.dat

用户如果希望使用自己的 sec.dat，可以参考修改 8909_fuseblower_OEM.xml8909_fuseblower_QC.xml8909_fuseblower_USER.xml，这些配置文件位于 \common\tools\sectools\config\8909。Sec.dat 根据这三个文件的配置生成。如果希望使用自己的 key 去生成 sec.dat。参考后面章节----“证书生成方法” 操作后,再操作本章内容。

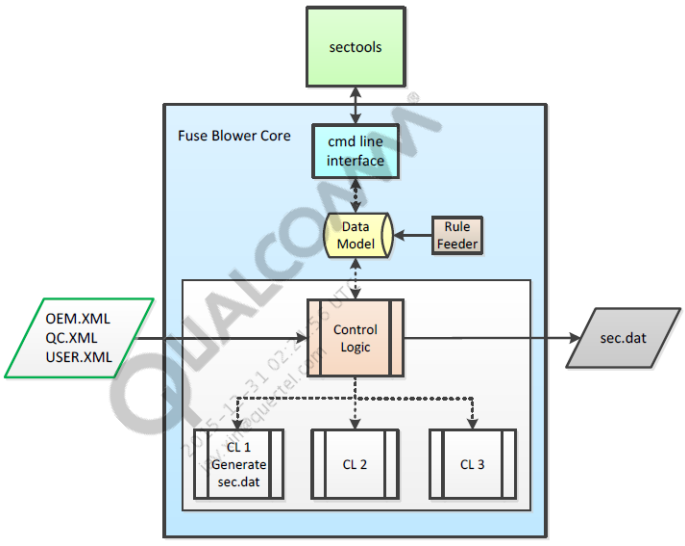
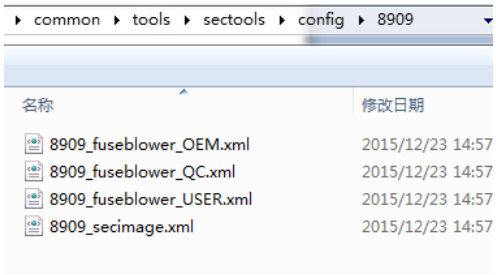


Figure 2-1 FuseBlower generate sec.dat with preconfigured XMLs



进入代码根目录 work 下，执行

python common\tools\sectools\sectools.py fuseblower –p 8909 –g -v

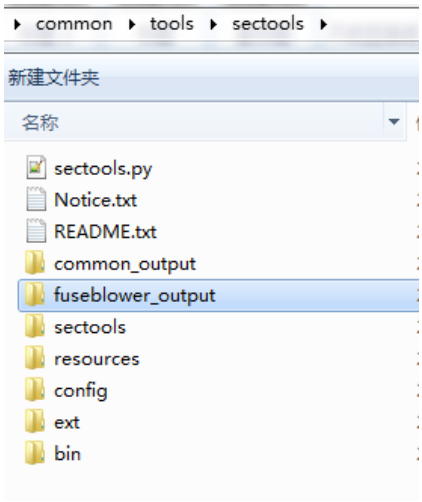
```
>python common\tools\sectools\sectools.py fuseblower -p 8909 -g
```

使用这个命令，将用\common\tools\sectools\config\8909 里面的配置文件生成 sec.dat

也可以指定详细的 QC, OEM,USER 配置文件的路径，

python common\tools\sectools\sectools.py fuseblower -e < path of xx_fuseblower_OEM.xml> -q < path of xx_fuseblower_QC.xml > -u < path of xx_fuseblower_USER.xml> –g -v

生成的 sec.dat 文件路径位于\common\tools\sectools\fuseblower_output\v1\sec.dat



4.1 将生成的 sec.dat 替换原始的 sec.dat，common\tools\sectools\resources\build\sec.dat。

制作 QFile Falt build 下载包时，将会对 common\tools\sectools\resources\build\sec.dat 进行打包.

如果需要验证，需要先刷入签名过的 img，确认可以开机正常后，再刷入生成的 esec.dat。sec.dat 只能有效输入一次。原始的 common\tools\sectools\resources\build\sec.dat 注意保存。

4.2 烧写 sec.dat

4.2.1 手动 load sec.dat

设备进入 fastboot 模式

执行 fastboot flash sec <sce.dat path>

5.证书生成方法

5.1 secure boot 的证书生成

CMD 下进入 work 工作目录,执行 env.bat

5.1.1 生成证书

进入 common\tools\sectools 路径下，
执行如下命令
$openssl genrsa -out oem_rootca.key -3 2048

$openssl req -new -key oem_rootca.key -x509 -out oem_rootca.crt -subj /C="US"/ST="CA"/L="SANDIEGO"/O="OEM"/OU="General OEM rootca"/CN="OEM ROOT CA" -days 7300 -set_serial 1 -config resources\data_prov_assets\General_Assets\Signing\openssl\opensslroot.cfg -sha256

$openssl x509 -in oem_rootca.crt -inform PEM -out qpsa_rootca.cer -outform DER

$openssl genrsa -out oem_attestca.key -3 2048

$openssl req -new -key oem_attestca.key -out oem_attestca.csr -subj /C="US"/ST="CA"/L="SANDIEGO"/O="OEM"/OU="General OEM attestation CA"/CN="OEM attestation CA" -days 7300 -config resources\data_prov_assets\General_Assets\Signing\openssl\opensslroot.cfg

$openssl x509 -req -in oem_attestca.csr -CA oem_rootca.crt -CAkey oem_rootca.key -out oem_attestca.crt -set_serial 5 -days 7300 -extfile resources\data_prov_assets\General_Assets\Signing\openssl\v3.ext -sha256

$openssl x509 -in oem_attestca.crt -inform PEM -out qpsa_attestca.cer -outform DER

在 common\tools\sectools 路径下将 oem_rootca.key 重命名为 qpsa_rootca.key ,将 oem_attestca.key 重命名为 qpsa_attestca.key
计算 Root certificate 的 hash 值
$openssl dgst -sha256 qpsa_rootca.cer | tee sha256rootcert.txt

创建 oem_certs 文件夹，将上面命令执行后生成的文件复制到下图路径下 common\tools\sectools\resources\data_prov_assets\Signing\Local\oem_certs。文件夹也可以命名为其他名称，需要和 8909_secimage.xml 中的 selected_cert_config 名称一致。
生成的证书列表如下

名称	路径	用途
qpsa_rootca.key	common\tools\sectools\resources\data_prov_assets\Signing\Local\oem_certs	root CA private key
qpsa_rootca.cer	common\tools\sectools\resources\data_prov_assets\Signing\Local\oem_certs	Root certificate.
qpsa_attestca.key	common\tools\sectools\resources\data_prov_assets\Signing\Local\oem_certs	attestation CA private key
qpsa_attestca.cer	common\tools\sectools\resources\data_prov_assets\Signing\Local\oem_certs	Attestation CA certificate

5.1.2Config.xml 文件内容修改

可以参考复制 common\tools\sectools\resources\data_prov_assets\Signing\Local\qc_dbgp_test\config.xml，在此文件上做如下修改，这个文件用于设置证书，key 等信息。

```
<METACONFIG>
    <is_mrc>False</is_mrc>

    <root_pre>True</root_pre>
    <attest_ca_pre>True</attest_ca_pre>
    <attest_pre>False</attest_pre>

    <root_cert>qpsa_rootca.cer</root_cert>
    <root_private_key>qpsa_rootca.key</root_private_key>

    <attest_ca_cert>qpsa_attestca.cer</attest_ca_cert>
    <attest_ca_private_key>qpsa_attestca.key</attest_ca_private_key>
```

<root_cert>填写当前文件夹下的 Root certificate 文件名</root_cert>
<root_private_key>填写当前文件夹下的 root CA private key 文件名</root_private_key>
<attest_ca_cert>填写当前文件夹下的 Attestation CA certificate 文件名</attest_ca_cert>
<attest_ca_private_key>填写当前文件夹下的 attestation CA private key 文件名</attest_ca_private_key>

5.1.3 修改 8909_secimage.xml

修改 common\tools\sectools\config\8909\8909_secimage.xml,将 selected_cert_config 修改为 oem_certs，oem_cert 是上一步骤中新添加 key 的文件夹名称。

```
        <selected_signer>local</selected_signer>
        <selected_encryptor></selected_encryptor>
        <selected_cert_config>oem_certs</selected_cert_config>
        <cass_capability>qti_ct_sbl_sha2</cass_capability>
```

8909_secimage.xml 文件配置签名文件。

5.1.4 修改 8909_fuseblower_USER.xml 文件

修改 common\tools\sectools\config\8909\8909_fuseblower_USER.xml 文件
修改 root_cert_hash，hash 值在
common\tools\sectools\resources\data_prov_assets\Signing\Local\oem_certs\sha256rootcert.txt 中。
SEC_BOOT1_PK_Hash_in_Fuse: set if PK HASH for SEC_BOOT1 is in fuse
SEC_BOOT2_PK_Hash_in_Fuse: set if PK HASH for SEC_BOOT2 is in fuse
SEC_BOOT3_PK_Hash_in_Fuse: set if PK HASH for SEC_BOOT3 is in fuse

根据下图内容做相应修改。

```
<module id="SECURITY_CONTROL_CORE">
    <entry ignore="false">
        <description>contains the OEM public key hash as set by OEM</description>
        <name>root_cert_hash</name>
        <value>53fb94dae6880a07ec492c540581de5ce36cde61ac9739cc1dadc3bdce71b97d</value>
    </entry>
    <entry ignore="true">
        <description>SHA256 signed root cert to generate root hash</description>
        <name>root_cert_file</name>
        <value>./../../resources/data_prov_assets/Signing/Local/oem_certs/qpsa_rootca.cer</value>
    </entry>
    <entry ignore="false">
        <description>PK Hash is in Fuse for SEC_BOOT1 : Apps</description>
        <name>SEC_BOOT1_PK_Hash_in_Fuse</name>
        <value>true</value>
    </entry>
    <entry ignore="false">
    <entry ignore="false">
    <entry ignore="false">
        <description>PK Hash is in Fuse for SEC_BOOT2 : MBA</description>
        <name>SEC_BOOT2_PK_Hash_in_Fuse</name>
        <value>true</value>
    </entry>
    <entry ignore="false">
    <entry ignore="false">
    <entry ignore="false">
        <description>PK Hash is in Fuse for SEC_BOOT3 : MPSS</description>
        <name>SEC_BOOT3_PK_Hash_in_Fuse</name>
        <value>true</value>
    </entry>
    <entry ignore="false">
```

root_cert_hash 填写 root certificate 的 hash 值，这个值生成 key 的步骤有操作过，可以在 sha256rootcert.txt 中读取

5.1.5 修改 8909_fuseblower_QC.xml 文件

QFPROM_RAW_OEM_SEC_BOOT_ROWn_LSB				
Bits	**Name**	**Description**		**Used for**
23:16	SEC_BOOT3	Bits	Name	For authentication information of MPSS (modem) image
		7	RESERVED	
		6	USE_SERIAL_NUM	
		5	AUTH_EN	
		4	PK_HASH_IN_FUSE	
		3:0	ROM_PK_HASH_INDEX	
15:8	SEC_BOOT2	Bits	Name	For authentication information of MBA (modem boot authenticator) image
		7	RESERVED	
		6	USE_SERIAL_NUM	
		5	AUTH_EN	
		4	PK_HASH_IN_FUSE	
		3:0	ROM_PK_HASH_INDEX	
7:0	SEC_BOOT1	Bits	Name	For authentication information of all other images: ▪ SBL ▪ RPM firmware ▪ TrustZone kernel ▪ APPSBL ▪ TrustZone Application images ▪ WCNSS (WLAN/RIVA) ▪ LPASS ▪ Sensors (DSPS) ▪ Video (Venus) ▪ Emergency downloader ▪ Debug watchdog
		7	RESERVED	
		6	USE_SERIAL_NUM	
		5	AUTH_EN	
		4	PK_HASH_IN_FUSE	
		3:0	ROM_PK_HASH_INDEX	


```
<fuse_region id="QFPROM_RAW_OEM_SEC_BOOT">
    <description></description>
    <fuse ignore="false" n="0">
        <address>0x00058098</address>
        <operation>BLOW</operation>
        <field id="SEC_BOOT1">
            <description></description>
            <owner>QC</owner>
            <value>0x20</value>
            <bits>7:0</bits>
        </field>
        <field id="SEC_BOOT2">
            <description></description>
            <owner>QC</owner>
            <value>0x20</value>
            <bits>15:8</bits>
        </field>
        <field id="SEC_BOOT3">
            <description></description>
            <owner>QC</owner>
            <value>0x20</value>
            <bits>23:16</bits>
        </field>
```

根据需要,设置 QFPROM_RAW_OEM_SEC_BOOT 中的 value 值意义,

23:16	SEC_BOOT3	OEM fuse settings for this secure boot configuration. This can be overridden with the Qualcomm settings if necessary. Bit 7: Reserved Bit 6: Use Serial Num for secure boot authentication (0: Use OEM ID (Default), 1: Use Serial Num) Bit 5: Authentication Enable (0: no auth, 1: auth required) Bit 4: PK Hash in Fuse (0: SHA-256 hash of root cert is ROM, 1: SHA-256 hash of root cert to use is in OEM_PK_HASH) Bits 3-0: ROM PK Hash Index (If PK Hash in Fuse is 0, then this index selects which of 16 keys in ROM to use)
15:8	SEC_BOOT2	OEM fuse settings for this secure boot configuration. This can be overridden with the Qualcomm settings if necessary. Bit 7: Reserved Bit 6: Use Serial Num for secure boot authentication (0: Use OEM ID (Default), 1: Use Serial Num) Bit 5: Authentication Enable (0: no auth, 1: auth required) Bit 4: PK Hash in Fuse (0: SHA-256 hash of root cert is ROM, 1: SHA-256 hash of root cert to use is in OEM_PK_HASH) Bits 3-0: ROM PK Hash Index (If PK Hash in Fuse is 0, then this index selects which of 16 keys in ROM to use)
7:0	SEC_BOOT1	OEM fuse settings for this secure boot configuration. This can be overridden with the Qualcomm settings if necessary. Bit 7: Reserved Bit 6: Use Serial Num for secure boot authentication (0: Use OEM ID (Default), 1: Use Serial Num) Bit 5: Authentication Enable (0: no auth, 1: auth required) Bit 4: PK Hash in Fuse (0: SHA-256 hash of root cert is ROM, 1: SHA-256 hash of root cert to use is in OEM_PK_HASH) Bits 3-0: ROM PK Hash Index (If PK Hash in Fuse is 0, then this index selects which of 16 keys in ROM to use)

如果希望开启 SEC_BOOTn，设置相应为的 value，默认值为 0x20，如果不希望开启验证，将值改为 0x00 即可。

例如，如果希望验证 mba 和 modem，不希望验证 sbl1 等其他 image，可以设置 SEC_BOOT1 的 value 值为 0x00 即可，SEC_BOOT2 和 SEC_BOOT3 的 value 值保持不变 0x20。

如果只希望验证 SEC_BOOT1 中的 image，设置 SEC_BOOT1 的 value 值为 0x20，SEC_BOOT2 和 SEC_BOOT3 的 value 值修改为 0x00 即可。

5.2 Android linux 下的证书生成

本节的操作环境为 Ubuntu 下，需要成功编译 android 源码后，在 android 源码路径下，执行以下操作。

5.2.1 生成 key

进入 android 源码路径

执行下面的命令，生成 OEM keypair

development/tools/make_key mykey '/C=US/ST=California/L=Mountain View/O=Android/OU=Android/CN=Android/emailAddress=android@android.com'

提示输入密码时候，可以按 enter 键，忽略输入密码。

在 android 源码路径下生成 mykey.pk8 和 mykey.x509.pem

5.2.2 生成 verity_key

执行下面的命令，生成 verity key 的相关工具，生成文件位于 out 路径下。
source build/envsetup.sh
lunch msm8909-userdebug
mmm system/extras/verity/
执行下面的命令
out/host/linux-x86/bin/generate_verity_key -convert mykey.x509.pem verity_key
会在 android 源码路径下生成 verity_key

```
barret@barret-pc:/data/sc20/LINUX/android$ ls
mykey.pk8  mykey.x509.pem  verity_key.pub
```

执行下面的命令，生成 keystore.img
java -Xmx512M -jar out/host/linux-x86/framework/KeystoreSigner.jar mykey.pk8 mykey.x509.pem keystore.img mypub.der
会在 android 源码路径下生成文件 keystore.img

5.2.3 替换原始 key

build/target/product/security/下有如下文件
verity.pk8 -- private key used to sign boot.img and system.img
verity.x509.pem -- certificate include pub key
verity_key -- pub key used in dm verity for system image
verity.pk8 and verity.x509.pem are used in signing the boot.img

将 mykey.pk8 替换 build/target/product/security/verity.pk8
mykey.x509.pem 替换 build/target/product/security/verity.x509.pem
verity_key.pub 替换 build/target/product/security/verity_key，


源码&编译&烧写方式V0.9_20160125.zip


SC10_升级包制作及原理V4.0_Law_20160125-.zip

6.附录

参考文档：80-NU861-1_QUALCOMM ANDROID SECURITY FEATURES

80-NL239-45_Application Note- Enabling Secure Boot

80-NM248-1_SECTOOLS- SECIMAGE TOOL USER GUIDE

80-NM248-3_FUSEBLOWER TOOL USER GUIDE

参考 Solution 00031142

Question:How to enable verfied boot and generate OEM's own key-pair
Answer:
1.To enabled build system to sign boot.img and recovery.img
please check
build/target/product/verity.mk
PRODUCT_SUPPORTS_VERITY := true
it may also enabled in device makefile:
e.g.
  device/qcom/msm8992/msm8992.mk
  PRODUCT_SUPPORTS_VERITY := true

This will enable LK build with VERIFIED_BOOT, then will do verification when load kernel.
bootable/bootloader/lk/AndroidBoot.mk
ifeq ($(PRODUCTS.$(INTERNAL_PRODUCT).PRODUCT_SUPPORTS_VERITY),true)
  VERIFIED_BOOT := VERIFIED_BOOT=1
else
  VERIFIED_BOOT := VERIFIED_BOOT=0
endif

2.The default dev key include public key and private key are put at build/target/product/security/
dev key is used to sign boot and recovery images, and the verity metadata table.
it is defined in build/target/product/verity.mk
PRODUCT_VERITY_SIGNING_KEY := build/target/product/security/verity

build/target/product/security/
verity.pk8 -- private key used to sign boot.img and system.img
verity.x509.pem -- certificate include pub key
verity_key -- pub key used in dm verity for system image

verity.pk8 and verity.x509.pem are used in signing the boot.img
see build/core/Makefile
$(INSTALLED_BOOTIMAGE_TARGET): $(MKBOOTIMG) $(INTERNAL_BOOTIMAGE_FILES)
$(BOOT_SIGNER) $(BOOTIMAGE_EXTRA_DEPS)
	$(call pretty,"Target boot image: $@")
	$(hide) $(MKBOOTIMG) $(INTERNAL_BOOTIMAGE_ARGS) $(BOARD_MKBOOTIMG_ARGS) --
output $@
	$(BOOT_SIGNER) /boot $@
$(PRODUCTS.$(INTERNAL_PRODUCT).PRODUCT_VERITY_SIGNING_KEY).pk8
$(PRODUCTS.$(INTERNAL_PRODUCT).PRODUCT_VERITY_SIGNING_KEY).x509.pem $@
	$(hide) $(call assert-max-image-size,$@,$(BOARD_BOOTIMAGE_PARTITION_SIZE))

some old release you see only 2 key file
verity_private_dev_key -- private key used to sign boot.img and system.img
verity_key -- pub key used in dm verity for system image

3. To generate OEM's new keypair
in your linux machine, make sure openssl is installed and openssl version is new enough
>openssl version
OpenSSL 1.0.1 14 Mar 2012
e.g.
cd your android home
>development/tools/make_key mykey '/C=US/ST=California/L=Mountain
View/O=Android/OU=Android/CN=Android/emailAddress=android@android.com'
don't input password, then mykey.pk8 and mykey.x509.pem will be generated in current folder.

4. To generate verity key for dm verity feature.
first make sure host tool are already built:
. build/envsetup.sh
lunch

mmm system/extras/verity/
the generated host app exist in out/host/linux-x86/framework/

system/extra/verity/generate_verity_key.c
generate_verity_key <path-to-key> | -convert <path-to-x509-pem> <path-to-key>

out/host/linux-x86/bin/generate_verity_key -convert mykey.x509.pem verity_key
then you can copy mykey.pk8, mykey.x509.pem, verity_key.pub to build/target/product/security/
rename them to verity.pk8, verity.x509.pem , verity_key to overwrite default dev key.

5. To generate keystore
if you want to replace default keystore with your own key.
in LK, there are 2 keystore, one is oem_keystore, the other is user_keystore
oem_keystore is built into LK, it is defined in oem_keystore.h
user_keystore is read from "keystore" partition.
LK will use oem keystore to verify the keystore partition, if verify passed, it read it to user_keystore and use it to verify boot.img and recovery.img.
google has removed user_keystore in later release, then don't need keystore partition, so you need to check your baseline code to choose.

To generate new keystore:
keystore_signer <privatekey.pk8> <certificate.x509.pem> <outfile> <publickey0.der>
system/extras/verity/keystore_signer is a script to call a java app.
so you can call it directly
first generate pub key
>openssl rsa -in mykey.pk8 -inform DER -pubout -outform DER -out mypub.der
then call java app
>java -Xmx512M -jar out/host/linux-x86/framework/KeystoreSigner.jar mykey.pk8 mykey.x509.pem keystore.img mypub.der
on old release with no pk8 key format:
    keystore_signer <PRIVATE_KEY> <KEYSTORE_IMG> <RSA_PUBLIC_KEY_DER>
>java -Xmx512M -jar out/host/linux-x86/framework/KeystoreSigner.jar verity_private_dev_key keystore.img mypub.der
KeystoreSigner.jar and other java app in out/host/linux-x86/framework/ can be generated by step 4.

now you got keystore.img
then flash keystore.img to keystore partition:
fastboot flash keystore keystore.img
if there is no keystore partition, you can skip this

To use same key for oem_keystore:
Generate oem_keystore.h for the Android boot loader (LK):
LK needs a header file (oem_keystore.h) that was generated with the contents of the OEM
a. Run the following code to generate this header file:
```
    function generate_oem_keystore_h()
    {
      echo \#ifndef __OEM_KEYSTORE_H
      echo \#define __OEM_KEYSTORE_H
  xxd -i $1 | sed -e 's/unsigned char .* = {/const unsigned char
OEM_KEYSTORE[] = {/g' -e 's/unsigned int .* =.*;//g'
      echo \#endif
    }
    generate_oem_keystore_h keystore.img > oem_keystore.h
```
b. Copy the oem_keystore.h file to the following location:
    bootable/bootloader/lk/platform/msm_shared/include

6.The code call path in LK code
boot_linux_from_mmc
boot_verifier_init
verify_signed_bootimg-> boot_verify_image-> verify_image_with_sig