高通平台二段式打包签名熔丝版本不开机问题

一、问题背景

平台：高通 `SM6225`
项目：`T726`
配置：将需要签名的镜以及签名工具单独打包给客户进行二段式签名(Secure Boot V3)熔丝版本。将签名返回的镜像重新构建刷机镜像，烧录所有镜像(`xbl.elf`、`tz.mbn`、`hyp.mbn`、`modem` 等）后，设备在熔丝下无法启动，且串口也没有log输出。

二、实现过程和问题回顾

（一）实现过程

将签名环境及待签名的镜像统一打包提供给客户，客户根据指引生成专属密钥，并将密钥置于签名包中，随后执行二次签名脚本对镜像进行签名操作。签名完 成后，客户将已签名的镜像返回，由我方重新打包生成完整的刷机镜像，最终烧录至设备以完成熔丝过程。

- **高通需要签名的基线列表**
  - smplap64.mbn
  - smplap32.mbn
  - prog_firehose_lite.elf
  - prog_firehose_ddr.elf
  - xbl_config.elf
  - xbl.elf
  - uefi_sec.mbn
  - tz.mbn
  - storsec.mbn
  - sec.elf
  - rpm.mbn
  - qupv3fw.elf
  - keymint.mbn
  - imagefv.elf
  - hyp.mbn
  - devcfg.mbn
  - abl.elf
  - NON-HLOS.bin
- **将这些需要签名的镜像和配置在编译的时候打包**
  - 脚本sign_pack.sh

```
  function package_sign_img(){
  bash ${ANDROID_SUBSYSTEM_DIR}/Divar.LA.3.0.1/common/sign_pack.sh
${CUSTOM_PROJECT_VARIANT} ${CUSTOM_PROJECT_NAME} }
```

（二）问题现象

执行完上述过程后，将签名的镜像刷入设备进行熔丝，开不了机并且串口log无消息吐出

- **归纳出来的现象：**
  - 烧录成功后设备黑屏无响应
  - 串口无任何打印，包括 `xbl` 阶段 log
  - QFIL 验证能识别设备处于 9008 模式
  - 更换未熔丝机台，刷入未签名镜像则可正常启动
  - 客户签名镜像具备合法签名，从结构上看未损坏

（三）预期目标

- 实现符合 Sec-Boot签名要求的刷机流程
- 二段式签名版本支持熔丝设备正常启动
- 提供稳定、安全的签名包供客户量产使用

三、处理过程分析

（一）步骤一：初步分析定位问题

- 首先分析未能启动阶段，串口无打印说明 **启动链条在 XBL 前已被阻断**；
- 结合 Sec-Boot 特性判断可能是签名验证失败导致 Boot ROM 阻止启动；
- 检查 `xbl.elf` 签名合法性；
- 比对客户返回镜像与原始 unsigned 镜像结构、offset、header、hash 值，确认是否被破坏；
- 使用 `secimage` 工具对原始镜像与签名镜像结构进行对比分析
- 使用openssl工具验证公钥私钥是否匹配
- 检查签名的xml文件配置

（二）步骤二：深入排查与尝试

没有客户的key，设备没有任何log输出，只能自己生成一套key任何从启动链开始逐步分析；

- **1、按照高通平台的要求使用 `openssl` 生成一套自己的key以及证书**

```
openssl genrsa -out qpsa_rootca.key 2048
openssl req -new -sha256 -key qpsa_rootca.key -x509 -out rootca_pem.crt \
  -subj "/C=US/ST=California/L=San Diego/OU=General Use Test Key (for testing 13
only)/OU=CDMA Technologies/O=QUALCOMM/CN=QCT Root CA 1" \
  -days 7300 -set_serial 1 -config opensslroot.cfg \
  -sigopt rsa_padding_mode:pss -sigopt rsa_pss_saltlen:-1 -sigopt digest:sha256
```

```
openssl x509 -in rootca_pem.crt -inform PEM -out qpsa_rootca.cer -outform DER
openssl x509 -text -inform DER -in qpsa_rootca.cer

openssl genrsa -out qpsa_attestca.key 2048
openssl req -new -key qpsa_attestca.key -out attestca.csr \
  -subj "/C=US/ST=CA/L=San Diego/OU=CDMA Technologies/O=QUALCOMM/CN=QUALCOMM
Attestation CA" \
  -config opensslroot.cfg
openssl x509 -req -in attestca.csr -CA rootca_pem.crt -CAkey qpsa_rootca.key \
  -out attestca_pem.crt -set_serial 5 -days 7300 -extfile v3.ext \
  -sigopt rsa_padding_mode:pss -sigopt rsa_pss_saltlen:-1 -sigopt digest:sha256
openssl x509 -inform PEM -in attestca_pem.crt -outform DER -out qpsa_attestca.cer

openssl dgst -sha384 qpsa_rootca.cer > sha384rootcert.txt
```

将生成的key放入签名包里，执行二次签名脚本，待签名成功后组包进行刷机熔丝，发现和上述一致的效果，证明不是客户key的问题，下一步对签名环境配置分析

- **2、检查 secimage config xml文件中需要配置的部分：**

  `path`：\config\divar\divar_fuseblower_USER.xml XML tags.

```
<entry ignore="false">
<description>SHA384 hash of the root certificate used for
signing</description>
<name>root_cert_sha384_hash0_file</name>
<!--配置成指定的key文件，EMOKEYS-->
<value>./../../resources/data_prov_assets/Signing/Local/OEMKEYS/
qpsa_rootca.cer</value>
</entry>
Or
<entry ignore="false">
<description>contains the OEM public key hash as set by OEM</
description>
<name>root_cert_hash0</name>
<!--配置成sha384rootcert.txt里面对应的哈希值-->
<value>bdaf51b59ba21d8a243792c0e183e88bddd369ccca58bc792a3e4c22eff329e8a
8c7 2d449559cd5f09ebfa5c7bf398c0</value>
</entry>

<entry ignore="false">
<description>The OEM hardware ID</description>
<name>oem_hw_id</name>
<!--配置成自己公司向高通申请到的OEM_ID-->
<value>0x0000</value> //Add the QTI OEM_ID.//
</entry>
<entry ignore="false">
<description>The OEM product ID</description>
<name>oem_product_id</name>
<!--可根据项目需求自选-->
<value>0x0000</value> //The OEM can select any value.//
</entry>
```

`path`:/config/divar/divar_secimage.xml

```
<general_properties>
<selected_signer>local_v2</selected_signer>
<selected_encryptor>unified_encryption_2_0</selected_encryptor>
<!--Set the path of the root key and certificate configuration for image signing-->
<selected_cert_config>OEM-KEYS</selected_cert_config>
<cass_capability>secboot_sha2_pss_subca1</cass_capability>
<key_size>2048</key_size>
<exponent>65537</exponent>
<!--Set OEM_ID and Model_ID-->
<oem_id>0x0000</oem_id>
<model_id>0x0000</model_id>
```

`path`:/config/divar/divar_fb_kp_secimage.xml

```
<oem_id>0x000</oem_id>
<model_id>0x0000</model_id>
```

经过排查分析，发现配置文件中的`oem_id`未配置成高通认证的ID；经过查阅知道我们公司向高通申请到的`oem_id`为`0x0235`,然后将配置文件中的`oem_id`配置成`0x0235`再对签名包重新运行签名脚本，组包后刷机熔丝，发现能正常开机起来！

- **3、更新配置文件，重新打包给客户签名**
  与客户的研发对接，拿到客户向高通申请得到的`oem_id`然后更新到配置文件里面，更新签名包给客户重新签名。将签名好的包重新组包，刷机熔丝，发现还是不能开机串口也没有log输出

（三）步骤三：进一步分析与优化

对比客户和自己生成的`key`，使用自己的`key`签名符合`PBL`校验要求,可以正常开机，同样的步骤替换成客户的`key`启动链条在`XBL`前已被阻断，可以推断出客户生成的`key`不符合高通平台的要求！

- **1、使用`openssl`工具对生成的`key`进行校验**

```
公钥证书的MD5：
openssl x509 -in qpsa_rootca.cer -inform der -noout -modulus | openssl md5
公钥的MD5：
openssl rsa -in qpsa_rootca.key -noout -modulus | openssl md5

私钥证书的MD5：
openssl x509 -in qpsa_attestca.cer -inform der -noout -modulus | openssl md5
私钥的MD5：
openssl rsa -in qpsa_attestca.key -noout -modulus | openssl md5
```

与客户研发对接，使用上面的指令对客户生成的`key`进行校验，发现客户的公钥和私钥都能匹配得上

- **2、检查客户生成key的指令**

```
openssl genrsa -out qpsa_rootca.key 2048
openssl req -new -sha256 -key qpsa_rootca.key -x509 -out rootca_pem.crt -subj
"/C=CN/ST=Shanghai/L=Shanghai/OU=tOS/O=Transsion/CN=Android/emailAddress=tos@transsion.
com" -days 7300 -set_serial 1 -config openssl root.cfg -sigopt rsa_padding_mode:pss -
sigopt rsa_pss_saltlen:-1 -sigopt digest:sha256
openssl x509 -in rootca_pem.crt -inform PEM -out qpsa_rootca.cer -outform DER
openssl x509 -text -inform DER -in qpsa_rootca.cer
```

发现客户生成的证书未严格遵循 Qualcomm PBL 验签要求，尤其在 `subject` 字段、扩展配置（如 `basicConstraints` 和 `keyUsage`）或签名格式上存在不兼容，导致 Boot ROM 拒绝验证。

让客户按照标准格式，重新生成一对rsa key放入签名包中，重新运行签名脚本生成签名镜像，组包然后刷机熔丝，发现可以正常启动，至此整个打包式的二段签名全部完成！

四、经验总结

（一）成功经验

- 构建完整的签名分析链路，包括对比 unsigned/signed 镜像结构、Header、签名 hash 值等；
- 明确高通Sec-Boot对不同镜像校验的粒度和位置，定位问题更有针对性；
- 引导客户严格依照 `sectools` 配置进行签名，避免“工具对了、参数错了”的问题。

（二）不足之处

- 前期未对签名参数及返回镜像进行自动化结构验证，导致需要反复人工对比；
- 签名文档未说明清楚客户签名所需严格匹配的熔丝参数，造成沟通成本上升；
- 缺少启动链失败情况下的 log trace 能力，完全依赖于串口且没有log输出、9008 判断阶段性失败，导致debug周期很长；

五、改进计划

（一）技术优化

- 建立自动化脚本，对客户签名返回镜像执行结构校验、证书链验证、secimage config 一致性比对；
- 在 `rawprogram.xml` 构建环节增加自动校验逻辑，确保关键镜像均为 `signed` 版本；
- 增加自动化打包工具中对 `secimage` 参数的版本提示和验证；
- 建立熔丝信息和签名配置之间的 Mapping 表，减少人工校对错误。

（二）流程完善

- 客户签名镜像返回后，引入镜像验证 checklist（结构校验、签名验证、hash 比对等）；
- 每次版本发布前进行 **熔丝设备验证回归测试**，确保二段式签名支持；
- 建立跨部门联调流程，避免签名参数不一致引起重复调试；
- 统一版本工具链、签名脚本版本，附带签名使用说明，避免客户误用。

（三）团队建设

- 针对 Secure Boot、sectools、签名机制开展专项技术培训，提高团队整体理解深度；
- 梳理高通二段签名流程、工具链、镜像结构文档，形成标准作业指导书（SOP）；
- 建立“关键路径镜像签名维护小组”，专人负责签名工具迭代、流程维护与客户支持；
- 建立“熔丝设备仿真环境”用于高风险签名包验证，避免量产阶段误烧死机。