

RK3588 SecurityBoot和AVB操作指南

文件标识: RK-YH-YF-307

发布版本: V1.1.0

日期: 2023-04-21

文件密级: ☐绝密 ☐秘密 ☐内部资料 ☒公开

免责声明

本文档按“现状”提供, 瑞芯微电子股份有限公司(“本公司”, 下同)不对本文档的任何陈述、信息和内容的准确性、可靠性、完整性、适销性、特定目的性和非侵权性提供任何明示或暗示的声明或保证。本文档仅作为使用指导的参考。

由于产品版本升级或其他原因, 本文档将可能在未经任何通知的情况下, 不定期进行更新或修改。

商标声明

“Rockchip”、“瑞芯微”、“瑞芯”均为本公司的注册商标, 归本公司所有。

本文档可能提及的其他所有注册商标或商标, 由其各自所有者所有。

版权所有 © 2023 瑞芯微电子股份有限公司

超越合理使用范畴, 非经本公司书面许可, 任何单位和个人不得擅自摘抄、复制本文档内容的部分或全部, 并不得以任何形式传播。

瑞芯微电子股份有限公司

Rockchip Electronics Co., Ltd.

地址: 福建省福州市铜盘路软件园A区18号

网址: www.rock-chips.com

客户服务电话: +86-4007-700-590

客户服务传真: +86-591-83951833

客户服务邮箱: fae@rock-chips.com

前言

概述

本文提供security boot和AVB的配置方法。

读者对象

本文档（本指南）主要适用于以下工程师：

技术支持工程师

软件开发工程师

修订记录

版本号	作者	修改日期	修改说明
V1.0.0	吴良清	2022-2-11	初始版本
V1.1.0	吴良清	2023-4-21	增加AVB解锁说明

目录

RK3588 SecurityBoot和AVB操作指南

1. 代码环境
2. 安全启动SecurityBoot操作步骤
 - 2.1 编译环境确认
 - 2.2 进入u-boot目录
 - 2.3 代码修改
 - 2.4 生成keys
 - 2.5 编译签名
 - 2.6 编译完整固件
 - 2.7 固件烧写
 - 2.8 判断是否熔断成功
 - 2.8.1 通过开机串口log判断
 - 2.8.2 烧写未签名的loader和uboot来判断
3. Android Verified Boot(AVB)操作步骤
 - 3.1 1. 编译avbtool工具
 - 3.2 生成atx_permanent_attributes.bin
 - 3.3 代码修改
 - 3.4 AVB key烧写
 - 3.4.1 方式一： AVB key烧写工具烧写
 - 3.4.2 方式二： AVB key集成到uboot代码内，跟固件一起烧写到机器内，不需要再用AVB key工具烧写AVB key
 - 3.4.2.1 该方式SDK默认没有支持，需要在uboot下打上补丁
 - 3.4.2.2 抽取公钥
 - 3.4.2.3 替换公钥
 - 3.5 固件编译
 - 3.6 固件烧写
 - 3.7 启动验证
 - 3.8 AVB严格校验的解锁步骤

1. 代码环境

安全启动和AVB需要在Android12 RKR4及之后版本才可以支持

2. 安全启动SecurityBoot操作步骤

2.1 编译环境确认

确认编译服务器的fdtput版本是否是1.4.5版本

```
#fdtput --version
#Version: DTC 1.4.5
```

如果fdtput版本小于1.4.5，请执行如下命令升级

```
sudo apt-get install device-tree-compiler
```

2.2 进入u-boot目录

```
cd u-boot
```

以下操作步骤都在u-boot目录下执行

2.3 代码修改

进入u-boot目录，打开对应平台的configs/rk3588_defconfig，选择如下配置：

```
//编辑configs/rk3588_defconfig文件，RK3588和RK3588s都是修改这个文件
vim configs/rk3588_defconfig
// 必选。
CONFIG_FIT_SIGNATURE=y
CONFIG_SPL_FIT_SIGNATURE=y
CONFIG_AVB_VBMETA_PUBLIC_KEY_VALIDATE=y

// 可选。
CONFIG_FIT_ROLLBACK_PROTECT=y           // boot.img防回滚
CONFIG_SPL_FIT_ROLLBACK_PROTECT=y       // uboot.img防回滚
```

2.4 生成keys

在u-boot目录执行如下操作生成keys:

```
mkdir -p keys
../rkbin/tools/rk_sign_tool kk --bits 2048 --out .
cp privateKey.pem keys/dev.key && cp publicKey.pem keys/dev.pubkey
openssl req -batch -new -x509 -key keys/dev.key -out keys/dev.crt
```

注意：该步骤执行一次即可，然后妥善保存这些keys。

2.5 编译签名

以下以RK3588为例

```
./make.sh rk3588 --spl-new --rollback-index-uboot 1 --burn-key-hash
```

说明：

--spl-new //重新打包签名后的spl

--rollback-index-uboot <版本号> //设置版本号，当步骤2中的config配置了防回滚时，需要增加这个编译选项，否则不需要

--burn-key-hash //加这个编译选项就会在烧写固件后开机的时候进行芯片熔断。

如果编译出现：

```
Can't load XXXXXX//.rnd into RNG
```

执行：

```
touch ~/.rnd
```

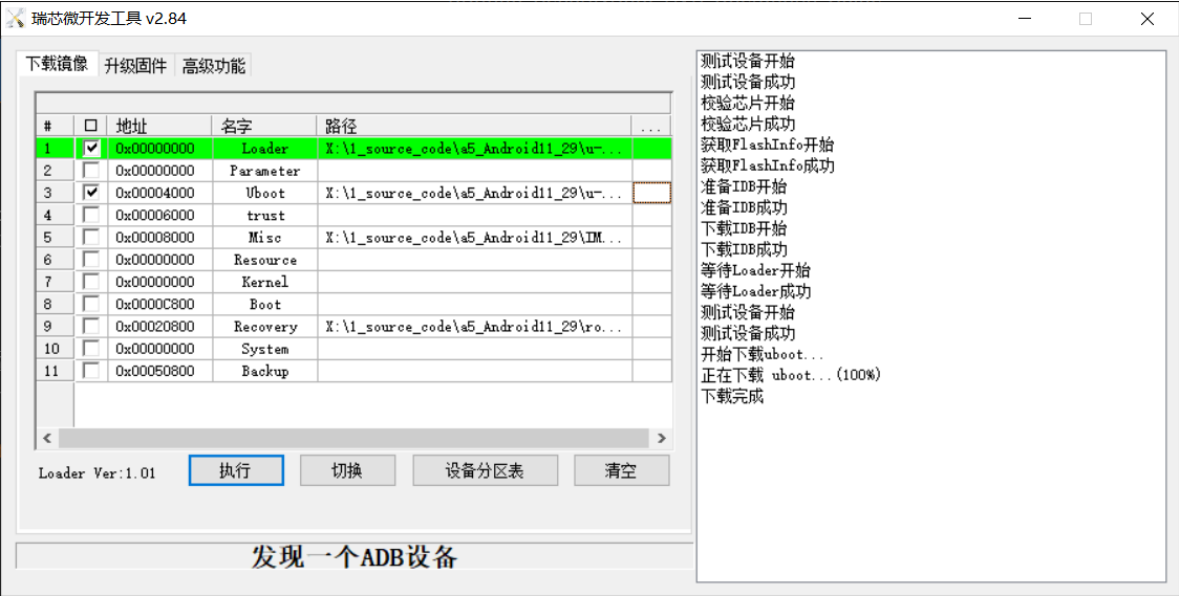
2.6 编译完整固件

按正常编译固件的方式编译其他固件（uboot和loader上面已经编译完，不需要重新编译），如用

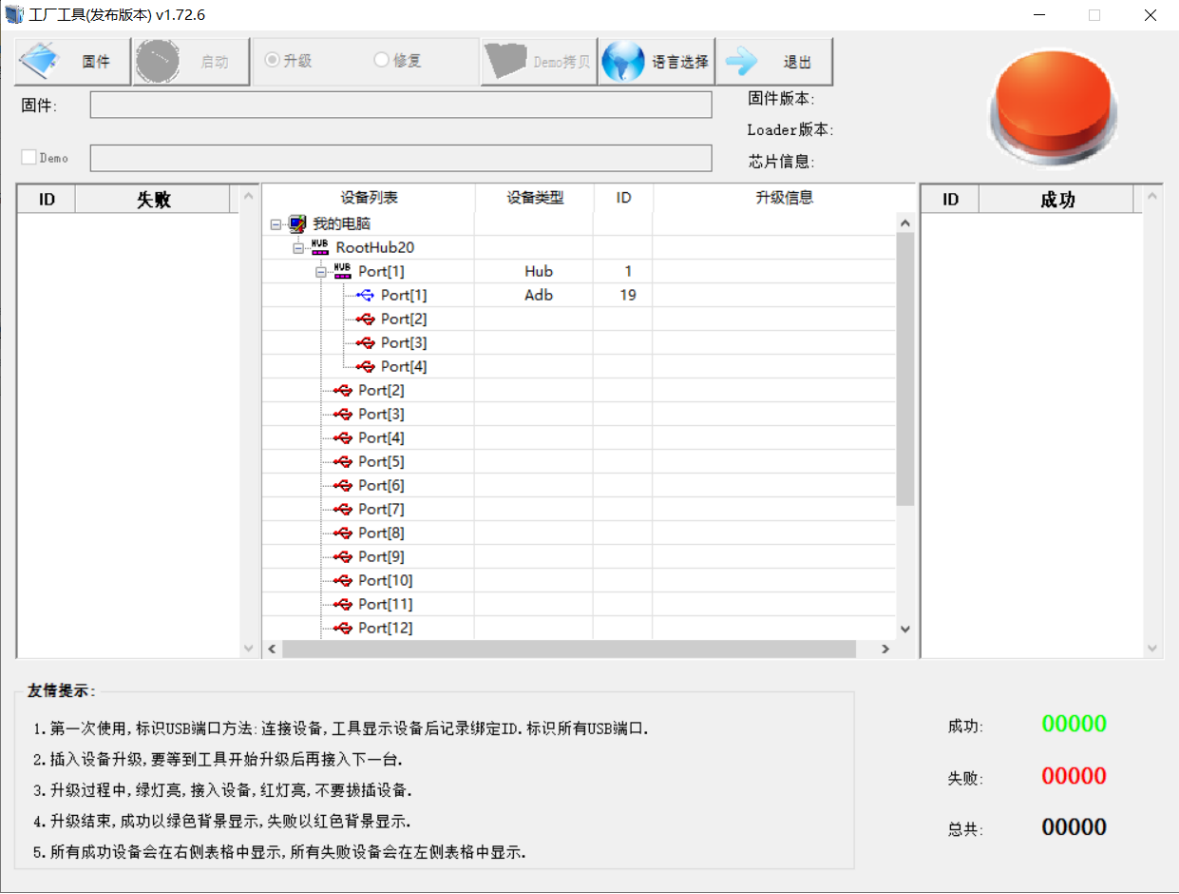
```
source build/envsetup.sh
lunch rk3588_s-userdebug
./build.sh -ACKup
```

2.7 固件烧写

开发验证阶段可以使用AndroidTool单独烧写uboot和loader即可



量产阶段使用量产工具烧写步骤5中编译生成的update.img



2.8 判断是否熔断成功

2.8.1 通过开机串口log判断

```
### Verified-boot: 0    //固件签名但是芯片没有熔断（1表示有熔断），没有进行hash有效性验证，
即编译的时候没有加 --burn-key-hash
sha256,rsa2048:dev+
rollback index: 1 >= 0(min), OK    //回滚版本，即编译时加--rollback-index-uboot
//下面这些是uboot完整性的校验
## Checking atf-1 0x00040000 ... sha256+ OK
## Checking uboot 0x00a00000 ... sha256+ OK
## Checking fdt 0x00b2a018 ... sha256+ OK
## Checking atf-2 0xfdcc9000 ... sha256+ OK
## Checking atf-3 0xfdc00000 ... sha256+ OK
## Checking optee 0x00200000 ... sha256+ OK
```

2.8.2 烧写未签名的loader和uboot来判断

3. Android Verified Boot(AVB)操作步骤

3.1 1. 编译avbtool工具

```
mma external/avb/ -j16
```

编译完成后生成:

```
out/host/linux-x86/bin/avbtool
```

3.2 生成atx_permanent_attributes.bin

- 修改产品ID

```
cd external/avb/test
```

```
diff --git a/test/avb_atx_generate_test_data b/test/avb_atx_generate_test_data
index 1b8bb2b..2220688 100755
--- a/test/avb_atx_generate_test_data
+++ b/test/avb_atx_generate_test_data
@@ -48,7 +48,7 @@ AVBTOOL=$(dirname "$0")/../../avbtool
 echo AVBTOOL = ${AVBTOOL}

 # Get a zero product ID.
-echo 00000000000000000000000000000000 | xxd -r -p - atx_product_id.bin
+echo 00000000000000000000000000000000123 | xxd -r -p - atx_product_id.bin

 # Generate key pairs.
if [ ! -f testkey_atx_prk.pem ]; then
```

```
cd -
```

注意：产品ID的位数为16位，数值可以自己定义

- 生成atx_permanent_attributes.bin

```
cd external/avb/test/data
```

```
../avb_atx_generate_test_data
```

```
cd -
```

执行以上操作后在external/avb/test/data生成：

- atx_permanent_attributes.bin
- atx_metadata.bin
- testkey_atx_pik.pem
- testkey_atx_prk.pem
- testkey_atx_psk.pem

注意：

- pem文件系统中默认有一个，如果需要重新生成，需要删除系统默认的文件，然后再执行上面的操作重新生成pem文件，建议客户自己重新生成
- 这个步骤一个产品只要执行一次就可以，请妥善保管上面生产的文件，在下面的步骤中会使用

3.3 代码修改

```
cd device/rockchip/rk3588
```

```
diff --git a/rk3588_s/BoardConfig.mk b/rk3588_s/BoardConfig.mk
index 24b415f..80fa60f 100644
--- a/rk3588_s/BoardConfig.mk
+++ b/rk3588_s/BoardConfig.mk
@@ -37,3 +37,7 @@ ifeq ($(strip $(BOARD_USES_AB_IMAGE)), true)
    include device/rockchip/common/BoardConfig_AB.mk
    TARGET_RECOVERY_FSTAB := device/rockchip/rk3588/rk3588_s/recovery.fstab_AB
  endif
+
+BOARD_AVB_ENABLE := true      //打开AVB功能
+BOARD_AVB_ALGORITHM := SHA256_RSA4096 //配置加密算法
+BOARD_AVB_KEY_PATH := external/avb/test/data/testkey_atx_psk.pem //秘钥存放路径
+BOARD_AVB_METADATA_BIN_PATH := external/avb/test/data/atx_metadata.bin //指定
metadata文件
+#BOARD_AVB_ROLLBACK_INDEX := 5    //配置防版本回滚，默认不开，根据需求开关，需要配合
uboot修改
```

```
cd -
```

```
cd u-boot
```

```
diff --git a/configs/rk3588_defconfig b/configs/rk3588_defconfig
index 3017921487..84197eeale 100644
--- a/configs/rk3588_defconfig
+++ b/configs/rk3588_defconfig
@@ -214,5 +214,9 @@ CONFIG_AVB_LIBAVB_AB=y
 CONFIG_AVB_LIBAVB_ATX=y
 CONFIG_AVB_LIBAVB_USER=y
 CONFIG_RK_AVB_LIBAVB_USER=y
+CONFIG_RK_AVB_LIBAVB_ENABLE_ATH_UNLOCK=y
+CONFIG_AVB_VBMETA_PUBLIC_KEY_VALIDATE=y
//选配
+CONFIG_ANDROID_AVB_ROLLBACK_INDEX=y //防回滚功能，需要改功能的才配置，需要配合device下
面的BOARD_AVB_ROLLBACK_INDEX一起配置
```

```
cd -
```

3.4 AVB key烧写

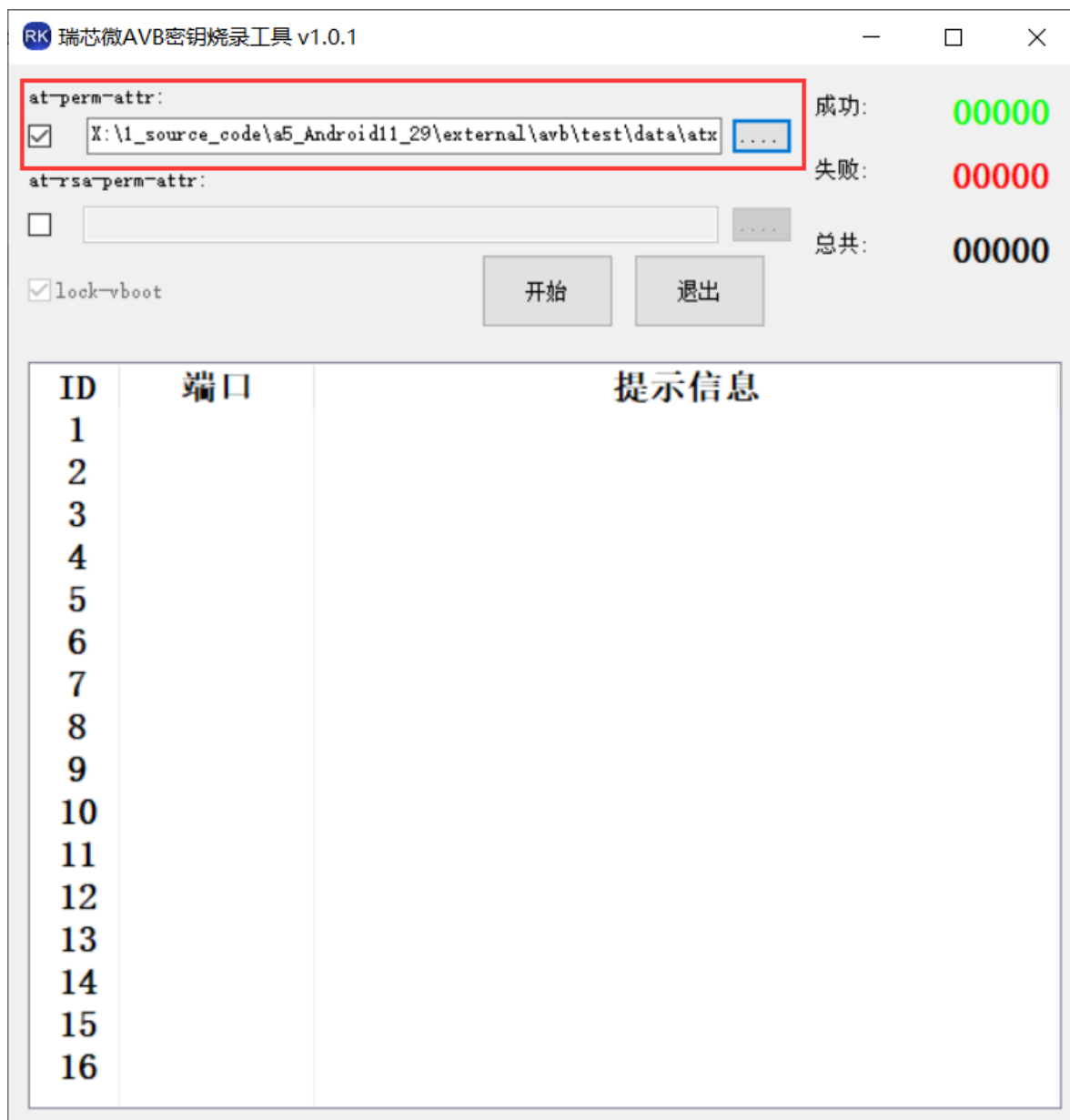
3.4.1 方式一：AVB key烧写工具烧写

烧写工具：AvbKeyWriter（RKTools/windows/AvbKeyWriter-v1.0.1.7z）

烧写源文件：步骤1中生产的external/avb/test/data生成atx_permanent_attributes.bin

烧写方式：

- 勾选at-perm-attr
- 导入步骤3中生产的external/avb/test/data生成atx_permanent_attributes.bin
- 待烧写设备进入loader模式
- 点击“开机按键进行烧写”



3.4.2 方式二： AVB key集成到uboot代码内，跟固件一起烧写到机器内，不需要再用AVB key工具烧写AVB key

这种方式是将key内置在uboot.img的固件里面，没有烧到安全分区中，当boot.img被替换后key就会失效，所以需要配合security boot来保证uboot分区的固件不被篡改或者替换。

注意：这种方式不支持严格AVB校验的解锁功能，所以需要解锁请使用方式一。

3.4.2.1 该方式SDK默认没有支持，需要在uboot下打上补丁

补丁路径：RKDocs/common/security/patch/u-boot/0001-avb-add-embedded-key.patch

```
cd u-boot
```

```
git am RKDocs/common/security/patch/u-boot/0001-avb-add-embedded-key.patch
```

```
cd -
```

3.4.2.2 抽取公钥

```
avbtool extract_public_key --key external/avb/test/data/testkey_atx_psk.pem --  
output avb_root_pub.bin  
xxd -i avb_root_pub.bin > external/avb/test/data/avb_root_pub.h
```

3.4.2.3 替换公钥

抽取的公钥(external/avb/test/data/avb_root_pub.h)替换 u-boot/lib/avb/libavb_user/avb_ops_user.c中的 avb_root_pub 数组

```
cd u-boot
```

```
vim lib/avb/libavb_user/avb_ops_user.c
```

```
/**  
 * Internal builds use testkey_rsa4096.pem  
 * OEM should replace this Array with public key used to sign vbmeta.img  
 * *  
 openssl genpkey -algorithm RSA -pkeyopt rsa_keygen_bits:4096 \  
 * -outform PEM -out avb_rsa4096.pem  
 * avbtool extract_public_key --key avb_rsa4096.pem --output avb_root_pub.bin  
 * xxd -i avb_root_pub.bin > avb_root_pub.h  
 */  
static const char avb_root_pub [] = {  
0x00, 0x00, 0x10, 0x00, 0x55, 0xd9, 0x04, 0xad, 0xd8, 0x04, 0xaf, 0xe3,  
0xd3, 0x84, 0x6c, 0x7e, 0x0d, 0x89, 0x3d, 0xc2, 0x8c, 0xd3, 0x12, 0x55,  
0xe9, 0x62, 0xc9, 0xf1, 0x0f, 0x5e, 0xcc, 0x16, 0x72, 0xab, 0x44, 0x7c,  
0x2c, 0x65, 0x4a, 0x94, 0xb5, 0x16, 0x2b, 0x00, 0xbb, 0x06, 0xef, 0x13,  
0x07, 0x53, 0x4c, 0xf9, 0x64, 0xb9, 0x28, 0x7a, 0x1b, 0x84, 0x98, 0x88,  
0xd8, 0x67, 0xa4, 0x23, 0xf9, 0xa7, 0x4b, 0xdc, 0x4a, 0x0f, 0xf7, 0x3a,  
0x18, 0xae, 0x54, 0xa8, 0x15, 0xfe, 0xb0, 0xad, 0xac, 0x35, 0xda, 0x3b,  
0xad, 0x27, 0xbc, 0xaf, 0xe8, 0xd3, 0x2f, 0x37, 0x34, 0xd6, 0x51, 0x2b,  
... ..  
}
```

```
cd -
```

3.5 固件编译

以上步骤执行完后可以进行完整的固件编译，以RK3566_r的产品为例进行编译：

- 带安全启动的方案

带安全启动的方案uboot要单独先编译，参考上面安全启动的操作步骤

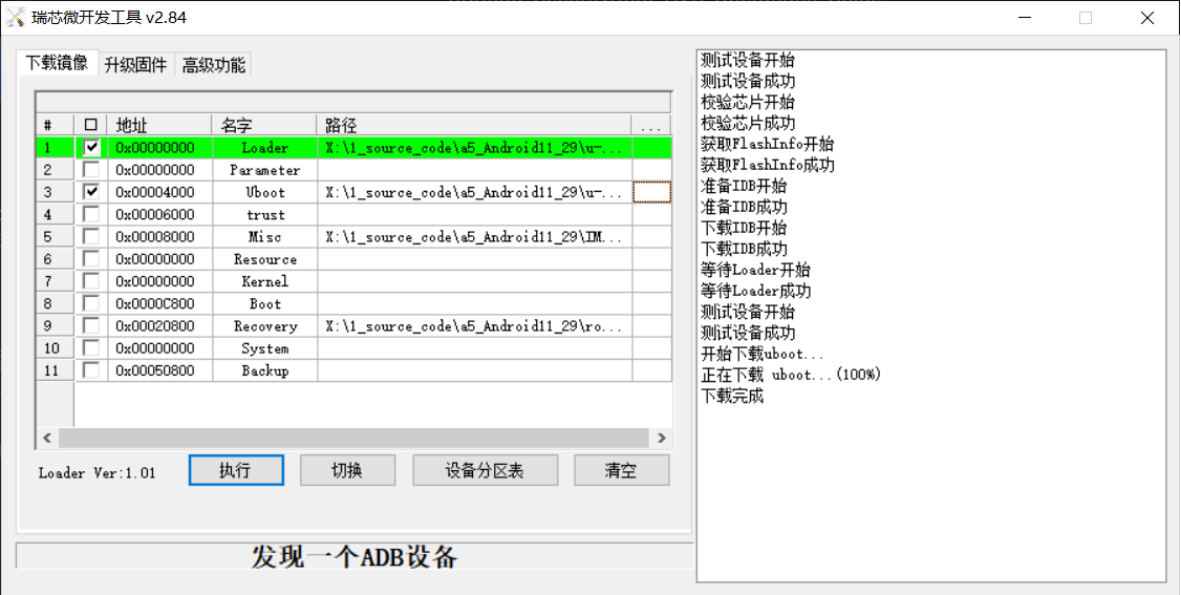
```
source build/envsetup.sh  
lunch rk3588_s-userdebug  
./build.sh -ACKup
```

- 不带安全启动的方案

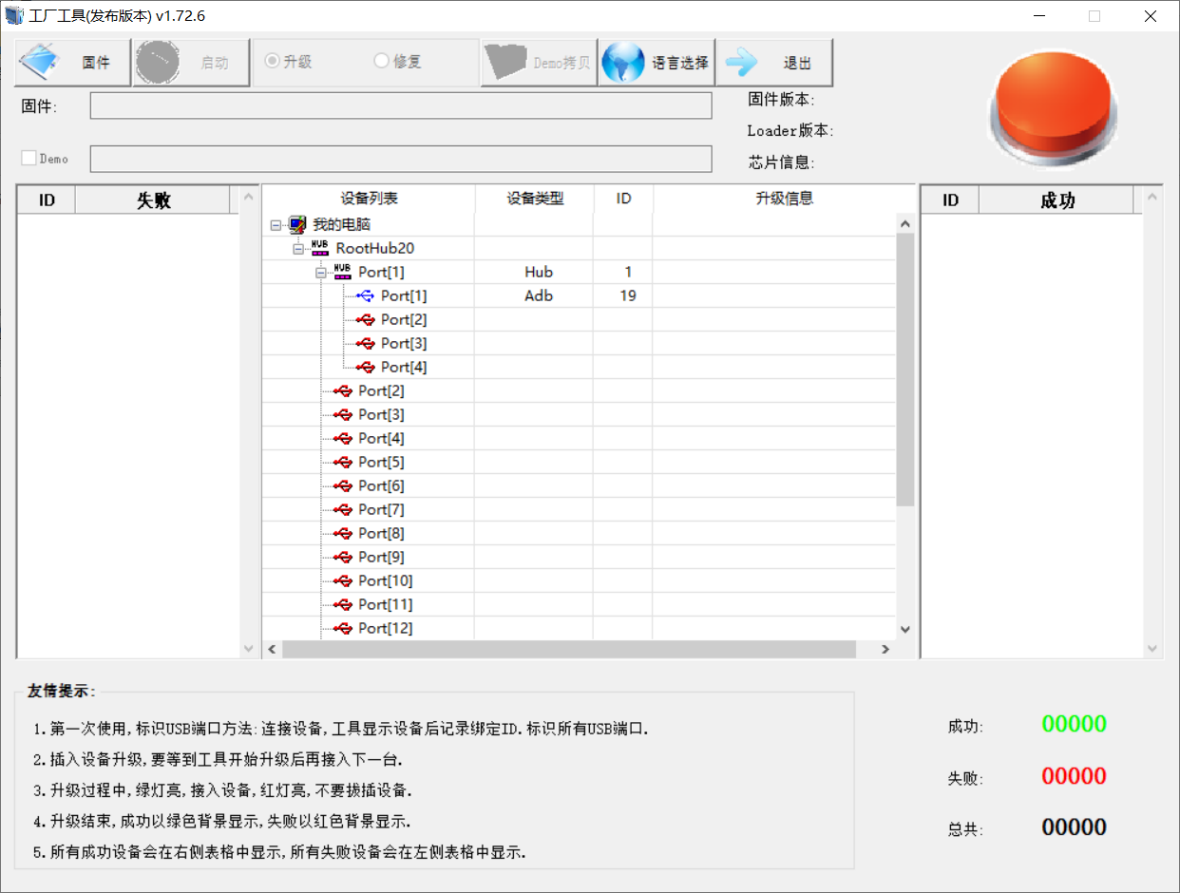
```
source build/envsetup.sh
lunch rk3588_s-userdebug
./build.sh -ACKUup
```

3.6 固件烧写

开发阶段用AndroidTool工具烧写编译出来的完整固件



量产阶段使用量产工具烧写步骤4中编译生成的update.img



3.7 启动验证

- uboot开机log确认

通过以上步骤后系统开机时在串口的log中u-boot阶段会有如下打印：

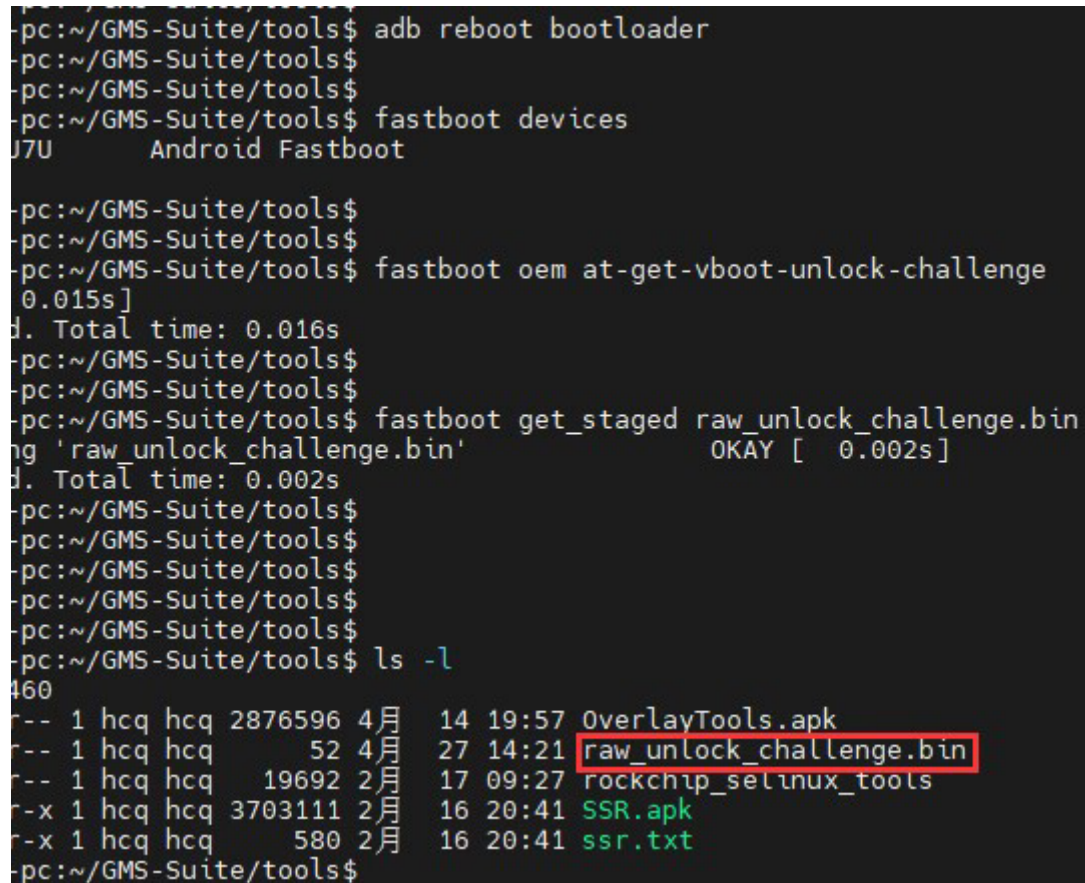
```
Vboot=0, AVB images, AVB verify
read_is_device_unlocked() ops returned that device is LOCKED
ANDROID: Hash OK
```

- 烧写非AVB固件或者其它的固件会无法开机

3.8 AVB严格校验的解锁步骤

- 设备进入 fastboot 模式，电脑端输入

```
adb reboot bootloader
fastboot oem at-get-vboot-unlock-challenge
fastboot get_staged raw_unlock_challenge.bin
```



```
pc:~/GMS-Suite/tools$ adb reboot bootloader
pc:~/GMS-Suite/tools$
pc:~/GMS-Suite/tools$
pc:~/GMS-Suite/tools$ fastboot devices
J7U      Android Fastboot

pc:~/GMS-Suite/tools$
pc:~/GMS-Suite/tools$
pc:~/GMS-Suite/tools$ fastboot oem at-get-vboot-unlock-challenge
0.015s]
d. Total time: 0.016s
pc:~/GMS-Suite/tools$
pc:~/GMS-Suite/tools$
pc:~/GMS-Suite/tools$ fastboot get_staged raw_unlock_challenge.bin
ng 'raw_unlock_challenge.bin'          OKAY [ 0.002s]
d. Total time: 0.002s
pc:~/GMS-Suite/tools$
pc:~/GMS-Suite/tools$
pc:~/GMS-Suite/tools$
pc:~/GMS-Suite/tools$
pc:~/GMS-Suite/tools$
pc:~/GMS-Suite/tools$
pc:~/GMS-Suite/tools$ ls -l
460
-- 1 hcq hcq 2876596 4月 14 19:57 OverlayTools.apk
-- 1 hcq hcq      52 4月 27 14:21 raw_unlock_challenge.bin
-- 1 hcq hcq  19692 2月 17 09:27 rockchip_selinux_tools
-x 1 hcq hcq 3703111 2月 16 20:41 SSR.apk
-x 1 hcq hcq   580 2月 16 20:41 ssr.txt
pc:~/GMS-Suite/tools$
```

获得带版本、Product Id 与 16 字节的随机数的数据，取出随机数生成 raw_unlock_challenge.bin

- 使用 avbtool 生成 unlock_credential.bin

根据raw_unlock_challenge.bin和 atx_product_id.bin，其中atx_product_id.bin是前面生成 atx_permanent_attributes.bin时生成的

```
python avb-challenge-verify.py raw_unlock_challenge.bin atx_product_id.bin
```

```
python avbtool make_unlock_credential --output=unlock_credential.bin --
intermediate_key_certificate=pik_certificate.bin --
unlock_key_certificate=puk_certificate.bin --challenge=unlock_challenge.bin --
unlock_key=testkey_puk.pem
```

其中avb-challenge-verify.py的实现代码如下，请复制代码另存为avb-challenge-verify.py

```
#!/user/bin/env python
"This is a test module for getting unlock_challenge.bin"
import sys
import os
from hashlib import sha256

def challenge_verify():
    if (len(sys.argv) != 3) :
        print "Usage: rkpublickey.py [challenge_file] [product_id_file]"
        return
    if ((sys.argv[1] == "-h") or (sys.argv[1] == "--h")):
        print "Usage: rkpublickey.py [challenge_file] [product_id_file]"
        return
    try:
        challenge_file = open(sys.argv[1], 'rb')
        product_id_file = open(sys.argv[2], 'rb')
        challenge_random_file = open('unlock_challenge.bin', 'wb')
        challenge_data = challenge_file.read(52)
        product_id_data = product_id_file.read(16)
        product_id_hash = sha256(product_id_data).digest()
        print("The challenge version is %d" %ord(challenge_data[0]))
        if (product_id_hash != challenge_data[4:36]) :
            print("Product id verify error!")
            return
        challenge_random_file.write(challenge_data[36:52])
        print("Success!")

    finally:
        if challenge_file:
            challenge_file.close()
        if product_id_file:
            product_id_file.close()
        if challenge_random_file:
            challenge_random_file.close()
if __name__ == '__main__':
    challenge_verify()
```

- 烧写unlock_credential.bin，电脑端输入

```
fastboot stage unlock_credential.bin
fastboot oem at-unlock-vboot
```

注意：此时设备状态一直处于第一次进入 fastboot 模式状态，在此期间不能断电、关机、重启。因为步骤 1.做完后，设备存储着生成的随机数，如果断电、关机、重启，会导致随机数丢失，后续校验 challenge signature 会因为随机数不匹配失败。

如果开启：

```
CONFIG_MISC=y  
CONFIG_ROCKCHIP_EFUSE=y  
CONFIG_ROCKCHIP_OTP=y
```

就会使用 CPUID 作为 challenge number，而 CPUID 是与机器匹配的，数据不会因为关机而丢失，生成的unlock_credential.bin 可以重复使用。省去重复生成unlock_challenge.bin，制作unlock_credential.bin的步骤。再次解锁步骤变为：

```
fastboot oem at-get-vboot-unlock-challenge  
fastboot stage unlock_credential.bin  
fastboot oem at-unlock-vboot
```