



# Android 电信安全 方案设计

文档版本 0B02

发布日期 2013-03-29

**版权所有 © 深圳市海思半导体有限公司 2012。保留一切权利。**

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。

## **商标声明**



**HISILICON**、海思和其他海思商标均为深圳市海思半导体有限公司的商标。

本文档提及的其他所有商标或注册商标，由各自的所有人拥有。

## **注意**

您购买的产品、服务或特性等应受海思公司商业合同和条款的约束，本文档中描述的全部或部分产品、服务或特性可能不在您的购买或使用范围之内。除非合同另有约定，海思公司对本文档内容不做任何明示或默示的声明或保证。

由于产品版本升级或其他原因，本文档内容会不定期进行更新。除非另有约定，本文档仅作为使用指导，本文档中的所有陈述、信息和建议不构成任何明示或暗示的担保。

## **深圳市海思半导体有限公司**

地址：                    深圳市龙岗区坂田华为基地华为电气生产中心                    邮编：518129

网址：                    <http://www.hisilicon.com>

客户服务电话：          +86-755-28788858

客户服务传真：          +86-755-28357515

客户服务邮箱：          [support@hisilicon.com](mailto:support@hisilicon.com)



# 前 言

## 概述

Android 是个开放的平台，且有大量第三方应用。在广电、电信领域应用时，引起运营商对安全问题的担忧。主要包含以下几方面：安全启动、APK 管控、密钥管理、应用权限管理、终端身份认证。

本文档主要基于中国电信的安全要求，详细描述了 Android 系统的安全方案设计，开发过程以及生产流程。

## 产品版本

与本文档相对应的产品版本如下。

产品名称	产品版本
Hi3716C 芯片	V101


## 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

## 符号约定

在本文中可能出现下列标志，它们所代表的含义如下。

符号	说明
 DANGER	表示有高度潜在危险，如果不能避免，会导致人员死亡或严重伤害。



符号	说明
 <b>WARNING</b>	表示有中度或低度潜在危险，如果不能避免，可能导致人员轻微或中等伤害。
 <b>CAUTION</b>	表示有潜在风险，如果忽视这些文本，可能导致设备损坏、数据丢失、设备性能降低或不可预知的结果。
 <b>TIP</b>	表示能帮助您解决某个问题或节省您的时间。
 <b>NOTE</b>	表示是正文的附加信息，是对正文的强调和补充。

## 修订记录

修订记录累积了每次文档更新的说明。最新版本的文档包含以前所有文档版本的更新内容。

修订日期	版本	修订说明
2013-03-29	02	增加 APK 管控方案
2013-01-22	01	第一次发布



# 目 录

前 言.....	iii
1 电信 Android 开发 .....	x
1.1 概述.....	x
1.2 开发方主体及责任.....	x
1.3 开发过程.....	xi
1.4 配置编译 fastboot.....	xii
1.5 系统编译.....	xiv
1.6 制作 bootargs 文件 .....	xiv
1.7 生成安全启动相关密钥.....	xiv
1.8 生成 DDR 参数配置文件 cfg.bin .....	xvi
1.9 签名 fastboot 等镜像.....	xvii
1.10 制作 recovery 升级包.....	xix
1.11 烧写安全启动密钥和关键数据 .....	xx
1.12 APK 安装管控.....	xxii
1.13 工厂生产 APK 开发.....	xxv
1.14 身份认证 API 说明 .....	xxx
2 电信安全要求概述.....	2-1
2.1 安全性目标.....	2-1
2.2 终端安全启动要求.....	2-1
2.2.1 系统安全引导过程.....	2-1
2.2.2 系统软件安全启动、升级.....	2-2
2.2.3 APK 安装管控.....	2-4
2.3 终端身份认证功能要求.....	2-4
2.3.1 终端身份认证过程.....	2-5
2.3.2 挑战字原文与挑战字的生成及验证.....	2-5
2.3.3 终端身份认证数据结构.....	2-5
2.3.4 挑战字返回结果的计算.....	2-6
3 海思安全方案概述.....	3-7
3.1 安全性目标.....	3-7

3.2 系统架构.....	3-7
3.3 系统签名校验启动流程.....	3-8
3.3.1 安全 Boot 签名校验.....	3-10
3.4 系统分区签名校验.....	3-12
3.4.1 Key Partition 分区介绍 .....	3-13
3.4.2 Key Partition 分区的签名校验 .....	3-14
3.4.3 非文件系统分区的签名校验.....	3-17
3.4.4 文件系统分区镜像的签名.....	3-22



## 插图目录

图 1-1 去除 SDIO 的复位 .....	错误！未定义书签。
图 1-2 Boorargs 生成示意图 .....	xiv
图 1-3 密钥生成工具 GenSecureData 界面 .....	xvi
图 1-4 DDR 参数配置文件生成工具 .....	xvi
图 1-5 HiSigA 的用户界面 .....	xvii
图 1-6 HiSigA 制作升级包过程 .....	xix
图 1-7 make_key 提示是否需要密码保护 .....	xxiii
图 1-8 make_key 生成最终密钥 .....	xxiii
图 1-9 工厂生产线示意图 .....	xxv
图 2-1 系统的安全引导过程 .....	2-2
图 2-2 系统软件安全启动、升级 .....	2-3
图 2-3 终端身份认证过程 .....	2-5
图 3-1 芯片功能结构图 .....	3-7
图 3-2 系统签名校验启动流程 .....	3-9
图 3-3 安全 Boot 在 Flash 的内部存储结构 .....	3-10
图 3-4 安全 Boot 的校验流程 .....	3-12
图 3-5 Key Partition 分区密钥结构 .....	3-13
图 3-6 Key Partition 分区密钥加密流程 .....	3-14
图 3-7 Key Partition 分区镜像结构图 .....	3-14
图 3-8 Key Partition 分区签名校验流程 .....	3-17
图 3-9 Kernel 等分区的签名校验示意图 .....	3-18
图 3-10 Kernel 等非文件系统分区镜像结构图 .....	3-18
图 3-11 kernel 等非文件系统分区的签名校验流程图 .....	3-21
图 3-12 文件系统签名的镜像结构图 .....	3-22







## 表格目录

表 1-1 支持电信 Android 安全的芯片列表 .....	xi
表 1-1 安全启动相关密钥列表.....	xv
表 1-2 关键数据长度和存储位置.....	xx
表 2-1 系统分区校验表.....	2-3



# 1 电信 Android 开发

## 1.1 概述

本章主要介绍了中国电信 Android 高级安全方案的开发过程，其他运营商的 Andorid 高级安全方案可以参考该规范。

## 1.2 开发方主体及责任

整个方案涉及的开发方及责任主要包括：

1. 电信
  - 负责制定安全技术标准
  - 评估芯片厂商的方案是否满足电信的要求
  - 验证机顶盒厂商的盒子

2. 海思
  - 根据电信的标准，提出满足电信要求的安全方案
  - 指导机顶盒厂商基于海思方案的开发

海思的交付件如下：

- 基于 Android SDK 的安全启动补丁。该补丁的主要作用是：
  - 在 fastboot 中增加对 bootargs、kernel、system 文件系统以及 recovery 的签名校验
  - 在 recovery 中增加升级时对升级包及升级文件的签名校验
- 中国电信方案签名工具 HiSigA。该签名工具的主要作用是：
  - 对 fastboot、kernel、system 文件系统、recovery 等需要进行签名校验的分区提供签名
  - 对需要进行升级的文件打包并且签名
- 身份认证库 libhi\_ctit.a。该库的主要作用是：
  - 提供终端身份认证时的解密、校验接口



- 提供工厂量产时对电信关键数据和安全启动 OTP 密钥的烧写
  - 海思安全启动密钥生成工具 GenSecureData.exe
    - 用于生成安全启动相关的 RSA 密钥等
  - 开发调试工具 write\_flash 和 write\_otp
    - 这两个工具的作用是在开发调试阶段烧写 OTP 中的安全启动密钥如 RSA Key，以及电信要求的关键数据
3. 机顶盒厂商
- 负责开发工厂生产的烧写工具 APK
4. 茁壮网络
- 负责集成海思的安全启动、升级方案，集成后把镜像提供给机顶盒厂商
- 茁壮联系人：王燕（[willa@ipanel.cn](mailto:willa@ipanel.cn)）
5. 阿网
- 负责开发电信前端的身份认证平台
  - 负责电信关键数据的生成
  - 负责工厂生产时 PC 端工具、云端工具的开发
- 阿网的联系人：陈樑（[chenliang@sanss.com](mailto:chenliang@sanss.com)）
6. 科升
- 负责身份认证 APK 的集成开发工具，将开发好的身份认证 APK 提供给机顶盒厂商
- 科升的联系人：奚工（[xixi@akazam.com](mailto:xixi@akazam.com)）

### 1.3 开发过程

机顶盒厂商使用海思的安全方案的开发过程主要包括如下的步骤：

步骤 1 向海思申请适用该方案的芯片，目前支持该方案的芯片类型如表 1-1。

表1-1 支持电信 Android 安全的芯片列表

芯片类型	芯片版本	Mark
Hi3716C	V101	Hi3716CRBCV101H*0

其中“\*”代表是否支持杜比。

步骤 2 向海思申请电信 Android 安全方案组件包。

步骤 3 将组件包合进海思的 Android SDK 中，编译生成 fastboot，kernel，system 和 recovery。

步骤 4 使用海思的 bootargs 制作工具 mkbootargs 制作生成 bootargs。

步骤 5 使用海思的密钥生成工具 GenSecureData.exe 生成密钥文件 Key\_for\_signature.txt 和 Key\_for\_verify.txt。



步骤 6 使用海思的签名工具 HiSigA 和步骤 5 中生成的密钥 Key\_for\_signature.txt 对 fastboot, kernel, system, recovery 和 bootargs 进行签名。并且将签名后生成的文件烧写进对应的 Flash 分区中。

步骤 7 向阿网申请电信要求的关键数据，将申请到的关键数据按照一定的数据格式粘贴到 Key\_for\_verify.txt 的末尾。



说明

每个机顶盒的烧写的关键数据是唯一的。

步骤 8 使用海思的烧写工具 write\_flash 和 write\_otp 烧写 RSA 密钥和关键数据。

步骤 9 断电重启，这个时候机顶盒应该可以正常启动。

步骤 10 向科升要求提供身份认证 APK，测试机顶盒的身份认证是否能够通过。

步骤 11 以上步骤完成了机顶盒在实验室的调试开发，在量产时，机顶盒厂商还应该开发烧写 APK，用于工厂烧写安全启动相关密钥和关键数据。

## 1.4 配置编译 fastboot

海思 Android 电信安全方案目前支持 SPI、Nand 的 Flash。只要是兼容海思芯片方案的 Flash，都可以在海思 Android 电信安全方案中使用。根据 fastboot 存放的 Flash 类型不同，配置可能有差别。

### 步骤 1 系统配置

打开根目录文件 config.mk:

- 选择大系统存放 Flash 类型，例如 flash（flash 是 NAND 器件）:

```
HiAndroid_FLASH_TYPE=NAND
```

- 选择芯片类型，（以 Hi3716C 芯片为例）:

```
HiAndroid_CHIP_TYPE=Hi3716C
```

- 配置选择 DDR 大小（以 DDR 为 512M 为例）:

```
HiAndroid_MEM_VERSION=512
```

### 步骤 2 配置 fastboot 存放的 Flash 类型

打开文件 bootable/bootloader/fastboot3.0/product/customer\_android.h:

- SPI: 打开宏定义

```
#define BOOT_FROM_SPI_FLASH
```

同时注释掉宏定义

```
#define BOOT_FROM_EMMC_FLASH
```

```
#define BOOT_FROM_NAND_FLASH
```

- Nand: 打开宏定义

```
#define BOOT_FROM_NAND_FLASH
```

同时注释掉宏定义



```
#define BOOT_FROM_EMMC_FLASH  
#define BOOT_FROM_SPI_FLASH
```

### 步骤 3 配置 bootargs 存放的偏移和大小

打开文件 bootable/bootloader/fastboot3.0/product/customer\_android.h。

- 配置 bootargs 偏移：修改以下宏定义为实际偏移

```
BOOTPARAM_OFFSET
```

- 配置 bootargs 大小：修改以下宏定义为实际大小

```
BOOTPARAM_SIZE
```

### 步骤 4 配置 Key Partition 存放的 Flash 类型

打开文件 bootable/bootloader/fastboot3.0/product/customer\_android.h。

- SPI：打开宏定义

```
#define KEYPARTITION_IN_SPI
```

同时注释掉宏定义

```
#define KEYPARTITION_IN_EMMC
```

```
#define KEYPARTITION_IN_NAND
```

- Nand：打开宏定义

```
#define KEYPARTITION_IN_NAND
```

同时注释掉宏定义

```
#define KEYPARTITION_IN_SPI
```

```
#define KEYPARTITION_IN_EMMC
```

### 步骤 5 配置 Key Partition 存放的偏移和大小

打开文件 bootable/bootloader/fastboot3.0/product/customer\_android.h。

- 配置 Key Partition 偏移：修改以下宏定义为实际偏移

```
KEYDATA_OFFSET
```

- 配置 Key Partition 大小：修改以下宏定义为实际大小

```
KEYDATA_SIZE
```

### 步骤 6 寄存器配置及 DDR 参数配置

不同类型的板子，寄存器和 DDR 参数的配置均不同。DDR 参数配置文件可以由海思 SDK 提供的表格文件生成。进入 bootable/bootloader/fastboot3.0/lowlevel\_reg\_info 目录，选择对应的表格文件生成配置文件。

将 reg\_info.bin 拷贝到 bootable/bootloader/fastboot3.0 根目录，并将名字改成 “.reg”

### 步骤 7 完成以上步骤后，编译 fastboot，参考以下命令：

```
cd bootable/bootloader/fastboot3.0/  
make ARCH=arm CROSS_COMPILE=arm-hisiv200-linux- godbox_config  
make ARCH=arm CROSS_COMPILE=arm-hisiv200-linux- -j
```



## 1.5 系统编译

在Android源码工程根目录中执行如下操作编译：

```
#source build/envsetup.sh
#lunch full_godbox-eng
#./mk-ics.sh
```

镜像将生成于目录 out/target/product/godbox。

## 1.6 制作 bootargs 文件

在海思的 Android 电信安全方案中，bootargs 需要经过 fastboot 签名校验后才能使用。同时，在海思的方案中，需要增加存放密钥的 Key Partition 分区，因此需要在 bootargs 中增加名字为“key”的分区。

bootargs 的制作工具为 mkbootargs，在 linux 环境上使用。

步骤 1 在文件 bootargs\_input.txt 中根据实际单板需要配置 bootargs。



**注意** 请在 bootargs 中务必增加“key”分区。所有参数必须放在一行，每个参数之间用符合“@”隔开。

步骤 2 在 Linux 环境中运行命令 ./mkbootargs bootargs\_input.txt。在当前目录下即可生成最终的 bootargs 文件 bootargs.bin。



目录 tools\mkbootargs 中有 SPI 和 EMMC flash 的 bootargs 配置供参考。

图1-1 Bootargs 生成示意图

```
00209458@STB-2285-8:~/bootargs/mkbootargs$ ./mkbootargs bootargs_input.txt
00209458@STB-2285-8:~/bootargs/mkbootargs$
```

## 1.7 生成安全启动相关密钥

安全启动密钥用于机顶盒启动时对 fastboot，kernel 等分区的校验。机顶盒中需要存储的安全启动相关密钥如表 1-1。



表1-1 安全启动相关密钥列表

Key	用途	存储位置	Host CPU 是否可读写
Root RSA Public Key	用于对 Extern RSA Public Key 做签名校验	OTP	N
BL_RootKey	用于加密 Key Partition AES CBC Key	OTP	N
Key Partition AES CBC Key	用于 Key Partition 分区签名	OTP	N
Extern RSA Public Key	用于 boot 分区签名校验	Flash	Y
AES CBC Encrypted Key	用于非 boot 分区签名校验，如 Kernel	Flash, key 分区中	Y



说明

Host CPU 是否可读写指的是数据烧写进 OTP 后。烧写数据时，OTP 中相关密钥会被锁定，Host CPU 不可再读写。

海思提供生成安全启动密钥的生成工具 GenSecureData.exe，由机顶盒厂商生成密钥文件 Key\_for\_signature.txt 和 Key\_for\_verify.txt，两个文件包含的数据和作用如下：

- Key\_for\_signature.txt:
  - 表 1-1 中的密钥数据。
  - 表 1-1 中 Root RSA Public Key 对应的私钥 Root RSA Private Key。
  - 表 1-1 中 Extern RSA Public Key 对应的私钥 Extern RSA Private Key。Key\_for\_signature.txt 用于对 fastboot 等分区镜像进行签名。参考 1.9 。
- Key\_for\_verify.txt:
  - 表 1-1 中的密钥数据。Key\_for\_verify.txt 用于在开发调试阶段烧写安全启动相关密钥进机顶盒中。参考 1.11 。

一般机顶盒厂商只需要运行该工具一次，即 Key\_for\_signature.txt 和 Key\_for\_verify.txt 对所有的机顶盒都一样，机顶盒厂商必须保管好这两个文件。



**注意** 机顶盒厂商务必保管好 Key\_for\_signature.txt 和 Key\_for\_verify.txt 文件，防止丢失和泄密。



图1-2 密钥生成工具 GenSecureData 界面



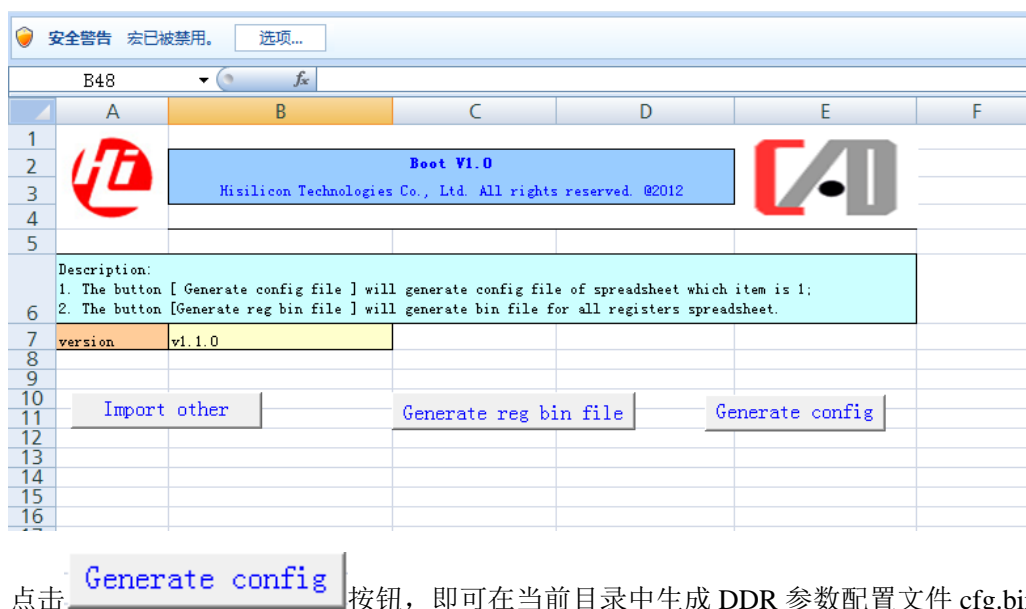
点击“OK”按钮，即可以在当前目录下生成 Key\_for\_signature.txt 和 Key\_for\_verify.txt。

## 1.8 生成 DDR 参数配置文件 cfg.bin

DDR 参数配置文件可以由海思 SDK 提供的表格文件生成，比如 hi3716cdmoverb\_hi3716cv100\_dds3\_1gbyte\_8bitx4\_4layers.xls。

DDR 参数配置文件的工具如图 1-3 所示：

图1-3 DDR 参数配置文件生成工具



点击 **Generate config** 按钮，即可在当前目录中生成 DDR 参数配置文件 cfg.bin。

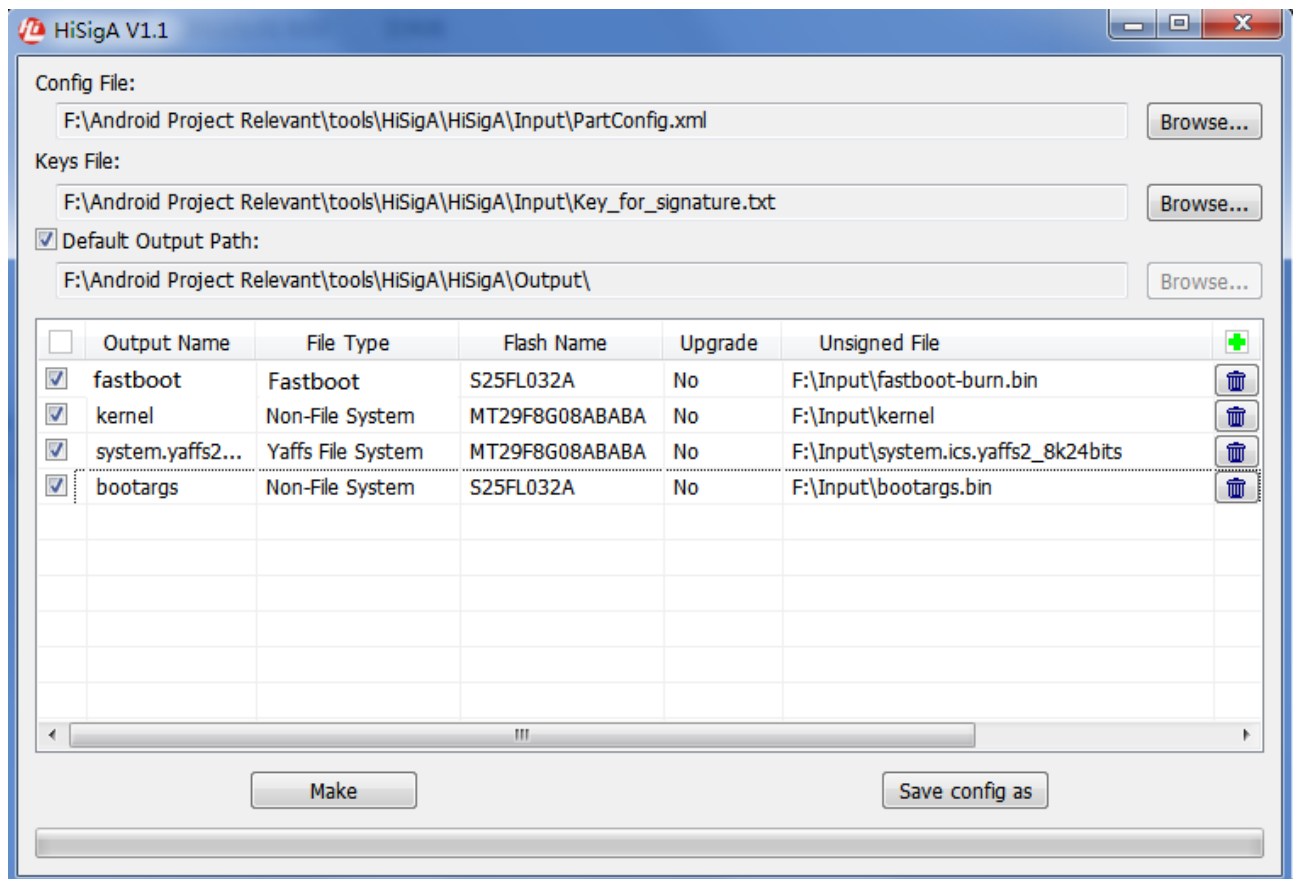




## 1.9 签名 fastboot 等镜像

经过以上准备后，接下来就可以对 bootargs, fastboot, kernel, system 和 recovery 进行签名。签名可以使用海思提供的签名工具 HiSigA 进行。同时，该签名工具也支持生成 recovery 升级需要的升级包。

图1-4 HiSigA 的用户界面



步骤 1 新增任意一个目录，将 1.4 - 1.8 章节中生成的以下文件放进该目录中：

- cfg.bin
- Key\_for\_signature.txt
- bootargs.bin
- fastboot-burn.bin
- kernel
- system
- userdata



### 注意

所有的输入文件必须位于同一个目录中，包括 cfg.bin, Key\_for\_signature.txt, kernel, system, bootargs, userdata 等输入文件。



步骤 2 **Config File:** 用户可以通过“Save config as”按钮保存本次签名的配置文件信息，下次使用时可以选择上次保留的配置文件恢复上次的配置。

步骤 3 **Keys File:** 选择步骤 1 中新增的目录中的 Key\_for\_signature.txt。

步骤 4 **Default Output Path:** 选择签名后的文件输出路径。

步骤 5 可以通过按钮“”增加需要签名的分区。

步骤 6 **Output Name:** 用户可以根据自己的需求输入签名后生成的文件名字。

步骤 7 **File Type:** 文件类型

- **None:** 选择该类型则不会对输入的文件进行签名，只是将其打包进升级包 update.zip 中，主要是在制作升级包时针对 userdata 分区。
- **Fastboot:** 如果是对 fastboot 进行签名，则选择该类型。
- **None-File System:** 对除了 fastboot 和文件系统之外的文件进行签名时，选择该类型，比如 bootargs, kernel, recovery 等。
- **Yaffs File System:** 对 yaffs 文件系统进行签名。
- **Ext4 File System:** 对 Ext4 文件系统进行签名。

步骤 8 **Flash Name:** 在文件夹 Config 中，FlashConfig.xml 保存了 flash 型号的配置信息，包括：

- FlashName
- FlashType
- PageSize
- BlockSize
- ECCSize

用户可以通过手工修改该文件来增加实际需要的 flash 型号信息，主要是：

- PageSize
- BlcokSize
- ECCSize

用户需要根据实际需要修改 FlashConfig.xml 来增加自己需要的 Flash 类型，并且在签名时根据需要选择相应的 Flash 类型。



说明

如果是 EMMC 类型，PageSize 和 BlockSize 应设置成一样的。

步骤 9 **Upgrade:** 如果是签名，不制作 update.zip 升级包，则选择 No；如果制作升级包，则选择 Yes。

步骤 10 **Unsigned File:** 选择需要签名的文件。



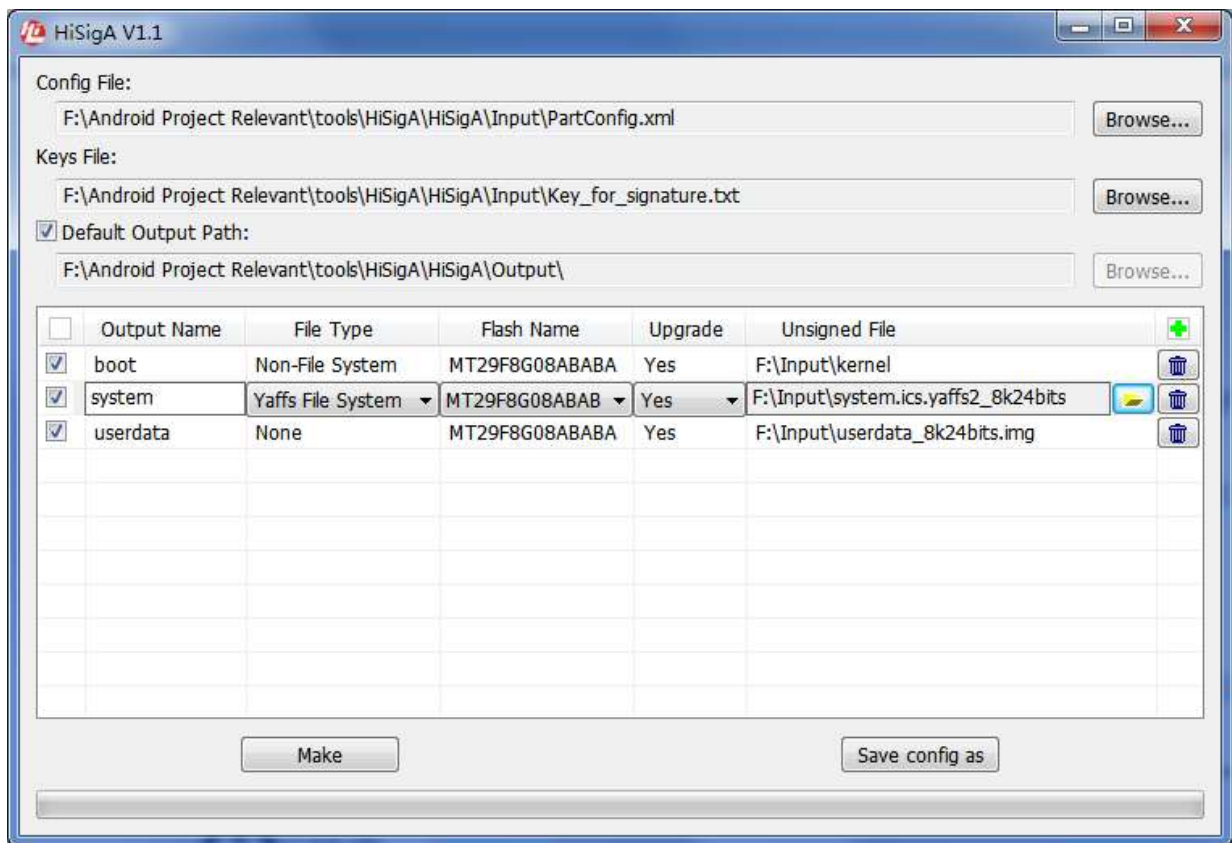
步骤 11 点击“**Make**”，签名工具即会在输出文件夹中输出对应的签名文件。签名工具在勾选上签名 fastboot 分区时，会自动生成 KeyPart.img，该文件是密钥文件，需要烧写到 Key Partition 分区中。

签名的文件生成后，使用 HiTool 将签名后的文件烧写到对应的分区中，比如 fastboot 烧写到 fastboot 分区。

## 1.10 制作 recovery 升级包

制作 recovery 升级包需要通过 HiSigA 工具。制作的步骤如下：

图1-5 HiSigA 制作升级包过程



步骤 1 按照章节 1.9 的步骤 1 – 步骤 8 做好准备。

步骤 2 将 bootable/recovery/mkrecovery/update/file/META-INF.zip、bootable/recovery/mkrecovery/update/目录中的 testkey.pk8、testkey.x509.pem 放在 input 目录，用于打包时对升级包签名

步骤 3 将需要打包进升级包的文件的 Upgrade 选择为“Yes”。注意：userdata 不需要进行签名，所以 File Type 必须选择为 None。

步骤 4 **Unsigned File:** 选择需要升级的文件。

步骤 5 点击“Make”在输出路径生成升级包 update.zip。



制作 recovery 升级包时，kernel 的输出名字必须为 boot，文件系统的输出名字为 system，userdata 的名字为 userdata。此要求是 Andorid recovery 功能的要求。

## 1.11 烧写安全启动密钥和关键数据

本节主要介绍如何在开发调试阶段如何将安全启动相关的密钥和身份认证相关的关键数据烧写进机顶盒中。

安全启动相关密钥参考 1.7 。

终端身份认证过程中需要使用到的关键数据的长度和存储位置见表 1-2。关于终端身份认证的详细介绍，参考 2.3 。

表1-2 关键数据长度和存储位置

序号	数据	长度 (BYTE)	位置	备注
1	RSA1024 公钥	128	OTP	向电信申请，每个机顶盒厂商都不一样，甚至同一个机顶盒厂商不同批次盒子都可能不一样
2	运营商标识	4	deviceinfo 分区	跟电信协商定义为字符串“CTIT”
3	终端 SN	24	OTP	向电信申请，每个盒子对应唯一的终端 SN
4	终端 IN	24	OTP	向电信申请，每个盒子对应唯一的终端 IN
5	Chip ID	4	OTP	由海思在芯片出厂前烧写到 OTP 中，每颗芯片的 Chip ID 唯一

安全启动相关密钥和关键数据需要同时烧写进机顶盒中，烧写的步骤如下：

步骤 1 使用章节 1.7 中生成的 Key\_for\_verify.txt，该文件中包含了安全启动相关密钥。

步骤 2 向电信申请关键数据，电信同意后，阿网会提供以下两部分数据：

- RSA1024 公钥，参考表 1-2 中序号 1，阿网会通过邮件等形式提供。阿网提供的 RSA 1024 公钥格式如下：



```
AE705B43CF6935C389062AED0BA506543514C579B18057D2D20C0E7AD7C1E67ED5DD03D6655
F34CE3F228A9923828D1DD4FEAFDD602EB2D844F4ADE21EE4EC710A77CB290BD827DF855735
3008D7412701164F2AEF476B73FEA8C26FD972EE4F5B474CCB1B521D1D4A73A339E2A7EEB1C
F90CE28AAB6C12369C9E337B57DCDB9
0184EB
```



**注意** 该数据为两行，第一行黑色字体为 RSA 公钥的 N 值，第二行蓝色字体为 RSA 公钥的 E 值。

- 阿网提供的终端 SN 和 IN 格式如下：

```
SNXXXXXXXXXXXXXXXXXXXXXXXXXXXX,NXXXXXXXXXXXXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXX,NBXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX,
HAXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
```

‘X’代表实际的数据。每个盒子的终端 SN 和 IN 不一样，一个 SN 号一行，例如有 200 台机顶盒，阿网将会提供一份 200 行以上数据的文件。

**步骤 3** 收到关键数据后，请按照以下格式组织后另起一行将数据粘贴到 Key\_for\_verify.txt。

- 红色字体为 RSA1024 公钥，参考表 1-2 序号 1。需要替换成步骤 2 中阿网提供的 RSA 公钥。
- 蓝色字体的部分固定不变。
- 绿色部分为终端 SN 和 IN，参考表 1-2 序号 3 和 4 的数据。需要替换成步骤 2 中阿网提供的 SN 和 IN 号。

如果需要烧写另一个机顶盒的关键数据，只需要将以下数据中最后一行的绿色部分的 SN 和 IN 替换成阿网提供的同一批次的另一个 SN 和 IN。红色和蓝色部分不变。

```
AE705B43CF6935C389062AED0BA506543514C579B18057D2D20C0E7AD7C1E67ED5D
D03D6655F34CE3F228A9923828D1DD4FEAFDD602EB2D844F4ADE21EE4EC710A77C
B290BD827DF8557353008D7412701164F2AEF476B73FEA8C26FD972EE4F5B474CCB1
B521D1D4A73A339E2A7EEB1CF90CE28AAB6C12369C9E337B57DCDB9
```

0184EB

CTIT

chipid:68616973

time:20120101000000

random:0000000000000000

aes:0D3CED9BEC10A777AEC23CCC353A8C08

```
SNXXXXXXXXXXXXXXXXXXXXXXXXXXXX,NXXXXXXXXXXXXXXXXXXXXXXXXXXXX
XXXXXXXX,NBXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX,HAXXXXXX
XXXXXXXXXXXXXXXXXXXXXXXXXXXX
```



**说明**

手工烧写的时候，阿网生成数据时是不知道 chipid 的，所以先用某个特定值（比如 68616973）来代替，而校验数据计算的时候，也是用这个 chipid。烧写完之后，需要上



报真实的 chipid，之后的正式认证的接口使用真实的 chipid，如果是在线烧写的话，就是另外一个数据格式，参考工厂生产 apk 开发。

步骤 4 使用海思提供的工具 write\_flash 和 write\_otp 烧写安全启动相关密钥和关键数据：

- ./write\_flash Key\_for\_verify.txt
- ./write\_otp Key\_for\_verify.txt

步骤 5 烧写完成后，需要将 SN 对应的芯片 ChipID 和 MAC 地址提供给上海电信（现网）和阿网（Demo 平台）。海思提供的库 libhi\_advca.a 中有获取芯片 Chip ID 接口，参考下面代码的使用：

```
HI_S32 Ret;  
HI_U32 u32ChipID;  
  
Ret = HI_UNF_ADVCA_Init();  
if (HI_SUCCESS != Ret)  
{  
    return HI_FAILURE;  
}  
  
Ret = HI_UNF_ADVCA_GetChipId(&u32ChipID);  
if (HI_SUCCESS != Ret)  
{  
    return HI_FAILURE;  
}  
  
(HI_VOID)HI_UNF_ADVCA_DeInit();
```

步骤 6 断电重启，机顶盒如果正常启动则表明安全启动相关密钥烧写正确。

步骤 7 向科升申请提供身份认证 APK，将该 APK 安装到系统中，跟电信确认步骤 5 中获取的 SN 对应的 Chip ID 和 MAC 地址是否已经导入电信现网或阿网 Demo 平台，断电重启后，盒子将会进行身份认证，身份认证通过则表明关键数据烧写正确。关于身份认证 APK 的更加详细的使用，请咨询科升。

## 1.12 APK 安装管控

Android APK 的发布是需要签名的。签名机制标明了 APK 的发行机构，Android 系统禁止更新安装签名不一致的 APK，防止已安装的应用被恶意的第三方覆盖或替换掉。

对于用户安装的 APK，在安装时，必须首先对 APK 进行签名校验，只有运营商签名过的 APK 才允许安装到系统中。

Android APK 管控的操作步骤如下：

步骤 1 运营商生成密钥和证书

- 运用 Android 自带的工具 make\_key，该工具在 Android 的 SDK 包 /development/tools 目录中。为了方便运营商，海思在组件包的 tools 目录中也保存了该工具。在 windows 或 linux 命令行下执行以下命令：





```
./make_key CTIT '/C=CN/ST=Shanghai/L=Mountain  
View/O=Android/OU=Android/CN=Android/emailAddress=JoeLee@huawei.com'
```

make\_key 的参数意义如下：

1. 生成的 key 的名字，在例子中我们假定生成的密钥名字为“CTIT”。
  2. 生成该密钥的公司的信息。
    - C: Country Name (2 letter code)
    - ST: State or Province Name (full name)
    - L: Locality Name (eg, city)
    - O: Organization Name (eg, company)
    - OU: Organizational Unit Name (eg, section)
    - CN: Common Name (eg, your name or your server's hostname)
    - emailAddress: Contact email address
- 运营商根据实际配置好参数后，运行该命令，工具提示是否需要密码保护，如图 1-6。这里我们不需要输入密码，直接敲回车，即可在当前目录下生成对应的密钥，如图 1-7。示例中的密钥名字为“CTIT”，因此生成的密钥为“CTIT.x509.pem”和“CTIT.pk8”。

图1-6 make\_key 提示是否需要密码保护

```
lizhong@android-164:~/SPC050_CA$ ./make_key CTIT '/C=CN/ST=Shanghai/L=Mountain View/O=Android/OU=Android/CN=Android/emailAddress=JoeLee@huawei.com'  
Enter password for 'CTIT' (blank for none; password will be visible):
```

图1-7 make\_key 生成最终密钥

```
Enter password for 'CTIT' (blank for none; password will be visible):  
creating CTIT.pk8 with no password  
Generating RSA private key, 2048 bit long modulus  
.....+++  
..+++  
e is 3 (0x3)
```



**注意** 运营商务必保管好这两个密钥文件，防止丢失和泄露。

## 步骤 2 运营商从步骤 1 生成的证书中提取公钥

利用海思发布包中提供的 ExtractPubKey 工具，提取证书中的公钥，在 windows 或者 linux 的命令行下执行如下命令：

```
java -jar ./ExtractPubKey.jar CTIT.x509.pem
```

在当前目录下生成一个 publickey.txt 的文件，内容是提取出来的公钥。运营商将此 publickey.txt 文件分发给机顶盒厂商。



说明

CTIT.x509.pem 是示例中生成的证书的名字，请按照实际生成的证书名字填写该参数。

## 步骤 3 机顶盒厂商从运营商获取公钥



机顶盒厂商从运营商那里获取到公钥 `publickey.txt` 后，需要在 `./frameworks/base/core/java/android/content/pm/PackageParser.java` 文件中增加对公钥的检验。修改代码参考如下：

- 在 `PackageParser` 类中增加 `mPublicKey` 字符串，将运营商提供的 `publickey.txt` 中的内容作为 `mPublicKey` 的值，长度为 512 个字节。

*/\*请根据运营商提供的实际公钥修改该变量的值\*/*

```
private static final String mPublicKey =
"c07dc9c0932868c0cbe6ffedf964839815407cbe9f124c38f6360e94442ab2020ca806abbf2
749383b0144174503bab95309e74d8397ea10c885036b4914adfc64e20a317029757bffeac61
3bb4d52234601354ae84a7a47fe7589d45c6f94addc57b812ed2f892d2d9a3c1f82bafacc5e8
8cf37faad1fbb2262ed2bdc6467e6b0281eda29ee5621e61337217da867bc01574f2f4be59b8
fbedb229a342cef3992502441f076e7275c023de3d69be1c3fb4aee8b596f1dea6ed515778e2
93ba8a71bc94c2214b6d1d59b82fe442a829f4897869b62c734a15c953ffe2ba17b791033f83
7d1745ce109020b05249ab00d4e0fcd72e64910ceff873db45be9985f";
```

- 在方法 `collectCertificates` 中增加以下红色字体部分的代码，这部分代码主要功能是从 APK 中提取签名该 APK 的公钥，并且跟电信提供的公钥进行对比，如果相同则进行 APK 的安装，否则返回安装失败：

```
if (certs != null && certs.length > 0) {
    final int N = certs.length;
    String key;
    int begin = 0;
    int end = 0;
    PublicKey publickey = null;
    pkg.mSignatures = new Signature[certs.length];
    for (int i=0; i<N; i++) {
        pkg.mSignatures[i] = new Signature(certs[i].getEncoded());
        publickey = certs[i].getPublicKey();
        begin = publickey.toString().indexOf("modulus:") + 9;
        end = publickey.toString().indexOf("public exponent:") - 5;
        key = publickey.toString().substring(begin, end);
        if (0 != key.compareToIgnoreCase(mPublicKey)) {
            mParseError = PackageManager.INSTALL_FAILED_VERIFICATION_FAILURE;
            Slog.e(TAG, "Package " + pkg.packageName + " certificate failed!");
            return false;
        }
    }
}
```

步骤 4 机顶盒厂商编译 Android 系统。

步骤 5 机顶盒厂商将系统编译出来后，需要将 `out/target/product/godbox/system/app` 目录下 Android 自带的 APK 发给运营商签名。

步骤 6 运营商使用 Android 的工具和步骤 1 中生成的密钥对机顶盒厂商的 APK 进行签名。签名的工具 `signapk.jar` 在 Android 编译后的 `out/host/linux-x86/framework/` 目录。为了方便运营商，海思在组件包的 `tools` 目录中提供了该工具 `signapk.jar`。

在 windows 或者 linux 系统的命令行下进入目录，执行以下命令：

```
java -jar ./signapk.jar CTIT.x509.pem CTIT.pk8 sample.apk sample_sign.apk
```





- CTIT.x509.pem 和 CTIT.pk8: 步骤 1 中产生的证书和私钥。运营商对 APK 签名时需要将其替换成实际生成的证书和私钥。
- sample.apk: 需要签名的 APK 包。
- sample\_sign.apk 签名后的 APK 名字。

步骤 7 机顶盒厂商将运营商签名后的 APK 放在 out\target\product\godbox\system\app 中。重新制作 system 分区镜像。制作镜像的方法参考 Android 发布包文档。



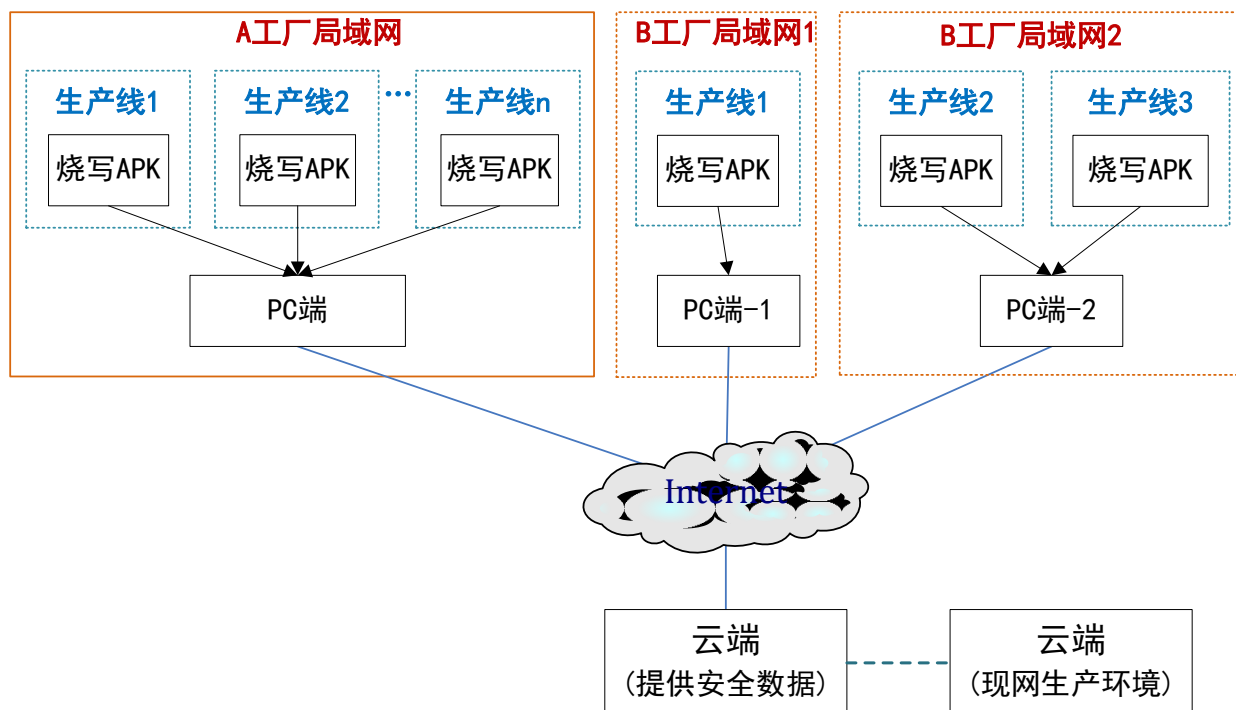
**注意** 机顶盒厂商或者其他 APK 开发者开发的 APK 必须提供给运营商签名，否则在安装过程中将提示失败。

## 1.13 工厂生产 APK 开发

前面的章节 1.11 介绍了如何在实验室将安全启动密钥和关键数据手工烧写进机顶盒中，该方案适合于调试开发阶段。

机顶盒量产时，需要由机顶盒厂商开发烧写 APK，完成跟 PC 端工具交互，请求 PC 端工具通过网络从电信云端获取关键数据，并且将关键数据烧写进机顶盒的功能。同时，该烧写 APK 也负责将安全启动相关密钥烧写进机顶盒中。工厂生产线的示意图如图 1-8。

图1-8 工厂生产线示意图





烧写 APK 的开发涉及的主体及责任如下：

#### 1. 芯片厂商（海思）：

- Key\_for\_verify.txt：使用章节 1.7 中生成的 Key\_for\_verify.txt。
- libhi\_ctit.a 库：提供两个接口供机顶盒厂商把传过来的安全数据进行解密校验后烧写到芯片中。libhi\_ctit.a 提供 debug 和 release 模式，对应 libhi\_ctit\_debug.a 和 libhi\_ctit\_release.a。
  - libhi\_ctit\_debug.a：对机顶盒厂商传进来的安全数据进行解析，并通过串口打印出解析出来的数据，但是不会将数据烧写进机顶盒中。供调试使用。
  - libhi\_ctit\_release.a：对机顶盒厂商传进来的安全数据进行解析烧写进机顶盒中。

#### 2. 机顶盒厂商：

- 烧写 APK：机顶盒厂商负责集成烧写 APK，烧写 APK 需要完成的功能如下：
  - 跟阿网提供的 PC 端工具交互，请求 PC 端工具向云端申请关键数据；
  - 接收到 PC 端工具传过来的关键数据后，调用海思提供的库接口烧写到芯片中；
  - 调用海思提供的库接口将 Key\_for\_verifx.txt 中的安全启动相关密钥烧写到芯片中；
- 关键数据烧写完成后，使用科升提供的身份认证 APK 验证身份认证是否通过。



说明

- 烧写 APK 跟 PC 端工具的交互协议请咨询阿网。
- 对工厂网络环境的要求，配置等请咨询阿网。

#### 3. 阿网：

- 负责给机顶盒厂商提供 PC 端工具，指导机顶盒厂商对工厂网络环境的配置。
- 负责指导机顶盒厂商完成烧写 APK 跟 PC 端工具的交互。
- 负责给电信提供云端。

工厂的生产流程大致如下（更加详细的流程请咨询阿网）：

步骤 1 电信分配 SN（起始 SN 和生产数量）

步骤 2 机顶盒生产工厂提供 MAC 地址，最好是连续的，即提供起始 MAC 地址和数量；如果是分段的，则需要额外转换为 MAC 地址清单

步骤 3 电信生成 SN 序列，每个 SN 绑定一个 MAC 地址，为每个 SN 生成一个安全数据 IN（IN 加密存储），电信将数据备份一次后导出将 SN 和 MAC 地址对应关系表，提供给盒子生产厂商，用于打印 SN 和 MAC 标签。



说明

在一次量产中，SN/MAC 数量应考虑生产过程中的损耗，申请时可以向电信申请多一点的 SN；如果烧写失败或者机顶盒在使用过程中损坏，则 SN/MAC 直接废弃，不需



要通知电信或者阿网。

**步骤 4** 向电信申请 RSA 1024 公钥，关键数据从云端下发下来是加密的。需要使用电信提供的 RSA 1024 公钥解密。

电信提供的 RSA 1024 公钥例子如下：

```
E=014D0E
N=B7DFAB06E24EF1D881465AFACCDCE57296262A69C8289B35F07DECA105EE63D7DE3455443E0837F20
267FF5A681858AAFEDBD3E2AADFA8C12B4BF667590DABA2DEE2C56936E54AD83E2DB3DCCC891498D4D8
FF5CC79E5B093D4E9EE6F9EA2922D4ED09A7EBBB2CD62125B9B031110E727C9A2D0B1D977AA1096A301
276CFB481
```



**注意** 该 RSA 1024 公钥每个机顶盒厂商都会不一样，并且，同一个机顶盒厂商，不同批次的盒子，可能也不一样。以上的数据仅为样例，不可烧写到机顶盒中。

**步骤 5** 工厂烧写 APK 芯片的 Chip ID 为参数向 PC 端请求指定 SN 的关键数据。海思提供的库 libhi\_advca.a 中有获取芯片 Chip ID 接口，接口文件在 device/hisilicon/godbox/driver/sdk/msp/hisecurity/advca/release/include/hi\_unf\_advca.h, 参考下面代码的使用：

```
HI_S32 Ret;
HI_U32 u32ChipID;

Ret = HI_UNF_ADVCA_Init();
if (HI_SUCCESS != Ret)
{
    return HI_FAILURE;
}

Ret = HI_UNF_ADVCA_GetChipId(&u32ChipID);
if (HI_SUCCESS != Ret)
{
    return HI_FAILURE;
}

HI_UNF_ADVCA_DeInit();
```

获取到的 Chip ID 是 unsigned int 类型的，需要转化为大端的字符数组。例如获取到的 Chip ID 为 0x12345678，传给 PC 端的格式如下：

```
HI_U8 u8ChipIdBuf[4];

u8ChipIdBuf[0] = 0x12;
u8ChipIdBuf[1] = 0x34;
u8ChipIdBuf[2] = 0x56;
u8ChipIdBuf[3] = 0x78;
```

**步骤 6** 从 PC 端获取到关键数据后，调用海思的库 libhi\_ctit\_release 中的库接口 HI\_UNF\_ADVCA\_SetCTITData 将关键数据烧写到机顶盒中。接口描述请参考文件



device\hisilicon\godbox\driver\sdk\msp\hisecurity\advca\release\include\hi\_unf\_ctit.h, 每个关键数据的长度为 327 字节。设置关键数据的代码参考如下:

```
HI_S32 Ret = HI_SUCCESS;
HI_CHAR s8Data[327]; //将步骤 6 中从 PC 端获取到的关键数据拷贝进该缓冲
HI_CHAR Rsa_E[] = "014D0B"; //数据仅供参考, 请替换成步骤 4 电信发送过来的 RSA 1024 公钥 E 值
HI_CHAR Rsa_N[] =
"B7DFAB06E24EF1D881465AFACCDCE57296262A69C8289B35F07DECA105EE63D7DE3455443E0837F202
67FF5A681858AAFEDBD3E2AADFA8C12B4BF667590DABA2DEE2C56936E54AD83E2DB3DCCC891498D4D8F
F5CC79E5B093D4E9EE6F9EA2922D4ED09A7EBBB2CD62125B9B031110E727C9A2D0B1D977AA1096A3012
76CFB481"; //数据仅供参考, 请替换成步骤 4 电信发送过来的 RSA 1024 公钥 N 值
Ret = HI_UNF_ADVCA_SetCTITData(s8Data, Rsa_E, Rsa_N, 327);
if(Ret != 0)
{
    Return HI_FAILURE;
}
```

步骤 7 调用海思的库 libhi\_ctit\_release 中的库接口 HI\_UNF\_ADVCA\_SetSCSDData 将 Key\_for\_verify.txt 中的安全启动相关密钥烧写到机顶盒中。安全启动相关密钥的数据长度为 1132 个字节。参考代码如下:

```
HI_S32 Ret = HI_SUCCESS;

/*数据仅供参考, 请替换成实际数据*/
HI_CHAR s8Data[] =
"85b278cd23855568d9da05d44c32e182f95fef69539c4c3bd6f79658fa2aba4737d2ada1610969ce0b
5dec9d00b5905edff6f1fae94642b91f5f34d79c5d59f906ef271cf9fec8834a448280e392f128c223b
89d0bb0c0920ff3da833b304322a30d88cbe48215f166ff5f132bd40095aa6e1c41526fa6a8499dccc3e
39e58be2243fbec39640d9bb99805de9d945d0defd90053ff7f4a30374cc18b669b61858a5cd352b57
5b2145cf2e61a6d44705d1c0aef1418484c42f79edb80b273cfce57676beb803af646fd80ddac0fc053
abb7b855a5e943a1dd607bef98f08fae61f59b01733588c51533cad00277e0b173b8678cc91808bdb29
186913867891c49010001513defe9765263b1f689e210880900188221e32d4ce953c415261169c66f6a
079932f62b24a01452604985b0ada9ec0aafd419cd35d7f7ccbe334289b1cb532d89f147d9ae60c1787
04e0bf7679f86634fcb0ca17639be2337951e3af79762e1a6cc0d5edad363382ac6fe97aaa2159b47f1
aaa12a62f756b1c754a2a5aa7b82eee209df62715fe7f39a089ccae5b9f3d1ac000c1a28cb848bd68a5
cb7413140236256dad6780cdeff8ad257f45218f299f9fc185b7da907600103c8fab58d018f728ba779
836422e06c3dd3f7d2528f8fcd8b53061a9b7630c1e6d1ad33df203450a45d9aaa69aba62felb011039
f5e5ede0af9d95198179d92f1174d5474020bcc318b15405dc8c43f381f8c02950ba0073f8a188506e4
89b57c578d201ee5010001348e0b649a2917eb46a277ae923b72ab";

Ret = HI_UNF_ADVCA_SetSCSDData(s8Data, 1132);
if(Ret != 0)
{
    Return HI_FAILURE;
}
```

步骤 8 验证终端数据是否正确。

- i. 获取挑战字: 调整的云端验证 SN 有效 (由于烧写客户端不连互联网, 无法进行 NTP 同步, 因此这里不验证时戳有效性), 下发挑战字 ChallengeCode = RSA1024(“CTIT”+T1+R1+T2+R2), TMAuthURL, Result, Description, 解密挑战字需用接口 HI\_CTIT\_RSA\_Decrypt()。



- ii. 获取 TMtoken: 终端管理客户端（由 PC 端代理）向云端请求 TMtoken, 携带参数 SN, AlgorithmID=1[SHA256], Authenticator=SHA256(SN+ChipID+IN+T2+R2)), 这里 SHA256 算法的接口用海思提供的接口 HI\_CTIT\_HASH256 ( ) 来运算; 云端验证 Authenticator 的有效时, 下发 TMtoken (TMtoken, 8640000, " ", " ", Result, Description), 更新 SN 状态为 “ready”, 记录日志; 否则更新 SN 状态为 “error”, 记录日志。



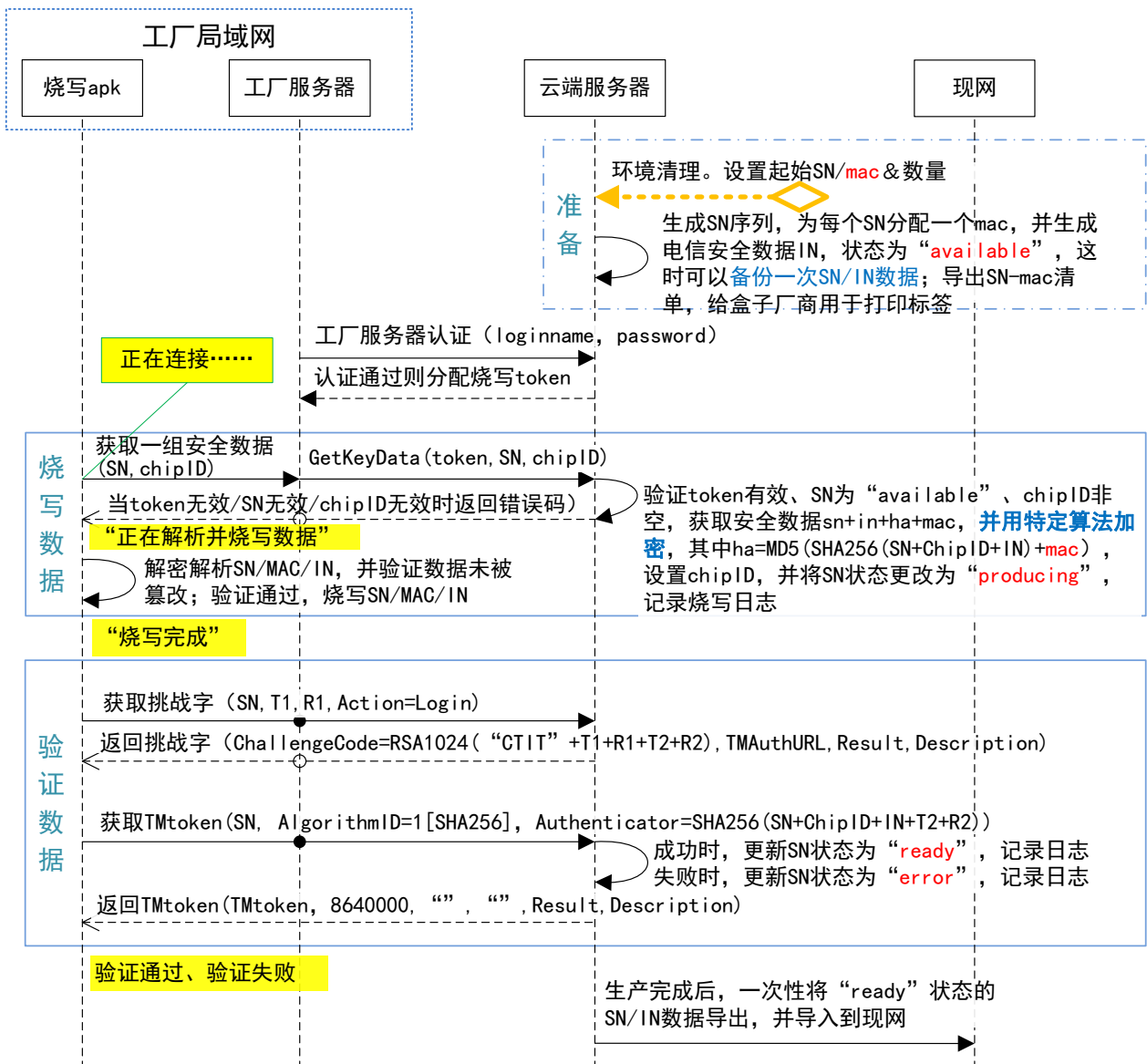
说明

身份验证详细过程需要参考《上海电信智能机顶盒安全生产工具（草案）》，海思提供身份验证 api 及说明。

步骤 9 断电重启, 验证机顶盒是否能够正常启动

步骤 10 使用科升的身份认证 APK, 验证身份认证是否能够通过。

4. 工厂生产流程图:



## 1.14 身份认证 API 说明

身份认证 APK 的开发由科升完成，海思负责提供身份认证 APK 需要使用的加解密 API，以下 API 供科升使用。海思提供的 API 在库 libhi\_ctit\_release.a 中，对应的.h 文件为 device\hisilicon\godbox\driver\sdk\msp\hisecurity\advca\release\include\hi\_unf\_ctit.h。本小结主要介绍海思提供的加解密 API 的使用。

- HI\_U32 HI\_CTIT\_readSN(HI\_U8\* u8data, HI\_U32 len);

获取机顶盒 SN 号，SN 号 24 个字符，末尾'\0'，至少要 25 个字符长度的空间。参考代码如下：

```
HI_U8 u8Sn[30];
```



```
HI_U32 Ret = HI_SUCCESS;
```

```
Ret = HI_CTIT_readSN(u8Sn, 24);
```

```
if(Ret != HI_SUCCESS)
```

```
{
```

```
    return HI_FAILURE;
```

```
}
```

- HI\_VOID HI\_CTIT\_random(HI\_U32 \*rand, HI\_U32 len);

产生随机数，参考代码如下：

```
HI_U32 rand[4];
```

```
memset(rand, 0, sizeof(rand));
```

```
HI_CTIT_random(rand, 16);
```

- HI\_U32 HI\_CTIT\_RSA\_Decrypt(HI\_U8 \*inputbuf, HI\_U8 \*outputbuf, HI\_U32 outputlen)

解密电信终端管理平台发送过来的挑战字，参考代码如下：

```
HI_U32 Ret = HI_SUCCESS;
```

```
HI_U8 u8ChallengeWord[128]; //实际数据为终端管理平台下发下来的挑战字
```

```
HI_U8 u8OutputBuf[64];
```

```
Ret = HI_CTIT_RSA_Decrypt(u8ChallengeWord, u8OutputBuf, 64);
```

```
if(Ret != HI_SUCCESS)
```

```
{
```

```
    return HI_FAILURE;
```

```
}
```



#### 说明

电信终端管理平台下发的挑战字为 256 个字节的字符串，需要转化为 128 个字节的 16 进制数组。例如管理平台下发的挑战字为 12AB45CD...，则应该转化为以下数据传给 API：

```
u8ChallengeWord[0] = 0x12;
```

```
u8ChallengeWord[1] = 0xAB;
```

```
u8ChallengeWord[2] = 0x45;
```

```
u8ChallengeWord[3] = 0xCD;
```

- HI\_U32 HI\_CTIT\_HASH256(HI\_U8 \*vendor, HI\_U8 time[14], HI\_U8 random[16], HI\_U8 \*outbuf, HI\_U32 out\_length);

身份认证 APK 请求安全 CPU 计算挑战字，将计算得到的挑战字返回给终端管理平台认证。参考代码如下：

```
HI_U32 Ret = HI_SUCCESS;
```

```
HI_U8 vendor[] = "CTIT"; //固定为 CTIT
```

```
HI_U8 u8Time[14]; // HI_CTIT_RSA_Decrypt 解密出来的平台时间戳
```

```
HI_U8 u8Random[16]; // HI_CTIT_RSA_Decrypt 解密出来的平台随机数
```

```
HI_U8 u8OutBuf[32];
```

```
Ret = HI_CTIT_RSA_Decrypt(vendor, u8Time, u8Random, u8OutBuf, 32);
```

```
if(Ret != SUCCESS)
```

```
{
```

```
    return HI_FAILURE;
```

```
}
```



说明

HI\_CTIT\_HASH256 返回的挑战字为 32 字节的 16 进制数，终端管理平台要求的为 64 字节的字符串，需要做转化。例如返回的挑战字为：

```
u8OutBuf[0] = 0x12;  
u8OutBuf[1] = 0xAB;  
u8OutBuf[2] = 0xEF;  
u8OutBuf[3] = 0x34;
```

则转化出来的 64 字节的字符串如下：

```
u8ChallengeStr = "12ABEF34....."。
```





# 2 电信安全要求概述

## 2.1 安全性目标

中国电信为了确保基于开放 Android 操作系统的业务在运行过程中软件的安全性，从而保证业务的安全性，对终端提出了相应的安全技术要求。

主要的安全技术要求包括以下两点：

- 终端安全启动要求
  - 系统安全引导过程
  - 系统软件（Kernel/文件系统/Recovery）安全启动、升级
  - APK 安装管控
- 终端身份认证功能要求

## 2.2 终端安全启动要求

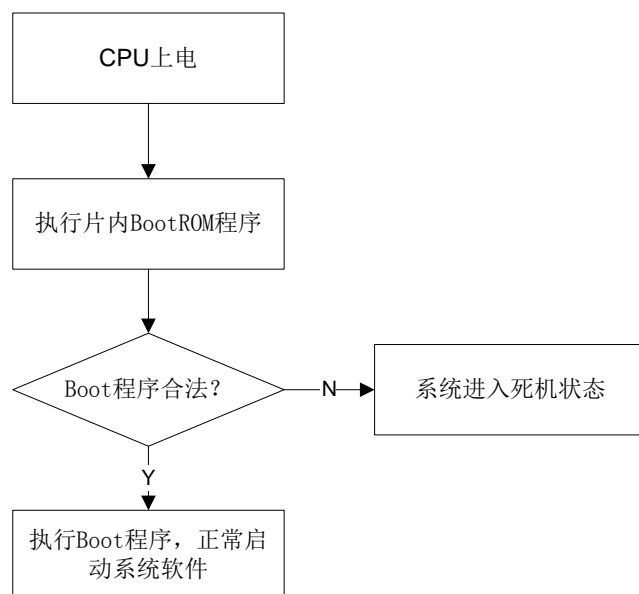
本章主要介绍了电信对终端启动，执行 Kernel 和文件系统以及升级过程的安全性要求。

### 2.2.1 系统安全引导过程

系统的安全引导过程必须符合以下基本流程：



图2-1 系统的安全引导过程



具体要求如下：

- CPU 对 Boot 的验证可采用安全、有效的验证算法，如基于 SHA256+RSA2048 的数字签名验证，或基于 AES128 加密的代码加密方法。海思对 Boot 的校验采用的是 SHA256+RSA2048。
- 用于验证的密钥必须确保安全性，如必须使用 OTP 区域存放密钥，如果使用存放在 Flash 中的密钥，则必须使用安全有效的加密与签名验证机制确保密钥的安全。海思对 Boot 进行签名校验的密钥存储在 OTP 中。
- 对引导程序的签名验证必须采用全覆盖签名验证。

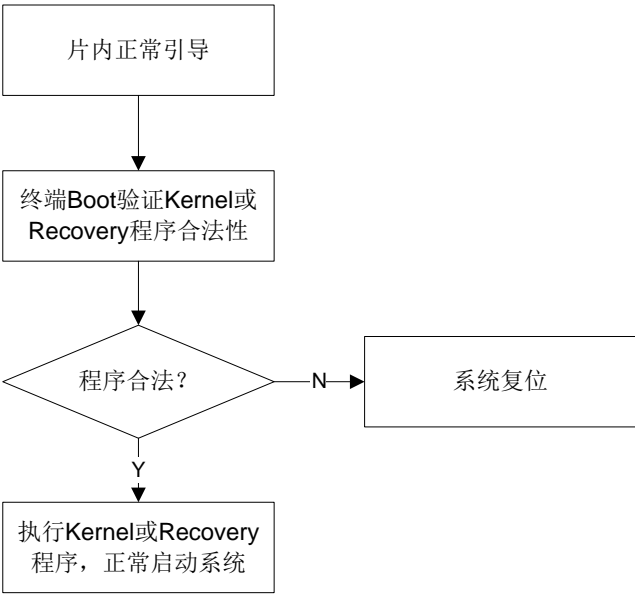
## 2.2.2 系统软件安全启动、升级

终端的 Boot 程序通过合法性验证后，它必须对后续执行的代码进行合法性验证。

根据系统状态，终端 Boot 可能执行 Kernel 或 Recovery 程序，其执行过程必须符合以下基本流程：



图2-2 系统软件安全启动、升级



具体要求如下：

- 终端 Boot 对 Kernel 或 Recovery 程序的合法性验证必须采用基于 SHA256+RSA2048 的数字签名验证，或基于 AES128 加密的代码加密方法。
- 用于验证的密钥必须确保安全性，如必须使用存放的 OTP 区域的密钥，如果使用存放在 Flash 中的密钥，则必须使用安全有效的加密与签名验证机制确保密钥的安全。
- 对 Kernel 或 Recovery 程序的签名验证必须采用全覆盖签名验证。
- Kernel 在加载文件系统之前必须对文件系统的数据进行合法性验证。为了提升系统启动速度，可以对文件系统采用随机抽样的方式进行合法性验证，比如将文件系统按照每 1M 计算一个签名值，校验时随机抽取一些数据块进行校验，保证每次启动时校验的数据不同。
- 在系统升级过程中，Recovery 程序必须对将要用于系统升级所有数据采用与 Boot 相同的合法性验证方法进行验证。

系统中的分区的校验情况见表 2-1 系统分区校验表。

表2-1 系统分区校验表

分区名	是否校验
fastboot	全校验
bootargs	全校验
key	全校验

recovery	全校验
deviceinfo	不校验
baseparam	不校验
logo	不校验
fastplay	不校验
cache	不校验
misc	不校验
kernel	全校验
system	随机校验，随机抽取一些数据块进行校验。
userdata	不校验
blackbox	不校验
sdcard	不校验

### 2.2.3 APK 安装管控

Android APK 的发布是需要签名的。签名机制在 Android 应用和框架中有着十分重要的作用。签名机制标明了 APK 的发行机构，Android 系统禁止更新安装签名不一致的 APK，防止已安装的应用被恶意的第三方覆盖或替换掉。站在软件安全的角度，通过比对 APK 的签名情况，判断此 APK 是否由官方发行，而不是被破解篡改过重新签名打包的盗版软件。

对于用户安装的 APK，在安装时，必须首先对 APK 进行签名校验，只有中国电信签名过的 APK 才允许安装到系统中。

## 2.3 终端身份认证功能要求

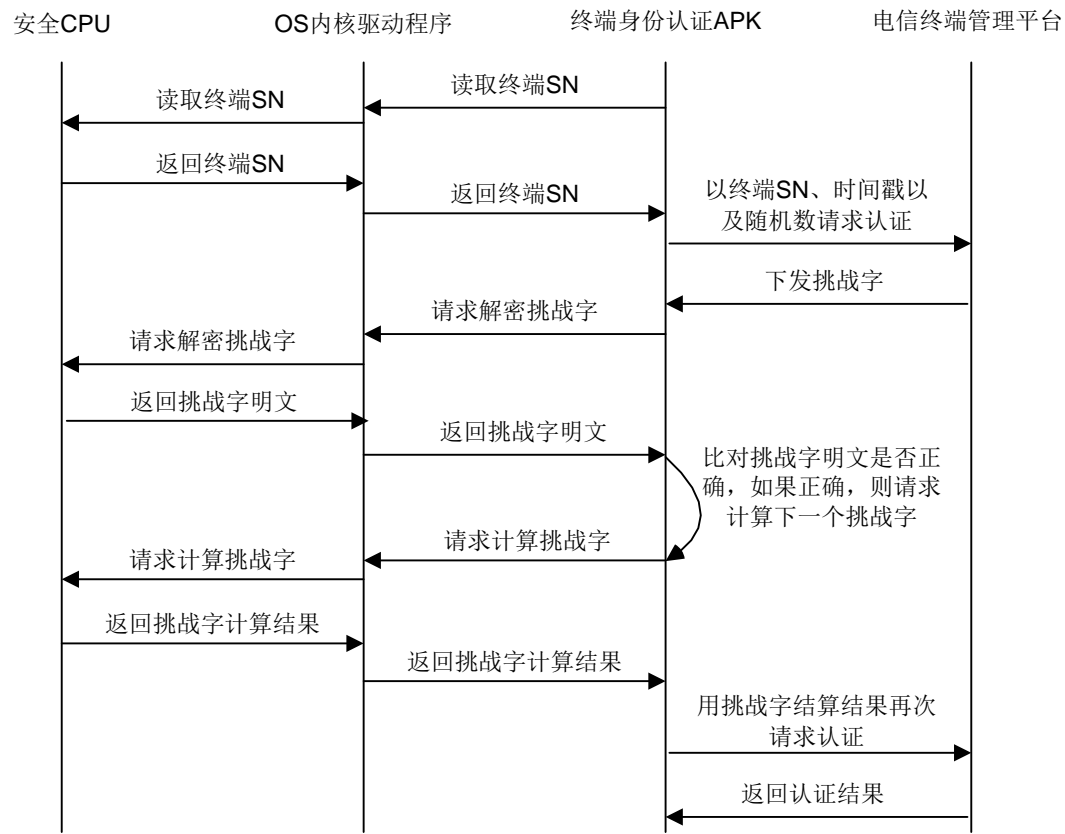
根据电信的要求，终端安全启动挂载完文件系统后，终端上的身份认证 APK 必须通过网络跟电信前端的终端认证平台进行认证，只有通过身份认证的终端才能接入电信的网络，身份认证失败的情况下终端禁止执行任何应用程序，并且通过菜单提示用户身份认证失败。



2.3.1 终端身份认证过程

终端身份认证过程如图 2-3 终端身份认证过程所示：(图中的安全 CPU 指海思芯片专有的一个 CPU，用于处理安全相关算法)

图2-3 终端身份认证过程



2.3.2 挑战字原文与挑战字的生成及验证

挑战字由电信终端管理平台对下列挑战字原文数据用 RSA1024 私钥加密产生：

运营商标识4字节	终端时间戳14字节	终端随机数16字节	平台时间戳14字节	平台随机数16字节
----------	-----------	-----------	-----------	-----------

上述挑战字原文共 64 字节，其中运营商标识符 4 字节，时间戳采用数字表示的时间形式，具体为 yyyymmddhhmmss，即 4 字节数字年，月日時分秒各 2 字节。终端随机数与终端时间戳由终端产生并在请求认证消息中发送到电信终端管理平台。

终端身份认证 APK 请求安全 CPU（海思芯片专有的 CPU）将挑战字解密，获得挑战字原文，然后终端身份认证 APK 对运营商标识、终端时间戳、终端随机数进行对比验证，验证正确后才能进行挑战字的计算。

2.3.3 终端身份认证数据结构

终端 SN 是一串存放在 OTP 中的终端唯一标识，长度 24 字节，Host CPU（ARM Cortex A9）不可读，只有安全 CPU 可以访问，具有唯一性，由以下几个部分组成：

厂商代码4字节	终端型号4字节	产品批次8字节	产品序号8字节
---------	---------	---------	---------

终端 IN 是一串存放在 OTP 中的终端唯一内部序列号，Host CPU 不可读，只有安全 CPU 可以访问，长度 24 字节，具有唯一性，由以下几部分组成：

厂商代码4字节	芯片批次8字节	芯片序号12字节
---------	---------	----------

2.3.4 挑战字返回结果的计算

挑战字返回结果由安全 CPU 计算，具体计算方法如下：

终端身份认证 APK 将挑战字原文中的运营商代码、平台时间戳与平台随机数发送给安全 CPU，安全 CPU 在收到挑战字计算请求后，验证运营商标识是否为“CTIT”后，用 OTP 中保存的 SN、Chip ID、IN，加上平台时间戳、平台随机数，用 SHA256 算法计算散列值作为挑战字计算的返回值。

上述 SHA256 算法的输入具体定义如下：

SN：24字节	ChipID：4字节	IN：24字节	平台时间戳：14字节	平台随机数：16字节
---------	------------	---------	------------	------------



## 3 海思安全方案概述

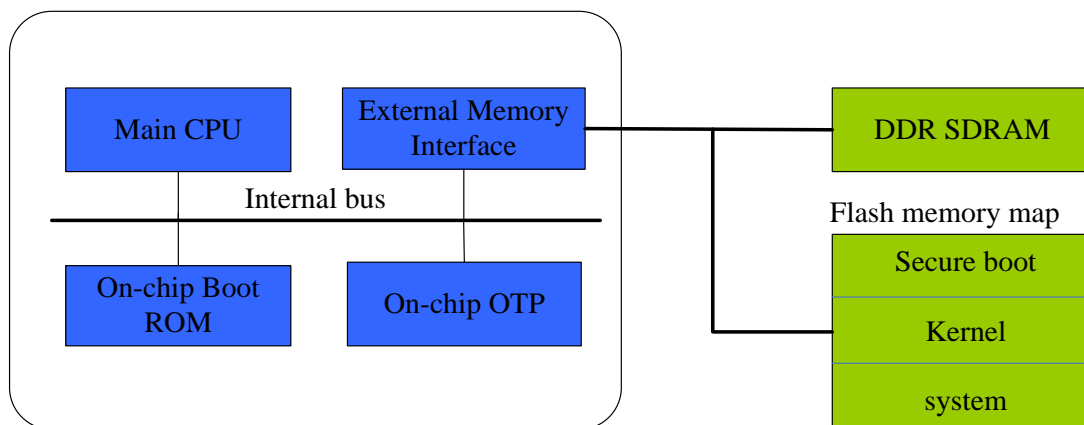
### 3.1 安全性目标

针对电信的安全要求，结合海思目前的安全芯片架构，提出符合电信要求的安全方案，增强海思芯片的竞争力。

### 3.2 系统架构

为了保证系统的安全性，作为系统的最主要的部件芯片必须具备一定的安全机制，该安全机制是保证整个系统安全性的基础。芯片的内部架构如图 3-1 所示：

图3-1 芯片功能结构图



- 芯片内部的 On-chip Boot ROM 固化了一段程序，该程序在芯片出厂后不可更改，也不能被读取。芯片每次上电复位后从 Boot ROM 开始执行，Boot ROM 负责对安全 Boot 进行签名校验，只有签名校验通过后，Boot ROM 才会跳转到安全 Boot 开始执行，系统开始启动，否则系统无法启动。
- 安全 Boot 开始执行后，会依次对 Bootargs, Kernel, System 等分区进行签名校验，只有签名校验通过后，Boot 才会开始引导 Kernel 和文件系统，整个系统开始运行起来。



- 系统运行过程中，如果有升级的需求，系统将会复位进入 Boot，Boot 对 recovery 进行签名校验，只有通过签名校验后，Boot 才会进入 recovery 进行升级，否则 Boot 将直接复位。
- On-chip OTP 用于保存一些不可被读取或不可被更改的数据。主要包括用于签名校验的 RSA Key，AES Key 和运营商要求的一些关键数据等信息，关于关键数据的描述，参考表 1-2 关键数据长度和存储位置。

**说明**

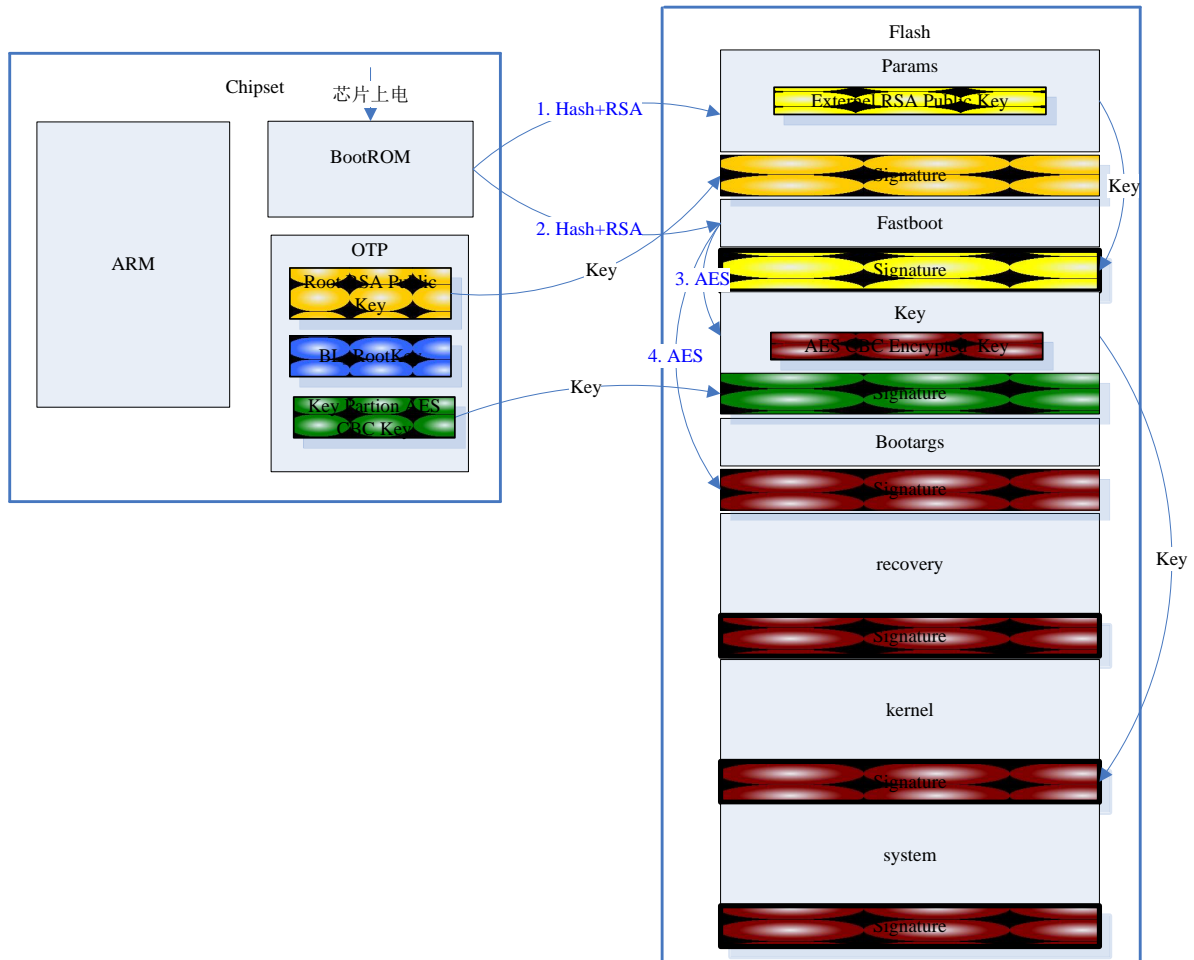
“签名”指的是对任意长度的数据，通过某一个密钥进行计算后，得出固定长度的数值。在验证时，使用对应的密钥对相同的数据进行计算，如果计算结果跟之前的一样，则验证通过；否则认为数据已经被篡改。

### 3.3 系统签名校验启动流程

整个系统启动的安全校验流程如图 3-2 所示：



图3-2 系统签名校验启动流程



关于 OTP 中存储的用于系统安全启动过程中使用的密钥请参考错误！未找到引用源。错误！未找到引用源。。

- 步骤 1 芯片上电后，Boot ROM 使用 OTP 中的 Root RSA Public Key 对 Flash 中的存储的 External RSA Public Key 进行校验。
- 步骤 2 如果 External RSA Public Key 校验通过后，使用 External RSA Public Key 对 fastboot 进行签名校验；否则系统复位。
- 步骤 3 Boot 校验通过后，系统开始执行 Boot；Boot 使用 OTP 中存储的 Key Partition AES CBC Key 对 Flash 中的 Key 分区进行签名校验；Key 分区中存储的是加密的 AES CBC Encrypted Key，该密钥用于签名校验 Bootargs，Kernel，System，Recovery 分区。
- 步骤 4 使用 OTP 中存储的 BL\_RootKey 对加密的 AES CBC Encrypted Key 进行解密，根据系统的运行状态，如果是需要升级的，则使用解密后的 AES CBC Encrypted Key 对 Recovery 进行校验；如果是正常启动的，则使用解密后的 AES CBC Encrypted Key 对 Kernel，System 进行签名校验。



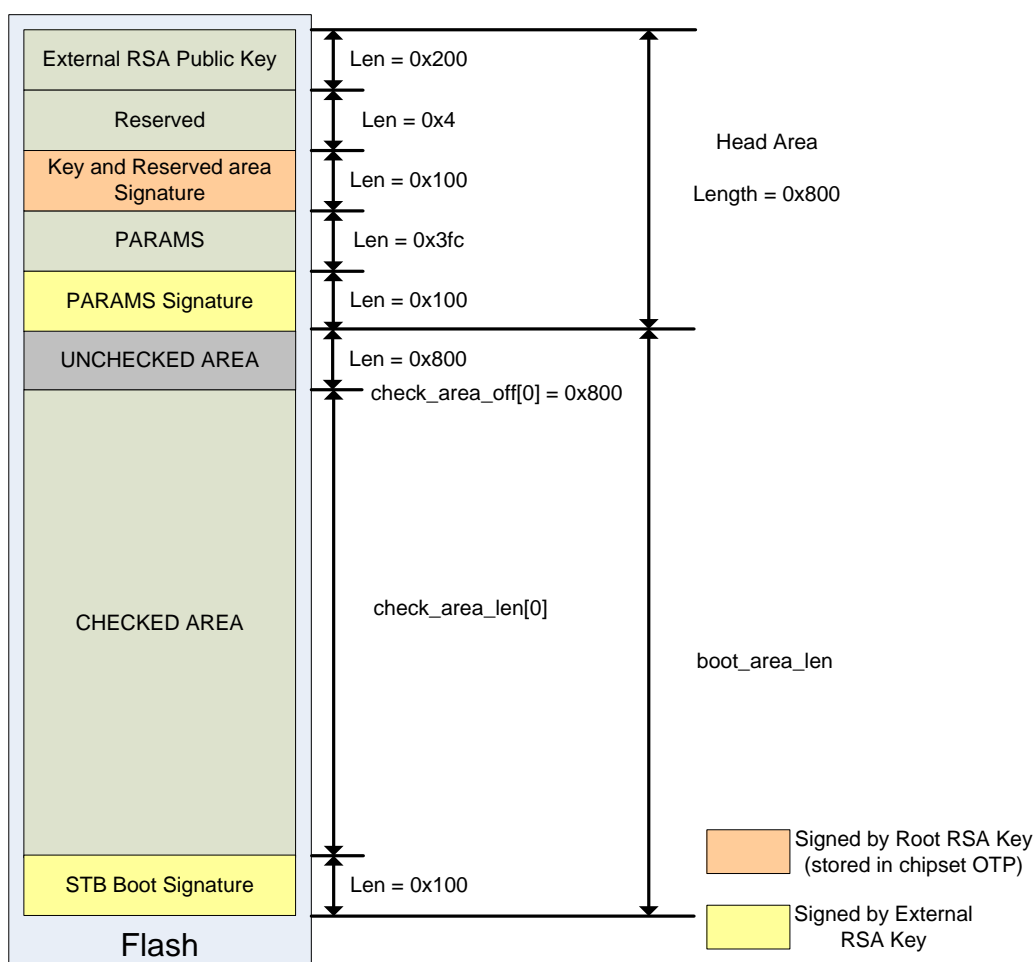
### 3.3.1 安全 Boot 签名校验

为了保证系统的安全性，芯片上电启动后，Boot ROM 首先对安全 boot 进行签名校验，只有签名校验通过后，Boot ROM 才会跳转到安全 Boot 开始执行。对 Boot 的签名校验采用的算法主要是 RSA 和 SHA256，遵循以下标准：

1. 安全 HASH 算法 SHA256，参照 NIST FIPS PUB 180-3.
2. 标准 RSA 签名校验算法 RSASSA\_PKCS1\_V1\_5，参照 PKCS#1 v2.1. RSA key 长度为 2048 bits.

安全 Boot 在 Flash 的内部存储结构如图 3-3 所示：

图3-3 安全 Boot 在 Flash 的内部存储结构



从图 3-3 中可以看出，安全 Boot 主要分为 2 个部分：

- Head Area 区：
  - Key Area：包含 External RSA Public Key、Reserved 保留区域及其签名。
  - Param Area：包含 DDR 的一些配置数据及其签名。
- Boot Area 区：



- Checked Area: 包含 Boot, 所在的 Flash 区域需要进行签名校验。
- Uncheck Area: 用于保存一些私有数据, 所在的 Flash 区域不需要签名校验。

用于校验 External RSA Public Key 的 Root RSA Key 存储于芯片内部的 OTP 区域中, Root RSA Key 不可以被更改, 且只允许被 Boot ROM 访问用于校验 External RSA Public Key。

External RSA Public Key 用于校验 Param Area 和 Boot Area, 对 Boot 的校验形成一个链状的校验机制, 从而保护 Boot 的安全性。



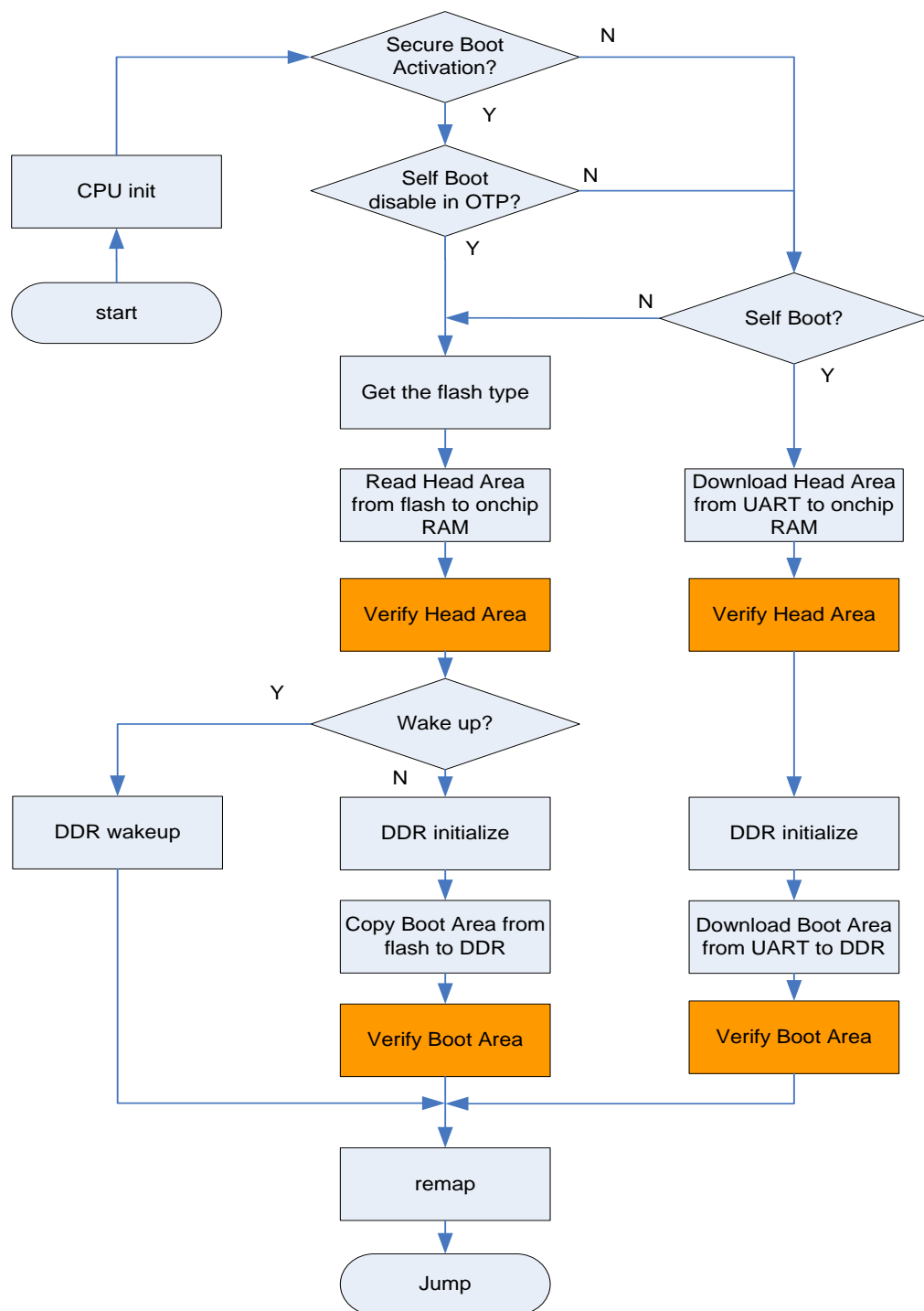
#### 说明

安全 Boot 的 Flash 存储结构中, 各个分区的偏移固定不可更改, 除存储 Boot 的 Check Area 分区长度随着 Boot 的实际大小可变外, 其他分区的长度固定不可改变。

安全 Boot 的校验流程如图 3-4 所示:



图3-4 安全 Boot 的校验流程



### 3.4 系统分区签名校验



系统需要进行签名校验的分区主要包括 bootargs、Kernel、System、Recovery、Key Partition 等分区。Boot 校验通过后，在启动 Kernel、System 或者 Recovery 前，必须先对 Kernel、System 或 Recovery 做校验，只有通过签名校验系统才能启动。同时，Boot 也必须对 bootargs 做签名校验。

Boot 对 Kernel、System、Recovery 等分区进行签名校验时采用的算法为 AES CBC-MAC。校验时采用的密钥存储于 Key Partition 分区中，有关 Key Partition 分区的介绍，请参考 3.4.1 Key Partition 分区介绍。

### 3.4.1 Key Partition 分区介绍

Key Partition 分区用于存储校验 Bootargs，Kernel，System 和 Recovery 等分区用到的密钥 AES CBC Encrypted Key。密钥的结构如图 3-5 Key Partition 分区密钥结构所示：

图3-5 Key Partition 分区密钥结构



- KEY 长度：Key 的长度，固定为 16 个字节。
- KEY ID：标示 Key，便于找到对应的 Key,每个签名镜像中含有这个 KEY ID
- KEY 密文：KEY 数据

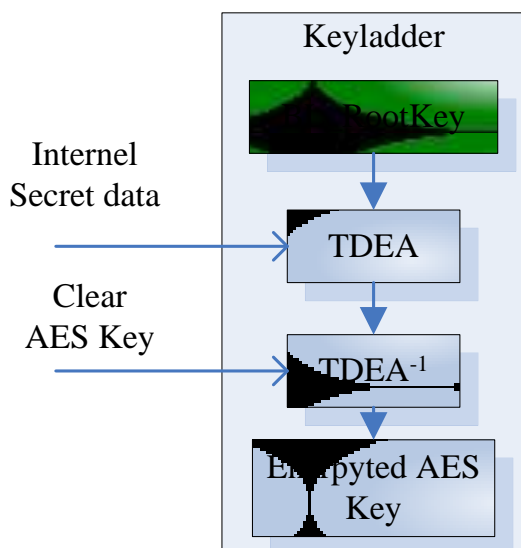
```
typedef struct hi_Key_S
{
    HI_U32 u32KeyLen;
    HI_U32 u32KeyId;
    HI_U8 u8Key[16];
} HI_Key_S;
```

目前的 KeyPartition 分区中仅存储一个密钥 AES CBC Encrypted Key，所有分区包括 Bootargs，Kernel，System 等使用相同的这个密钥进行签名校验。

Key Partition 分区中存储的密钥使用 OTP 中存储的密钥 BL\_RootKey 进行加密，保证数据的安全。加密的过程如图 3-6 Key Partition 分区密钥加密流程所示：



图3-6 Key Partition 分区密钥加密流程



BL\_RootKey 存储在 OTP 中，Host CPU 不可读写，只有安全 CPU 可以访问。

### 3.4.2 Key Partition 分区的签名校验

Key Partition 分区由安全 Boot 对该分区进行签名校验，签名校验采用的算法为 AES-CBC MAC。用于校验该分区的密钥称为 Key Partition AES CBC Key。Key Partition AES CBC Key 位于 OTP 中，Host CPU 不可读写，只有安全 CPU 可以访问。

#### 3.4.2.1 Key Partition 分区的签名

Key Partition 分区签名后的镜像如图 3-7 Key Partition 分区镜像结构图所示：

图3-7 Key Partition 分区镜像结构图



Key Partition 镜像由三部分组成：

1. Header
2. 密钥
3. 签名



Header 包含以下信息:

1. 魔术字, 用于识别 Header
2. 版本号
3. 页大小
4. 块大小
5. OOB
6. 总长度
7. 镜像偏移
8. 镜像大小
9. 签名数据长度
10. 密钥个数
11. Head 偏移
12. 签名偏移
13. 签名长度
14. 密钥是否加密
15. deviceinfo 分区偏移
16. 保留位

```
typedef struct hi_Andorid_KeyPartition_Head_S
```

```
{  
    char u8MagicNumber[32];  
    char u8Version[8];  
    unsigned int u32PageSize;  
    unsigned int u32BlockSize;  
    unsigned int u32Oob;  
    unsigned int u32TotalLen;  
    unsigned int u32ImageOffset;  
    unsigned int u32ImageLen;  
    unsigned int u32SignDataLen;  
    unsigned int u32KeyNum;  
    unsigned int u32HeadOffset;  
    unsigned int u32SignatureOffset;  
    unsigned int u32SignatureLen;  
    unsigned int u32bIsKeyEncrypt;  
    unsigned int u32DeviceinfoOffset;  
    unsigned char u8Reserved[32];  
}
```



```
} HI_Android_KeyPartition_Head_S;
```

签名工具需求:

1. 签名规则

签名范围: (Header + 填充) + (密文 + 填充), Header + 填充, Signature + 填充, 密文 + 填充都要求进行页对齐。

2. 签名算法

AES CBC-MAC

3. 签名的 Key

Key Partition AES CBC Key, 存储在 OTP。

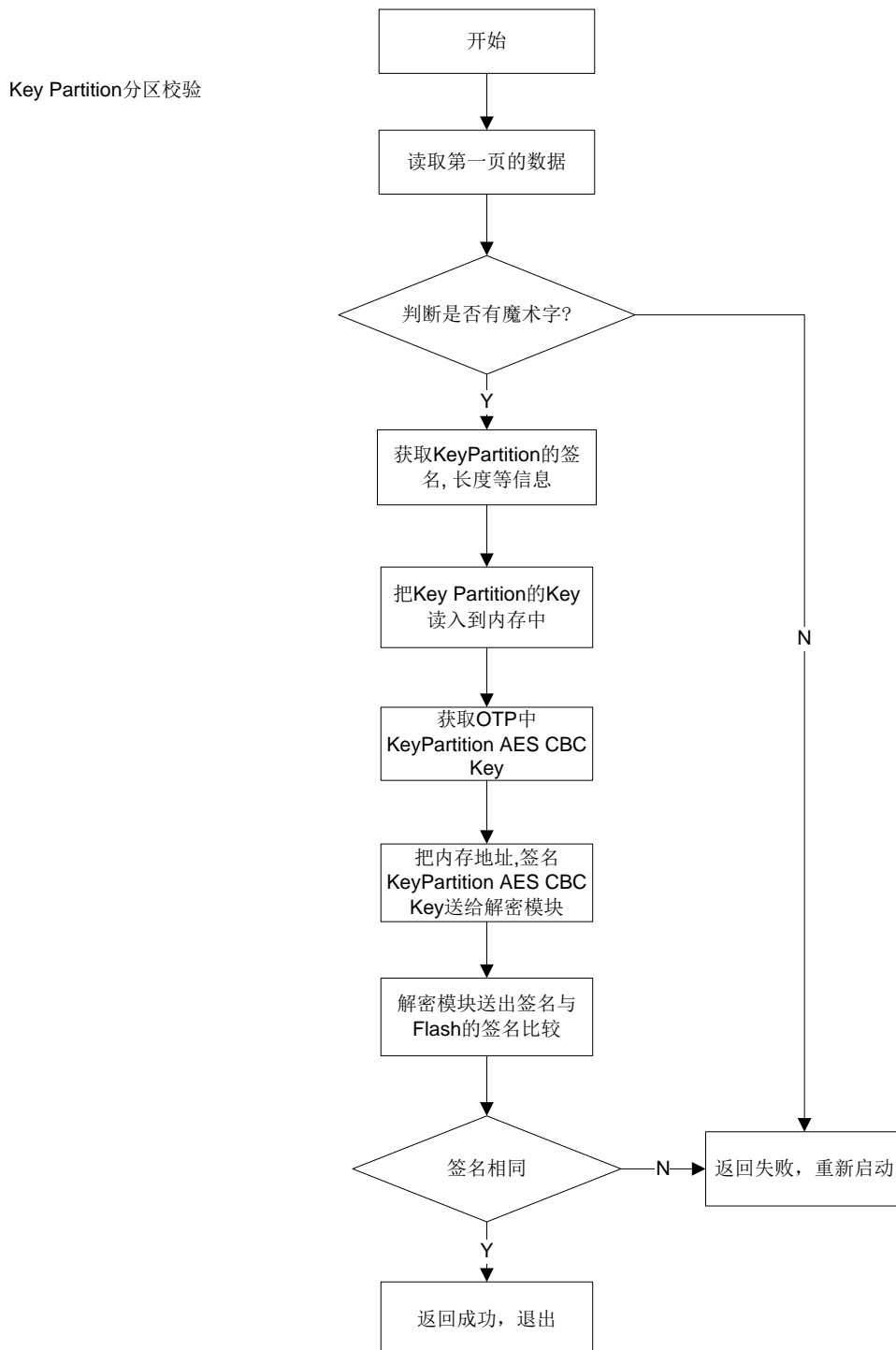
### 3.4.2.2 Key Partition 分区的校验

Key Partition 分区进行签名校验的流程如图 3-8 所示:





图3-8 Key Partition 分区签名校验流程

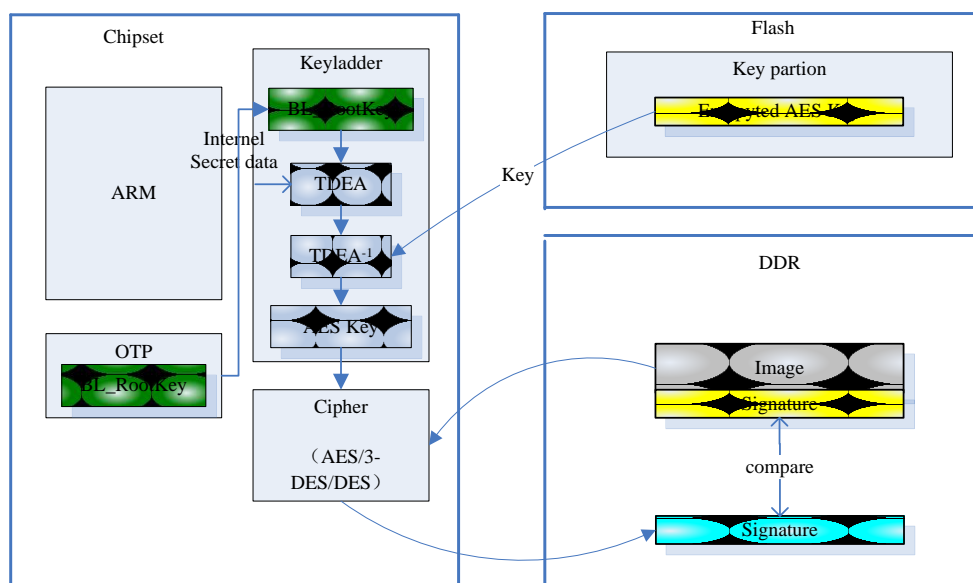


### 3.4.3 非文件系统分区的签名校验

Kernel 等非文件系统分区由安全 boot 进行签名校验，签名校验采用的算法为 AES-CBC MAC，密钥为 Key Partition 分区中存储的 AES CBC Encrypted Key。对 Kernel 等分区进行签名校验的示意图如图 3-9 所示：



图3-9 Kernel 等分区的签名校验示意图



### 3.4.3.2 Kernel 等非文件系统分区镜像的签名

kernel、recovery、bootargs 等非文件系统分区镜像生成签名时，是基于整个镜像的内容。

图3-10 Kernel 等非文件系统分区镜像结构图



签名后的镜像由三部分组成：

2. Header
3. 原始镜像 + 填充
4. 签名

Header 包含以下信息：

1. 魔术字，用于识别 Header
2. 版本号
3. 页大小
4. 块大小
5. OOB



6. 总长度
7. 镜像偏移
8. 镜像大小
9. 签名数据长度
10. 密钥 ID
11. Head 偏移
12. 签名偏移
13. 签名长度
14. 保留位

```
typedef struct hi_Andorid_CAImpHead_S
```

```
{  
  
    char u8MagicNumber[32];  
  
    char u8Version[8];  
  
    unsigned int u32PageSize;  
  
    unsigned int u32BlockSize;  
  
    unsigned int u32Oob;  
  
    unsigned int u32TotalLen;  
  
    unsigned int u32ImageOffset;  
  
    unsigned int u32ImageLen;  
  
    unsigned int u32SignDataLen;  
  
    unsigned int u32KeyId;  
  
    unsigned int u32HeadOffset;  
  
    unsigned int u32SignatureOffset;  
  
    unsigned int u32SignatureLen;  
  
    unsigned char u8Reserved[32];  
  
} HI_Android_CAImpHead_S;
```

签名规则:

1. 签名范围: (Header + 填充) + (镜像 + 填充); Header + 填充, 镜像 + 填充都要求页对齐。
2. 签名算法: AES CBC-MAC
3. 签名 key: AES CBC Encrypted Key, 存储在 flash 的 key partition 分区中。



说明

Kernel 等非文件系统分区镜像签名时需要在原始镜像前添加一个 Header, Boot 在启动 kernel 时会自动跳过该 Header, 因此, 非文件系统分区签名后的镜像烧写地址跟原始的镜像烧写地址一样。

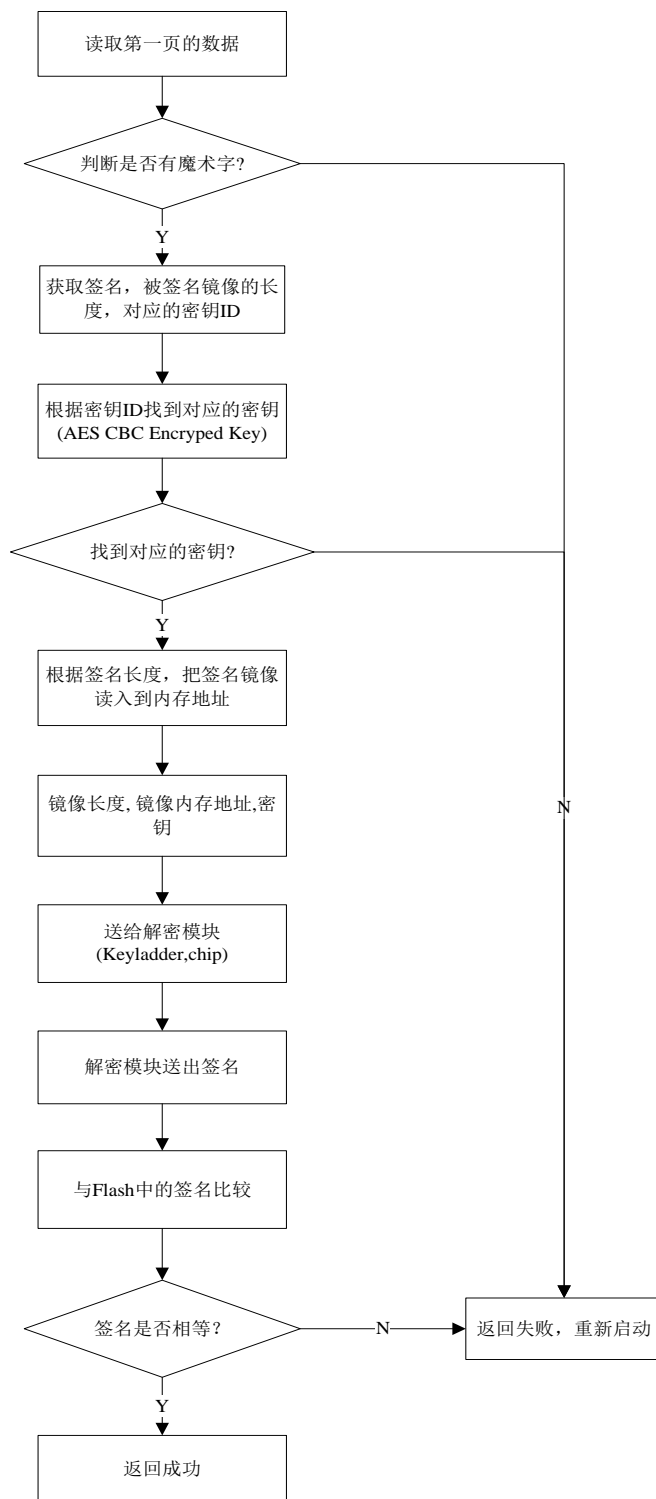


### 3.4.3.3 Kernel 等非文件系统分区镜像的校验

kernel 等非文件系统分区的签名校验如图 3-11 所示：



图3-11 kernel 等非文件系统分区的签名校验流程图





### 3.4.4 文件系统分区镜像的签名

由于文件系统这种镜像体积较大（system 镜像在 Android 上具有 100-200M），整分区校验耗时较长，对启动速度影响较大。目前采用伪随机抽样校验的方式：签名时将整个分区分块进行签名，签名结果附在镜像尾部。签名校验时随机抽样若干块进行签名校验。

#### 3.4.4.1 yaffs（ext4）文件系统签名

图3-12 文件系统签名的镜像结构图



签名后的镜像由三部分组成：

1. 原始镜像 + 填充
2. Header + 填充
3. 签名 + 填充

Header 包含以下信息：

1. 魔术字，用于识别 Header
2. 版本号
3. 页大小
4. 块大小
5. OOB
6. 总长度
7. 镜像偏移
8. 镜像大小
9. 签名数据长度
10. 密钥 ID
11. Head 偏移
12. 签名偏移
13. 签名长度
14. 保留位

```
typedef struct hi_Andorid_CAIImgHead_S
```



```
{  
    char u8MagicNumber[32];  
    char u8Version[8];  
    unsigned int u32PageSize;  
    unsigned int u32BlockSize;  
    unsigned int u32Oob;  
    unsigned int u32TotalLen;  
    unsigned int u32ImageOffset;  
    unsigned int u32ImageLen;  
    unsigned int u32SignDataLen;  
    unsigned int u32KeyId;  
    unsigned int u32HeadOffset;  
    unsigned int u32SignatureOffset;  
    unsigned int u32SignatureLen;  
    unsigned char u8Reserved[32];  
} HI_Android_CAImpHead_S;
```

签名工具需求:

#### 1. 签名规则

- 签名范围: 镜像中的抽样位置 + (Header + 填充)
- Nand 器件对齐要求:

由于 Nand 器件 OOB 数据会改变, 因此签名校验时不能包含 OOB 数据。同时, 由于签名算法采用的是 AES CBC-MAC, 算法要求签名数据必须 16 字节对齐, 因此, Nand 器件上的数据对齐格式要求如下:

- 整个文件要求(块+OOB\*块的页数)对齐
- 镜像+填充要求(页+OOB)对齐
- Head+填充要求(页+OOB)对齐, Header且位于最后一块的第一页
- Signature+Header要求(页+OOB)对齐, 且位于最后一块的第二页

- EMMC 器件对齐要求:

- 块对齐: EMMC器件的块默认是512byte

#### 2. 签名算法

- AES CBC-MAC

#### 3. Key

- AES CBC Encrypted Key, 存储在 flash 的 key partition 分区中。



签名校验方法：

fastboot 从分区尾部向前扫描，通过签名头部魔术字识别签名头部，获取镜像大小和签名信息，然后对镜像内容做签名校验。由于文件系统通过 OOB 识别有效文件内容，签名数据并不破坏文件系统完整性。