



# 上海电信智能机顶盒安全方案 使用指南

文档版本 01

发布日期 2015-03-06

**版权所有 © 深圳市海思半导体有限公司 2015。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总部                    邮编：518129

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

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



# 前 言

## 概述

上海电信提出了针对智能机顶盒的安全规范《上海电信智能终端安全技术要求.pdf》，用于指导芯片厂商和机顶盒厂商智能机顶盒的开发。

上海电信智能机顶盒安全方案主要包括两部分：

- 安全启动
- 身份认证

本文档详细描述了上海电信智能机顶盒的安全方案设计，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
Hi3798M	V1XX0XX

## 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
 <b>DANGER</b>	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。
 <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。

## 作者信息

章节号	章节名称	作者信息
全文	全文	F00172091

## 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
2015-03-06	00B01	第一次临时发布。 支持 Hi3798MV100 上海电信方案。



# 目 录

前 言.....	iii
1 安全启动.....	1-1
1.1 概述.....	1-1
1.2 系统签名校验启动流程.....	1-1
1.3 系统分区签名校验.....	1-2
1.3.1 非文件系统分区的签名校验.....	1-2
1.3.2 文件系统分区镜像的签名.....	1-3
1.4 各分区校验情况说明.....	1-3
2 身份认证.....	2-1
2.1 开发方主体及责任.....	2-1
2.2 开发过程.....	2-3
2.3 终端身份认证功能.....	2-3
2.3.1 终端身份认证过程.....	2-3
2.3.2 挑战字原文与挑战字的生成及验证.....	2-4
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.....	2-5
2.4 业务安全数据存储说明.....	2-5
2.5 工厂生产 APK 开发.....	2-6
2.6 身份认证 API 说明.....	2-12
2.7 维护说明.....	2-17



## 插图目录

图 1-1 系统签名校验启动流程.....	1-1
图 1-2 Kernel 等非文件系统分区镜像结构图.....	1-3
图 2-1 身份认证各方关系图.....	2-1
图 2-2 终端身份认证过程.....	2-4
图 2-3 挑战字原文数据 .....	2-4
图 2-4 终端 SN .....	2-5
图 2-5 终端 IN .....	2-5
图 2-6 SHA256 算法的输入具体定义.....	2-5
图 2-7 工厂生产线示意图.....	2-7
图 2-8 工厂生产流程示意图.....	2-8



## 表格目录

表 1-1 各分区校验情况说明.....	1-3
表 2-1 业务安全数据存储说明.....	2-6



# 1 安全启动

## 1.1 概述

安全启动（含安全升级）是上海电信智能机顶盒安全方案的关键环节，用于保护机顶盒系统安全，防止机顶盒被刷机或业务相关应用被篡改。

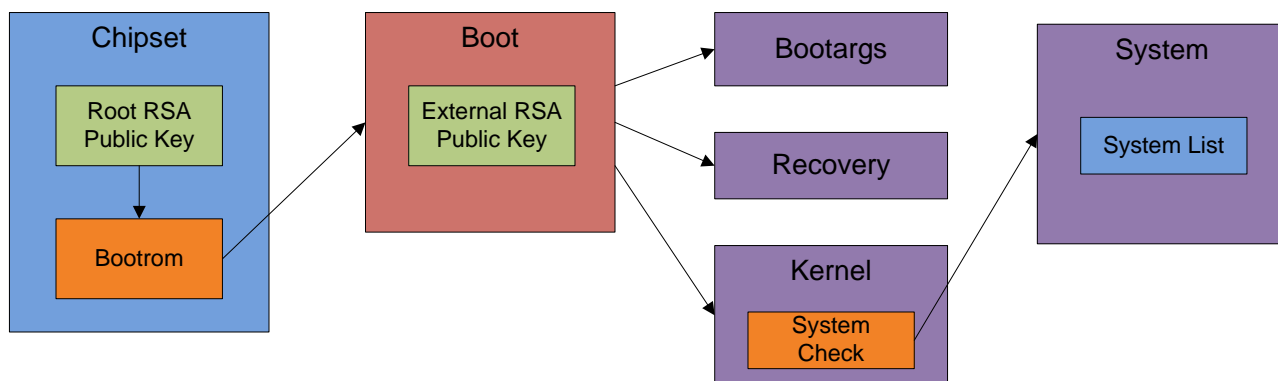
海思安全启动和校验方案采用“海思智能机顶盒 2 级安全方案”，基于海思安全芯片提供的硬件保护机制，对机顶盒的 boot, bootargs, kernel, recovery 镜像以及 Android 系统（system）提供可靠的安全保护。

具体开发指导细节，请参考《海思智能机顶盒 2 级安全方案 使用指南》。

## 1.2 系统签名校验启动流程

安全启动方案采用海思 Android 2 级安全方案，整个系统启动的安全校验流程如下图所示：

图1-1 系统签名校验启动流程



步骤 1 芯片上电后，片内 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，再使用 External RSA Public Key 对 fastboot 进行签名校验；如果校验通过则执行下一步操作，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 External RSA Public Key 校验 Bootargs, Kernel, Recovery 分区。
- 步骤 4 Kernel 校验完成后，Kernel 挂载 System 分区，Kernel 中的一个后台进程首先校验 System 分区中的 System.list 文件，然后根据该文件描述的信息，按文件方式逐个校验所有文件。
- System.list 文件内容主要存储 System 文件路径和文件对应的 HASH 值，由机顶盒厂商在编译系统镜像的时候生成，然后使用 External RSA Private Key 对 System.list 文件进行签名。
- 结束

## 1.3 系统分区签名校验

除 boot 镜像需要签名校验外，系统还需要进行签名校验的分区主要包括 bootargs、Kernel、Recovery 等分区，并对 System 分区进行按文件校验。Boot 校验通过后，在启动 Kernel 或者 Recovery 前，必须先对 Kernel 或 Recovery 做校验，只有通过签名校验系统才能启动。System 校验在后台按照文件方式进行，一旦出现校验失败，立即复位系统。

Boot 对 Kernel、Recovery 等分区和 System.list 文件进行签名校验时采用的算法为 SHA256+RSA2048，System 分区文件按照 SHA1 算法进行对比校验。校验时采用的密钥为 External RSA Public Key。

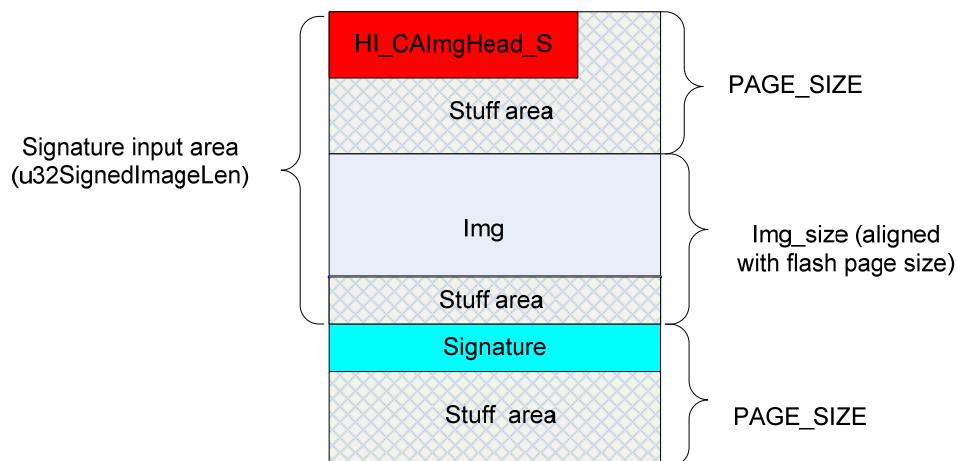
### 1.3.1 非文件系统分区的签名校验

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 SHA256+RSA2048，密钥为 External RSA Public Key。

非文件系统镜像签名如[图 1-2](#)所示，镜像前面增加签名头部信息，主要包括镜像长度等信息，签名数据放在镜像尾部。



图1-2 Kernel 等非文件系统分区镜像结构图



## 1.3.2 文件系统分区镜像的签名

由于文件系统镜像体积较大（system 镜像在 Android 上可能会占用 200M 以上的空间），整分区校验耗时较长，对启动速度影响较大。目前采用文件校验的方式，优点是启动时在后台校验，对启动速度几乎没有影响，并且能够支持 System 分区的差分升级。

制作文件系统时生成 system 分区文件列表 system.list，并计算每个文件的 HASH 值，然后使用 External RSA Private Key 对 system.list 进行签名。

Kernel 启动后，内核态后台校验程序先对 system.list 本身进行校验，校验成功后再对 system.list 里的每个文件依次校验。

## 1.4 各分区校验情况说明

表1-1 各分区校验情况说明

分区名		校验方式	校验密钥		
			名称	存放位置	Owner
fastboot	Key 区	全校验	Root RSA Public Key	存放在 OTP 中，CPU 不可读	机顶盒厂商
	参数区和 boot 区	全校验	External RSA Public Key	Flash 签名 Boot 镜像的 key 区	机顶盒厂商
bootargs		全校验	External RSA Public Key	Flash 签名 Boot 镜像的 key 区	机顶盒厂商
recovery		全校验	External RSA Public Key	Flash 签名 Boot 镜像的 key 区	机顶盒厂商
deviceinfo		不校验	-	-	-
baseparam		不校验	-	-	-



分区名	校验方式	校验密钥		
logo	不校验	-	-	-
fastplay	不校验	-	-	-
cache	不校验	-	-	-
misc	不校验	-	-	-
kernel	全校验	External RSA Public Key	Flash 签名 Boot 镜像的 key 区	机顶盒厂商
system	按文件校验	External RSA Public Key	Flash 签名 Boot 镜像的 key 区	机顶盒厂商
userdata	不校验	-	-	-
sdcard	不校验	-	-	-

说明：

- 两对校验用非对称密钥 Root RSA Key 和 External RSA Key 均为机顶盒厂商使用海思 Windows 签名工具 CASignTool 生成。
- 所有分区的签名，均为机顶盒厂商编译 SDK 时自动完成。



## 2 身份认证

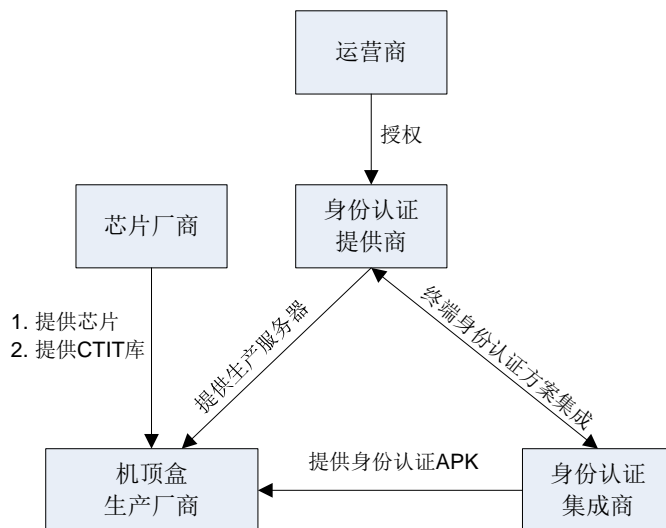
上海电信身份认证规范用于智能机顶盒身份合法性确认，整个方案主要包括工厂生产，身份认证，业务安全数据存储三大部分内容。

- 工厂生产部分负责云端业务安全数据的部署，终端业务安全数据申请及烧写。
- 身份认证部分负责机顶盒运行时的身份合法性确认。
- 业务安全数据存储负责业务安全数据的解析，安全存储和使用。

### 2.1 开发方主体及责任

身份认证各方关系如图 2-1 所示。

图2-1 身份认证各方关系图



整个身份认证方案涉及的开发方及责任主体包括：

- 运营商（上海电信）
  - 负责制定安全技术标准



- 评估芯片厂商的方案是否满足运营商（电信等）的要求
- 验证机顶盒厂商的盒子。
- 身份认证提供商（阿网）
  - 负责开发运营商（电信等）前端的身份认证平台
  - 负责电信关键数据的生成
  - 负责机顶盒生产时 PC 端工具、云端工具的开发
- 身份认证集成商（科升）

负责身份认证 APK 的集成开发，将开发好的身份认证 APK 提供给机顶盒厂商。
- 芯片制造商（海思等）
  - 根据运营商（电信等）的标准，提供满足运营商（电信等）要求的安全启动和身份认证业务安全数据处理方案。
  - 指导机顶盒厂商基于海思方案的开发。

海思的交付件如下：

- 支持上海电信安全启动特性的 Android SDK 软件包及对应工具。
- 身份认证库 libhi\_ctit\_\*.a。该库的主要作用是：
  - 提供工厂生产时运营商（上海电信）业务安全数据的烧写功能。
  - 提供终端身份认证时的解密、校验等功能。
- 身份认证库 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：对生产 APK 传进来的业务安全数据进行解析和校验，并通过串口打印出解析出来的数据和校验结果，但是不会将数据烧写进机顶盒中。仅供调试生产 APK 使用。
  - libhi\_ctit\_release.a：

功能一：对生产 APK 传进来的业务安全数据进行解析和校验，校验通过则烧写到芯片 OTP 中。

功能二：对身份认证 APK 传进来的身份认证数据进行解密和 HASH 计算，HASH 结果通过身份认证 APK 回传给云端服务器进行机顶盒身份确认。



### 警告

在 debug 数据确认无误的情况下才可以使用 release 库烧写业务安全数据。

- 机顶盒厂商
  - 负责集成海思的安全启动方案。
  - 负责集成测试海思 CTIT 库和科升身份认证 APK。
  - 负责开发工厂生产的烧写 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 端工具向云端申请业务安全数据；



- 接收到 PC 端工具传过来的业务安全数据后，调用海思 libhi\_ctit\_release.a 库烧写接口解析并烧写业务安全数据到芯片 OTP 中。

## 2.2 开发过程

机顶盒厂商使用海思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，支持该方案的芯片类型请参考本文“前言”中的“产品版本”说明。
- 步骤 2 向海思申请支持上海电信安全方案的软件包。
- 步骤 3 参考《海思智能机顶盒 2 级安全方案 使用指南》生成安全启动相关密钥，编译生成各分区镜像的签名；然后根据该文档的指导，将各分区镜像烧写到机顶盒 Flash，并把安全启动相关数据到芯片 OTP 中。根据阿网的指导和帮助，搭建工厂生产环境，开发业务安全数据烧写 APK，与运营商云端服务器成功建立通讯。
- 步骤 4 提供开发调试机顶盒对应的 SN 和 MAC 地址给阿网，阿网在云端服务器生成开发调试用的业务安全数据。
- 步骤 5 生产 APK 从云端服务器获取到业务安全数据后，调用海思 debug 库 libhi\_ctit\_debug.a 的烧写接口，与阿网确认数据解析正确。
- 步骤 6 若数据解析正确，生产 APK 再调用海思 release 库 libhi\_ctit\_release.a 的烧写接口，将业务安全数据烧写到芯片 OTP 中。
- 步骤 7 断电重启，这个时候机顶盒应该可以正常启动。
- 步骤 8 集成科升提供的身份认证 APK，测试确认机顶盒的身份认证是否能够通过。
- 步骤 9 工厂生产 APK 的具体开发指导，请参考 [2.5 工厂生产 APK 开发](#)。

----结束

## 2.3 终端身份认证功能

本章节内容描述身份认证功能，请参考上海电信规范《上海电信智能终端安全技术要求.pdf》中的第 4.5 章节“终端身份认证功能要求”，若有不符，以上海电信规范为准。

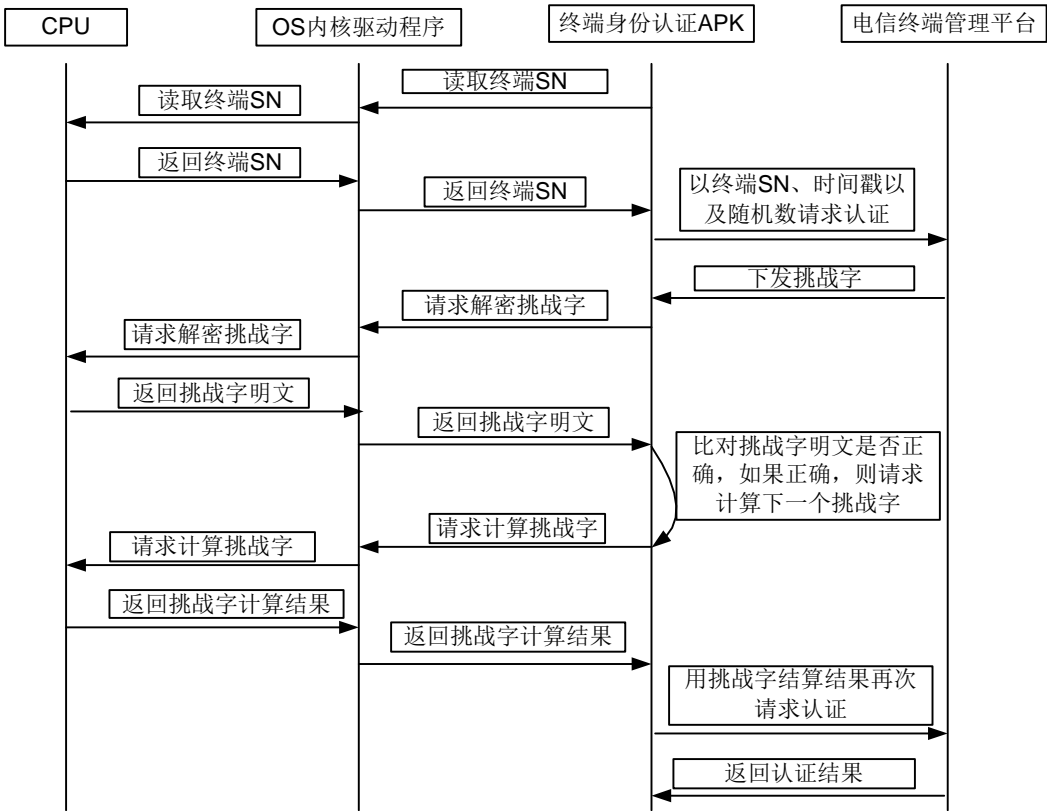
根据电信的要求，终端安全启动挂载完文件系统后，终端机顶盒上的身份认证 APK 必须通过网络跟电信前端的终端认证平台进行认证，只有通过身份认证的终端才能接入电信的网络，身份认证失败的情况下终端禁止执行任何应用程序，并且通过菜单提示用户身份认证失败。

### 2.3.1 终端身份认证过程

终端身份认证过程如[图 2-2](#)所示。



图2-2 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

挑战字由终端管理平台对下列挑战字原文数据用 RSA1024 私钥加密产生，如图 2-3 所示。

图2-3 挑战字原文数据

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 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，由如[图 2-4](#) 所示几个部分组成。

图2-4 终端 SN

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 是一串普通域不可读的安全存储的序列号，长度 24 字节，具有唯一性，由如[图 2-5](#) 所示几部分组成：

图2-5 终端 IN

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域计算，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 CPU，CPU 安全域在收到挑战字计算请求后，验证运营商标识是否正确，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[图 2-6](#) 所示。

图2-6 SHA256 算法的输入具体定义

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
---------	------------	---------	------------	------------

## 2.4 业务安全数据存储说明

终端身份认证过程中需要使用到的业务安全数据的存储情况说明见[表 2-1](#)。





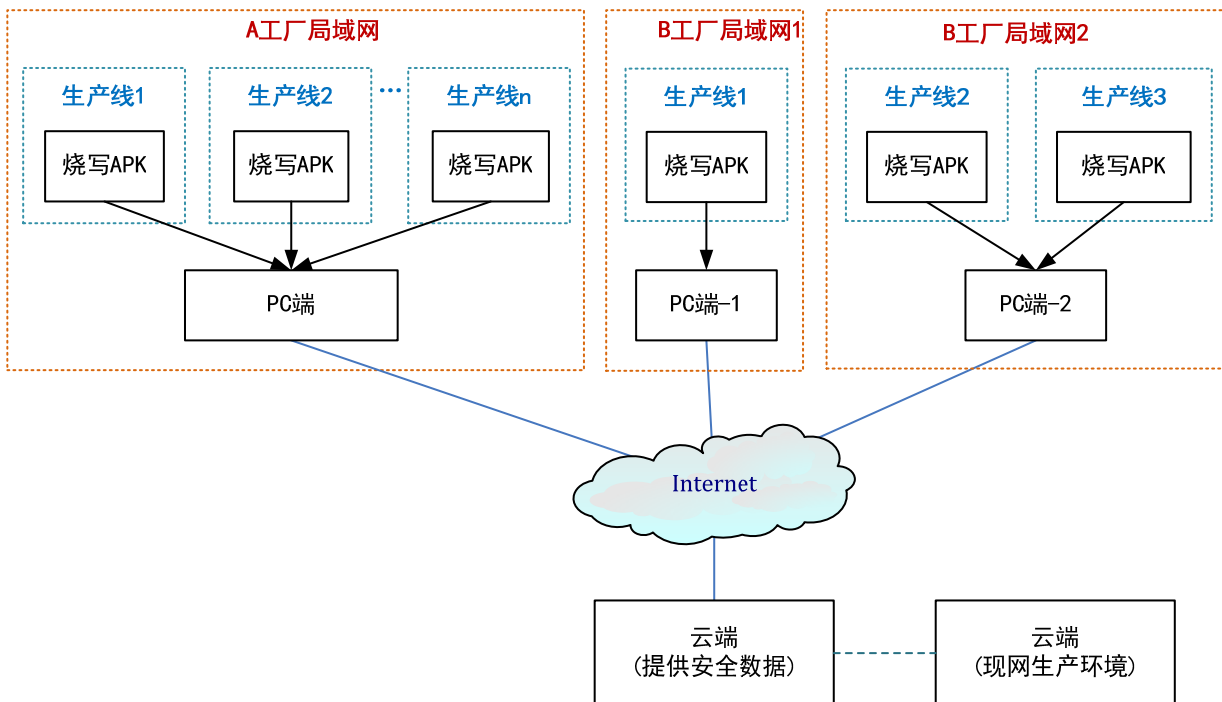
表2-1 业务安全数据存储说明

序号	数据	长度 (BYTE)	位置	是否加密	备注
1	RSA1024 公钥	256+6	OTP	加密	向运营商（电信等）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	OTP	加密	跟运营商（电信等）协商定义，例如电信天翼为字符串“CTIT”。
3	终端 SN	24	OTP	加密	向运营商（电信等）申请，每个盒子对应唯一的终端 SN。
4	终端 IN	24	OTP	加密	向运营商（电信等）申请，每个盒子对应唯一的终端 IN。
	MAC	12	OTP	明文	MAC 地址由机顶盒厂商提供给运营商，生产时由云端服务器随业务安全数据下发，用于业务安全数据的校验。
5	Chip ID	4	OTP	明文	由海思在芯片出厂前烧写到 OTP 中，每颗芯片的 Chip ID 唯一。

## 2.5 工厂生产 APK 开发

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业务安全数据，然后将业务安全数据烧写进机顶盒。工厂生产线的示意图如图 2-7 所示。

图2-7 工厂生产线示意图



烧写 APK 的开发涉及的主体及责任如下：

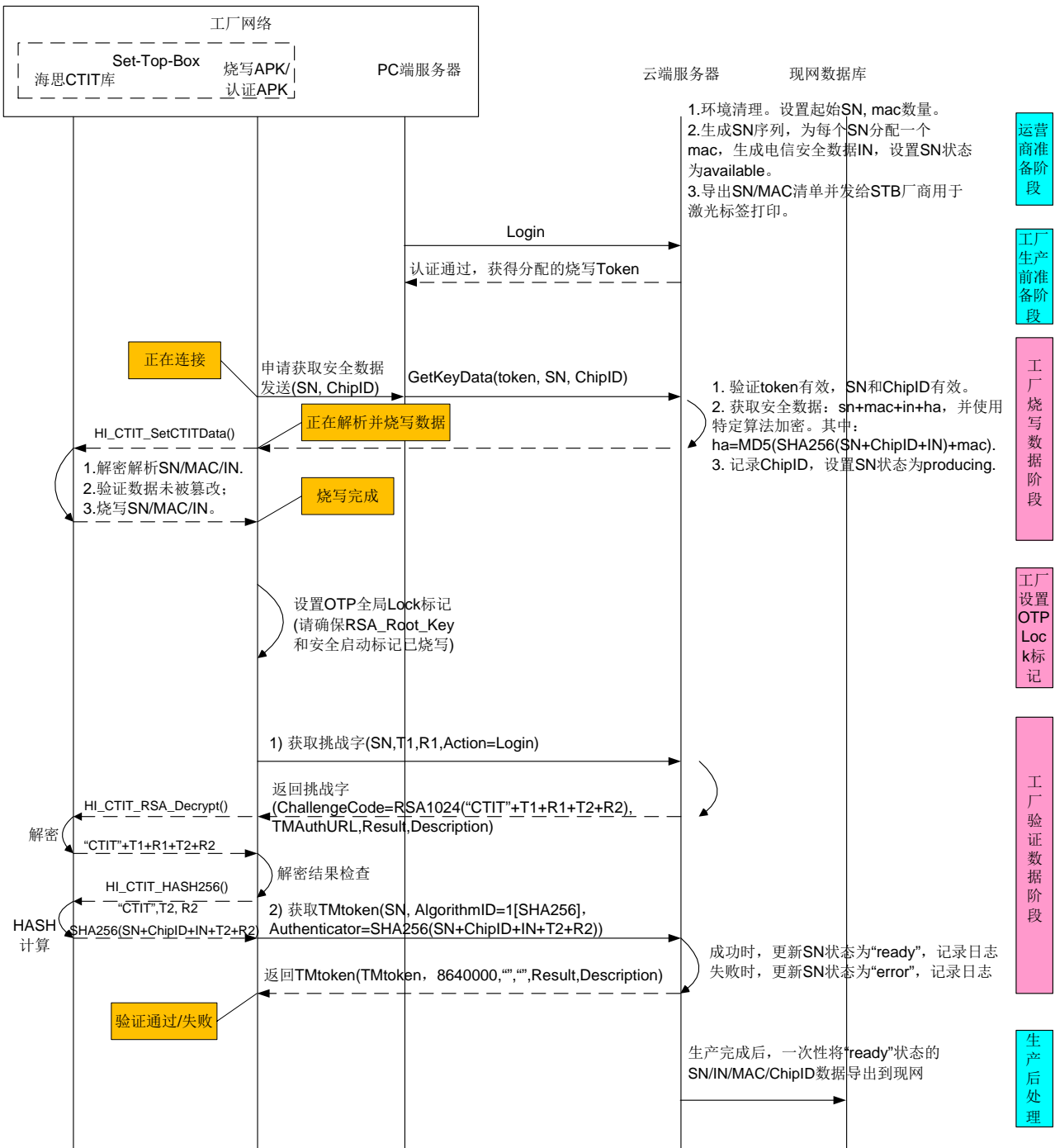
- 芯片厂商（海思）：
  - 提供身份认证库 libhi\_ctit\_xxxx.a 库：提供 API 接口供机顶盒厂商把传过来的安全数据进行解密校验后烧写到芯片中。身份认证库分为 debug 和 release 模式，对应 libhi\_ctit\_debug.a 和 libhi\_ctit\_release.a。
    - libhi\_ctit\_debug.a：对烧写 APK 传进来的安全数据进行解析和校验，并通过串口打印出解析出来的数据，但是不会将数据烧写进机顶盒中。供调试使用。
    - libhi\_ctit\_release.a：对烧写 APK 传进来的安全数据进行解析和校验，校验 OK 则烧写进机顶盒中。
- 机顶盒厂商：
  - 烧写 APK：机顶盒厂商负责集成产线烧写 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 端工具向云端申请业务安全数据；
    - 接收到 PC 端工具传过来的业务安全数据后，调用海思身份认证库接口解析和校验业务安全数据并烧写到芯片中；
  - 业务安全数据烧写完成后，使用科升提供的身份认证 APK 验证身份认证是否通过。
- 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解决方案。
- 科升：  
负责提供身份认证 APK 给机顶盒厂商，通过与云端服务器的身份认证交互，用于确认机顶盒业务安全数据烧写正确。

工厂的生产流程如图 2-8 所示（更加详细的流程请咨询阿网）：

图2-8 工厂生产流程示意图





工厂生产步骤:

- 步骤 1 机顶盒厂商提供生成 SN 的基本信息给运营商, 运营商 (电信等) 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 运营商 (电信等) 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, 申请时可以向运营商 (电信等) 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 向运营商 (电信等) 申请 RSA 1024 公钥, 业务安全数据从云端下发下来是加密的。需要使用运营商 (电信等) 提供的 RSA 1024 公钥解密。

运营商 (电信等) 提供的 RSA 1024 公钥例子如下:

E=014D0B

N=B7DFAB06E24EF1D881465AFACCDCE57296262A69C8289B35F07DECA105EE63D7DE34554

43E0837F20267FF5A681858AAFEDBD3E2AADFA8C12B4BF667590DABA2DEE2C56936E54AD8

3E2DB3DCCC891498D4D8FF5CC79E5B093D4E9EE6F9EA2922D4ED09A7EBBB2CD62125B9B03

1110E727C9A2D0B1D977AA1096A301276CFB481



#### 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 工厂使用扫描枪扫描 SN (事先打印的条码), 烧写 APK 使用 SN 和芯片的 Chip ID 为参数向 PC 端请求指定 SN 的业务安全数据。海思提供的库 libhi\_common.so 中有获取芯片 Chip ID 接口, 参考代码如下:

```
HI_S32 Sample_CTIT_GetChipID(HI_U32 *pu32ChipID)
{
    HI_S32      Ret = HI_SUCCESS;
    HI_SYS_CHIP_ATTR_S stChipAttr = {0};

    if (pu32ChipID == HI_NULL)
    {
        printf("pu32ChipID is null pointer\n");
        return HI_FAILURE;
    }

    Ret = HI_SYS_Init();

    //获取 64 位 chip id
```



```

Ret |= HI_SYS_GetChipAttr(&stChipAttr);
if (HI_SUCCESS != Ret)
{
    printf("HI_SYS_GetChipAttr err, Ret=%#x\n", Ret);
    (HI_VOID)HI_SYS_DeInit();
    return Ret;
}

// 转换成 32 位
*pu32ChipID = HI_SYS_CRC32(&(stChipAttr.u64ChipID),
sizeof(stChipAttr.u64ChipID));

(HI_VOID)HI_SYS_DeInit();
return HI_SUCCESS;
}

```



### 注意

上海电信规范定义的 ChipID 是 32bit 的，而海思芯片提供的 ChipID 为 64bit，不能直接截取，需要调用 HI\_SYS\_CRC32() 函数进行 64bit 到 32bit 的转换以确保唯一性。

获取到的 Chip ID 是 32bit unsigned int 类型的，生产 APK 在需要转化为大端格式全大写字母的字符串发给云端服务器。例如获取到的 Chip ID 为 0x12ab35cd，传给云端服务器的字符串应该为“12AB35CD”。

如果以 16 进制格式的字节数组存储，存储方式应当如下：

```

HI_U8 u8ChipIdBuf[4];

u8ChipIdBuf[0] = 0x12;
u8ChipIdBuf[1] = 0xab;
u8ChipIdBuf[2] = 0x35;
u8ChipIdBuf[3] = 0xcd;

```

- 步骤 6** 烧写 APK 通过 PC 从云端获取到业务安全数据后，调用海思 libhi\_ctit\_release 库中的函数 HI\_CTIT\_SetCTITData 解析及校验业务安全数据，若校验通过这个函数则将业务安全数据烧写到机顶盒中。从云端获取的业务安全数据的长度为 327 字节，烧写业务安全数据的参考代码如下：

```

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[] =
"B7DFAB06E24EF1D881465AFACCDCE57296262A69C8289B35F07DECA105EE63D7DE345544
3E0837F20267FF5A681858AAFEDBD3E2AADFA8C12B4BF667590DABA2DEE2C56936E54AD83
E2DB3DCCC891498D4D8FF5CC79E5B093D4E9EE6F9EA2922D4ED09A7EBBB2CD62125B9B031

```



```
110E727C9A2D0B1D977AA1096A301276CFB481"; //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_CTIT_SetCTITData(s8Data, Rsa_E, Rsa_N, 327);  
if(Ret != 0)  
{  
    Return HI_FAILURE;  
}
```

步骤7 调用接口 HI\_S32 HI\_UNF\_ADVCA\_SetOtpFuse()设置 OTP 全局 Lock 标记。参考代码如下：

```
HI_S32 sample_set_otp_global_lock_flag(HI_VOID)  
{  
    HI_S32 Ret;  
    HI_UNF_ADVCA_OTP_FUSE_E enOtpFuse;  
    HI_UNF_ADVCA_OTP_ATTR_S stOtpAttr;  
  
    Ret = HI_UNF_ADVCA_Init();  
    if (HI_SUCCESS != Ret)  
    {  
        printf("HI_UNF_ADVCA_Init failed\n");  
        return Ret;  
    }  
  
    enOtpFuse = HI_UNF_ADVCA_OTP_GLOBAL_LOCK_ACTIVATION;  
    Ret = HI_UNF_ADVCA_SetOtpFuse(enOtpFuse, &stOtpAttr);  
    if (HI_SUCCESS != Ret)  
    {  
        printf("set OTP global lock flag failed, ret=0x%x\n", Ret);  
        HI_UNF_ADVCA_DeInit();  
        return Ret;  
    }  
  
    (HI_VOID)HI_UNF_ADVCA_DeInit();  
    return HI_SUCCESS;  
}
```

判断 OTP 全局 Lock 标记是否烧写的方法：

- 方法一：  
通过 HI\_UNF\_ADVCA\_GetOtpFuse()接口获取，enOtpFuse 设置为 HI\_UNF\_ADVCA\_OTP\_GLOBAL\_LOCK\_ACTIVATION。
- 方法二：



查看 OTP 的 0x3[7] (第 3 字节第 7bit)是否设置为 1。

```
root@Hi3798MV100:/ # cat /proc/msp/otp
-----Hisilicon OTP Info-----
OTP read all over:
0000: 20000000 00800001 00000000 00a00001
0010: 00480400 00000000 00000800 00000011
```

图 2-8 中 OTP 数据按 32bit 整型数据显示，红色字体数据对应的地址为 0x3。



### 注意

OTP 全局 Lock 标记一旦设置，OTP 将无法再写入任何数据。请确保其它所有烧写 OTP 的操作（如烧写安全启动相关的 Root\_RSA\_Public\_Key，安全启动标记，HDMI 输出保护对应的 HDCP\_Root\_Key 等）都在设置 OTP 全局 Lock 标记前完成。

步骤 8 断电重启，验证机顶盒是否能够正常启动。

步骤 9 使用科升的身份认证 APK，验证身份认证是否能够通过。

----结束

## 2.6 身份认证 API 说明

身份认证 APK 的开发由科升完成后提供给机顶盒厂商集成测试，海思负责提供身份认证 APK 需要使用的加解密 API，以下 API 供身份认证 APK 使用，海思提供的 API 在库 libhi\_ctit\_release.a 中。

具体身份认证流程和数据说明，请参考 [2.3 终端身份认证功能](#) 以及 [图 2-8 工厂生产流程示意图](#) 中身份相关。

本小结主要介绍海思提供的加解密 API 的使用。

### HI\_CTIT\_readSN

#### 【描述】

获取机顶盒 SN。

#### 【语法】

```
HI_U32 HI_CTIT_readSN(HI_U8* u8data, HI_U32 len);
```

#### 【参数】



参数名称	描述	输入/输出
u8data	保存返回的 SN 的 buffer。 返回的 SN 为 24 字节的字符串。	输出
len	保存 SN 的 buffer 长度。	输入

#### 【返回值】

返回值	描述
HI_SUCCESS	获取 SN 成功
其它	获取 SN 失败

#### 【需求】

库: libhi\_ctit\_release.so

#### 【注意】

无

#### 【举例】

```
HI_U8 u8Sn[24];  
HI_U32 Ret = HI_SUCCESS;  
  
Ret = HI_CTIT_readSN(u8Sn, 24);  
if(Ret != HI_SUCCESS)  
{  
    return Ret;  
}
```

#### 【相关主题】

无

## HI\_CTIT\_random

#### 【描述】

获取随机数。

#### 【语法】

```
HI_VOID HI_CTIT_random(HI_U32 *rand, HI_U32 len);
```

#### 【参数】





参数名称	描述	输入/输出
rand	保存返回的随机数的 buffer。 随机数为 16 进制格式。	输出
len	需要获取的随机数长度，单位为字节。 有效值为 16。	输入

**【返回值】**

返回值	描述
HI_SUCCESS	获取随机数成功
其它	获取随机数失败

**【需求】**

库：libhi\_ctit\_release.so

**【注意】**

无

**【举例】**

```
HI_U32 rand[4];  
memset(rand, 0, sizeof(rand));  
HI_CTIT_random(rand, 16);
```

**【相关主题】**

无

## HI\_CTIT\_RSA\_Decrypt

**【描述】**

解密身份认证 APK 从云端服务器获取的挑战字。

**【语法】**

```
HI_U32 HI_CTIT_RSA_Decrypt(HI_U8 *inputbuf, HI_U8 *outputbuf, HI_U32  
outputlen);
```

**【参数】**



参数名称	描述	输入/输出
inputbuf	身份认证 APK 从云端服务器获取的挑战字，16 进制格式。 需要把云端服务器下发的 256 字节的字符串转换为 128 个字节的 16 进制数据。	输入
outputbuf	解密后的挑战字原文，字符串格式。 挑战字原文格式请参考图 2-3 挑战字原文数据。	输出
outputlen	解密后的挑战字的长度，单位为字节，有效值为 64。	输入

#### 【返回值】

返回值	描述
HI_SUCCESS	解密成功
其它	解密失败

#### 【需求】

库：libhi\_ctit\_release.so

#### 【注意】

云端服务器下发的挑战字为 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字符串 12AB45CD...，则应该转化为以下数据传给 HI\_CTIT\_RSA\_Decrypt()函数：

```
u8ChallengeWord[0] = 0x12;  
u8ChallengeWord[1] = 0xAB;  
u8ChallengeWord[2] = 0x45;  
u8ChallengeWord[3] = 0xCD;
```

#### 【举例】

```
HI_U32 Ret = HI_SUCCESS;  
HI_U8 u8ChallengeWord[128]; //实际数据为云端服务器下发下来的16进制挑战字  
HI_U8 u8OutputBuf[64];  
Ret = HI_CTIT_RSA_Decrypt(u8ChallengeWord, u8OutputBuf, 64);  
if(Ret != HI_SUCCESS)  
{  
    return Ret;  
}
```

#### 【相关主题】



无

## HI\_CTIT\_HASH256

## 【描述】

计算挑战字的返回结果。

## 【语法】

```
HI_U32 HI_CTIT_HASH256(HI_U8 *vendor, HI_U8 time[14], HI_U8 random[16],  
HI_U8 *outbuf, HI_U32 out_length);
```

## 【参数】

参数名称	描述	输入/输出
vendor	从解密出的挑战字里获取的运营商标识。 4 字节字符串格式。	输入
time	从解密出的挑战字里获取的 14 字节的平台时间戳 T2。 字符串格式。	输入
random	从解密出的挑战字里获取的 16 字节的平台随机数 R2。 字符串格式。	输入
outbuf	挑战字返回结果，为业务安全数据及 T2, R2 的散列值 SHA256(SN+ChipID+IN+T2+R2)。 16 进制格式。	输出
Out_length	挑战字返回结果（散列值）长度，单位为字节，有效值 为 32 字节。	输出

## 【返回值】

返回值	描述
HI_SUCCESS	计算挑战字返回结果成功
其它	计算挑战字返回结果失败

## 【需求】

库：libhi\_ctit\_release.so

## 【注意】



HI\_CTIT\_HASH256 返回的挑战字计算结果（散列值）为 32 字节的 16 进制数，身份认证 APK 需要转化为 64 字节的字符串（必须为全大写字母）发送给云端服务器。例如返回的数据为：

```
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：

```
u8OutputStr = "12ABEF34....."。
```

#### 【举例】

```
HI_U32 Ret = HI_SUCCESS;  
HI_U8 vendor[]; // HI_CTIT_RSA_Decrypt解密出来的运营商标识。  
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];  
..... //解密挑战字，检查及处理解密结果  
Ret = HI_CTIT_HASH256(vendor, u8Time, u8Random, u8OutBuf, 32);  
if(Ret != SUCCESS)  
{  
    return Ret;  
}
```

#### 【相关主题】

无

## 2.7 维护说明

因所有业务安全有关数据全部烧写在 OTP 中，因此机顶盒维修时，若 Flash 损坏，可直接更换 Flash，然后烧写 Flash 各分区镜像，不影响身份认证功能的正常执行。