智能电视操作系统　第2部分：安全

1　范围

本文件规定了智能电视操作系统的安全体系、安全机制等相关技术要求。
本文件适用于智能电视操作系统的研发、生产和应用。

2　规范性引用文件

本文件没有规范性引用文件。

3　术语和定义

本文件没有需要界定的术语和定义。

4　缩略语

下列缩略语适用于本文件。
API　应用程序编程接口（Application Programming Interface）
APP　应用程序（Application）
ATR　应答复位（Answer To Reset）
CA　证书认证机构（Certification Authority）
DCAS　可下载条件接收系统（Downloadable Conditional Access System）
DoS　拒绝服务（Denial of Service）
DRM　数字版权管理（Digital Rights Management）
DVB　数字视频广播（Digital Video Broadcasting）
ECW　加密的控制字（Encrypted Control Words）
EK　加密的密钥（Encrypted Key）
HAL　硬件抽象层（Hardware Abstract Layer）
IPSec　互联网安全协议（Internet Protocol Security）
OS　操作系统（Operating System）
OTP　一次性写入（One Time Programming）
REE　富执行环境（Rich Execution Environment）
ROM　只读存储器（Read-Only Memory）
TVOS　智能电视操作系统（Smart Television Operating System）
SELinux　Linux强制访问控制安全系统（Security-Enhanced Linux）
TAPP　可信应用（Trusted Application）
TEE　可信执行环境（Trusted Execution Environment）
VPN　虚拟专用网络（Virtual Private Network）

5 通则

TVOS安全是由TVOS REE、TVOS TEE和硬件三种安全环境组成的安全架构。

TVOS REE是TVOS的第一道安全防线，承载和运行各种应用程序，防范应用程序的安全风险，如运行用户交互界面应用程序，防范用户交互界面应用程序的安全风险等。TVOS REE应依靠TVOS TEE和基于硬件安全根的安全计算和处理，以保证TVOS REE具备足够的安全性。硬件安全根由基于安全芯片的安全硬件环境通过TEE来提供。TVOS TEE是中间安全防线，构建在安全操作系统之上，通过TVOS REE和 TVOS TEE之间的安全通信接口，实现与TVOS REE侧安全计算和基本安全能力处理相关的所有核心安全计算和处理。安全硬件环境是最后一道安全防线，是TVOS的安全根，具有三条安全防线中最强的安全级别。所有关键密钥和密码算法引擎都嵌入在硬件环境中。TVOS REE和TVOS TEE根据业务安全需求基于安全硬件环境提供安全资源。

6 安全架构

6.1 基础安全架构

6.1.1 通则

TVOS基础安全架构描述了基本安全能力形成TVOS REE安全、TVOS TEE安全和硬件安全的方式，见图1。

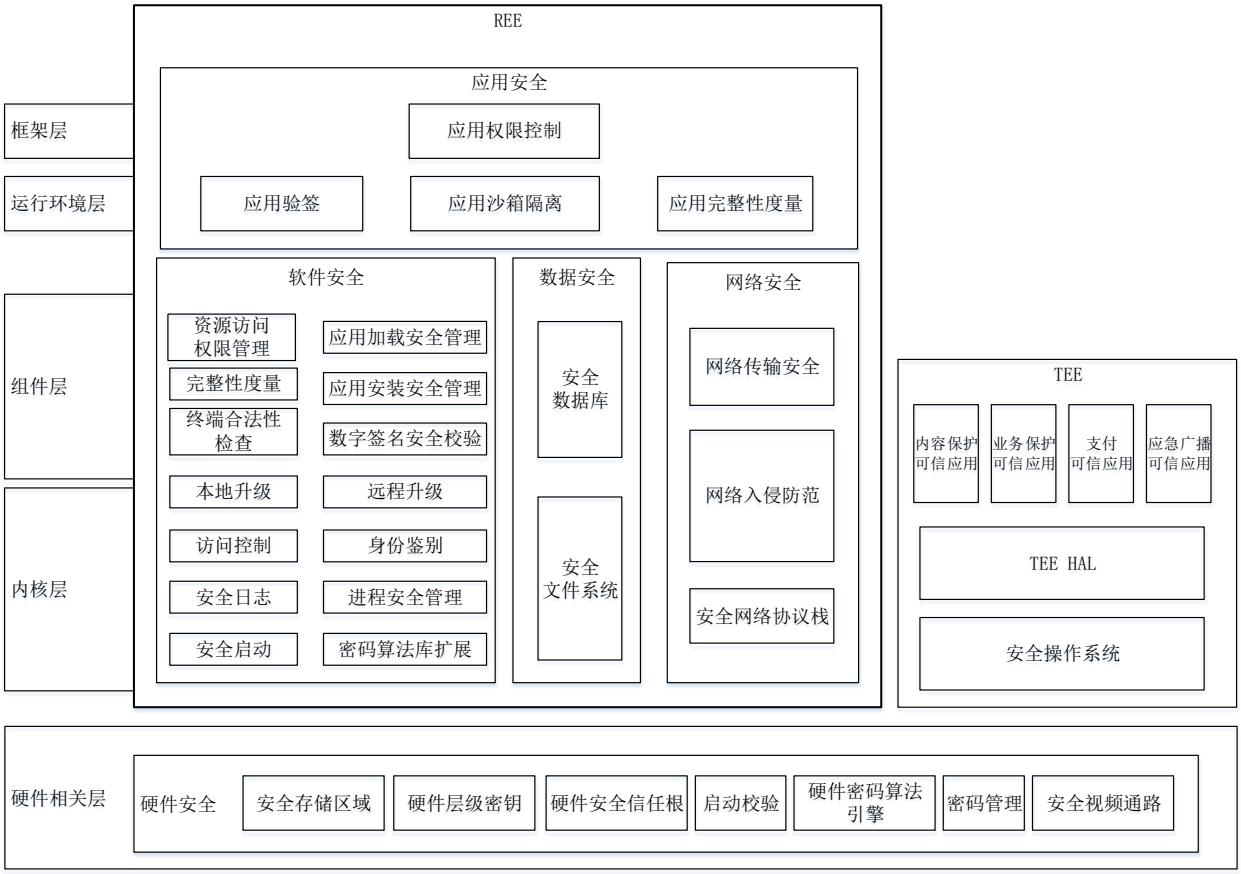

图 1　基础安全架构

TVOS基础安全架构应基于TVOS软件架构和安全机制，与TVOS硬件安全、软件安全、网络安全、数据安全和应用安全能力集成协同，形成纵深防御的TVOS安全功能。

TVOS硬件安全能力为软件安全、网络安全、数据安全、应用安全等基础安全能力构建提供支撑。软件安全、网络安全、数据安全等基础安全能力应基于硬件基础安全能力构建，为应用基础安全能力构建提供支撑。应用基础安全能力应基于软件安全、网络安全、数据安全等基础安全能力构建。

6.1.2　TVOS REE 安全

TVOS REE安全应在硬件安全的基础上提供软件安全、网络安全、数据安全和应用安全能力。TVOS REE通过安全机制与TVOS TEE通信，完成业务保护和内容保护等。

6.1.3　TVOS TEE 安全

TVOS TEE是一个可信和安全的计算环境，包括可信和安全的硬件和软件。

TVOS可信安全硬件支持安全芯片、安全内存访问控制、安全总线连接、安全中断、安全时钟、安全随机数、安全加解密引擎、安全视频路径。

TVOS可信安全软件包括安全操作系统和TEE HAL。安全操作系统提供内存管理、安全时间、任务调度、中断、任务通信、加解密功能，支持内存隔离、版本防回退、安全存储、TAPP动态加载。TEE HAL是一个硬件抽象层API，用于支持各种可信应用程序，如DRM TAPP和DCAS TAPP，TEE硬件抽象层接口具体定义见第10章。

6.1.4　硬件安全

TVOS应提供硬件安全能力，以实现基于安全芯片的TVOS软件安全、网络安全、数据安全和应用安全。这些硬件安全功能应包括安全存储区、安全信任根和硬件密码算法引擎。

6.2　沙箱隔离安全架构

应基于TVOS软件架构和基础安全架构对每个TVOS应用进行沙箱隔离，为每个TVOS应用构建一个相应的应用沙箱。每个TVOS应用沙箱应由一个相应的TVOS应用进程构建，该应用进程由相应的TVOS应用及其所调用的TVOS应用框架层相关功能接口单元实例、相应的应用执行环境和所调用的相关功能组件客户端实例界定。TVOS应用沙箱可根据需要采用TVOS基础安全能力和安全机制进行安全加固。

应基于TVOS软件架构和基础安全架构对每个TVOS功能组件进行沙箱隔离，为每个TVOS功能组件构建一个相应的组件沙箱。每个TVOS组件沙箱应由一个相应的TVOS组件进程构建，该组件进程由相应的TVOS功能组件服务端及其所调用的其他功能组件客户端实例和硬件抽象层相关功能接口单元实例界定。TVOS组件沙箱可根据需要采用TVOS基础安全能力和安全机制进行安全加固。

TVOS应用和TVOS功能组件模块的沙箱隔离安全架构见**错误!未找到引用源。**。

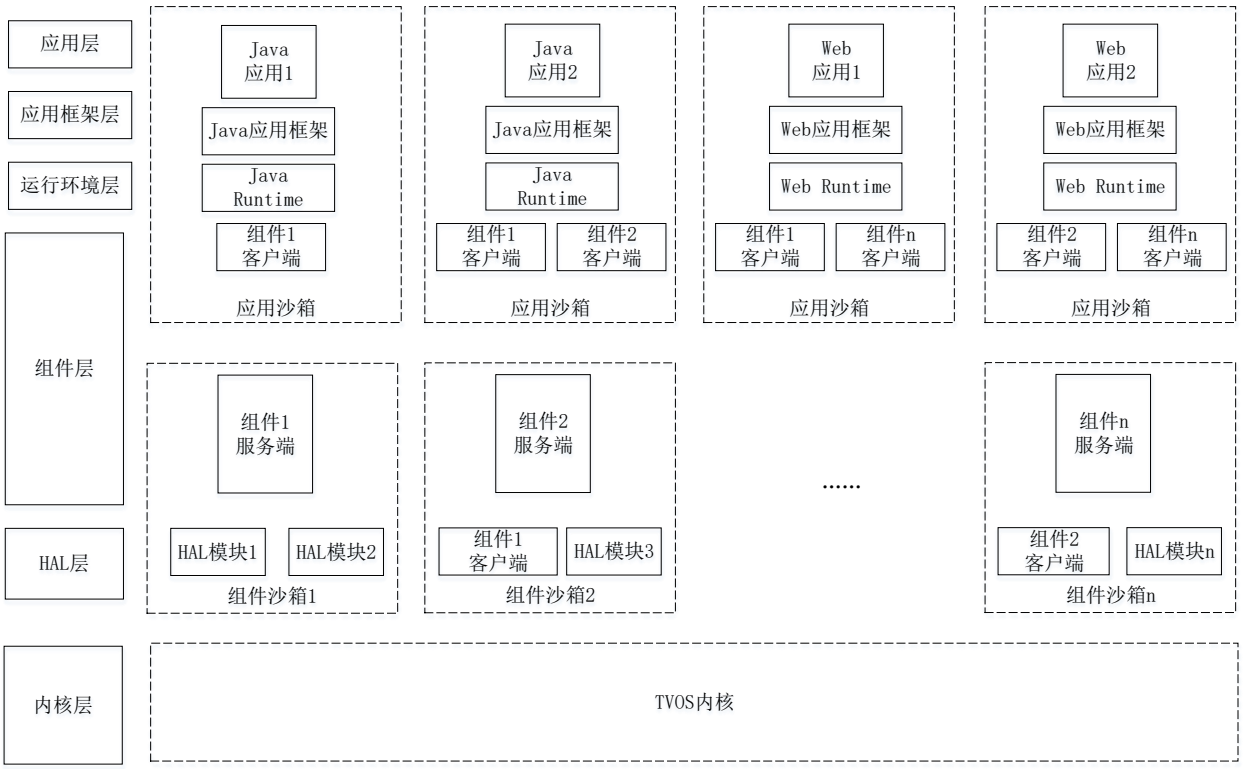

图 2　沙箱隔离安全架构

7　基础安全能力要求

7.1　基础硬件

应基于安全芯片等底层硬件为TVOS的软件安全、网络安全、数据安全和应用安全的实现提供TEE、安全存储区域、硬件层级密钥机制、硬件安全信任根、启动校验、硬件密码算法引擎、密码管理和安全视频路径等硬件安全能力。

7.2　基础软件

应能保障系统软件运行的安全，包括系统软件所控制的硬件、计算和数据等系统资源的安全。TVOS应具备如下的软件安全能力。

a)　访问控制：应具备对不同软件模块和应用访问相关数据和文件、调用相关软件模块和操作相关设备资源进行权限配置和管理的能力，使相应软件模块和应用只能按照所指定的安全访问控制权限访问相关数据、调用相关软件模块、操作相关设备资源，防止相应软件模块和应用获取安全敏感数据、执行未授权操作、调用未授权的软件模块，以及防止非授权入侵代码的执行等。应支持SELinux安全访问控制机制。

b)　资源访问权限管理：应具备对资源访问的权限进行控制和管理的能力，且能保存和查询应用资源访问权限的信息。所有的主体对资源客体的访问应基于SELinux权限控制机制经过明确的许可，访问主体对资源客体的资源访问权限应由相关安全策略所定义的安全上下文决定；所有客体与主体间均由一个唯一的安全上下文进行关联；在运行期间，特定的进程访问主体与进程间通信通道、文件和网络主机等资源客体之间的访问控制关系不可自主修改。

c) 数字签名安全校验：应具备基于数字签名技术对应用在安装或运行时进行安全校验的能力，支撑将安全信任链校验机制传递至应用软件。
d) 远程升级：应进行基于安全信任链校验机制的安全校验，只有通过安全校验的系统软件版本才允许在设备上进行远程升级。
e) 本地升级：应进行基于安全信任链校验机制的安全校验，只有通过安全校验的系统软件版本才允许在设备上进行本地升级。

7.3 基础网络

应能保障系统运行的网络通信安全，包括网络传输安全和网络入侵防范等。
应具备支持IPSec协议的VPN。
应具备防DoS攻击、IP地址黑白名单、服务端口接入控制名单、网络流量管理统计等功能。

7.4 基础数据

应能保障系统软件和应用软件运行所产生数据的安全。
应具备通过安全文件系统进行数据安全存储的能力，确保用户数据不被恶意泄漏和篡改、实现对安全敏感信息的保护等。

7.5 基础应用

应能保障合法应用的安全运行、管理应用的安装，包括拒绝非法应用安装、管理应用的系统资源访问权限和控制应用的资源访问等。
应用软件应在进行基于安全信任链校验机制的安全校验，确保应用软件的合法性和完整性后，才能在TVOS上安装运行。

8 基本安全功能要求

8.1 内容安全

TVOS内容安全应基于TVOS安全机制和硬件信任根建立DRM的安全信任链,通过使用密钥安全存储与管理机制、授权信息安全解密与存储管理机制以及安全视频路径保护措施，实现基于硬件安全的终端内容安全授权、安全解密、安全解码和安全播放与输出。
TVOS内容安全应能通过TVOS DRM组件与DRM APP和DRM TAPP及媒体引擎组件协同实现。DRM APP为JAVA形态或WEB形态。内容安全的功能实现见附录A。

8.2 业务安全

TVOS业务安全应基于TVOS安全机制和硬件信任根建立DCAS的安全信任链，通过DCAS根密钥派生、密钥安全存储管理、数据解密和安全存储、安全视频路径保护措施等，实现基于硬件安全的终端业务条件接收安全授权、安全解密解扰、安全解码和安全播放。
TVOS业务安全应能通过TVOS DCAS组件与DCAS APP和DCAS TAPP及媒体引擎组件和数字电视组件协同实现。DCAS APP为JAVA形态或WEB形态。业务安全的功能实现见附录B。

8.3 安全启动

应基于硬件安全和安全信任链校验机制对启动加载软件、操作系统和应用程序逐级进行安全校验，只有全部通过安全校验后，系统才能安全启动，设备方能进入正常工作状态。

安全启动以安全芯片为基础，TVOS系统实例与安全芯片硬件一一绑定；安全启动的信任链以安全芯片为起点，经终端Bootloader引导程序至TVOS内核和文件系统，信任链的每一环节只有通过数字签名校验后方可启动。

安全启动流程通常为：系统上电后，SoC的ROM将加载Bootloader引导程序，然后再加载TVOS TEE部分并创建安全区内存，再后加载TVOS REE部分并创建普通区内存，每一级软件的装载和引导都应通过数字证书信任机制合法性和完整性的安全校验，只有在任意一个环节的安全校验通过后，该环节方可启动，该环节的后一环节方可进入校验状态，只有全部环节都通过安全校验后，系统方可启动，任何一步的验证失败都将导致系统启动失败。安全启动的实现见附录C。

8.4 安全升级

TVOS的安全升级包括操作系统的安全升级和应用的安全升级。

操作系统的安全升级应进行基于安全信任链校验机制的安全校验，只有通过安全校验的操作系统版本才允许进行升级。操作系统的升级不准许回滚。

应用软件的安全升级应进行基于安全信任链校验机制的安全校验，只有通过安全校验的应用软件版本才允许进行升级。安全应用的升级不准许回滚。

安全升级的实现见附录D。

9 安全机制

9.1 TEE

TEE应基于TVOS可信安全硬件和可信安全软件实现，为TVOS提供可信安全计算环境。

TVOS可信安全硬件应基于安全芯片支持安全内存访问控制、安全总线连接、安全中断、安全时钟、安全随机数、安全加解密引擎等功能，应支持安全视频路径。

TVOS可信安全软件应包括安全OS和TEE HAL。安全OS应具备内存管理、安全时间、任务调度、中断、任务通信、加解密等功能，应支持内存隔离、版本防回滚、安全存储、TAPP动态加载；TEE HAL应支持DRM TAPP、DCAS TAPP等。

9.2 硬件安全信任根

硬件安全信任根作为内置于安全芯片,不可改写安全存储区域中的安全信息,应为系统安全启动、系统软件安全升级、应用软件下载安装的安全信任链校验机制信任根。

9.3 数字证书安全信任机制

数字证书安全信任机制应支持基于X.509进行分级的安全信任校验,每级安全信任校验应采用对应层级的数字证书。

9.4 安全信任链校验机制

安全信任链校验机制应采用数字证书安全信任机制，对信任链路中各环节的软件合法性和完整性进行校验，信任链路应自底向上从安全芯片开始，包括启动加载软件和操作系统，至应用软件。

在信任链路中，各环节软件应被数字签名，且应使用预置在信任链路前一环节软件中的密钥逐级进行安全校验，其中，启动加载软件应使用硬件安全信任根进行校验。只有信任链路中各环节软件都通过安全校验，安全信任链路才建立。

安全信任链校验机制应保障信任链路中启动加载软件、操作系统和应用程序的来源合法性和完整性。

9.5 安全视频路径

TVOS安全视频路径应包括安全解密/解扰、安全解码、安全缓存、安全显示或输出保护等环节，涵盖加密视频内容从解密/解扰开始直至最终显示的保护路径各环节。

视频内容在TVOS安全视频路径的任一环节及各环节之间的传输都应在TVOS TEE可信执行环境下获得安全保护。

10 TVOS TEE 硬件抽象层接口

10.1 通则

TVOS硬件抽象层接口应支持全球平台组织编程接口，包括GP TEE Client API和GP TEE Internel API。GP TEE Client API在TVOS REE中提供了打开会话、调用命令和关闭会话API，客户端应用（CA）通过调用命令API来访问TEE侧的可信应用（TA）的安全功能；GP TEE Internel API为上层TAPP提供基础的加解密、安全时间、签名/验证等接口。安全操作系统应使用白名单等访问控制机制，保护CA与TA通信，确保CA及TA身份唯一性，确保CA授权访问TA。

TVOS硬件抽象层接口针对广播电视媒体业务需要，应扩展了TVOS自有TEE应用编程接口，包括融合CA TEE接口、DRM TEE接口等。

10.2 融合 CA TEE 接口

10.2.1 安全芯片层级密钥驱动接口

安全芯片层级密钥驱动接口包括层级密钥初始化、层级密钥反初始化、读取安全芯片标识、通过SoC层级密钥计算挑战应答结果、设置解扰参数及密钥、触发解扰、停止解扰等接口，具体接口函数应符合E.1中的规定。

10.2.2 硬件安全模块应用程序接口

硬件安全模块应用程序接口包括获取硬件安全模块软件版本号、获取硬件安全模块基本信息、获取硬件安全模块诊断信息、获取硬件安全模块存储容量及读写能力信息、获取最近一次硬件安全模块被激活的时间、获取激活信息中CA厂商专属的数据、生成激活请求数据、设置硬件安全模块消息、与硬件安全模块建立安全认证通道、读取硬件安全模块数据、写入数据到硬件安全模块、读取硬件安全模块存储的位置信息、读取硬件安全模块公共存储区、向硬件安全模块公共存储区写入数据、设置硬件安全模块中重加密算法、硬件安全模块控制字CW、关闭硬件安全模块安全认证通道等接口。具体接口函数应符合E.2中的规定。

10.2.3 加解密及签名校验接口

加解密及签名校验接口包括验证SM2数字签名、SM3散列、SM2加密、SM4加密、SM4解密等接口。具体接口函数应符合E.3中的规定。

10.2.4 智能卡 TEE 接口

智能卡TEE接口包括智能卡复位、智能卡通讯接口。具体接口函数应符合E.4中的规定。

10.3 DRM TEE 接口

DRM TEE接口包括密码算法、安全存储、安全内存、安全时间、输出控制、密钥烧写等接口。具体接口函数应符合附录F中的规定。

附 录 A
（资料性）
内容保护功能实现

TVOS 通过 DRM 应用、DRM 组件、媒体引擎组件、DRM TAPP 的协同工作实现 DRM 功能，见图 A.1。

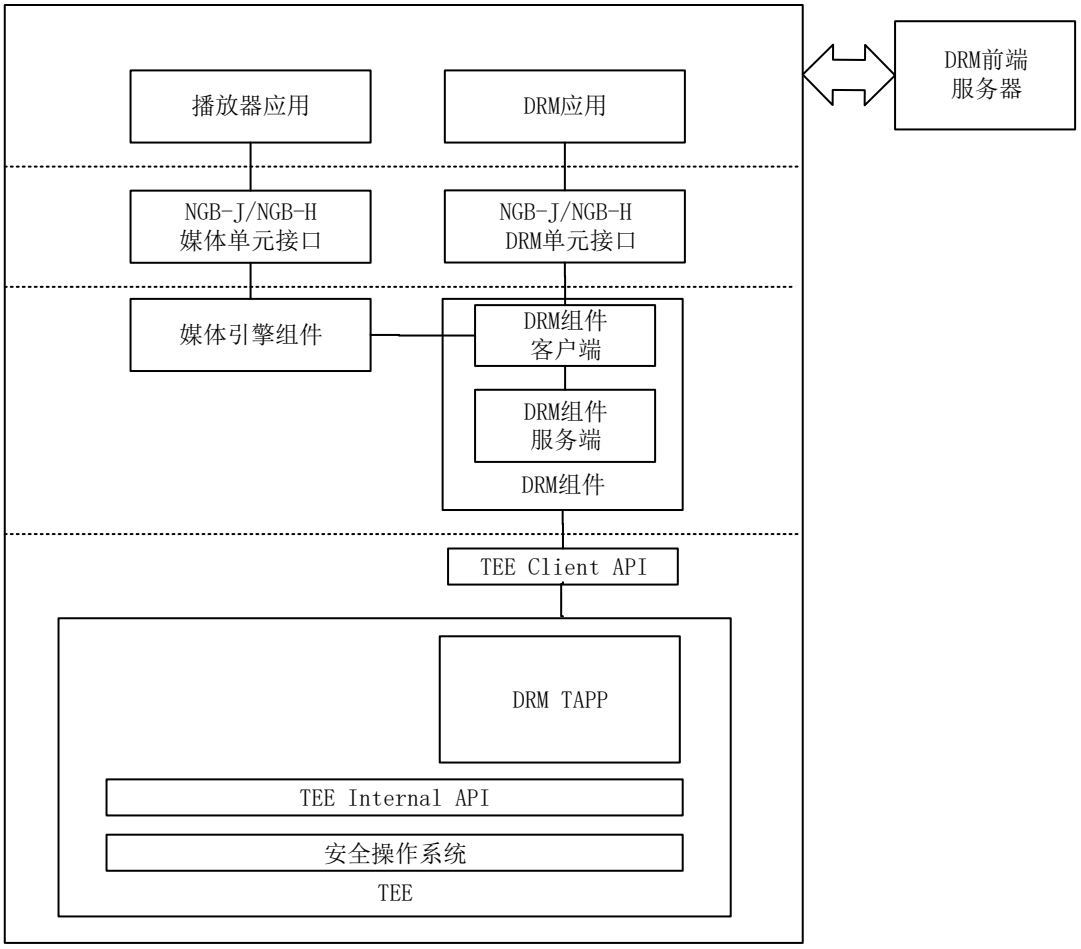

图 A.1 内容保护功能实现

DRM 功能实现方法如下：

a) 播放器应用播放 OTT 内容时，通过媒体单元接口将媒体内容的 URL 发送给媒体引擎组件，由媒体引擎组件获取播放文件列表和 DRM 信息；
b) 媒体引擎组件将 DRM 信息发送给 DRM 组件，由 DRM 组件根据 DRM 应用标识将 DRM 信息转发到 DRM 应用；
c) DRM 应用解析 DRM 信息，向 DRM 服务器获取内容授权，并将获取到的许可证发送给 DRM TAPP；
d) 媒体引擎组件将加密媒体内容及其密钥信息经 DRM 组件发送给 DRM TAPP；
e) DRM TAPP 解析许可证，根据密钥信息判断授权，从许可证中获取内容密钥，使用内容密钥解密加密媒体内容；
f) 媒体引擎组件将解密后的媒体内容发送给底层硬件音视频解码器，对解密后的媒体内容进行解码，并通过输出保护机制保护输出。

附 录 B
（资料性）
业务保护功能实现

B.1 基于JAVA的功能实现

基于 JAVA 的数字电视业务保护功能实现见图 B.1。

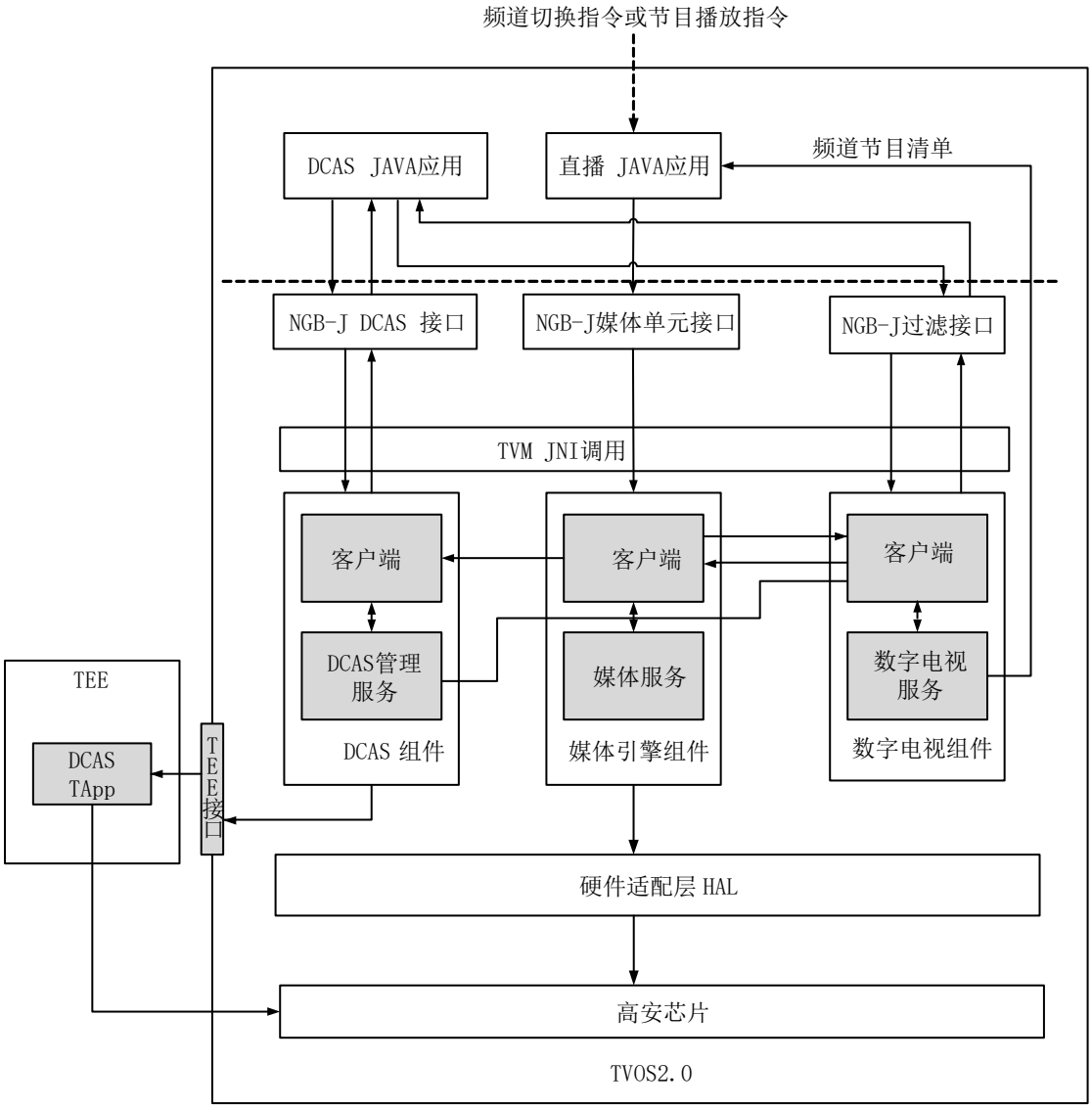

图 B.2 基于 JAVA 的业务保护功能实现

基于 JAVA 的数字电视业务保护功能实现步骤如下：

a) 用户使用直播应用观看节目，切换到某个频道；直播应用调用 NGB-J 媒体单元的相应接口，并将频道的 DVB 三要素（original network id、transport stream id、service id）传给 NGB-J 媒体单元。

b) NGB-J 媒体单元将 DVB 三要素传递给媒体播放组件。
c) 媒体播放组件根据 DVB 三要素从 DTV 组件获取 freeCAMode 标识，判断频道是否加扰，如果是加扰频道，则获取 casId、ecmPid、emmPid、streampath。
d) 如果频道是非加扰的，则媒体播放组件直接调用底层驱动正常播放；如果是加扰的，则媒体播放组件创建播放的 PipeLine，并将 audioPid、videoPid 和 casId、ecmPid、emmPid、streampath 一起传给 DCAS 组件。
e) DCAS 组件根据 casId 选择相应的 DCAS 应用,并通过 DCAS API 将 audioPid、videoPid、casId、ecmPid、emmPid、streampath 传给 DCAS 应用。
f) DCAS 应用进行相关初始化，并根据 ecmPid 和 emmPid 通过 NGB-J Section 单元调用 DTV 组件获取 ecm Data 和 emm Data，同时 DCAS 应用将 streampath、audioPid、videoPid 和 casId 传给 DCAS 组件，DCAS 组件将这些参数通过 TEE Client API 送到 Secure OS 中的 TAPP 模块。
g) DCAS 应用将获得的 ecm Data、emm Data 传给 DCAS 组件，DCAS 组件将 ecm data、emm data 通过 TEE Client API 送到 Secure OS 中的 TAPP 模块。
h) TAPP 模块在 Secure OS 中解析 ecm data、emm data 获得 EK2、EK1、ECW 并设置给高安芯片，实现加扰频道的解扰。

B.2 基于WEB的功能实现

基于 WEB 的数字电视业务保护功能实现见图 B.2。

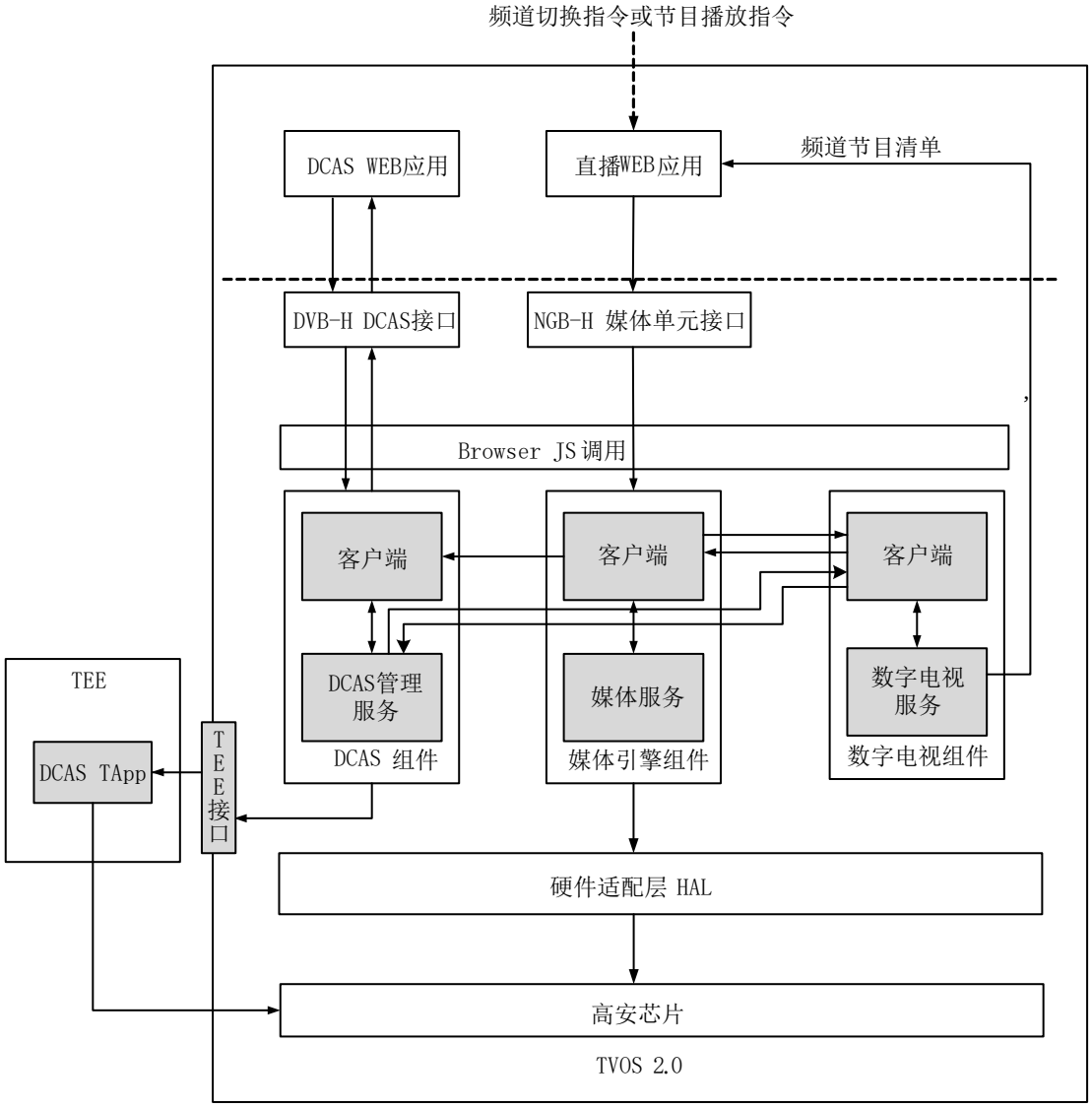

图 B.2　基于 WEB 的业务保护功能实现

基于 WEB 的数字电视业务保护功能实现步骤如下：

a) 用户使用直播应用观看节目，切换到某个频道；直播应用调用 NGB-H 媒体单元的相应接口，并将频道的 DVB 三要素（original network id、transport stream id、service id）传给 NGB-H 媒体单元。

b) NGB-H 媒体单元将 DVB 三要素传递给媒体播放组件。

c) 媒体播放组件根据 DVB 三要素从 DTV 组件获取 freeCAMode 标识，判断频道是否加扰，如果是加扰频道，则获取 casId、ecmPid、emmPid、streampath。

d) 如果频道是非加扰的，则媒体播放组件直接调用底层驱动正常播放；如果是加扰的，则媒体播放组件创建播放的 PipeLine，并将 audioPid、videoPid 和 casId、ecmPid、emmPid、streampath 一起传给 DCAS 组件。

e) DCAS 组件根据 casId 选择相应的 DCAS 应用，并通过 DCAS API 将 audioPid、videoPid、casId、ecmPid、emmPid、streampath 传给 DCAS 应用。

f) DCAS应用进行相关初始化,并根据ecmPid和emmPid通过DCAS组件调用DTV组件获取ecm Data和 emm Data，同时 DCAS 应用将 audioPid、videoPid 和 casId 传给 DCAS 组件，DCAS 组件将这些参数通过 TEE Client API 送到 Secure OS 中的 TAPP 模块。

g) DCAS 应用将 ecm Data、emm Data 传给 DCAS 组件。

h) DCAS 组件将 ecm data、emm data 通过 TEE Client API 送到 Secure OS 中的 TAPP 模块。

i) TAPP 模块在 Secure OS 中解析 ecm data、emm data 获得 EK2、EK1、ECW 并设置给高安芯片，实现加扰频道的解扰。

附 录 C
（资料性）
系统安全启动

TVOS安全启动包括安全芯片到引导程序的信任构建、引导程序到TVOS TEE的信任构建和TVOS TEE到TVOS REE的信任构建三个步骤。

a) 安全芯片到引导程序的信任构建：由 ROM 固化代码从 Flash 存储器读取 BL_KEY1 和使用 BL_KEY0 对 BL_KEY1 的签名，通过预埋在安全芯片 OTP 区域中的 BL_KEY0 验证 BL_KEY1 的合法性，验证成功后加载引导程序和使用 BL_KEY1 对引导程序的签名，通过 BL_KEY1 验证引导程序的合法性，验证成功后将系统控制权交给引导程序。

b) 引导程序到 TVOS TEE 的信任构建：由引导程序加载 Secure OS 镜像和使用 BL_KEY1 对 Secure OS 镜像的签名，通过 BL_KEY1 验证 Secure OS 镜像的合法性，验证成功后将系统控制权交给 Secure OS。

c) TVOS TEE 到 TVOS REE 的信任构建：由引导程序装载 TVOS 内核及系统镜像和使用 BL_KEY1 对 TVOS 内核及系统镜像的签名，通过 BL_KEY1 验证 TVOS 内核及系统镜像的合法性，验证成功后将系统控制权交给操作系统。

上述验证过程任何一步的验证失败则导致系统启动失败。TVOS安全启动流程见图C.1。

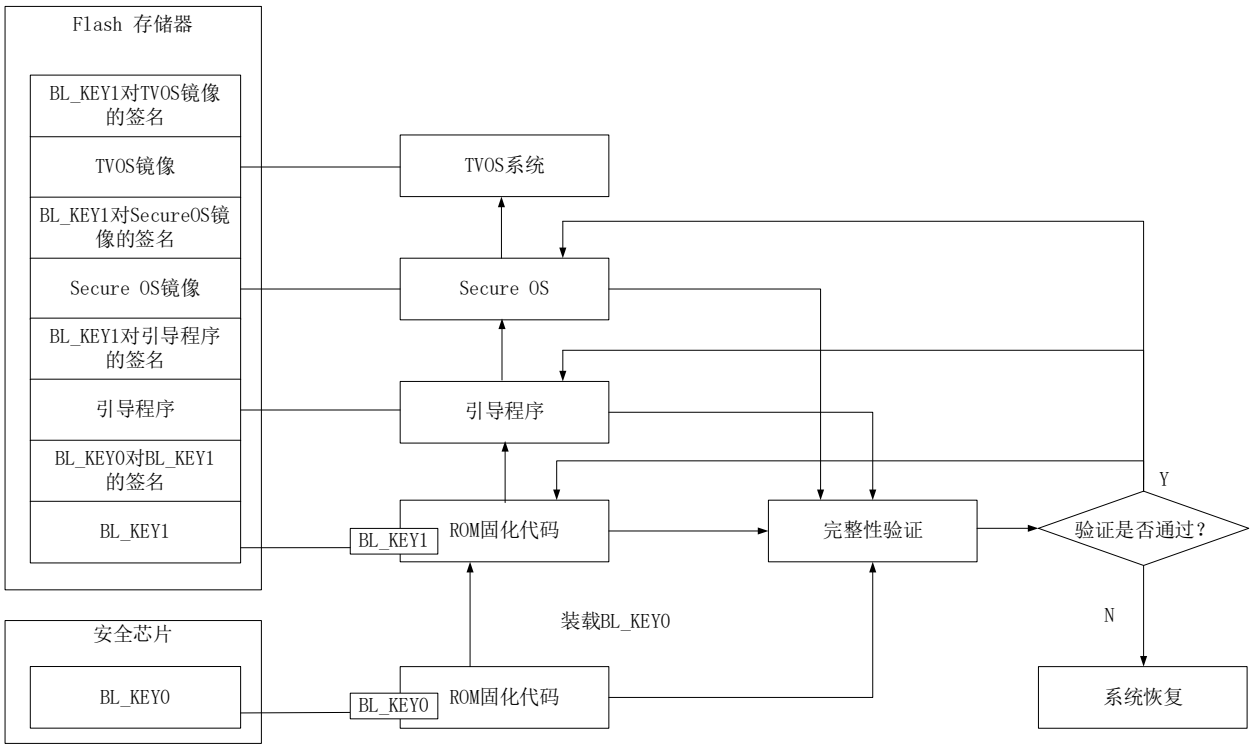

图 C.3 TVOS 安全启动流程

附 录 D
（资料性）
系统安全升级

TVOS的安全升级依靠模块的签名和校验来保证。所有的升级文件，包括系统升级、内置应用升级和第三方应用升级都按照运营商的要求签名保证安全，升级软件时校验签名，通过以后才能进行升级。运行在终端的系统各分区镜像也按照运营商的要求进行签名，并在系统启动时进行签名校验，如果校验不通过则自动重启终端。

系统通过内置升级软件模块检测升级，其升级触发功能按照服务器前端配置的参数来决定TVOS终端是否需要升级。为了保证系统安全升级，所配置的升级参数需要保证唯一性。

TVOS终端在升级前检查终端系统镜像是否符合运营商制定的安全签名规范，保证终端能在安全的环境下正确启动升级软件模块。终端上电后，先启动引导模块。引导模块签名校验通过后根据升级标志判断是否进入系统升级软件模块。升级有OTA下载（通过cable）和IP下载两种模式。所有的升级流经过签名后才能下发，终端进入升级后先下载数据，下载完成后先校验签名再进行擦写和程序烧录。升级过程中，若有断电或信号故障，不能损坏升级软件模块，当终端重新上电或恢复信号后，重新进行系统升级功能。

系统安全升级流程见图D.1。

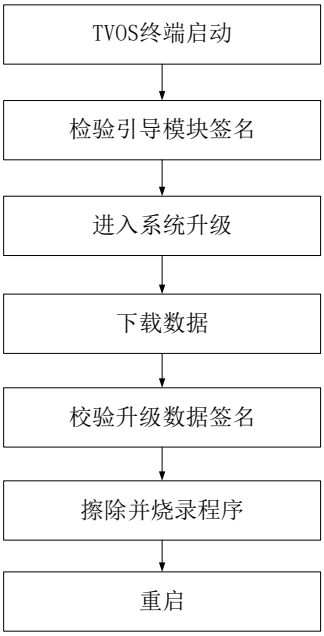

图 D.4　系统安全升级流程

附 录 E
（规范性）
融合 CA TEE 接口

E.1 安全芯片层级密钥驱动接口

DCAS用户端软件从安全侧调用层级密钥驱动接口，实现对层级密钥的配置。

E.1.1 数据类型及结构定义

E.1.1.1 基本数据类型

```
typedef unsigned char           TEE_KLAD_BYTE;
typedef unsigned short          TEE_KLAD_USHORT16;
typedef unsigned long           TEE_KLAD_ULONG32;
typedef unsigned char           TEE_KLAD_BOOLEAN;
```

E.1.1.2 接口返回值枚举类型

```
typedef enum
{
    TEE_KLAD_OK,
    TEE_KLAD_FAIL,
    TEE_KLAD_UNMATCH_CHAN
}TEE_KLAD_STATUS;
```

E.1.2 接口定义

E.1.2.1 TEE_KLAD_Init

层级密钥初始化。

原型：

```
TEE_KLAD_STATUS TEE_KLAD_Init(void)
```

输入：无。

输出：无。

E.1.2.2 TEE_KLAD_DeInit

层级密钥反初始化。

原型：

```
TEE_KLAD_STATUS TEE_KLAD_DeInit(void)
```

输入：无。

输出：无。

E.1.2.3 TEE_KLAD_GetChipId

读取安全芯片标识。

原型：

```
TEE_KLAD_STATUS TEE_KLAD_GetChipId( TEE_KLAD_BYTE *chipId)
```

输入：无。

输出：芯片标识，8字节缓存，应由调用此接口的应用程序负责分配和释放。

E.1.2.4 TEE_KLAD_GetResponseToChallenge

通过SoC层级密钥计算挑战应答结果。

原型：

```
TEE_KLAD_STATUS TEE_KLAD_GetResponseToChallenge
(
TEE_KLAD_BYTE *nonce,
TEE_KLAD_BYTE nonceLength,
int keyDescriptorsLength,
TEE_KLAD_BYTE *keyDescriptors,
TEE_KLAD_BYTE *response,
TEE_KLAD_BYTE *responseLength
)
```

输入：

——nonce：挑战应答数据；

——nonceLength：挑战应答数据长度；

——keyDescriptorsLength：密钥描述符长度；

——keyDescriptors：密钥描述符信息。

输出：

——response：计算所得的挑战应答结果；

——responseLength：挑战应答结果的长度。

其中密钥描述符中会用到的相关描述符如下：

——层级密钥描述符字节：

- 0：ENCRYPTION_KEY_DSCR_TAG = 0x03；
- 1：描述符长度；
- 2：层级密钥级别，对于挑战应答计算，取值为 2；
- 3：层级密钥长度；
- 4-n：被加密的层级密钥值，n 为层级密钥长度+3。

——密钥加密算法描述符字节：

- 0：ENCRYPTION_SCHEME_DSCR_TAG = 0x04；
- 1：描述符长度 = 2；
- 2-3：密钥加密算法枚举：0=3DES、1=AES、2=SM4。

——CA 供应商标识描述符字节：

- 0：VENDOR_ID_DSCR_TAG = 0x05；
- 1：描述符长度 = 2；
- 2-3：CA 供应商标识。

E.1.2.5 TEE_KLAD_SetDescrambler

设置解扰参数及密钥，并触发解扰。

原型：

```
TEE_KLAD_STATUS TEE_KLAD_SetDescrambler
(
int streamPathLength,
TEE_KLAD_BYTE *streamPath,
int numberOfStreamPids,
TEE_KLAD_USHORT16 *streamPids,
int OddkeyDescriptorsLength,
TEE_KLAD_BYTE *OddkeyDescriptor,
int EvenkeyDescriptorsLength,
TEE_KLAD_BYTE *EvenkeyDescriptor
)
```

输入：

——streamPathLength：解扰节目流路径信息长度；

——streamPath：解扰节目流路径信息；

——numberOfStreamPids：解扰节目流中所包含的音视频 Pid 总数；

——streamPids：音视频 Pid 列表；

——OddkeyDescriptorsLength：奇数密钥描述符长度；

——OddkeyDescriptor：奇数密钥描述符；

——EvenkeyDescriptorsLength：偶数密钥描述符长度；

——EvenkeyDescriptor：偶数密钥描述符。

其中奇数和偶数密钥描述符中会用到的相关描述符如下：

控明文制字CW描述符字节：

0：CLEAR_CW_DSCR_TAG = 0x01

1：描述符长度

2-n：明文控制字

被加密的控制字CW描述符字节：

0：ENCRYPTED_CW_DSCR_TAG = 0x02

1：描述符长度

2-n：被加密的控制字。为解密出CW，一些附加的描述符也会在此处提供。

层级密钥描述符字节：

0：ENCRYPTION_KEY_DSCR_TAG = 0x03

1：描述符长度

2：层级密钥级别，CW是0级，其他密钥可以是1，2…等级别

3：层级密钥长度

4-n：被加密的层级密钥值

密钥加密算法描述符字节：

0：ENCRYPTION_SCHEME_DSCR_TAG = 0x04

1：描述符长度 = 2

2-3：密钥加密算法枚举：0=3DES、1=AES、2=SM4

CA供应商标识描述符字节：

0：VENDOR_ID_DSCR_TAG = 0x05
1：描述符长度 = 2
2-3：CA供应商标识
解扰算法描述符字节：
0：DESCRAMBLING_ALGORITHM_DSCR_TAG = 0x07
1：描述符长度 = 2
2-3：解扰算法枚举值：0=DVB-CSA2、1=CSA3
输出：无。

E.1.2.6 TEE_KLAD_StopDescrambler

停止解扰。
原型：

```
TEE_KLAD_STATUS TEE_KLAD_StopDescrambler
(
int streamPathLength,
TEE_KLAD_BYTE *streamPath,
int numberOfStreamPids,
TEE_KLAD_USHORT16 *streamPids
)
```

输入：
——streamPathLength：解扰节目流路径信息长度；
——streamPath：解扰节目流路径信息；
——numberOfStreamPids：解扰节目流中所包含的音视频 Pid 总数；
——streamPids：音视频 Pid 列表。
输出：无。

E.2 硬件安全模块应用程序接口

E.2.1 数据类型及结构定义

E.2.1.1 基本数据类型

HSM接口返回值类型：typedef unsigned int HSM_Result。
基本数据类型如下：

```
typedef unsigned int uint32_t;
typedef unsigned short uint16_t;
typedef unsigned char unit8_t;
```

E.2.1.2 接口返回值枚举类型

```
typedef enum {
HSM_RESULT_OK = 0, /* 访问HSM成功 */
HSM_RESULT_ERROR_SECURITY = 1, /* 应用程序无访问HSM的权限 */
HSM_RESULT_ERROR_INVALID_PARAMETERS = 2,  /* 参数校验失败 */
HSM_RESULT_ERROR_NOT_SUPPORTED = 3,   /* 操作类型不支持 */
```

```
HSM_RESULT_ERROR_OUT_OF_RANGE = 4,    /* 长度或偏移量超过范围 */
HSM_RESULT_ERROR_DRIVER = 5,    /* HSM驱动内部错误导致访问失败 */
HSM_RESULT_ERROR_IO = 6,    /* 读写HSM操作失败 */
HSM_RESULT_ERROR_BUSY = 7,    /* HSM设备忙，稍后再试 */
HSM_RESULT_ERROR_API_COMMUNICATION = 8,    /* HSM通讯层错误 */
HSM_RESULT_ERROR_INSUFFICIENT_BUFFER = 9,  /* 缓冲区太小 */
HSM_RESULT_ERROR_OPERATION_FAILED = 10    /* 一般性错误 */
} HSM_Result;
```

E.2.2 接口定义

E.2.2.1 TEE_HSM_GetSoftwareVersion

获得硬件安全模块软件版本号。

版本号构成方式建议如下：一个至少8个字符的固定前缀，再接上一个变化的版本信息串，比如：“DCAS HSM Version: 34.9.2b”。

原型：

```
HSM_Result TEE_HSM_GetSoftwareVersion(int* versionLength,char* version)
```

输入：

——versionLength：版本号字符串缓冲区长度。

输出：

——versionLength：调用本接口后，输出实际版本号字符串长度；

——version：版本号字符串，各 HSM 厂商自行定义其格式。

返回值：

——HSM_RESULT_OK：访问成功；

——其他：访问失败，具体失败原因见 E.2.1.2。

E.2.2.2 TEE_HSM_GetHsmGeneralInfo

获得硬件安全模块基本信息。

原型：

```
HSM_Result TEE_HSM_GetHsmGeneralInfo
(
uint8_t* hsmStatus,
int*     hsmIdLength,
uint8_t* hsmId
)
```

输入：

——hsmIdLength：hsmId 缓冲区长度。

输出：

——hsmStatus：硬件安全模块激活状态，0 表示未激活，1 表示已激活，2 表示待激活中间状态；

——hsmIdLength：实际读取的 hsmId 长度；

——hsmId：硬件安全模块 Id。

返回值：

——HSM_RESULT_OK：访问成功；

——其他：访问失败，具体失败原因见 E.2.1.2。

E.2.2.3 TEE_HSM_GetHsmDiagnosticInfo

获得硬件安全模块诊断信息。

原型：

```
HSM_Result TEE_HSM_GetHsmDiagnosticInfo
•
uint8_t* activeChipIdLength,
uint8_t* activeChipId,
uint16_t* activeCasVendorId,
int* hsmDeviceCertificateLength,
uint8_t* hsmDeviceCertificate,
int* hsmVendorCertificateLength,
uint8_t* hsmVendorCertificate
)
```

输入：

——activeChipIdLength：激活芯片 Id 缓冲区长度；

——hsmDeviceCertificateLength：硬件安全模块证书缓冲区长度；

——hsmVendorCertificateLength：硬件安全模块供应商证书缓冲区长度。

输出：

——activeChipIdLength：实际激活芯片 Id 长度；

——activeChipId：激活芯片 Id，当芯片内部没有烧写 ChipId 时，返回全 0；

——activeCasVendorId：激活 HSM 芯片的 CAS 供应商标识，当芯片内没有 CAS 标识时，返回全 0；

——hsmDeviceCertificateLength：实际读取的硬件安全模块证书长度；

——hsmDeviceCertificate：硬件安全模块证书内容；

——hsmVendorCertificateLength：实际读取的硬件安全模块供应商证书长度；

——hsmVendorCertificate：硬件安全模块供应商证书内容。

返回值：

——HSM_RESULT_OK：访问成功；

——其他：访问失败，具体失败原因见 E.2.1.2。

E.2.2.4 TEE_HSM_GetHsmLastTimeStamp

获得最近一次硬件安全模块被激活的时间。

原型：

```
HSM_Result TEE_HSM_GetHsmLastTimeStamp(uint32_t* timestamp)
```

输入：无。

输出：

——timestamp:最后一次收到的可被接受的激活消息的时间,如果芯片从未被激活过,则返回全 0。

返回值：

——HSM_RESULT_OK：访问成功；

——其他：访问失败，具体失败原因见 E.2.1.2。

E.2.2.5 TEE_HSM_GetHsmActivationInfo

从硬件安全模块中读取激活信息中CA供应商专属的数据。

原型：

```
HSM_Result TEE_HSM_GetHsmActivationInfo
(
unint16_t vendorId,
int* casPropDataLength,
uint8_t* casPropData
)
```

输入：

——vendorId：CA 供应商标识；

——casPropDataLength：CA 专属数据缓冲区长度。

输出：

——casPropDataLength：CA 专属数据实际长度。

返回值：

——HSM_RESULT_OK：访问成功；

——其他：访问失败，具体失败原因见 E.2.1.2。

E.2.2.6 TEE_HSM_GenerateActivationRequest

生成激活请求数据。

原型：

```
HSM_Result TEE_HSM_GenerateActivationRequest
(
uint16_t vendorId,
int vendorCertificateLength,
uint8_t* vendorCertificate,
int chipIdLength,
uint8_t* chipId,
int longitude,
int latitude,
uint32_t timestamp,
int* activationRequestLength,
uint8_t* activationRequest
)
```

输入：

——vendorId：CA 供应商标识；

——vendorCertificateLength：CA 供应商证书长度；

——vendorCertificate：CA 供应商证书；

——chipIdLength：芯片标识长度；

——chipId：芯片标识；

——longitude：终端所在位置经度，取值为实际经度乘以 106；

——latitude：终端所在位置纬度，取值为实际纬度乘以 106；

——timestamp：系统时间（如北斗时间），表示为自 1970 年 1 月 1 日 0 点 0 分 0 秒以来的秒数；

——activationRequestLength：激活请求消息缓冲区长度。
输出：
——activationRequestLength：生成的激活请求消息实际长度；
——activationRequest：生成的激活请求消息数据。
返回值：
——HSM_RESULT_OK：访问成功；
——其他：访问失败，具体失败原因见E.2.1.2。

E.2.2.7 TEE_HSM_SetMessage

将收到的主激活消息或辅助激活消息送入HSM进行解析，并将结果存储到HSM中。
原型：

```
HSM_Result TEE_HSM_SetMessage
(
uint16_t vendorId,
int vendorCertificateLength,
uint8_t* vendorCertificate,
int messageLength,
uint8_t* message
)
```

输入：
——vendorId：CA供应商标识；
——vendorCertificateLength：CA供应商证书长度；
——vendorCertificate：CA供应商证书；
——messageLength：激活消息长度；
——message：激活消息内容。
输出：无。
返回值：
——HSM_RESULT_OK：访问成功；
——其他：访问失败，具体失败原因见E.2.1.2。

E.2.2.8 TEE_HSM_OpenSac

与硬件安全模块建立安全认证通道。
原型：

```
HSM_Result TEE_HSM_OpenSac
(
uint16_t vendorId,
int vendorCertificateLength,
uint8_t* vendorCertificate,
int chipIdLength,
uint8_t* chipId,
int PairKLength,
uint8_t* PairK,
```

```
    int randomNonceLength,
    uint8_t* randomNonce,
    int* hsmSacHandleLength,
    uint8_t* hsmSacHandle
    )
```

输入：

——vendorId：CA 供应商标识；

——vendorCertificateLength：CA 供应商证书长度；

——vendorCertificate：CA 供应商证书；

——chipIdLength：芯片标识长度；

——chipId：芯片标识；

——PairKLength：配对密钥长度；

——PairK：配对密钥；

——randomNonceLength：随机数缓冲区长度；

——randomNonce：随机数缓冲区。

输出：

——hsmHandleLength：hsm 访问句柄长度，CAS 用户端可信应用软件应分配 16 字节的缓冲区；

——hsmSacHandle：hsm 访问句柄，CAS 用户端可信应用软件通过此句柄对 HSM 进行读写操作。

返回值：

——HSM_RESULT_OK：访问成功；

——其他：访问失败，具体失败原因见 E.2.1.2。

E.2.2.9 TEE_HSM_Read

从硬件安全模块中读取数据。

原型：

```
HSM_Result TEE_HSM_Read
(
uint16_t vendorId,
int hsmSacHandleLength,
uint8_t* hsmSacHandle,
uint32_t offset,
uint32_t* length,
unint8_t* data
)
```

输入：

——vendorId：CA 供应商标识；

——hsmHandleLength：硬件安全模块访问句柄长度；

——hsmSacHandle：硬件安全模块访问句柄；

——offset：读取数据所在位置的偏移量；

——length：期望读取的数据长度。

输出：

——length：实际读取的数据长度；

——data：实际读取的数据内容。

返回值：

——HSM_RESULT_OK：访问成功；

——其他：访问失败，具体失败原因见 E.2.1.2。

E.2.2.10 TEE_HSM_Write

将收到的主激活消息或辅助激活消息送入HSM进行解析，并将结果存储到HSM中。

原型：

```
HSM_Result TEE_HSM_Write
(
uint16_t vendorId,
int hsmSacHandleLength,
uint8_t* hsmSacHandle,
uint32_t offset,
uint32_t* length,
uint8_t* data
)
```

输入：

——vendorId：CA 供应商标识；

——hsmHandleLength：硬件安全模块访问句柄长度；

——hsmSacHandle：硬件安全模块访问句柄；

——offset：写入数据所在目标位置的偏移量；

——length：期望写入的数据长度；

——data：实际写入的数据。

输出：

——length：实际写入的数据长度。

返回值：

——HSM_RESULT_OK：访问成功；

——其他：访问失败，具体失败原因见 E.2.1.2。

E.2.2.11 TEE_HSM_ReadPositionParameters

从硬件安全模块读取存储所设置的位置信息。

原型：

```
HSM_Result TEE_HSM_ReadPositionParameters
(
uint16_t vendorId,
int hsmSacHandleLength,
uint8_t* hsmSacHandle,
int* longitude,
int* latitude,
uint32_t* radius
)
```

输入：

——vendorId：CA 供应商标识；

——hsmHandleLength：硬件安全模块访问句柄长度；

——hsmSacHandle：硬件安全模块访问句柄。

输出：

——longitude：设置位置的经度，取值为实际经度乘以 106；

——latitude：设置位置的纬度，取值为实际纬度乘以 106；

——radius：实际位置与设置位置之间的距离，取值单位为十米。

返回值：

——HSM_RESULT_OK：访问成功；

——其他：访问失败，具体失败原因见 E.2.1.2。

E.2.2.12 TEE_HSM_ReadPublicSecureStorage

从硬件安全模块的公共区域读取数据。

原型：

```
HSM_Result TEE_HSM_ReadPublicSecureStorage
(
uint32_t offset,
uint32_t* length,
unint8_t* data
)
```

输入：

——offset：读取数据所在位置的偏移量；

——length：期望读取的数据长度。

输出：

——length：实际读取的数据长度；

——data：实际读取的数据。

返回值：

——HSM_RESULT_OK：访问成功；

——其他：访问失败，具体失败原因见 E.2.1.2。

E.2.2.13 TEE_HSM_WritePublicSecureStorage

向硬件安全模块的公共区域写入数据。

原型：

```
HSM_Result TEE_HSM_WritePublicSecureStorage
(
uint16_t vendorId,
int hsmSacHandleLength,
uint8_t* hsmSacHandle,
uint32_t offset,
uint32_t* length,
unint8_t* data
```

```
    )
```

输入：

——vendorId：CA 供应商标识；

——hsmHandleLength：硬件安全模块访问句柄长度；

——hsmSacHandle：硬件安全模块访问句柄；

——offset：写入数据目标地址的偏移量；

——length：期望写入的数据长度；

——data：实际写入的数据。

输出：

——length：实际写入的数据长度。

返回值：

——HSM_RESULT_OK：访问成功；

——其他：访问失败，具体失败原因见 E.2.1.2。

E.2.2.14 TEE_HSM_ChangeCwEncryptionScheme

设置硬件安全模块中重加密算法，应与SoC层级密钥所用算法一致。

只有当HSM默认使用的重加密算法不被SoC层级密钥支持时，才需调用此接口。

如需调用此接口，应在第一次调用TEE_HSM_GenerateCW接口前进行。

如果没有调用此接口，则HSM使用经过TEE_HSM_GenerateCW接口设置的层级密钥算法对CW进行重加密。

原型：

```
HSM_Result TEE_HSM_ChangeCwEncryptionScheme
(
uint16_t vendorId,
int hsmSacHandleLength,
uint8_t* hsmSacHandle,
uint16_t socSchemeId
)
```

输入：

——vendorId：CA 供应商标识；

——hsmHandleLength：硬件安全模块访问句柄长度；

——hsmSacHandle：硬件安全模块访问句柄；

——socSchemeId：代表重加密算法的数值，0 表示 3DES，1 表示 AES，2 表示 SM4。

输出：无。

返回值：

——HSM_RESULT_OK：访问成功；

——其他：访问失败，具体失败原因见 E.2.1.2。

E.2.2.15 TEE_HSM_GenerateCW

由硬件安全模块生成后续由终端安全芯片使用的被加密的控制字CW。

原型：

```
HSM_Result TEE_HSM_GenerateCW
```

```
(
unint16_t vendorId,
int hsmSacHandleLength,
uint8_t* hsmSacHandle,
uint16_t schemeId,
int keyL2Length,
uint8_t* keyL2,
int keyL1Length,
uint8_t* keyL1,
int keyL0Length,
uint8_t* keyL0,
int CWLength,
uint8_t* CW
)
```

输入：

——vendorId：CA 供应商标识；

——hsmHandleLength：硬件安全模块访问句柄长度；

——hsmSacHandle：硬件安全模块访问句柄；

——schemeId：硬件安全模块中层级密钥的算法，0 表示 3DES，1 表示 AES，2 表示 SM4；

——keyL2Length：层级密钥体系中 2 级密钥长度；

——keyL2：2 级层级密钥；

——keyL1Length：层级密钥体系中 1 级密钥长度；

——keyL1：1 级层级密钥；

——keyL0Length：层级密钥体系中 0 级密钥长度；

——keyL0：0 级层级密钥；

——CWLength：控制字缓冲区长度。

输出：

——CWLength：控制字实际长度；

——CW：生成的被加密后的控制字。

返回值：

——HSM_RESULT_OK：访问成功；

——其他：访问失败，具体失败原因见 E.2.1.2。

E.2.2.16 TEE_HSM_CloseSac

关闭与硬件安全模块之间的安全认证通道。

原型：

```
HSM_Result TEE_HSM_CloseSac
(
uint16_t vendorId,
int hsmSacHandleLength,
uint8_t* hsmSacHandle
)
```