GY

中华人民共和国广播电影电视行业标准

GY/T 308—2017

单向可下载条件接收系统技术规范

Technical specification of downloadable conditional access system for unidirectional network

2017－11－09 发布　　2017－11－09 实施

国家新闻出版广电总局　　发布

目　次

前　言

本标准按照GB/T 1.1—2009给出的规则起草。

本标准由全国广播电影电视标准化技术委员会（SAC/TC 239）归口。

本标准起草单位：国家广播电影电视总局广播科学研究院、中央电视台、北京永新视博数字电视技术有限公司、北京数码视讯科技股份有限公司、深圳市海思半导体有限公司、杭州国芯科技股份有限公司、湖南国科微电子股份有限公司、上海高清数字科技产业有限公司、国家广播电视网工程技术研究中心、北京安视网信息技术有限公司。

本标准主要起草人：盛志凡、丁文华、杨勋、王强、严海峰、方中华、宿玉文、李望舒、邹峰、解伟、张晶、熊彬、郭永伟、饶丰、郑力铮、刘晶磊、叶丰、黄新军、王旭升、王茵、张鹏。

引　言

本标准的发布机构提请注意，声明符合本标准时，可能使用涉及本标准有关内容的相关授权的和正在申请的专利如下：

序号	标准章条号	专利名称
1	4、6、7、附录B、C	加密控制字的保护方法、硬件安全模块、主芯片和终端
2	4、5、6、7、附录A、B、C	一种可下载可替换条件接收系统
3	4、5、6、7	一种条件接收系统发送端的根密钥生成方法
4	4、5、6、7	一种条件接收系统接收端的根密钥生成方法、模块、芯片及接收终端

本标准的发布机构对于该专利的真实性、有效性和范围无任何立场。

该专利持有人已向本标准的发布机构保证，他愿意同任何申请人在合理且无歧视的条款和条件下，就专利授权许可进行谈判。该专利持有人的声明已在本标准的发布机构备案，相关信息可以通过以下联系方式获得：

专利权利人	联系地址	联系人	邮政编码	电话	电子邮件
国家广播电影电视总局广播科学研究院	北京市复兴门外大街2号	杨勃	100866	010-86094200	yangqing@abs.ac.cn
中央电视台	北京市复兴路11号	蔡贺	100859	010-68506440	caihe@cctv.com
北京永新视博数字电视技术有限公司	北京市海淀区上地东路5号京蒙高科大厦B座4层	张晶	100085	010-62971199	zhangjing@novel-supertv.com
北京数码视讯科技股份有限公司	北京市海淀区上地五街数码视讯大厦	熊彬	100085	010-82345800	xiongbin@sumavision.com
深圳市海思半导体有限公司	广东省深圳市龙岗区坂田华为基地	严海峰	518129	0755-28780808	yanhaifeng@hisilicon.com
北京安视网信息技术有限公司	北京市西城区南礼士路66号建威大厦1202	张鹏	100026	010-87521564	pengz2@nds.com

请注意除上述专利外，本标准的某些内容仍可能涉及专利。本标准的发布机构不承担识别这些专利的责任。

本标准涉及到使用密码的部分应符合国家密码的管理政策。

单向可下载条件接收系统技术规范

1 范围

本标准规定了单向可下载条件接收系统的系统架构和功能、安全机制、总体要求和终端系统等内容。

本标准适用于单向广播电视网的可下载条件接收系统。

2 规范性引用文件

下列文件对于本标准的应用是必不可少的。凡是注日期的引用文件，仅所注日期的版本适用于本标准。凡是不注日期的引用文件，其最新版本（包括所有的修改单）适用于本标准。

GY/Z 175—2001 数字电视广播电视条件接收系统规范

GY/T 255—2012 可下载条件接收系统技术规范

GM/T 0002—2012 SM4 分组密码算法

GM/T 0003—2012 SM2 椭圆曲线公钥密码算法

GM/T 0004—2012 SM3 密码杂凑算法

GN/T 0005—2012 随机性检测规范

GM/T 0015—2012 基于 SM2 密码算法的数字证书格式规范

3 术语、定义、缩略语和约定

3.1 术语和定义

下列术语和定义适用于本标准。

3.1.1

DCAS 用户端应用软件 DCAS App

运行在终端软件平台之上的DCAS应用程序，终端部署后，该程序可以通过在线推送等方式升级或替换。

3.1.2

DCAS 用户端可信应用软件 DCAS trusted App

运行在终端可信执行环境中的DCAS应用程序。终端部署后，该程序可以通过在线推送等方式升级或替换。

3.1.3

DCAS 用户端软件 DCAS client software

一种由DCAS用户端应用软件和DCAS可信应用软件构成，并由二者在终端软件平台之上协同工作实现的终端应用软件。

3.1.4

DCAS 用户端软件数据 DCAS client software data

DCAS用户端软件在运行中需要保存或更新的数据，其中包括CA授权信息，CA私有数据,定位信息等DCAS相关的关键数据。

3.1.5

DCAS 管理器 DCAS manager

负责注册DCAS用户端软件、支撑DCAS用户端应用软件与DCAS用户端可信应用软件信息交互以及接收与转发DCAS授权控制和管理信息等功能的软件模块。

3.1.6

安全芯片密钥还原函数 secure chipset key de-obfuscation

用于将加密的安全芯片密钥还原成安全芯片密钥的算法。

3.1.7

层级密钥 key ladder

一种保证控制字安全传送的结构化多级密钥机制。

3.1.8

传输流过滤 transport stream filtering

通过一定的过滤机制，从传输数据流中提取符合过滤条件的数据。

3.1.9

复用 multiplex

在一个物理频道中承载一个或多个业务或事件的所有数据流。

3.1.10

控制字 control word

用于加解扰的控制信息。

3.1.11

加扰 scramble

通常在广播前端的条件接收系统控制下改变或控制被传送业务码流的某些特性,使得未经授权的接收者不能得到正确的业务码流。

3.1.12

解扰 descramble

解扰是加扰的逆过程。

3.1.13

根密钥 root key

用于层级密钥首层的密钥。

3.1.14

哈希值 hash value

使用杂凑算法对任一数值进行计算后得到的值为哈希值。

3.1.15

启动加载软件 bootloader

用于接收终端启动后初始化硬件和加载主软件的程序

3.1.16

挑战应答 challenge response

DCAS用户端软件通过DCAS管理器使用终端安全芯片层级密钥进行运算的一种流程。

3.1.17

挑战应答数据 Nonce

由DCAS前端系统发出的用于挑战应答的数据。

3.1.18

硬件安全模块 hardware secure module

一种具备控制字处理、授权访问和安全存储等支撑单向接收终端硬件层次安全增强功能的安全芯片。

3.1.19

终端安全芯片 terminal secure chipset

一种具备密钥派生和层级密钥等安全功能的码流处理芯片。

3.1.20

终端软件平台 terminal software platform

一种运行于接收终端上、集成了各种硬件驱动、具备各种终端应用API、并可对终端应用按一定安全要求进行下载和启动，以及为终端应用提供安全运行环境的软件平台。

3.2 缩略语

API 应用程序编程接口(Application Programming Interface)
CA 条件接收（Conditional Access）
CAT 条件接收表（Conditional Access Table）
CAS 条件接收系统（Conditional Access System）
ChipID 芯片标识（Chipset Identification）
CPU 中央处理器（Central Process Unit）
CREEK 重加密密钥（Crypto-toolkit Re-encryption Key）
CW 控制字（Control Word）

DCAS 可下载条件接收系统（Downloadable Conditional Access System）
ECM 授权控制信息（Entitlement Control Message）
ECMG 授权控制信息发生器（Entitlement Control Message Generator）
ECW 加密的控制字（Encrypted Control Word）
EMM 授权管理信息（Entitlement Management Message）
EMMG 授权管理信息发生器（Entitlement Management Message Generator）
EPG 电子节目指南（Electronic Program Guide）
ESCK 加密的安全芯片密钥（Encrypted Secure Chipset Key）
GP TEE 标准化平台（Global Platform）
HSM 硬件安全模块 (Hardware Security Module)
HSMID 硬件安全模块标识 (Hardware Security Module Identification)
KDF 密钥派生函数(Key Derivation Function)
NVM 非易失性存储器(Non-Volatile Memory)
OTP 一次性可编程（One Time Programmable）
PairK 安全认证通道配对密钥（Pairing Key）
PID 包标识（Packet Identification）
SAC 安全认证通道 (Secure Authenticated Channel)
SCK 安全芯片密钥（Secure Chipset Key）
SCKv 安全芯片密钥厂商派生密钥（Secure Chipset Key Vendor）
Seedv 掩码密钥厂商派生密钥（Seed Vendor）
SI 业务信息（Service Information）
SMK 掩码密钥（Secret Mask Key）
SoC 终端安全芯片(System on Chip)
TEE 可信执行环境(Trusted Execution Environment)
Vendor_SysID 条件接收厂商系统标识（Vendor System Identification）

4 系统架构概述

4.1 DCAS 系统架构

DCAS 是一套完整的端到端业务保护系统，具有传统条件接收系统所有的授权控制和管理功能，能够与传统条件接收系统共存，系统灵活性高。接收终端可以通过下载 DCAS 用户端软件，实现在不同 DCAS 系统终端间的灵活切换，从而实现终端业务保护水平化，即接收终端不需要更换硬件和全面升级软件，就可以适配和支持不同条件接收系统厂家相关条件接收前端系统，以及可以适配和支持同一条件接收系统厂家不同版本的条件接收前端系统。

DCAS 安全机制综合运用 DCAS 密钥机制、软硬件安全处理技术和安全管理手段，对 DCAS 前端、终端安全芯片、硬件安全模块、用户端软件以及安全芯片密钥植入模块进行安全保护，保证端到端系统安全。

DCAS 由前端、终端和安全数据管理平台组成。DCAS 架构图见图 1。

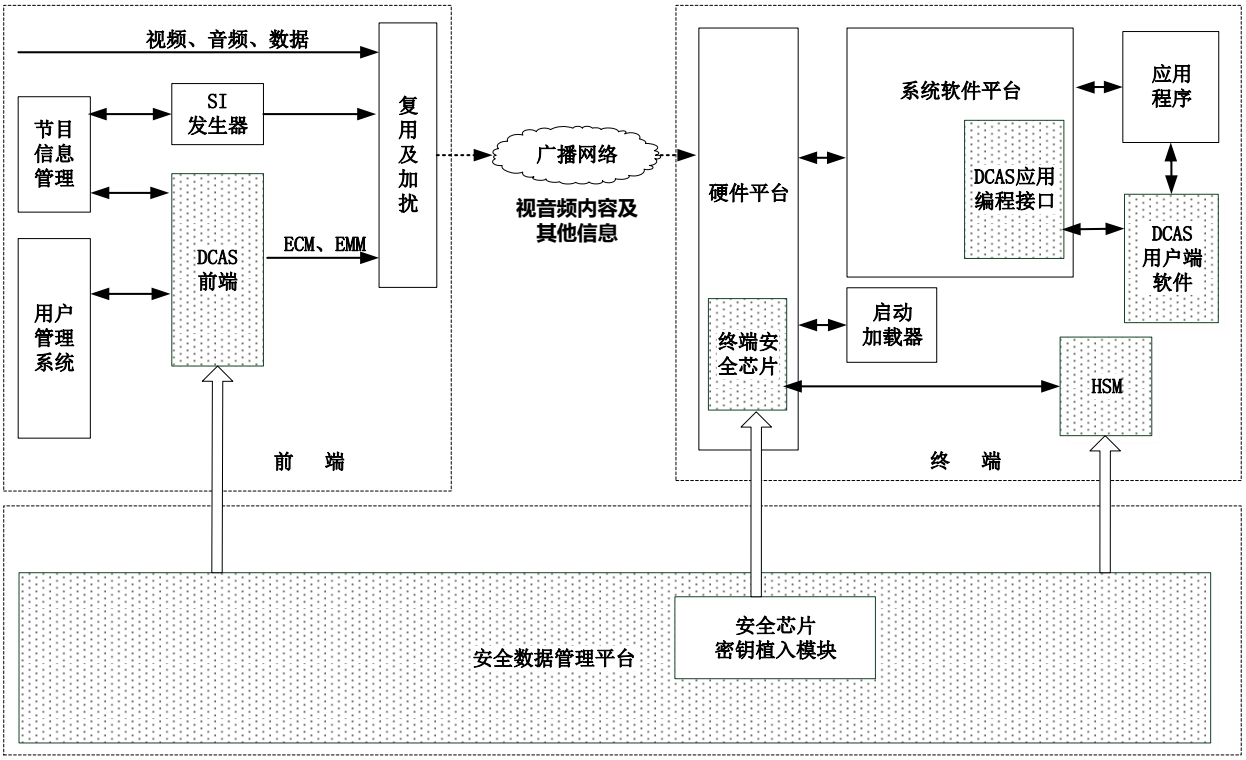

图1 DCAS 架构图

a）DCAS 前端

DCAS 前端对输入的音视频流进行加扰，通过广播信道发送条件接收的授权等信息，完成业务的加密保护传送和合法授权控制管理，是实现 DCAS 各项功能的基础。DCAS 前端主要包括 ECMG、EMMG、密钥管理和其他模块等。

b）DCAS 终端

DCAS 终端对用户的授权进行合法性验证，解扰受保护的业务，实现业务的条件接收。终端软件平台可以安全地下载、更新和替换 DCAS 用户端软件。DCAS 终端主要包括终端安全芯片、HSM、DCAS 用户端软件和终端软件平台的 DCAS 应用编程接口。

c）安全数据管理平台

DCAS 安全数据管理平台生成并管理 DCAS 相关密钥，向 DCAS 前端提供 SCKv 和 Vendor_SysID 等必要信息，通过密钥植入模块向终端安全芯片提供 ChipID、ESCK、BL_KEY0 等必要信息，向 HSM 提供根证书。

4.2 DCAS 前端系统

DCAS前端功能结构图见图2。

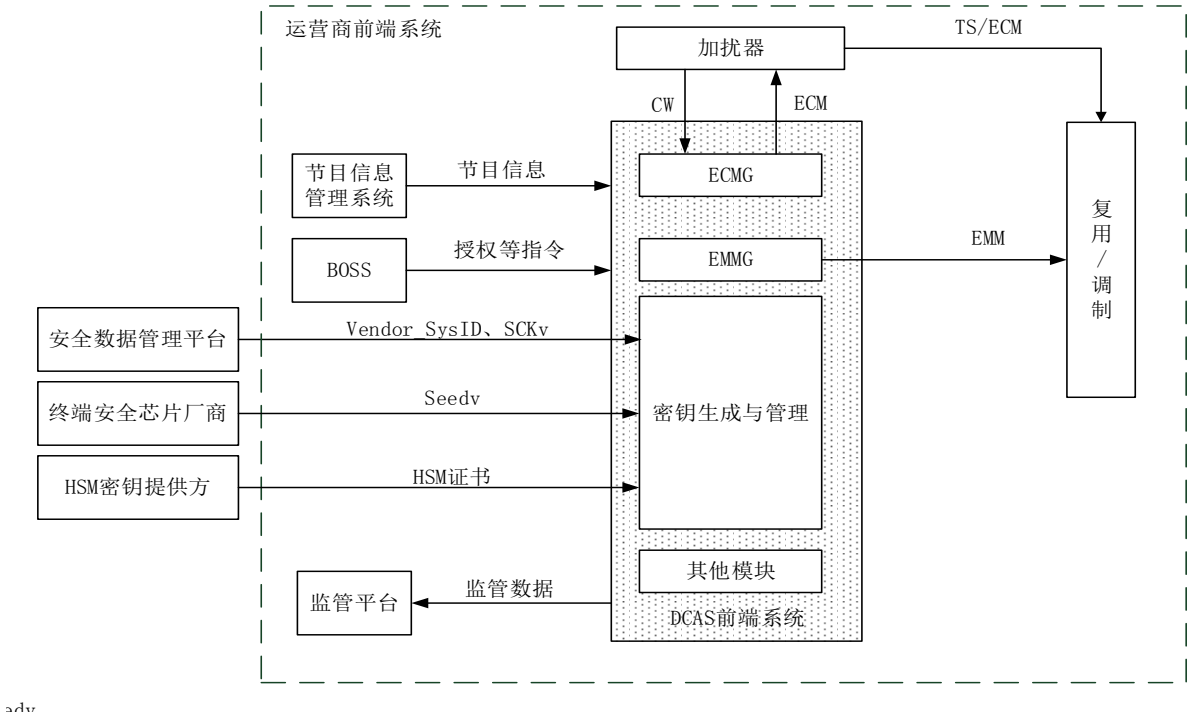

图2　DCAS 前端功能结构图

主要模块包括：

a)　ECMG

ECMG 实现和加扰器连接，ECMG 接收加扰器送来的 CW，生成 ECM 信息返回给加扰器，基本功能应满足 GY/Z 175-2001 中附录 E 同密技术中的要求。

b)　EMMG

EMMG 生成 EMM 信息，并通过与复用器接口发送 EMM。EMMG 的基本功能应满足 GY/Z 175-2001 中附录 E 同密技术中的要求。

c)　密钥生成与管理

根据安全数据管理平台和终端安全芯片厂商提供的终端安全芯片信息，生成并管理根密钥和层级密钥。根据安全数据管理平台和 HSM 密钥提供方提供的 HSM 信息，进行 HSM 相关数据的加密和签名。层级密钥的生成和管理由 DCAS 前端系统自行处理。

d)　其他模块

DCAS 还包括网管模块、监管模块和对外接口等其他模块。

4.3　DCAS 终端系统

DCAS终端功能结构见图3。

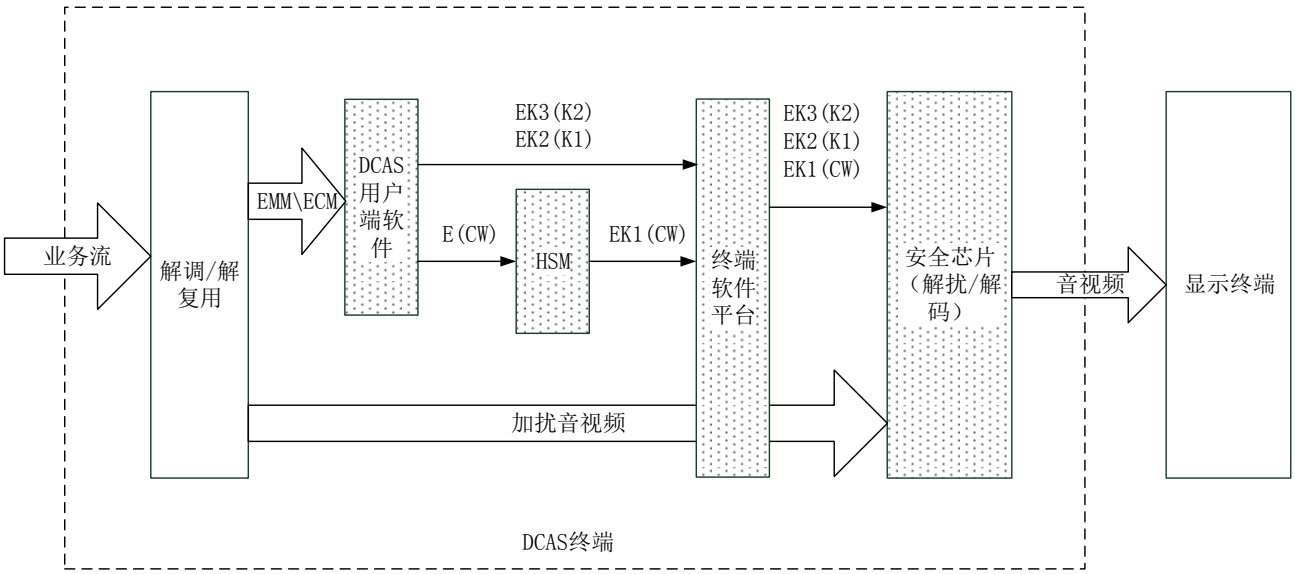

图3　DCAS 终端功能结构图

主要模块包括：

a)　DCAS 用户端软件

DCAS 用户端软件是运行在 DCAS 终端软件平台的应用程序，以标准应用程序接口的方式与平台以及其他模块通信，DCAS 用户端软件可以被下载、更新和替换。DCAS 用户端软件负责解析并处理 EMM 数据，向终端安全芯片和 HSM 提供层级密钥数据。

b)　终端软件平台

终端软件平台是运行在在终端安全芯片硬件及相应驱动软件之上的公共软件，提供 DCAS 用户端软件所需的标准应用程序接口，保证 DCAS 用户端软件下载、启动以及运行过程中的完整性、可靠性和安全性。

c)　终端安全芯片

终端安全芯片接收层级密钥和加扰业务流，对层级密钥进行解密，并对加扰业务流进行解扰和解码，输出解码后的视音频。

d)　HSM

HSM 是提供安全存储服务和算法工具的独立安全芯片，通过层级密钥和重加密对控制字进行保护，通过与终端安全芯片的协同工作，实现对单向 DCAS 系统的安全增强。

4.4　DCAS 安全数据管理平台

安全数据管理平台生成并植入终端安全芯片密钥和安全数据，并对密钥和安全数据进行维护和管理。主要功能包括：

a)　生成终端安全芯片密钥

终端安全芯片密钥生成：生成 ChipID 和 ESCK 文件，通过安全的方式送到安全芯片密钥植入模块，用于把密钥写入终端安全芯片中。

根密钥派生中间密钥生成：生成 ChipID 和 SCKv 文件，通过安全的方式送到条件接收厂商，用于生成根密钥 K3。

终端安全芯片启动校验密钥生成：生成 BL_KEY0 密钥对，将公钥送到终端安全芯片厂商，把密钥写入终端安全芯片中。

b) 植入终端安全芯片密钥

安全数据管理平台通过放置在芯片厂商生产线上的密钥植入模块,将保存在密钥植入模块中的芯片密钥，包括 ChipID 和 ESCK 等，利用相应芯片生产线上的安全传输协议，写入终端安全芯片。

c) 启动校验密钥签名

使用 BL_KEY0 对 BL_KEY1 持有者的 BL_KEY1 公钥进行签名,使终端安全芯片可以校验 BL_KEY1 持有者。

d) 管理 HSM 证书

生成 HSM 根证书，签发下级 HSM 厂商证书和 CA 厂商证书

5 安全机制

5.1 密钥机制

5.1.1 密钥模型

DCAS 系统的密钥机制包括根密钥派生机制、层级密钥机制、安全数据管理机制和业务加解扰机制，与 GY/T 255—2012 中第 5 章的密钥机制相同。DCAS 系统的密钥模型见图 4。

根密钥派生机制使终端安全芯片能实时运用其内置的根密钥派生模块派生出个性化根密钥;安全数据管理机制通过数据分散安全管理保障了根密钥生成管理的安全性;根密钥派生机制和安全数据管理机制密切协同，使不同 DCAS 系统可安全地基于同一终端安全芯片的根密钥派生模块派生出各自个性化的根密钥。

层级密钥机制实现控制字在传输过程中的保护,是以根密钥派生机制派生获得的根密钥为第一级密钥，对控制字进行逐层加解密保护，同时层级密钥中还包含挑战应答功能以实现使用终端安全芯片层级密钥进行运算。对内容的保护是通过采用层级密钥机制输出控制字实现的业务加解扰机制。

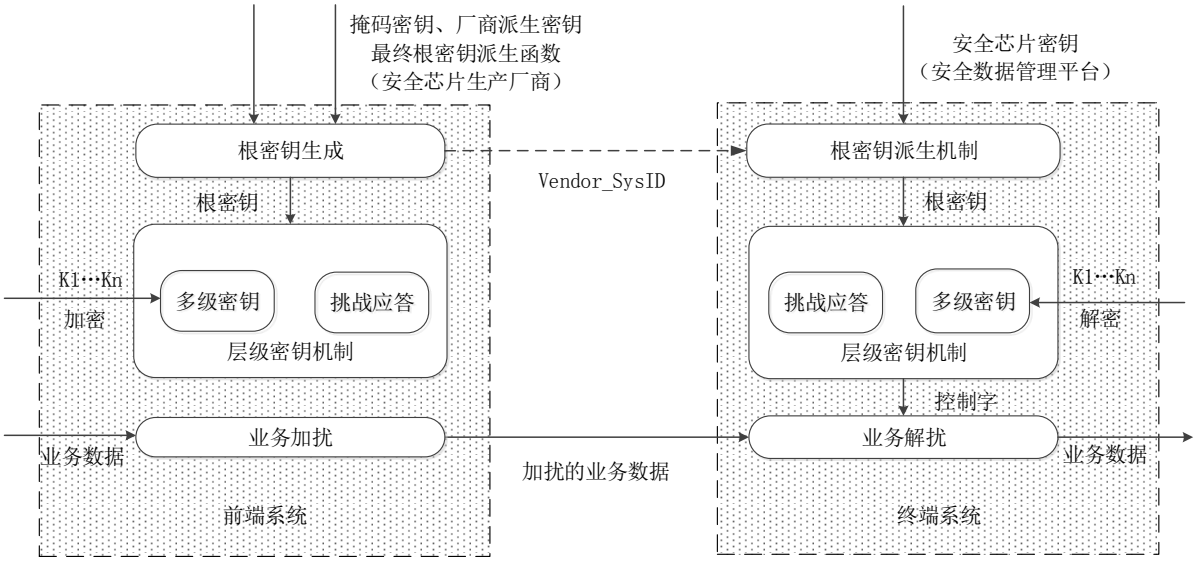

图4　密钥模型

5.1.2　根密钥派生机制

根密钥派生机制包括前端根密钥派生和终端根密钥派生。

a)　前端根密钥生成

DCAS 前端系统采用终端安全芯片生产厂商和安全数据管理平台提供的必要数据和相关算法，在终端使用前为每个终端派生出对应终端安全芯片的根密钥。

b)　终端根密钥派生

DCAS 终端利用 DCAS 用户端软件，为终端安全芯片提供相关必要信息，并通过终端安全芯片内预置的根密钥派生模块派生出根密钥。

5.1.3　层级密钥机制

层级密钥机制包含多层密钥功能和挑战应答功能。

多层密钥功能是指按照层级方式对控制字进行逐层解密，保证控制字在使用和传输过程的安全。

挑战应答功能是向多层密钥的某一层级发送运算数值，然后终端安全芯片按指定算法返回计算结果。

5.1.4　安全数据管理机制

安全数据管理机制是 DCAS 密钥机制的核心组成部分。它运用安全信息分散管理的方法，将前端和终端根密钥派生中所需的信息交由安全数据管理平台、条件接收厂商和芯片厂商分别管理，保证了 DCAS 系统的安全性及中立性。

5.1.5　业务加解扰机制

业务加解扰机制实现业务数据从前端到终端的安全传递。广播节目加解扰机制具体技术要求见 GY/Z 175-2001。

5.2 DCAS 终端硬件安全机制

DCAS 终端硬件安全机制通过终端安全芯片和 HSM 保证。具体技术要求见 7.3 和 7.4。

5.3 DCAS 终端软件安全机制

5.3.1 信任链

DCAS 用户端软件安全机制基于一种自底向上的信任链，从终端安全芯片至启动加载软件、终端软件平台和 DCAS 用户端软件，采用数字签名技术建立信任链，只有在在信任链中的每个环节都通过签名校验后，信任链的后一环节方可启动。DCAS 用户端软件除应通过签名安全验证外，在运行时还应有数据安全保证机制。

启动加载软件在运行之前由终端安全芯片对其进行数据来源可靠性验证和数据完整性验证。

终端软件平台下载和运行之前由启动加载软件在本地对其进行数据来源可靠性验证和数据完整性验证。

DCAS 用户端软件在下载和运行之前需要由终端软件平台对其进行数据来源可靠性验证和数据完整性验证。

具体技术要求见附录 A。

5.3.2 DCAS 用户端软件数据安全

DCAS 用户端软件在终端存储数据时，终端要保证数据存储的安全。

DCAS 用户端软件在存储关键数据时应该通过安全认证通道使用 HSM 提供的数据存储功能。

5.4 终端可信执行环境

DCAS 用户端可信应用软件运行在可信执行环境 TEE 中，TEE 基于可信安全硬件和可信安全软件为 DCAS 提供可信安全计算环境。

可信安全硬件通过安全内存访问控制、安全总线连接、安全中断、安全时钟、安全随机数和安全层级密钥处理模块等功能，提供可信的安全硬件环境。

可信安全软件包括安全 OS 和 TEE HAL，通过内存管理、安全时间、任务调度、中断、任务通信、加解密等功能，实现内存隔离、版本防回滚、安全存储、TApp 动态加载等功能，提供可信的安全软件环境。

6 总体要求

6.1 DCAS 前端系统要求

DCAS 前端应符合 GY/Z 175—2001 中第 6 章的要求，同时应支持以下功能：

a) 应支持通过单向信道实现 EMM 传输等功能；
b) 应能生成终端安全芯片和 HSM 的根密钥和层级密钥；
c) DCAS 前端生成的 EMM 和 ECM 应符合 DCAS 密钥机制的要求。

6.2 安全芯片密钥植入模块要求

安全芯片密钥植入模块负责将派生根密钥所需的 ChipID、ESCK、BL_KEY0 等必要信息植入终端安全芯片中，在安全芯片生产线上使用，其要求为：

a) 应支持 ChipID、ESCK、BL_KEY0 等必要信息的导入和导出；
b) 应支持安全存储敏感数据；
c) 应具有足够的安全性和防攻击能力。

6.3 DCAS 用户端软件要求

DCAS 用户端软件是可下载、可替换的终端软件模块，能够控制用户接收包括视频、音频和数据在内的数字广播服务,应支持以下功能：

a) 应实现和终端软件平台之间的标准应用程序接口；
b) 应通过与终端软件平交互实现条件接收功能；
c) 应支持终端 HSM 设备的使用；
d) 应具有足够的安全性和防攻击能力。

6.4 终端安全芯片要求

终端安全芯片是集成在接收终端中，实现层级密钥解密和码流处理的芯片，其要求为：

a) 应包含 OTP 区域，支持 Chip_ID、ESCK 等重要数据的安全存储；
b) 应支持根密钥 SCK 的还原；
c) 应支持通过派生方式生成最终根密钥 K3；
d) 应支持层级密钥机制，保证控制字在芯片内的安全传输；
e) 应包括音视频解码模块；
f) 应包括符合 DVB 标准的解扰模块；
g) 应支持启动模块签名验证功能；
h) 应支持可信执行环境；
i) 应支持以下算法：
   1) SM4 算法，遵循 GM/T 0002—2012 的要求；
   2) SM2 算法，遵循 GM/T 0003—2012 的要求；
   3) SM3 算法，遵循 GM/T 0004—2012 的要求。

6.5 硬件安全模块要求

硬件安全模块是集成在接收终端中,独立于终端安全芯片的安全芯片，用于提供算法工具和安全存储服务，其要求为：

a) 应包含随机数发生器，遵循 GM/T 0005—2012 的要求；
b) 应支持层级密钥机制；
c) 应参与控制字的运算；
d) 应支持数据签名和验证功能；
e) 应支持数据加解密功能；
f) 应支持安全认证通道功能；
g) 应包含可锁定的存储区域，该区域锁定后为只读区域；
h) 应包含 CA 存储区域，支持安全认证通道访问控制；
i) 应支持以下算法：
   1) SM4 算法，遵循 GM/T 0002—2012 的要求；
   2) SM2 算法，遵循 GM/T 0003—2012 的要求；

3) SM3 算法，遵循 GM/T 0004—2012 的要求。

6.6 终端软件平台要求

终端软件平台是在终端安全芯片硬件之上的公共软件，其要求为：

a) 应支持 DCAS 用户端软件的下载、更新和替换；
b) 应提供 DCAS 用户端软件所需的标准应用程序接口；
c) 应保证 DCAS 用户端软件下载、启动和运行过程中的完整性、可靠性和安全性；
d) 应支持可信执行环境。

6.7 安全要求

6.7.1 终端安全要求

DCAS 终端应遵循以下要求，从而有效地保护 DCAS 系统安全：

a) 应能保证本标准中涉及的密钥、证书和软件的保密性和完整性；
b) 应能保证解密内容不被输出、拦截、转发或复制；
c) 应能保证硬件的完整性,终端安全芯片和 HSM 不能轻易被移除或更换；
d) 应能保证软硬件接口不能被轻易破坏或规避。

6.7.2 终端安全芯片安全要求

6.7.2.1 OTP 区域

终端安全芯片的 OTP 区域用于存储 ChipID、ESCK 和 BL_KEY0 等信息，应符合以下要求：

a) OTP 区域完成写入后，其内容将不能更改；
b) SCK 是派生根密钥所需的必要安全信息，SCK 不能直接存储在 OTP 区域中；
c) OTP 区域中的安全信息应能被终端安全芯片中的安全校验机制检测、校验是否被篡改，发现安全信息被篡改后，DCAS 功能模块应立即停止工作。

6.7.2.2 SCK 还原函数

终端安全芯片的SCK还原函数应符合以下要求：

a) 应该是密码学上足够安全的单向函数；
b) 必须完全由内部硬件逻辑电路实现，外部软件无法截取 SCK 或 ESCK；
c) SCK 还原函数只能被根密钥派生模块使用。

6.7.2.3 根密钥派生模块

终端安全芯片的根密钥派生模块由 SCK 初步处理函数、厂商分离函数、最终根密钥派生函数等子模块组成。根密钥派生模块应符合以下要求：

a) 根密钥派生模块必须采用硬件逻辑电路或独立安全运算单元方式实现，使用专有的计算、存储资源进行独立工作，其他任何单元不能干涉派生模块的逻辑、运行和结果；
b) 根密钥派生模块运行过程的任何中间结果不应输出或被读取至外部模块；
c) SCK 初步处理函数应使用具有一定强度的单向函数，以保证在已知 Vendor_SysID 和 SCKv 的情况下，SCK 不能够被还原；
d) 厂商分离函数子应使用具有一定强度的单向函数，以保证在已知 Vendor_SysID 和 Seedv 的情况下，SMK 不能够被还原；

e) 最终根密钥派生函数应使用具有一定强度的单向函数，使得当知道根密钥 K3 和任一输入参数时，不能还原另一个输入参数，例如，当知道 K3 和 SCKv 时，不能还原 Seedv；
f) SMK 应不可被外部模块读取或篡改，且长度至少为 128 比特；
g) 终端安全芯片可同时支持多种最终根密钥派生函数。

6.7.2.4 层级密钥

终端安全芯片的层级密钥模块由多级密钥机制和挑战应答认证机制组成，层级密钥模块应符合以下要求：

a) 必须采用硬件逻辑电路或独立安全运算单元方式实现，使用专有的计算、存储资源进行独立工作；
b) 除解扰模块可以从层级密钥模块得到控制字之外，层级密钥模块的逻辑、运行和结果不能被其他任何模块干扰和使用；
c) 层级密钥运行过程的任何中间结果不应输出或被读取至外部模块；
d) 在挑战应答过程中，不能暴露层级密钥中的任何密钥。

6.7.3 HSM 安全要求

6.7.3.1 总体要求

HSM应符合以下要求：

a) HSM 具备安全机制，防止 HSM 软件被篡改；
b) HSM 应具备电压脉冲侦测、内部屏蔽罩等防物理攻击机制。

6.7.3.2 安全存储

HSM安全存储应符合以下要求：

a) 应具备 HSM 私钥防窃取功能，即 HSM 私钥应不能被外界获取；
b) 应具备内容防篡改机制，一旦内容写入锁定存储区域后，相应内容应不能被更改；
c) 应具备侦测安全机制，可侦测存储于锁定存储区内的内容是否被篡改，一旦发现相应内容被篡改，应立即使 HSM 停止工作；
d) 应具备反激活数据安全保护机制，在收到反激活命令时，首先应擦除所有 SAC 认证存储区域的内容，之后再迁入非激活状态；
e) 应具备抵御电子脉冲干扰功能，即存储区域的读写操作应能抵御电子脉冲干扰。

6.7.3.3 安全认证通道

HSM安全认证通道应符合以下要求：

a) 用于建立 SAC 的随机数应不能以明文的方式暴露在 SoC 的可信执行环境之外；
b) 应具备握手和数据传输的检测安全机制，在握手和数据传输过程中，一旦 HSM 在所收到的消息中检测到任何错误，HSM 应不做响应；
c) 应具备抵御重传攻击、中间人攻击的安全设计。

6.7.3.4 序列化

HSM的序列化应符合以下要求：

a) 在生产序列化过程中，HSM 私钥不得以明文形式存储于 HSM 之外；
b) 在生产序列化完成后，HSM 私钥不得以任何形式存储于 HSM 之外。

7 终端系统

7.1 终端系统架构

DCAS 终端系统架构见图 5，由硬件平台（包括终端安全芯片和硬件安全模块）、终端软件平台、DCAS 用户端软件及其他应用程序等终端软硬件模块组成。本标准仅规定与 DCAS 相关的终端安全芯片、HSM、DCAS 应用编程接口和 DCAS 用户端软件等终端软硬件模块。

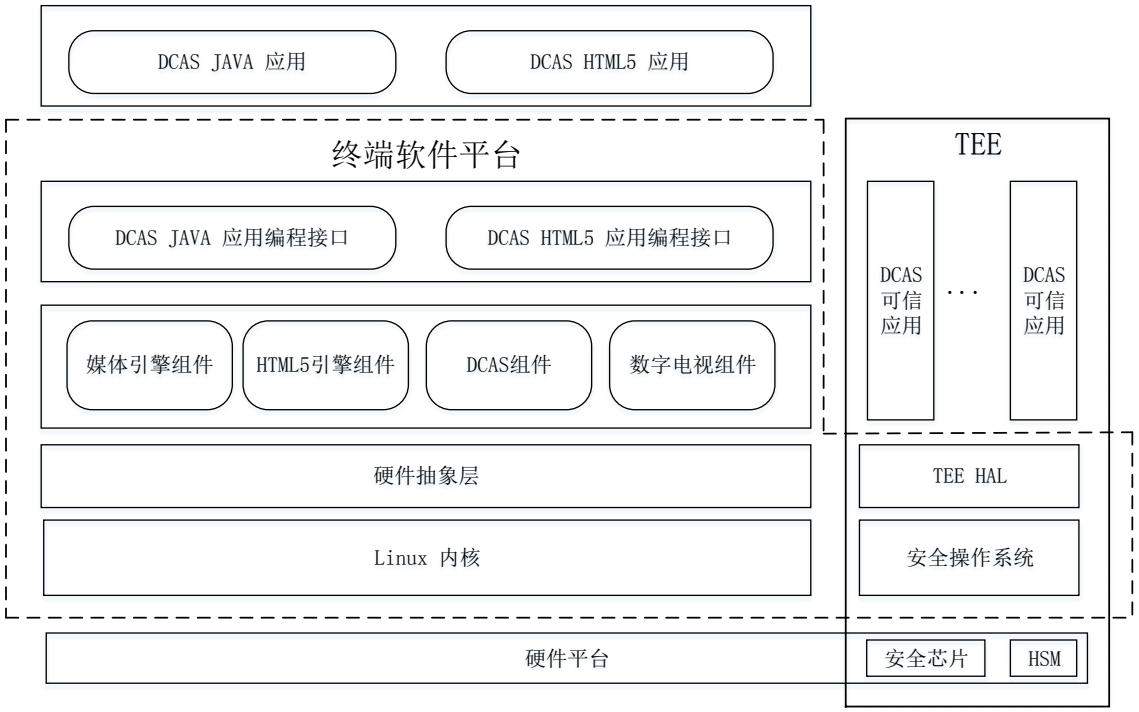

图5 DCAS 终端系统架构

a) 终端安全芯片

终端安全芯片提供层级密钥模块和根密钥生成模块保证终端传输数据安全性和条件接收系统的独立性，具体实现参见 7.3。

b) 硬件安全模块

硬件安全模块通过参与层级密钥处理，授权访问和安全存储等手段提供硬件层次的安全增强。具体实现参见 7.4。

c) 终端软件平台

终端软件平台提供支持 DCAS 用户端软件运行的应用编程接口。

终端软件平台可以是智能电视操作系统，也可以是基于 Linux 等操作系统的终端中间件。

d) DCAS 应用编程接口

DCAS 应用编程接口支撑 DCAS 管理器实现对 DCAS 用户端软件的管理，支撑 DCAS 用户端软件实现 ECM/EMM 等 CA 数据处理以及与 EPG 等应用之间的数据交互，其接口描述见附录 B。。

e) DCAS 用户端软件

DCAS 用户端软件实现对 ECM、EMM 等数据解析和处理，其接口描述见附录 B。

DCAS 用户端软件由 DCAS 用户端应用软件和 DCAS 可信应用软件构成，DCAS 用户端软件功能由 DCAS 用户端应用软件和 DCAS 用户端可信应用软件协同实现。

DCAS 用户端软件可以下载到终端软件平台上，与其他终端软件平台应用并存。

7.2 DCAS 应用编程接口

DCAS 应用编程接口实现 DCAS 用户端软件与终端软件平台之间的数据交互。提供一般的 Java、Javascript 接口和安全环境接口，具体实现方式见附录 B。

一般接口包括：

a) 过滤接口。DCAS 用户端软件通过调用过滤接口实现对 ECM、EMM 及 CAT 的接收。

b) DCAS 管理接口。DCAS 用户端软件通过 DCAS 管理接口实现向终端软件平台的注册，并接收来自终端软件平台 DCAS 管理器的解扰请求。

安全环境接口包括：

a) HSM 接口。DCAS 用户端软件通过调用 HSM 接口实现对安全存储区的访问及安全控制字的生成。

b) 层级密钥接口。DCAS 用户端软件通过调用层级密钥接口将加密的密钥载入终端安全芯片完成对码流的解扰。

c) 辅助信息接口。DCAS 用户端软件通过该接口获取终端实时位置信息等辅助信息。

d) 其他 GP 扩展接口。DCAS 用户端软件通过该接口实现加解密、签名校验、内存管理和调试打印等功能。

7.3 终端安全芯片

7.3.1 终端安全芯片工作流程

终端安全芯片功能见图6，终端安全芯片包含：OTP、根密钥派生、层级密钥和解扰及解码等模块。

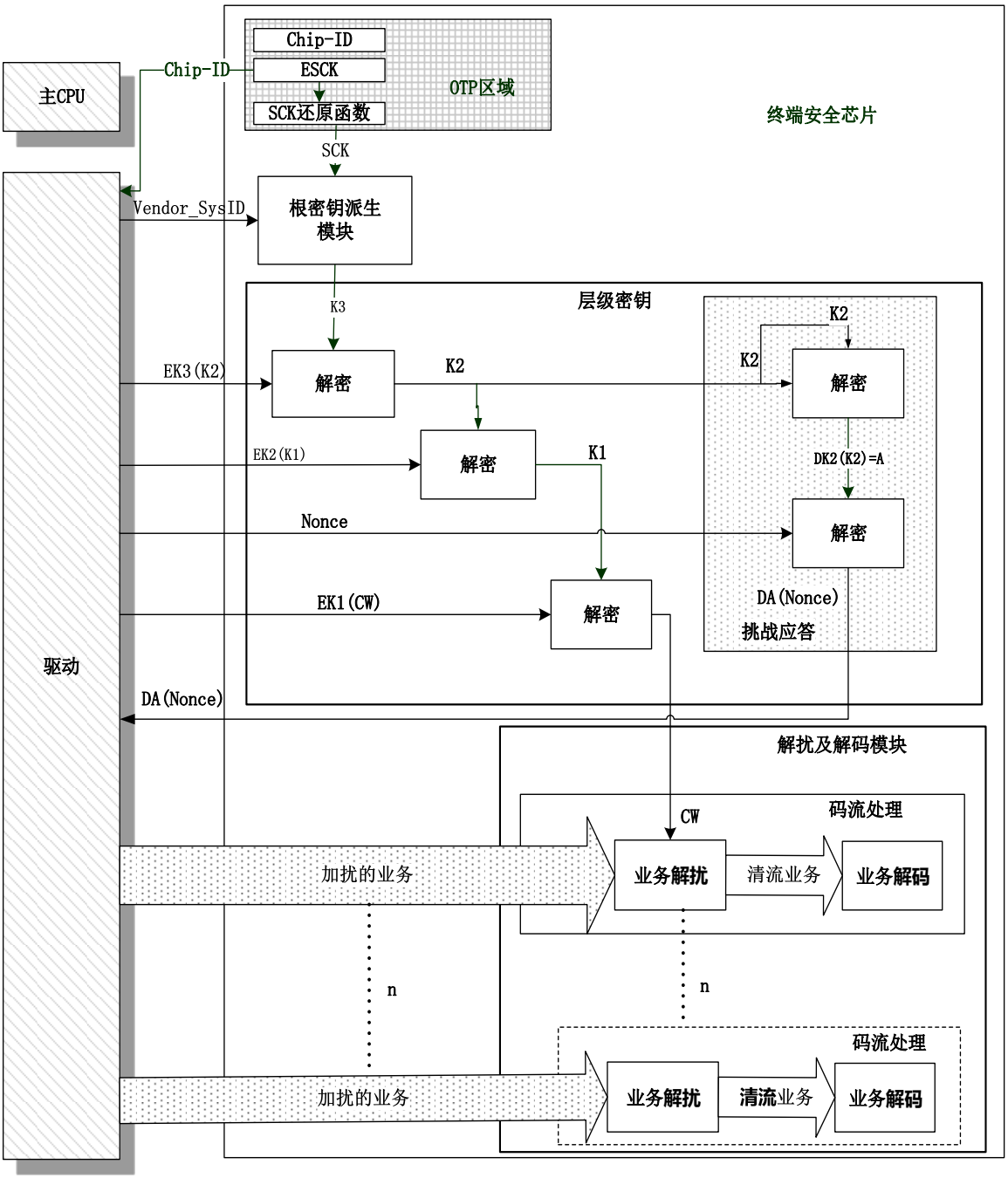

图6　终端安全芯片功能框图

终端安全芯片通过根密钥派生模块生成根密钥 K3，通过层级密钥模块保障控制字和其他密钥的安全传输及终端安全芯片的合法性。

终端安全芯片的功能不能由主 CPU 实现。

终端安全芯片工作流程：芯片上电后，OTP 通过内置的 ESCK 及 SCK 还原函数生成 SCK,并提供给根密钥派生模块,用于生成根密钥 K3。层级密钥模块接收根密钥 K3 用于相关密钥的解密及挑战应答流程。层级密钥模块包含两个方面的功能：对输入的加密密钥实现分层解密；处理挑战信息（Nonce），并生成应答响应（DA（Nonce））。最终解密获得的 CW 被送至解扰及解码模块进行业务解扰和解码。

7.3.2　根密钥派生模块

根密钥派生模块由一组硬件逻辑模块构成，通过其中嵌入的派生机制，并结合输入参数来实现根密钥的派生。任何一家条件接收厂商均通过此派生机制生成不同的根密钥。此方式规避了嵌入单一根密钥带来的安全风险。

根密钥派生模块功能见图 7，根密钥派生模块包括：SCK 初步处理函数、厂商分离函数和最终根密钥派生函数。根据外部输入的 SCK 和 Vendor_SysID，派生出某一条件接收厂商的根密钥。

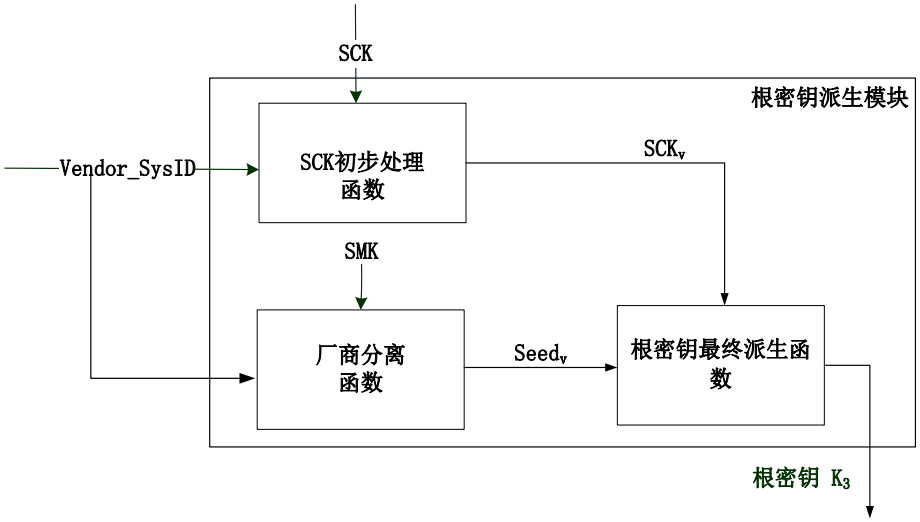

图7　根密钥派生模块功能框图

SCK 初步处理函数模块基于 SCK 和输入的 Vendor_SysID，生成 SCKv。该函数中应使用具有一定强度的密码算法，此强度应保证在已知 Vendor_SysID 和 SCKv 的情况下，SCK 不能够被还原。SCK 初步处理函数应由终端安全芯片厂商提供并由安全数据管理平台认证。

厂商分离函数使用 Vendor_SysID 和 SMK 作为输入值，生成 Seedv。该函数中应使用具有一定强度的密码算法，此强度应保证在已知 Vendor_SysID 和 Seedv 的情况下，SMK 不能够被还原。厂商分离函数应由终端安全芯片厂商提供并由安全数据管理平台认证。

最终根密钥派生函数依据 SCKv 和 Seedv 的输入值派生出根密钥 K3。在实施过程中，最终根密钥派生函数应采用单向函数，使得当知道 K3 和任一输入参数时，不能还原另一个输入参数。例如，当知道 K3 和 SCKv 时，不能还原 Seedv。最终根密钥派生函数应由终端安全芯片厂商提供并由安全数据管理平台认证。

——Vendor_SysID：用于标识条件接收系统，由安全数据管理平台分配，长度为 2 字节

——SCKv：16 字节

——SMK：16 字节，芯片厂商提供的门级数据

——Seedv：16 字节

——终端安全芯片可同时支持多种最终根密钥派生函数。

7.3.3　层级密钥

7.3.3.1　三层密钥机制

层级密钥模块功能框图见图8。

本标准规定的终端安全芯片应支持三层密钥机制，更多层级的密钥机制，其安全要求和技术细节不在本标准的描述范围内。

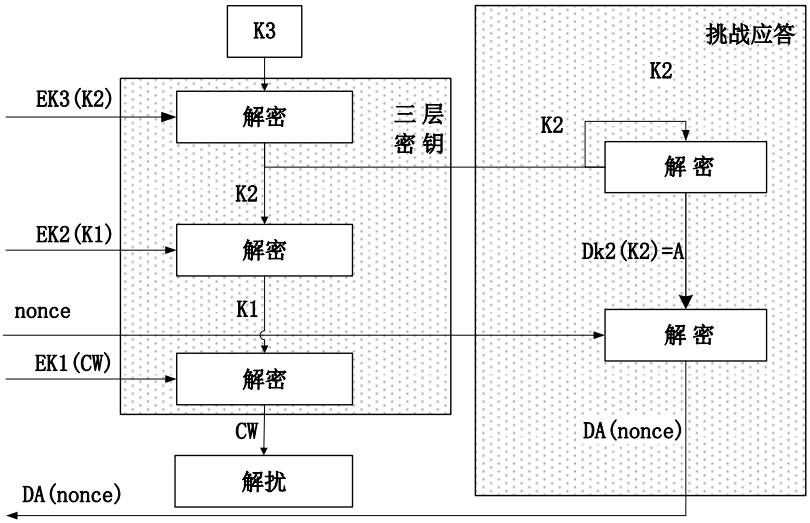

图8　层级密钥模块功能框图

三层密钥机制确保控制字在终端的传输安全。

三层密钥机制通过从根密钥派生模块获得的根密钥 K3，依次解密 EK3(K2)、EK2(K1)、EK1(CW)获得解扰所需的控制字；同时 K2 配合挑战信息（Nonce）生成响应应答。

终端安全芯片应遵循以下流程解密加扰业务：

a)　应接收密文 EK3(K2)，使用 K3 解密该密文，并生成 K2；

b)　应接收密文 EK2(K1)，使用 K2 解密该密文，并生成 K1；

c)　应接收密文 EK1(CW)，使用 K1 解密该密文，并生成 CW；

d)　CW 用于解密加扰业务。

EK3(K2)表示用密钥 K3 加密的数据 K2。

EK2(K1)表示用密钥 K2 加密的数据 K1。

EK1(CW/Key)表示用密钥 K1 加密的数据 CW。

K3 是派生根密钥，长度 16 字节。

K2 是用于解密 K1 的密钥，长度为 16 字节。

K1 是用于解密 CW 的密钥，长度为 16 字节。

CW 是用于解扰业务的密钥，长度为 8 或 16 字节。

层级密钥机制应支持 SM4 算法，使用 128 比特，模式为 ECB。

7.3.3.2　挑战应答

终端安全芯片应支持挑战应答机制，该机制可以被用于一些安全功能。

挑战应答机制应遵循以下流程：

a)　终端安全芯片应通过 API 驱动接口接收 Vendor_SysID、EK3(K2)和 Nonce。

b)　终端安全芯片利用派生的 K3 解密 EK3（K2）得到 K2。

c)　终端安全芯片应通过 K2 解密 K2 生成 DK2(K2),记为 A。

d)　终端安全芯片应使用 A 解密 Nonce，生成 DA(Nonce)。

e)　终端安全芯片通过 API 驱动接口返回 DA(Nonce)。

DK2(K2)表示用用密钥 K2 解密 K2 的过程。

A 表示 DK2(K2)的结果，长度为 16 个字节。

Nonce 挑战信息，长度为 16 字节。

DA(Nonce)表示用 A 解密随机数 Nonce 得到的结果。

7.3.4 OTP 区域

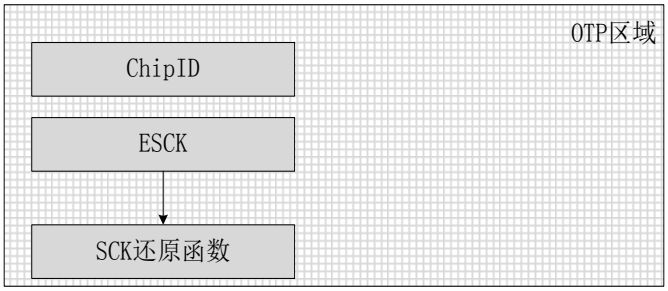

图9 OTP 区域功能框图

OTP 区域功能框图见图 9，用于存储 ChipID、ESCK 和 SCK 还原函数等信息。SCK 还原函数逻辑电路读取 OTP 区域中的 ESCK，将 ESCK 还原成 SCK，并将 SCK 提供给根密钥派生模块。

SCK 是派生根密钥所需的必要安全信息，SCK 不能直接存储在 OTP 区域中。

a) SCK 是终端安全芯片密钥，每芯片唯一，长度为 16 字节，由安全数据管理平台生成。

b) ESCK 是加密后的 SCK，由安全数据管理平台提供，存储于 OTP 区域，长度等于 SCK 长度。

c) ChipID 是终端安全芯片的公开标识符，包括了表明芯片厂商和型号的信息，以及芯片的一个全球唯一标识符，长度 8 字节，由安全数据管理平台分配，数据格式如表 1 所示。

表1 ChipID 数据格式

数据名称	长度 比特	数据类型
芯片厂商 ID（Chip Manufacturer ID）	8	uimsbf
芯片类型（Chip Type）	12	uimsbf
保留（Reserved）	12	uimsbf
芯片标识	32	uimsbf

芯片厂商 ID：芯片厂商的唯一标识，8 比特。

芯片类型：某一芯片厂商生产的芯片型号标识，由安全数据管理平台分配，12 比特。

保留位：12 比特。

芯片标识：某一芯片制造厂商生产的某一芯片类型芯片的序列标识，32 比特，此字段对任意芯片全球唯一，无论厂商和类型是否相同。

7.4 硬件安全模块

7.4.1 HSM 架构

HSM是单向可下载条件接收系统的一个核心功能组件，参与控制字的运算，可以用作代替传统条件接收系统终端的智能卡硬件；与作为各CA厂商高度私有化软硬件实体的智能卡不同，HSM是一种标准化的安全部件，可在不同的CA厂商之间共享。

HSM的基本架构见图10，HSM包含：层级密钥处理模块、算法工具和安全存储区域等模块，可与终端安全芯片建立安全认证通道，保证数据传输的安全性。

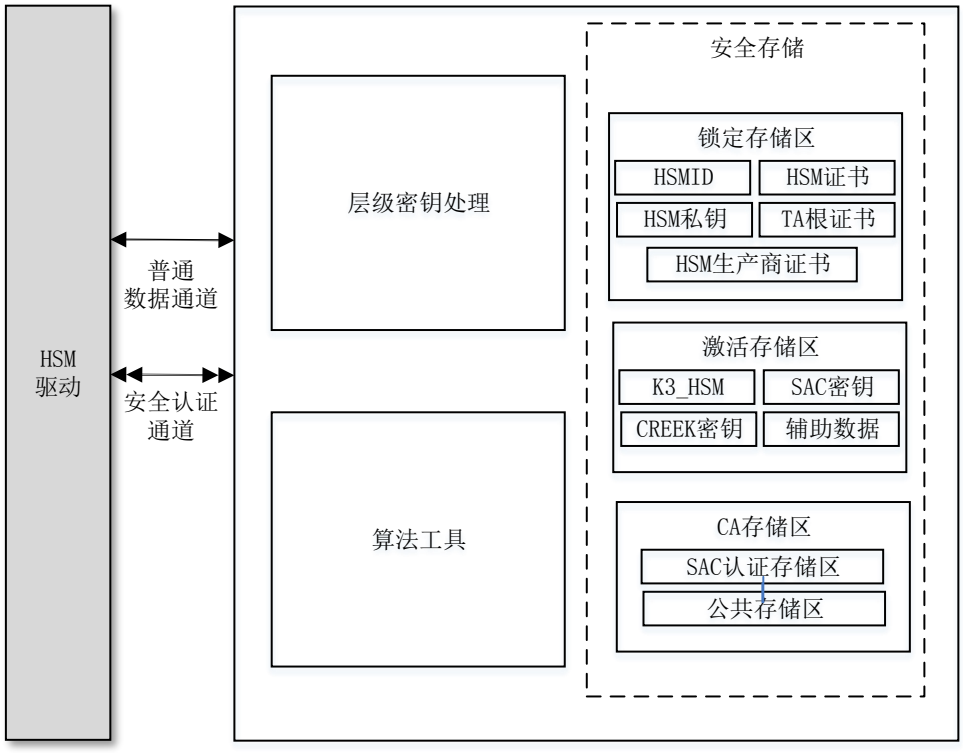

图10　HSM 基本架构

7.4.2　HSM 的激活

HSM在被CA使用前需要进行激活，非激活状态下HSM的功能将被限制，在收到前端发出的激活消息并完成激活后，HSM的安全功能才能被使用。激活的基本流程如图11所示。

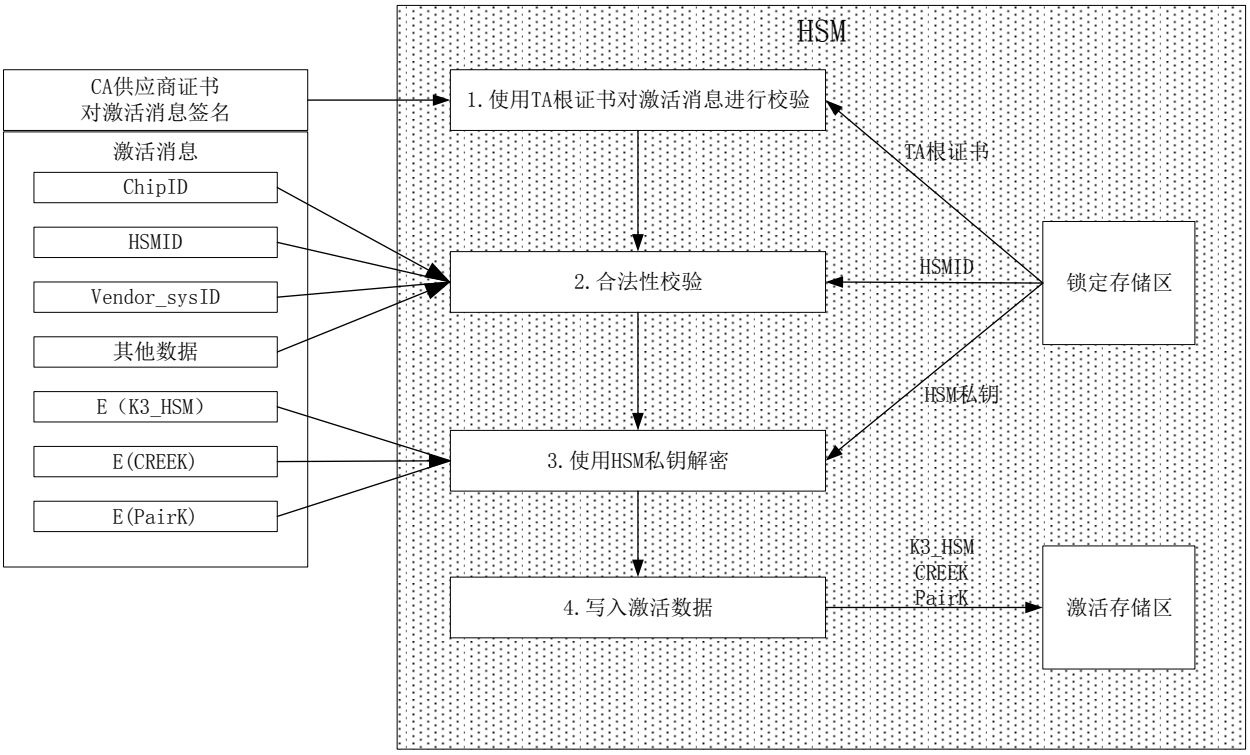

图11　HSM 基本激活流程

激活消息被CA厂商证书签名。HSM收到激活消息后，首先，HSM使用TA根证书对激活消息进行校验，然后对ChipID、HSMID、Vendor_sysID和版本号、时间戳等其他数据进行合法性校验，校验通过后，使用HSM私钥对加密的K3_HSM等密钥进行解密，将密钥存储至激活存储区。

HSM激活的详细流程和要求见附录C。

7.4.3　层级密钥处理模块

层级密钥处理是HSM的核心功能，使用K3_HSM和CREEK对前端发送来的层级密钥数据进行解密和重加密，生成参与终端安全芯片控制字运算的层级密钥数据。

典型的HSM层级密钥机制使用三层对称加解密算法，其访问通过运行在TEE环境中的HSM驱动接口进行，输出结果通过SAC返回给运行在TEE环境中的DCAS客户端可信应用软件，最终送至终端安全芯片。处理流程如图12所示。

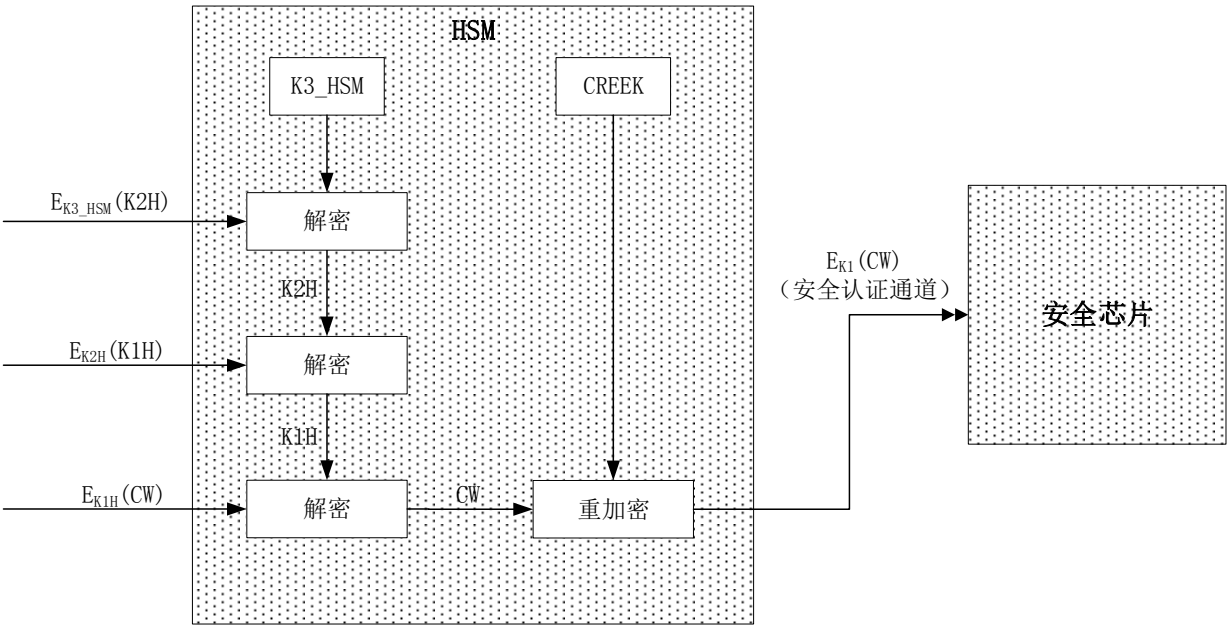

图12　HSM 层级秘钥处理流程

三层密钥机制确保控制字在终端的传输安全，层级密钥处理模块进行的相关操作，都必须通过 SAC 进行调用，即 SAC 建立之前，层级密钥处理模块是不可用的。

三层密钥机制通过从锁定存储区获得的根密钥 K3_HSM，依次解密 E_{K3_HSM}(K2H)、E_{K2H}(K1H)、E_{K1H}(CW)获得解扰所需的控制字,使用 CREEK 对 CW 进行重加密。

终端安全芯片应遵循以下流程解密：

a)　应接收密文 E_{K3_HSM}(K2H)，使用 K3_HSM 解密该密文，并生成 K2H；

b)　应接收密文 E_{K2H}(K1H)，使用 K2H 解密该密文，并生成 K1H；

c)　应接收密文 E_{K1H}(CW)，使用 K1H 解密该密文，并生成 CW；

d)　使用 CREEK 加密 CW，生成用于加扰业务 EK1(CW)。

E_{K3_HSM}(K2H)表示用密钥 K3_HSM 加密的数据 K2H。

E_{K2H}(K1H)表示用密钥 K2H 加密的数据 K1H。

E_{K1H}(CW)表示用密钥 K1H 加密的数据 CW。

K3_HSM 是激活下发的密钥，长度 16 字节。

K2H 是用于解密 K1H 的密钥，长度为 16 字节。

K1H 是用于解密 CW 的密钥，长度为 16 字节。

CREEK 是用于重加密的密钥，长度为 16 字节。

层级密钥机制应支持 SM4 算法，使用长度为 128 比特的数据块，模式为 ECB。

7.4.4　算法工具

HSM内置了加解密、签名和验证等可共享的算法工具，CA厂商可以利用这些算法工具开发和实现其安全解决方案，提高在共享平台下的安全性。

HSM算法工具的具体实现本标准不作定义。

7.4.5　安全存储区

HSM的安全存储区由锁定区、激活存储区和CA存储区组成。

7.4.5.1 锁定存储区

HSM内NVM某些区域的具有锁定机制，写入数据后该区域应被锁定，锁定区域成为只读区域，不可再被修改或删除。

锁定区域的数据包括：TA根证书、HSMID、HSM芯片证书、HSM厂商证书以及HSM私钥数据。

TA根证书是安全数据管理平台颁发的根证书，是自签名证书，HSM厂商证书证书是TA根证书的下级证书，HSM芯片证书是HSM厂商证书证书的下级证书。HSM芯片证书中包含HSM公钥，HSM私钥也存储在锁定存储区中，如表2所示，证书格式见附录C。

表2 锁定存储区数据

密钥名称	长度 字节	类型	写入方式	功能描述
HSM 私钥	32	SM2 私钥	生产序列化	用于签名和加密
HSM 证书	可变长度	SM2 公钥及签名	生产序列化	用于 HSM 的合法身份识别与校验
TA 根证书	可变长度	SM2 公钥及签名	固定硬编码	用于其他证书的校验

HSMID 是硬件安全模块芯片的公开标识符，包括了表明硬件安全模块芯片厂商和型号的信息，以及芯片的全球唯一标识符，长度 8 字节，数据格式如表 3 所示。

表3 HSMID 数据格式

数据名称	长度 比特	数据类型
HSM 芯片厂商 ID（HSM Chip Manufacturer ID）	8	uimsbf
HSM 芯片类型（HSM Chip Type）	6	uimsbf
国密算法标识	1	uimsbf
HSM 芯片厂商自定义数据标识	5	uimsbf
保留（Reserved）	12	uimsbf
HSM 芯片标识	32	uimsbf

HSM芯片厂商ID：芯片厂商的唯一标识，8比特。

HSM芯片类型：某一芯片厂商生产的芯片型号标识，6比特。

国密算法标识：标志芯片是否支持国密算法,1表示支持国密算法；0表示不支持国密算法，1比特。

HSM芯片厂商自定义数据标识：5比特，内容和用法由HSM厂商自行定义。

保留位：此数据为全“0”，12比特。

HSM芯片标识：某一HSM芯片制造厂商生产的某一HSM芯片类型芯片的序列标识，32比特，此字段对任意芯片全球唯一，无论厂商和类型是否相同。

7.4.5.2 激活存储区

激活存储区是存储激活数据和激活相关数据的区域，包括：K3_HSM、CREEK、PairK和辅助数据等，如表4所示。

表4　激活存储区数据

密钥名称	长度 字节	类型	写入方式	功能描述
SAC 配对密钥(PairK)	16	对称密钥	HSM 激活	用以建立安全认证通道
加解密引擎根密钥（K3_HSM）	16	对称密钥	HSM 激活	用于层级密钥运算

7.4.5.3　CA 存储区

CA存储区是存储CA私有数据的区域，分为SAC认证存储区和公共存储区两个部分。

SAC认证存储区通过SAC访问，读写必须通过SAC进行，支持对任意数据在指定偏移地址的写入和读取。

公共存储区写入操作必须通过SAC进行，读出操作可以不通过SAC进行。

7.4.6　安全通道

安全认证通道是建立在HSM与SoC之间的安全数据通路，只能在TEE下使用。安全认证通道的建立依赖于HSM芯片的激活，即只有HSM激活后，SAC才可以建立成功。安全认证通道的使用包括握手认证和数据通信两个阶段。

安全认证通道的要求见附录C。

附 录 A
（规范性附录）
DCAS 用户端软件下载与启动加载安全机制

A.1 信任链基本原则

DCAS用户端软件启动加载安全机制基于一种自底向上的安全信任链。

该安全信任链采用数字签名技术建立，自底向上包括终端安全芯片、启动加载软件、终端软件平台和DCAS用户端软件。

DCAS用户端软件启动加载安全机制要求信任链中的每个环节都必须按照自底向上的次序进行签名校验，只有在当前环节的签名校验通过以后，方可启动信任链下一环节的安全校验。只有当信任链中所有环节的签名校验全部通过后，DCAS用户端软件方可启动加载。

在整个信任链机制中：

a) 终端安全芯片 OTP 区域中应预置用于验证启动加载软件的验证密钥；

b) 信任链中每个环节软件都应内置用于校验下一环节软件的验证密钥；

c) 信任链中每个环节的软件都应被对应于验证密钥的私钥数字签名；

d) 每个环节软件都应包含其相应的数字签名信息；

e) 当前环节软件应在完成对下一环节软件安全验证后，才能启动加载下一环节软件。

DCAS 用户端软件的下载、加载和运行应始终遵循上述安全信任链机制。

A.2 启动签名验证

CPU应执行一定的安全代码，检查在CPU外部Flash中已签名的启动加载软件。启动加载软件则需要验证已签名的终端软件平台，终端软件平台需要验证已签名的DCAS用户端软件。

启动加载软件验证过程如图A.1所示。

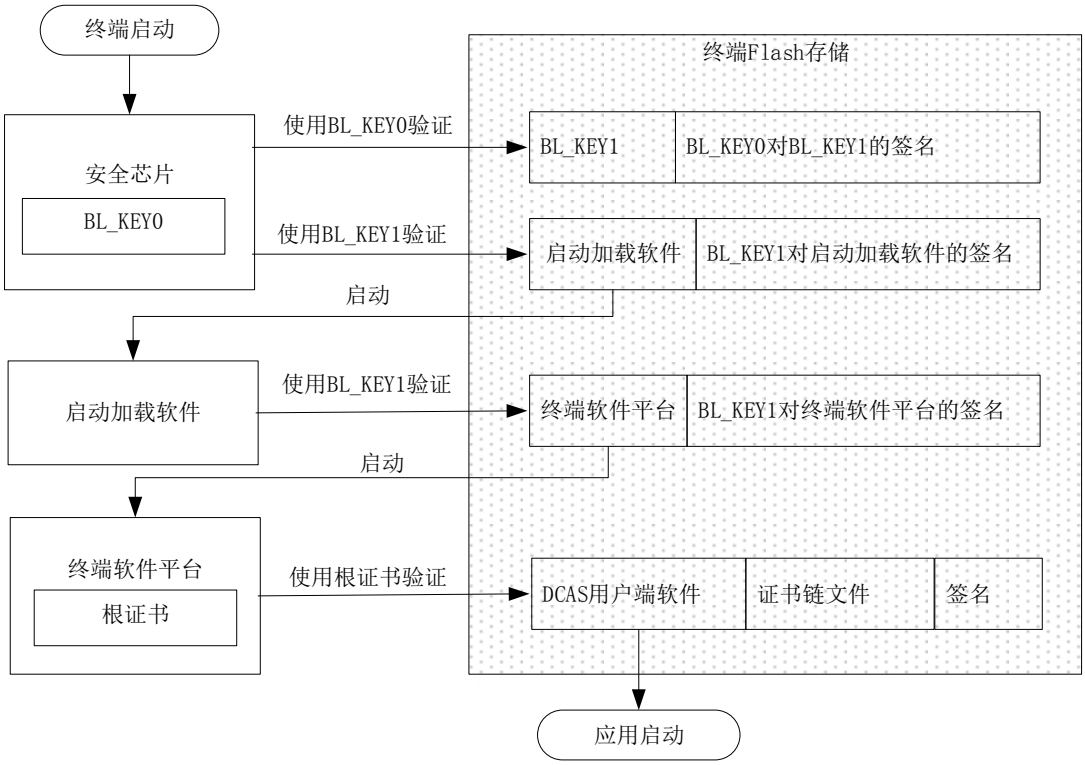

图 A.1　启动加载软件验证过程

终端安全芯片应在芯片安全区域中嵌入由安全数据管理平台分配的公钥，这里记做 BL_KEY0。安全数据管理平台在任何情况下不应泄露 BL_KEY0 的私钥。图 A.1 中的 BL_KEY0 指其公钥。

终端安全芯片在启动时应执行其安全区域中的代码从终端读取额外的公钥，这里记做 BL_KEY1，以及 BL_KEY1 的签名，并应使用 BL_KEY0 验证 BL_KEY1 的签名是否正确。图 A.1 中的 BL_KEY1 指其公钥。

在成功验证 BL_KEY1 后，终端安全芯片应使用 BL_KEY1 对启动加载软件进行验证。

启动加载软件应使用 BL_KEY1 对终端主软件的签名进行验证。

主软件应使用其内嵌的应用程序根证书对 DCAS 用户端软件进行验证。

上述过程提到的所有签名使用的算法包括 SM3 杂凑算法和 SM2 算法。

A.3　DCAS用户端软件下载替换

终端平台软件应可通过启动加载软件升级或应用下载的方式替换DCAS用户端软件，包括DCAS用户端应用软件和DCAS用户端可信应用软件，下载替换流程如下：

a)　前端系统通知终端软件平台启动DCAS下载替换；

b)　启动加载软件或应用管理器下载DCAS用户端软件；

c)　启动加载软件或应用管理器对下载的DCAS用户端软件进行签名验证；

d)　启动加载软件或应用管理器替换验证成功后DCAS用户端软件；

e)　终端重新启动系统；

f)　前端系统通知替换后的DCAS用户端软件启动HSM再激活流程，详见附录C.3.6；

g） HSM被再激活后，替换后的DCAS用户端软件应可与HSM协同工作，DCAS用户端软件完成下载替换。

A.4 密钥管理

启动加载软件密钥说明见表 A.1。

表 A.1 启动加载软件密钥说明

密钥名称	密钥属主	被签名	用于签名	备注
BL_KEY0	安全数据管理平台	N/A	BL_KEY1	公钥被嵌入芯片
BL_KEY1	运营商	BL_KEY0	启动加载软件 终端软件平台	

对于 BL_KEY0：

安全数据管理平台应管理其内部 BL_KEY0 的数据库，应负责分发 BL_KEY0 至芯片厂商，用于嵌入指定的芯片组。

对于 BL_KEY1：

BL_KEY1 所有者是运营商。为了保证终端软件可以被 BL_KEY1 正确签名，BL_KEY1 所有者可自行签名或者向其他可信机构进行密钥托管。

安全数据管理平台应提供使用 BL_KEY0 签名 BL_KEY1 的方法和过程。BL_KEY1 所有者应向安全数据管理平台提供终端的具体信息，这些信息包括芯片型号和 BL_KEY1 的公钥。安全数据应将 BL_KEY1 的公钥使用 BL_KEY0 的私钥签名，并将签名结果连同 BL_KEY1 的公钥一并返还给 BL_KEY1 所有者，用于将 BL_KEY1 的公钥及签名结果置入终端设备。

A.5 启动加载软件安全要求

启动加载软件安全要求主要针对启动过程和下载过程两个方面。芯片应支持从 RAM 中启动功能（BFR）。应满足信任链机制，所有由启动加载软件装入，启动，下载，更新的软件组件应被签名和验证。启动加载软件只负责验证平台软件，由平台软件下载的应用软件应由平台软件验证。

对于启动过程：

启动加载软件应在成功验证后续软件组件签名后，开始执行其代码。启动加载软件应在每一次后续软件组件重启后重新验证其签名。只有存储在终端 Flash 中的软件组件可以被执行。所有软件组件的签名验证和执行应在 RAM 中进行。验证后的软件组件，如有需要可以进行解压缩然后运行。

对于下载过程：

启动加载软件只有在 RAM 中验证下载的软件镜像签名正确后，方可将所下载的软件及其签名写入 Flash。软件下载成功后，终端应立即执行一次完全重启。如果所下载的软件超过启动加载软件分配的 Flash 最大尺寸，启动加载软件应拒绝该软件，并执行重启。应避免降级可升级软件组件。

A.6 启动加载软件和终端安全芯片的性能要求

为了保证终端的用户体验，应采用终端安全芯片为杂凑算法提供硬件加速。

附 录 B
（规范性附录）
DCAS 应用编程接口

B.1 Java应用程序接口

标准的 Java 虚拟机方案已在业界被广泛用于下载和启动应用。在此之上，增加对于 DCAS 基本实现的支持，可提供 DCAS 用户端应用的运行环境，如图 B.1 所示。

DCAS 用户端软件是一个运行于支持 Java 运行环境终端软件平台上的 Xlet 应用;运行已下载的 DCAS 用户端软件，可实现对业务解扰。

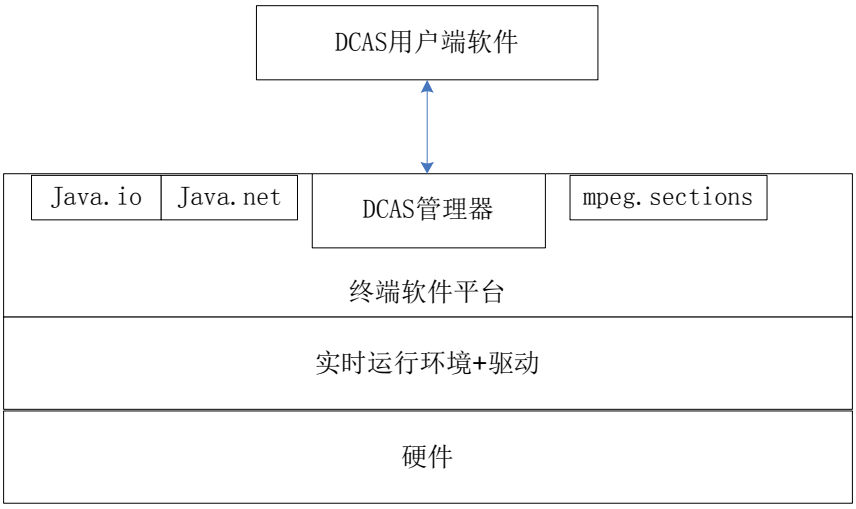

图 B.1 DCAS JAVA 应用程序接口示意图

B.1.1 应用程序接口类型

B.1.1.1 CAS管理器API

终端软件平台定义了 CA 模块管理器（DCAS Manager），来实现管理针对业务解扰的请求。

DCAS Manager 包括终端软件平台上层 API、终端软件平台底层 API、扩展应用 API。

B.1.1.1.1 终端软件平台上层API

终端软件平台上层 API 定义了 CA 模块管理器，来实现管理针对业务解扰的请求（也就是解扰视音频流）。一个 DCAS 用户端软件应用必须将 CA 模块在 CA 模块管理器中注册，用来接收网络中来自终端设备终端软件平台发出的解扰请求。

DCAS用户端软件需要终端软件平台实现DCAS终端软件平台上层API。

B.1.1.1.2 终端软件平台底层API

DCAS用户端软件需要终端软件平台实现以下类型的API集，其中除了现有Java API，还包括对于终端安全芯片访问所需的扩展API。

B.1.1.1.3 扩展应用API

通过扩展DCAS应用API，终端软件平台上的CAS管理模块可以同Java应用进行最基本的CA信息传输，而不受在Java应用及DCAS应用之间使用IXC的限制。

DCAS用户端软件需要终端软件平台为应用实现DCAS扩展API。

B.1.1.1.4 可分离安全设备API

通过此API，DCAS应用可以和可分离安全设备通讯。

B.1.1.2 网络API

DCAS 用户端软件可以使用 Java 网络 API 来访问网络资源，比如与前端 CA 服务器的互联。

DCAS 用户端软件需要终端软件平台根据 Java.net 中的定义实现现有 Java 网络 API。

B.1.1.3 MPEG Section过滤API

DCAS 用户端软件使用 MPEG Section 过滤 API 来载入用于 CA 的 MPEG Section。CA 相关的数据可包括 ECM，EMM 及 CAT 表。

DCAS 用户端软件需要终端软件平台根据 org.davic.mpeg.sections ，org.davic.mpeg.TransportStream 和 org.davic.net.tuning.NetworkInterface 中的定义实现 MPEG Section 过滤 API。

B.1.1.4 永久存储API

DCAS 用户端软件可以使用现有的 Java API 来访问终端存储，包括将数据保存在非易失性存储器中。

DCAS 用户端软件需要终端软件平台根据 java.io 的定义来实现永久存储 API。

DCAS 用户端软件可以在永久存储的文件系统中通过使用特定的目录来保存数据。终端软件平台需要提供适当的函数来读取该存储文件系统的根目录名称。

B.1.2 应用接口调用时序

本条提供了使用 DCAS API 的两种情况：CASModule 注册和频道选择切换，CASModule 注册如图 B.2 所示，频道选择切换如图 B.3 所示。

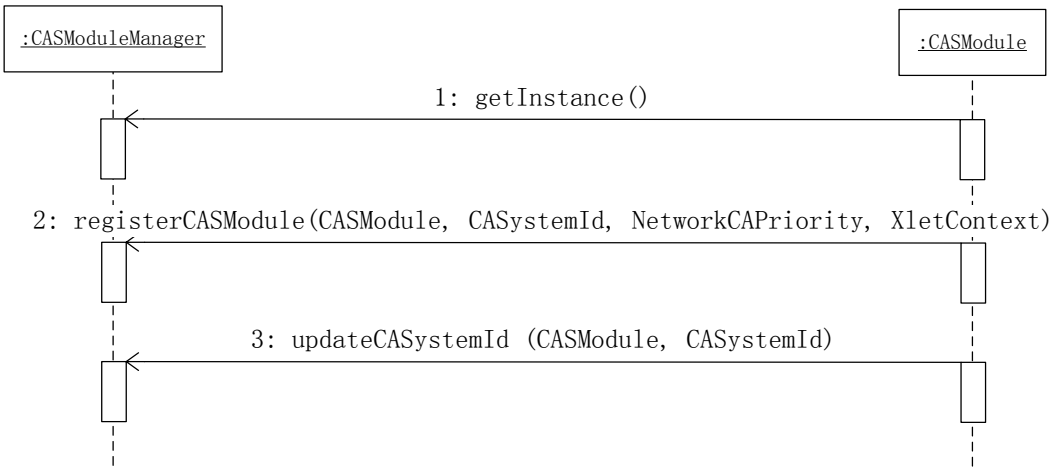

图 B.2　CA 模块在 CASModuleManager 中的注册

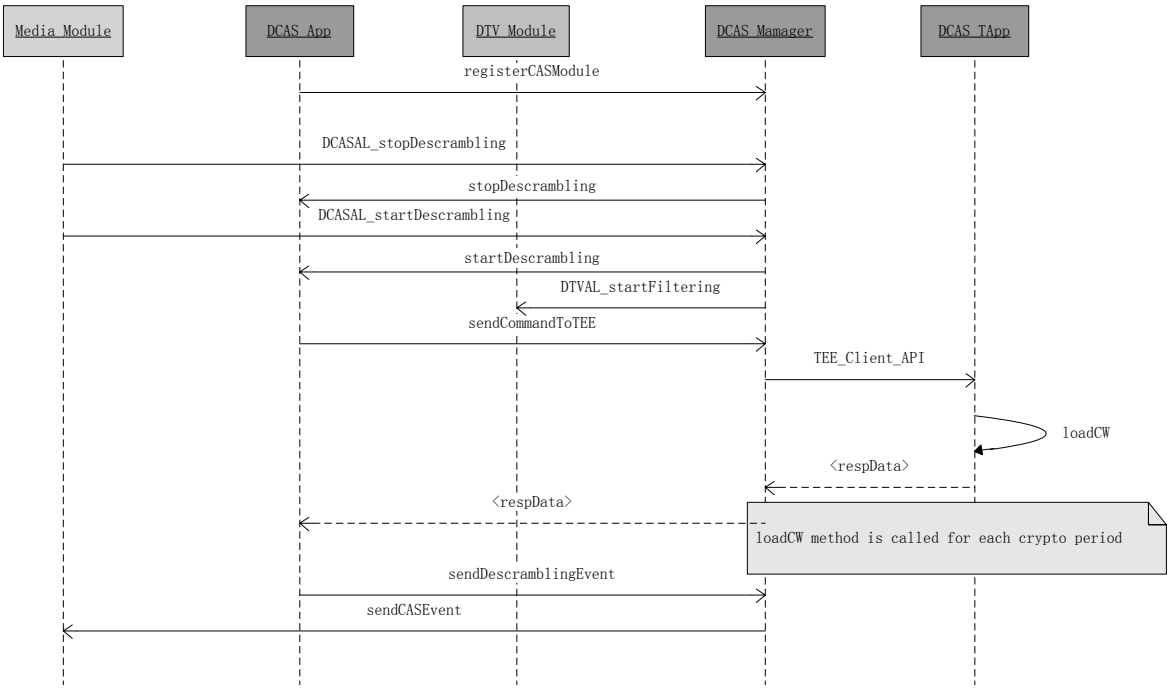

图 B.3　频道选择切换

B.1.3　应用程序接口描述

应用程序接口名称见表 B.1。

表 B.1　应用程序接口

API 名称	包名称
终端软件平台上层 API	org.ngb.net.cas.module
终端软件平台底层 API	org.ngb.net.cas.controller

表 B.1（续）

API 名称	包名称
应用扩展 API	org.ngb.net.cas.event
可分离安全设备 API	org.ngb.net.cas.detachable
网络 API	java.net
Section 过滤 API	org.davic.mpeg.sections
永久性存储 API	java.io

B.1.4 org.ngb.net.cas.module包

org.ngb.net.cas.module包提供了DCAS终端软件平台上层API，TVOS需实现此包。

Org.ngb.net.cas.module包概要见表B.2。

表 B.2 org.ngb.net.cas.module 包概要

接口	
CASModule	用于表示请求解扰一组基本流的CASModule对象。
CASDataUtils	用来获取和设置CA信息，读写DCAS数据。
CADescriptor	提供了CA描述符的信息，给定业务的PMT中的可能提供CA描述符。此外CAT中也会有CA描述符的出现。
CAServiceComponentInfo	用于提取特定CA业务成分的信息，如ECM PID和用于装载控制字的DescramblerContext。
CASPacketListener	DCAS应用通过本接口来接收带外CAS Packets (如 EMMs)。
CASSession	提供CAS会话相关信息。
CAStatus	DCAS应用在每次DescramblerContext中的解扰状态发生变化时发送CAStatus，本状态用来指示解扰成功与否。如果所解扰成分中任何一个发生解扰失败，本状态必须汇报整个业务的解扰请求失败。当终端软件平台收到一个新的CAStatus,它应通过本段描述的CAS事件通知其他应用。
CATListener	DCAS应用需实现该接口，使用CAT中的CA描述符来过滤带内EMM。
CATNotifier	DCAS应用使用本方法注册用于获取CAT更新通知的监听器。
类	
CASModuleManager	用来注册所有被DCAS应用实现的CASModule。
CASPermission	任一DCAS应用必须获取CASPersmission方可访问CASModuleManager，本机制用于确保只有被网络运营商授权的DCAS应用方可使用DCAS API。

B.1.4.1 接口org.ngb.net.cas.module.CASModule

B.1.4.1.1 方法

B.1.4.1.1.1 startDescrambling

原型：public void startDescrambling(CASSessioncasSession,

CAServiceComponentInfo[]casci)

描述：本方法由终端软件平台调用，请求 CASModule 解扰给定会话中的一组基本流。

DCAS 应用从 CAS 会话中可以得到相关 NetworkInterface 对象。

从 NetworkInterface 中，可以获取当前 TransportStream 对象，用于 org.davic.mpeg.sectionsAPI 进行 ECM section 过滤。

参数：

Cassession:请求解扰操作的会话

casci：CA 业务组件信息数组。该数组可以用于获取相关 ECM PID,以及用以 DCAS 装载控制字的 DescramblerContext 对象

返回：

无

B.1.4.1.1.2 updateDescrambling

原型：public void updateDescrambling(CASSessioncasSession,

CAServiceComponentInfo[]casci)

描述：终端软件平台调用本方法来更新 CASModule 中的解扰组件列表。

根据请求，CASModule 将开始解扰添加到数组的组件，并停止解扰移除的组件。

对于更新后不变的成分，不发生任何变化。

注：本方法很少被调用，通常在一个会话中由于 PMT 变化而发生。

终端软件平台还可在 CAS 会话如会话类型发生变化时通过调用本方法通知 CASModule。

参数：

casSession：请求解扰操作的会话

casci：CA 业务组件信息数组，该数组可以用于获取相关 ECM PID,以及用以 DCAS 装载控制字的 DescramblerContext 对象

返回：

无

B.1.4.1.1.3 stopDescrambling

原型：public void stopDescrambling(CASSession casSession)

描述：终端软件平台调用本接口请求 CASModule 停止解扰给定会话中的所有组件。

参数：

casSession 请求解扰操作的会话

返回：

无

B.1.4.1.1.4 getCAInfo

原型：public String getCAInfo(int cmdId, String data)

描述：终端软件平台调用本接口获取 CA 信息。

参数：

cmdId 命令的唯一标识，可根据实际项目扩展

data 查询参数

返回：

CA 信息数据

B.1.4.1.1.5 setCAInfo

原型：public int setCAInfo(int cmdId, String data)

描述：终端软件平台调用本接口设置 CA 信息。

参数：

cmdId 命令的唯一标识，可根据实际项目扩展

data CA 信息数据

返回：

设置成功返回 0

B.1.4.2 接口org.ngb.net.cas.module.CASDataUtils

本接口描述用于表示获取 CA 相关信息的通用接口。

B.1.4.2.1 方法

B.1.4.2.1.1 getCAInfo

原型：public String getCAInfo(int casId, int cmdId, String data)

描述：终端软件平台响应用户操作向 DCAS 应用获取 CA 信息。

参数：

casId 指定的 CAS 厂商

cmdId 命令的唯一标识，可根据实际项目扩展

data 查询参数

返回：

CA 信息数据

B.1.4.2.1.2 setCAInfo

原型：public int setCAInfo(int casId, int cmdId, String data)

描述：终端软件平台响应用户操作设置 CA 信息给 DCAS 应用。

参数：

casId 指定的 CAS 厂商

cmdId 命令的唯一标识，可根据实际项目扩展

data CA 信息数据

返回：

设置成功返回 0

B.1.4.2.1.3 getData

原型：public String getData(int casId, int cmdId, int[] type)

描述：终端软件平台响应用户操作向 DCAS 管理器获取数据。

参数：

casId 指定的 CAS 厂商

cmdId 命令的唯一标识，可根据实际项目扩展

type 数据类型引用

返回：

获取的数据

B.1.4.2.1.4 setData

原型：public int setData(int casId, int cmdId, int type, String data)

描述：终端软件平台响应用户操作设置 CA 信息给 DCAS 应用。

参数：

casId 指定的 CAS 厂商

cmdId 命令的唯一标识，可根据实际项目扩展

data DCAS 相关数据

type 数据类型

返回：

设置成功返回 0

B.1.4.3 接口org.ngb.net.cas.module.CADescriptor

描述：本接口提供了 CA 描述符的信息，给定业务的 PMT 中的可能提供 CA 描述符。此外 CAT 中也会有 CA 描述符的出现。

B.1.4.3.1 方法

B.1.4.3.1.1 getCASystemId

原型：public int getCASystemId()

描述：本方法返回 CA 描述符的 CASystemId。

参数：

无

返回：

CASystemId

B.1.4.3.1.2 getPid

原型：public int getPid()

描述：本方法返回 CA 描述符中的 PID (ECM PID 或 EMM PID)。

参数：

无

返回：

PID 值

B.1.4.3.1.3 getPrivateData

原型：public byte[] getPrivateData()

描述：本方法返回 CA 描述符终端的私有数据数组。

参数：

无

返回：

privateData

B.1.4.4 接口org.ngb.net.cas.module.CAServiceComponentInfo

描述：本接口用于提取特定 CA 业务成分的信息，如 ECM PID 和用于装载控制字的 DescramblerContext。

B.1.4.4.1 方法

B.1.4.4.1.1 getDescramblerContext

原型：public DescramblerContext getDescramblerContext()

描述：本方法返回用于 DCAS 应用装载控制字的 DescramberContext 对象，在 PMT 的组件循环中出现多次相同 CA 描述符（相同的 ECMPID 和私有数据）的情形下，应该只存在唯一一个 DescramblerContext。

参数：

无

返回：

DescramblerContext

B.1.4.4.1.2 getCADescriptor

原型：public CADescriptor getCADescriptor()

描述：本方法返回业务组件相关的 CA 描述符，CADescriptor 实例由 PMT 中的 CA 信息产生。

参数：

无

返回：

一个 CADescriptor

B.1.4.4.1.3 getComponentStreamPIDs

原型：public int[] getComponentStreamPIDs()

描述：本方法返回一个由 PMT 中描述的基本流数组，数组元素的顺序应同 getComponentStreamType 返回的数组元素顺序一致。

参数：

无

返回：

ES（基本流）PID 数组

B.1.4.4.1.4 getComponentStreamTypes

原型：public int[] getComponentStreamTypes()

描述：本方法返回 PMT 中的流类型数组，流类型值应符合 MPEG 标准。数组元素的顺序应同 getComponentStreamPID 返回的数组元素顺序一致。

参数：

无

返回：

流类型数组

B.1.4.4.1.5 getServiceIdentifiers

原型：public int[] getServiceIdentifiers()

描述：本方法返回对象所关联的业务标识数组，业务标识的表现形式由具体的使用环境而定。

参数：

无

返回：

ServiceID 数组

B.1.4.5 接口org.ngb.net.cas.module.CASPacketListener

B.1.4.5.1 描述

DCAS 应用通过本接口来接收带外 CAS Packets (如 EMMs)。

DCAS 应用根据给定的 CA 系统标识通过 CASModuleManager 类提供的 registerCasPacketListener 方法注册本监听器。

CA 系统标识由参数 casId 表示。

CAS 包的接收依赖终端软件平台来实现。

B.1.4.5.2 方法

B.1.4.5.2.1 casPacketArrived

原型：public void casPacketArrived(int casId, byte [] casPacketData,
byte [] casPacketHeader)

描述：DCAS 应用通过已注册的监听器来获得 CAS 包。

参数：

casId CA 系统标识

casPacketData CAS 包数据

casPacketHeader 依赖终端软件平台的 CAS 包头

返回：

无

B.1.4.6 接口org.ngb.net.cas.module.CASSession

本接口提供 CAS 会话相关信息。

B.1.4.6.1 常量域——会话类型

B.1.4.6.1.1 TYPE_PRESENTATION

public static final int TYPE_PRESENTATION = 0x00000001

B.1.4.6.1.2 TYPE_RECORDING

public static final int TYPE_RECORDING = 0x00000002

B.1.4.6.1.3 TYPE_BUFFERING

public static final int TYPE_BUFFERING = 0x00000004

B.1.4.6.2 方法

B.1.4.6.2.1 getType

原型：public int getType()
描述：本方法返回本会话的操作类型。
参数：
无
返回：
操作类型，可以是本接口中定义的值之一或组合
例如 - 本方法返回 0x00000003，即是类型(0x00000001)和 (0x00000002)的组合

B.1.4.6.2.2 getNetworkInterface

原型：public org.davic.net.tuning.NetworkInterface getNetworkInterface()
描述：本方法返回同 CAS 会话相关联的 NetworkInterface，DCAS 应用可以从 CAS 会话中获取相关 NetworkInterface 对象。使用 NetworkInterface，DCAS 应用可得到先 TransportStream 对象，用于调用 org.davic.mpeg.sections 应用接口进行 ECM Section 过滤。
参数：
无
返回：
一个 NetworkInterface 对象

B.1.4.6.2.3 getAssociatedService

原型：public java.lang.Object getAssociatedService()
描述：本方法返回 CAS 会话相关的业务。
参数：
无
返回：
一个 Service 对象

B.1.4.6.2.4 getServiceContext

原型：public java.lang.Object getServiceContext()
描述：本方法返回 CAS 会话相关的 ServiceContext。
注意：在某些实现中 ServiceContext 没有实际意义，本方法将返回 null。
参数：
无
返回：
一个 ServiceContext 对象

B.1.4.7 接口org.ngb.net.cas.module.CAStatus

B.1.4.7.1 描述

原型：public interface CAStatus
描述：DCAS 应用当调用 CASModuleManager 中的 sendDescramblingEvent 方法时使用本接口
DCAS 应用在每次 DescramblerContext 中的解扰状态发生变化时发送 CAStatus
本状态用来指示解扰成功与否
如果所解扰成分中任何一个发生解扰失败，本状态必须汇报整个业务的解扰请求失败
当终端软件平台收到一个新的 CAStatus,它应通过本段描述的 CAS 事件通知其他应用
具体应用接口在扩展应用接口部分说明

B.1.4.7.2 方法

B.1.4.7.2.1 isSuccess

原型：public boolean isSuccess()
描述：本方法返回解扰请求的状态。
参数：
无
返回：
如果解扰成功返回 true,解扰失败返回 false

B.1.4.7.2.2 getCAToken

原型：public int getCAToken()
描述：本方法返回用于其他应用通过 IXC 向 DCAS 应用查询网络相关信息的参数。
参数：
无
返回：
CA 令牌

B.1.4.8 接口org.ngb.net.cas.module.CATListener

DCAS 应用需实现该接口,使用 CAT 中的 CA 描述符来过滤带内 EMM。DCAS 应用需要通过 CATNotifier 接口中定义的 registerCATListener 注册本监听器。

B.1.4.8.1 方法

B.1.4.8.1.1 catUpdate

原型：public void catUpdate(CADescriptor desc,
org.davic.net.tuning.NetworkInterface ni)
描述：本接口用于通知 DCAS 应用特定网络接口上的 CAT 更新。
DCAS 应用可以通过 NetworkInterface 对象，获取现 TransportStream 对象。
TransportStream 对象可用 org.davic.mpeg.sections 应用程序接口来实现 EMM section 过滤。
终端软件平台将 CAT 更新通知到跟 casId 相匹配并且注册过的 CAT 监听器。

注意:

如果CAT不再被过滤（在成功过滤后）；或者CA 描述符在CAT上被删除时，终端软件平台应当调用 catUpdate(null , theNetworkInterface)。

参数:

desc The CA descriptor. DCAS应用通过CASDescriptor对象获取EMM PID

ni CAT更新所在的NetworkInterface

返回:

无

B.1.4.9 接口org.ngb.net.cas.module.CATNotifier

B.1.4.9.1 描述

原型：public interface CATNotifier

描述：DCAS应用使用本方法注册用于获取CAT更新通知的监听器

DCAS应用使用CAT信息过滤带内EMM

B.1.4.9.2 方法

B.1.4.9.2.1 registerCATListener

原型：public void registerCATListener(int casId, CATListener catListener)

描述：DCAS应用调用本方法注册一个CATListener。

参数:

casId CA系统标识

catListener 由于注册的CATListener

返回:

无

B.1.4.9.2.2 unregisterCATListener

原型：public void unregisterCATListener(CATListener catListener)

描述：DCAS应用调用本方法取消注册某个CATListener。

参数:

catListener 需要取消注册的CATListener

返回:

无

B.1.4.10 类org.ngb.net.cas.module.CASModuleManager

用来注册所有被DCAS应用实现的CASModule。

B.1.4.10.1 方法

B.1.4.10.1.1 getInstance

原型：public static CASModuleManager getInstance()

throws java.lang.SecurityException

描述：本方法用于获取一个 CASModuleManager 单例。

参数：

无

返回：

CASModuleManager 实例

异常处理：

java.lang.SecurityException-当安全策略被强制启用，但调用方没有被赋予一个 org.ngb.net.ca.module.CASPermission，抛出此异常。

B.1.4.10.1.2 registerCASmodule

原型：public void registerCASModule(CASModule aModule,
int caSystemId,
int networkCAPriority,
java.lang.Object context)
throws java.lang.IllegalArgumentException

描述：本方法用于 DCAS 应用在终端软件平台上注册一个 CASModule。

参数：

aModule:需要注册的 CASModule

caSystemId：CASModule 管理的 caSystemId

networkCAPriority：用于在超过一个 CASModule 在 CASModuleManager 中注册时使用，运营商可根据每个 CASModule 决定是否该参数可选，当优先级策略启用时，为运营商需要为每个 CASModule 指定优先级。终端软件平台应选择已注册，具有最高优先级，并且所管理的 caSystemId 在 PMT 中有相应 CA 描述符的 CASModule 发出解扰请求。当优先级策略禁用时，DCAS 应用应给该参数置零，CASModule 的决定方法由终端软件平台自行实现

context：希望进行注册 CASModule 的 DCAS 应用的 Context，用于终端软件平台决定 DCAS 应用的身份

返回：

无

异常处理：

java.lang.IllegalArgumentException-如果指定的 CASModule 实例已被注册，抛出此异常。

B.1.4.10.1.3 updateCASystemId

原型：public void updateCASystemId(CASModule aModule, int caSystemId)
throws java.lang.IllegalArgumentException

描述：本方法用于 DCAS 应用向应用软件平台更新某 CASModule 中的 CASystemId。

参数：

aModule - 指定 CASModule

caSystemId -module 管理的新 caSystemId

返回:

无

异常处理：

java.lang.IllegalArgumentException 如果给定的 CASModule 实例尚未注册。

B.1.4.10.1.4 sendDescramblingEvent

原型：public void sendDescramblingEvent(CASModule aModule,
CASSession casSession,CAStatus aCAStatus)throws
java.lang.IllegalArgumentException

描述：本方法被用于 DCAS 应用向终端软件平台返回一个 CAStatus 当某个 DescramblerContext 每次由于相应服务中的加扰成分的加扰组件改变而变化时,DCAS 应用必须发送 CAStatus 用于指示解扰是否成功。如果任何一个组件解扰失败，CAStatus 必须通知此服务的解扰失败。

当终端软件平台收到一个新的 CAStatus 时，应继续通过在扩展 API 中定义的 CAS Event 继续将该信息传至相应应用。

参数：

aModule：指定 CASModule

casSession：解扰请求操作的会话

aCAStatus：需要发送的 CAStatus

返回：

无

异常处理：

java.lang.IllegalArgumentException-如果所给的 CASModule 实例没有被注册。

B.1.4.10.1.5 unregisterCASModule

原型：public void unregisterCASModule(CASModule aModule)
throws java.lang.IllegalArgumentException

描述：本方法用于 DCAS 应用从终端软件平台上取消某 CASModule 的注册。

参数：

aModule 需要取消注册的 CASModule

返回：

无

异常处理：

java.lang.IllegalArgumentException 如果给定的 CASModule 尚未注册。

B.1.4.10.1.6 getChipControllers

原型：public ChipController[] getChipControllers()

描述：本方法用于 DCAS 应用从终端软件平台请求可被使用的芯片控制器列表，本方法为每个终端安全芯片返回一个芯片控制器。很多终端只支持单个芯片控制器，这种情况下，返回的数组中只含有一个元素。

参数：

无

返回：

一个芯片控制器数组

B.1.4.10.1.7 setcurrentController

原型：public void setCurrentController(CASModule aModule,
ChipController aChipController)throws
ava.lang.IllegalArgumentException

描述：本方法用于设置根据所给 CAModule 进行解扰操作所缺省使用的芯片控制器。如果本方法没有被调用，在 CASModuleManager 的选择无需指定。

参数：

aModule 指定 CASModule

aChipController CASModule-所使用的缺省芯片控制器

返回：

无

异常处理：

java.lang.IllegalArgumentException 如果给定的 CASModule 尚未注册。

B.1.4.10.1.8 setCCIBits

原型：public void setCCIBits(CASModule aModule,CASSession casSession,
int cciBits)

描述：本方法用于设定终端对于本业务进行拷贝保护所需拷贝控制信息数据（CCI bits），CCI 比特信息的定义由终端软件平台指定并解释执行。

参数：

aModule 指定 CASModule

casSession 解扰请求会话

cciBits 当前服务使用的 CCI 比特值

返回：

无

B.1.4.10.1.9 setServiceListFilter

原型：public void setServiceListFilter(int filterData)

描述：本方法用于向终端软件平台提供用于业务列表过滤的参数，业务列表参数的具体含义由终端软件平台指定并执行。

参数：

filterData 业务列表过滤参数

返回：

无

B.1.4.10.1.10 registerCASPacketListener

原型：public void registerCASPacketListener(int casId,
CASPacketListener casPacketListener) throws
java.lang.IllegalArgumentException

描述：本方法用于 DCAS 应用注册一个 CAPacketListener，CAPacketListener 由终端软件平台调用，并向 DCAS 应用传送 CAS 数据包（如 EMM）。CA 系统标识由参数 casID 表示， CAS 数据包的接收由终端软件平台各自实现。

参数：

casId CasystemID

casPacketListener 需要注册的 CASPacketListener

返回：

无

异常处理：

java.lang.IllegalArgumentException 如果给定 casId 已经注册过 listener。

B.1.4.10.1.11 unregisterCASPacketListener

原型：public void unregisterCASPacketListener(
CASPacketListener casPacketListener)
throws java.lang.IllegalArgumentException

描述：本方法用于 DCAS 应用取消 CASPacketListener 的注册。

参数：

casPacketListener 需要取消注册的 listener

返回：

无

异常处理：

java.lang.IllegalArgumentException 如果给定 CASPacketListener 尚未注册。

B.1.4.10.1.12 getDetachableSecurityDevices

原型：public DetachableSecurityDevice[] getDetachableSecurityDevices()

描述：本方法用于 DCAS 应用获得可分离设备（智能卡等）的对象句柄。

参数：

无

返回：

一个 DetachableSecurityDevice 对象

B.1.4.10.1.13 receiveOsdMsg

原型：public void receiveOsdMsg(byte[] msg, int[] flags)

描述：显示 OSD 信息，其参数的具体含义和具体项目相关。

参数：

msg - OSD 信息内容，可包括文本外的描述信息

flags - OSD 类型指示

返回：

无

B.1.4.10.1.14 showFingerMsg

原型：public void showFingerMsg(CASModule aModule, CASSession casSession, byte[] msg)

描述：显示 OSD 信息，其参数的具体含义和具体项目相关。

参数：

aModule - 指定 CASModule

casSession - 请求解扰操作的会话

msg - 指纹消息，为 NULL 时取消指纹显示

返回：

无

B.1.4.10.1.15 receiveTuningAlert

原型：public void receiveTuningAlert (int[] serviceIdentifiers, int[] flags)

描述：应急广播。在某些项目中应急广播的参数不是通过 CA 系统来下发的，在此类情况下不必实现此函数。

参数：

serviceIdentifiers - 一组用于标示应急广播频道参数的数值。数值的含义由具体项目定义

flags - 可用于表示应急广播类型的参数

返回：

无

B.1.4.10.1.16 getCATNotifier

原型：public CATNotifier getCATNotifier()

描述：本方法被 DCAS 应用调用来获取 CAT notifer 对象，DCAS 应用可以通过 CAT notifier 注册获取 CAT 更新通知的监听器。DCAS 应用需要 CAT 信息来过滤带内 EMM。

参数：

无

返回：

CAT Notifier 对象

B.1.4.11 类org.ngb.net.cas.module.CASPermission

B.1.4.11.1 描述

原型：public class CASPermission extends java.security.BasicPermission

描述：任一 DCAS 应用必须获取 CASPersmission 方可访问 CASModuleManager。

本机制用于确保只有被网络运营商授权的 DCAS 应用方可使用 DCAS API。

B.1.4.11.2 方法

B.1.4.11.2.1 CASPermission

原型：public CASPermission(String name)

描述：创建一个新 CASPermission. Name 字符串现不使用，应设为空字符串。

参数：

Name-本 CASPermission 的名称

返回：

无

B.1.4.11.2.2 CASPermission

原型：public CASPermission(String name, String actions)

描述：创建一个新 CASPermission. Name 字符串现不使用，应设为空字符串，actions 字符串现不使用，应设为空（null）。本构造方法用于 java.security.Policy 对象实例化一个新 Permission objects。

参数：

name 本 CASPermission 的名称

actions action 列表

返回：

无

B.1.5 org.ngb.net.cas.controller包

org.ngb.net.cas.controller 包提供了 DCAS 终端软件平台底层 API，TVOS 需实现此包。

Org.ngb.net.cas.controller 包概要见表 B.3。

表 B.3 org.ngb.net.cas.controler 包概要

接口	
DescramblerContext	用来控制终端安全芯片解扰功能的组件。可以实例化多个 DescramblerContext 来使用不同密钥解扰多个码流的情形。
ChipController	用来控制终端安全芯片执行的组件。
类	
Key	一个基本密码密钥。用来决定K-LAD使用的密码算法及密码函数的输出参数。
CWKey	解扰密钥或控制字。

B.1.5.1 接口org.ngb.net.cas.controller.DescramblerContext

B.1.5.1.1 描述

原型：public interface DescramblerContext

描述：表示用来控制终端安全芯片解扰功能的组件。可以实例化多个 DescramblerContext 来使用不同密钥解扰多个码流的情形

B.1.5.1.2 方法

B.1.5.1.2.1 loadCW

原型：public void loadCW(int Vendor_SysID, CWKey cwKey, Key[] levelKeys, int schemeId)
throws CADriverException

描述：本方法用于通知终端软件平台向解扰器装入控制字，并向终端安全芯片装入所需密钥。

一个解扰器通道是一个被单个控制字解扰的所有流的逻辑集合。

解扰器通道的使用依赖于 DescrablerContext。

此外，DCAS 应用应该通知终端软件平台当前控制字已失效（例如，由于授权原因），终端软件平台应当停止相应的解扰行为。

在这种情况下，DCAS 应用会提供一个 null CWKey。

参数：

Vendor_SysID：该值用于标识 CA 厂商在控制器中用于支持根密钥派生。终端安全芯片的根密钥由该值派生. 否则 Vendor_SysID 被忽略

cwKey：控制字. 如果控制字是明文，levelKeys 参数被忽略。如果 cwKey 为 null, 即 DCAS 应用没有提供有效的控制字

levelKeys：用于置入终端安全芯片的多级密钥。密钥数组的索引等于终端安全芯片中的绝对位置。在数组中特定元素值为 Null 表明终端安全芯片中相应位置不应装入密钥

即：levelKey[0]是 Key 1（被 Key 2 加密）；levelKey[1]是 Key 2（被 Key 3 加密）；levelKey[2]不必使用

schemeId：本 schemeId 用于指定终端安全芯片的加密算法，ChipController 接口中定义了方式（scheme）值的列表。如果控制器只支持一种方式，则该值被忽略

返回：

无

异常处理：CADriverException 如果装入失败。

B.1.5.1.2.2 overrideChipController

原型：public void overrideChipController(ChipController aChipController)
throws CADriverException

描述：本方法用于 DCAS 应用请求终端软件平台覆盖缺省终端安全芯片层级密钥的实现。可通过调用 CASModuleManager 的 setCurrentController 方法设置）。如果本方法没有被调用，终端安全芯片将使用缺省控制器。本方法只用于实现了多个终端安全芯片的终端安全芯片系统中。

参数：

aChipController 所要覆盖的控制器

返回：

无

异常处理:

CADriverException 如果操作失败。

B.1.5.2 接口org.ngb.net.cas.controller.Chipcontroller

B.1.5.2.1 描述

原型：public interface ChipController

描述：表示控制终端安全芯片执行的组件。

B.1.5.2.2 常量域

B.1.5.2.2.1 SCHEME_TDES

public static final int SCHEME_TDES=0

描述：用于指示终端安全芯片应使用 TDES 的值。

B.1.5.2.2.2 SCHEME_AES

public static final int SCHEME_AES=1

用于指示终端安全芯片应使用 AES 的值。

B.1.5.2.2.3 PROCESSING_MODE_REGULAR

public static final int PROCESSING_MODE_REGULAR=0
用于指示终端安全芯片认证应答算法中不需进行额外处理的值。

B.1.5.2.2.4 PROCESSING_MODE_POST_PROCESSING

Public static final int PROCESSING_MODE_POST_PROCESSING=1
用于指示终端安全芯片认证应答算法中需要实施后处理阶段的值。

B.1.5.2.3 方法

B.1.5.2.3.1 getPublicId

原型：public byte[] getPublicId() throws CADriverException
描述：本方法返回终端安全芯片的公共标识。
参数：
无
返回：
终端安全芯片的公共标识 publicId
异常处理：
CADriverException 如果访问终端安全芯片驱动过程中有通信错误。

B.1.5.2.3.2 getChipType

原型：public byte[] getChipType() throws CADriverException
描述：本方法返回终端安全芯片的类型标识。
参数：
无
返回：
终端安全芯片类型
异常处理：
CADriverException 如果访问终端安全芯片驱动过程中有通信错误。

B.1.5.2.3.3 getChipControllerProperty

原型：public java.lang.String getChipControllerProperty(java.lang.String propertyName) throws CADriverException
描述：本方法根据提供的终端安全芯片属性名称，返回该属性对应的值。本功能在本接口中被预留，可用于读出控制器中将来扩增的属性。现阶段没有定义任何属性名称。
参数：
propertyName 属性名称
返回：
属性值
异常处理：

CADriverException 如果访问终端安全芯片驱动过程中有通信错误。

B.1.5.2.3.4 authenticate

原型：public byte[] authenticate(int Vendor_SysID, byte[] challenge, Key[] levelKeys, int schemeId, int processingMode) throws CADriverExceptio

描述：本方法用于认证终端安全芯片中的层级密钥机制，终端安全芯片应根据送入的随机握手信息计算认证信息。

参数：

Vendor_SysID 该值用于标识 CA 厂商. 在控制器中用于支持根密钥派生。终端安全芯片的根密钥由该值派生. 否则 Vendor_SysID 被忽略

challenge：握手信息，随机数

levelKeys：层级密钥所需各级密钥.密钥数组的索引等于终端安全芯片中的绝对位置。在数组中特定元素值为 Null 表明终端安全芯片中相应位置不应装入密钥。即：levelKey[0]是 null；levelKey[1]是 Key 2（被 Key 3 加密）；levelKey[2]不必使用

schemeId 本 schemeId 用于指定终端安全芯片的加密算法

如果控制器只支持一种方式，则该值被忽略

processingMode：用于指定计算应答结果计算过程中是否实施额外

后处理过程的值，如果控制器只支持没有后处理模式，则该参数被忽略

返回：

终端安全芯片所计算的应答响应

异常处理：

CADriverException 如果访问终端安全芯片驱动过程中有通信错误。

B.1.5.2.3.5 encryptData

```
原型：public void encryptData(int Vendor_SysID,
                              CWKey cwKey,
                              Key[] levelKeys,
                              int schemeId,
                              int encryptionId,
                              byte[] src,
                              int srcPos,
                              byte[] dest,
                              int destPos,
                          int length) throws CADriverException
```

描述：本方法调用芯片功能来加密内存中的数据。

参数：

Vendor_SysID 本参数用于标示 CA 厂商。安全芯片用此数值来派生根密钥

cwKey 加密用的控制字。如果控制字没有加密，之后的 levelKeys 将被忽略

levelKeys 层级密钥

数组中密钥的索引值等于其在层级密钥中的绝对位置。数组中的 Null 元素表明在此层级位置没有 key 需要被设置

schemeId 层级密钥所使用的加密算法。如果芯片只支持一种算法，则此参数将被忽略

encryptionId 数据加密/解密的算法。如果芯片只支持一种算法，则此参数将被忽略

src 源数据数组

srcPos 源数据数组的起始位置

dest 目的数据数组

destPos 目的数据数组的起始位置

length 需要处理的数据字节数

异常处理：

层级密钥通讯错误时，抛出 CADriverException 异常。

B.1.5.2.3.6 decryptData

原型：public void decryptData(int Vendor_SysID,
CWKey cwKey,
Key[] levelKeys,
int schemeId,
int encryptionId,
byte[] src,
int srcPos,
byte[] dest,
int destPos,
int length) throws CADriverException

描述：本方法调用芯片功能来解密内存中的数据。

参数：

Vendor_SysID 本参数用于标示 CA 厂商。安全芯片用此数值来派生根密钥

cwKey 解密用的控制字。如果控制字没有加密，之后的 levelKeys 将被忽略

levelKeys 层级密钥。数组中密钥的索引值等于其在层级密钥中的绝对位置。数组中的 Null 元素表明在此层级位置没有 key 需要被设置

schemeId 层级密钥所使用的加密算法。如果芯片只支持一种算法，则此参数将被忽略

encryptionId 数据加密/解密的算法。如果芯片只支持一种算法，则此参数将被忽略

src 源数据数组

srcPos 源数据数组的起始位置

dest 目的数据数组

destPos 目的数据数组的起始位置

length 需要处理的数据字节数

返回：

无

异常处理：

层级密钥通讯错误时，抛出 CADriverException 异常。

B.1.5.3 类org.ngb.net.cas.controller.Key

B.1.5.3.1 描述

原型：public class Key
描述：表示一个基本密码密钥。用来决定 K-LAD 使用的密码算法及密码函数的输出参数。

B.1.5.3.2 方法

B.1.5.3.2.1 Key

原型：public Key(byte[] value, boolean encrypted)
参数：
value 密钥的值
encrypted-密钥是否被加密的标志，true 表示密钥已被加密，false 表示密钥是明文

B.1.5.3.2.2 getKeyValue

原型：public byte[] getKeyValue()
描述：本方法返回密钥的值。
参数：
无
返回：
密钥的值

B.1.5.3.2.3 isEncrypted

原型：public boolean isEncrypted()
描述：本方法返回 true 时，表示密钥是加密的，false 表示密钥未加密。
参数：
无
返回：
true 密钥是加密的，false 表示密钥未加密

B.1.5.4 类org.ngb.net.cas.controller.CWKey

B.1.5.4.1 描述

原型：public class CWKey extends Key
描述：表示解扰密钥或控制字

B.1.5.4.2 常量域

B.1.5.4.2.1 PARITY_EVEN

```
public static final int PARITY_EVEN = 0
```

B.1.5.4.2.2 PARITY_ODD

```
public static final int PARITY_ODD = 1
```

B.1.5.4.3 方法

B.1.5.4.3.1 CWKey

原型：public CWKey(byte[] value, boolean encrypted, int parity)

描述：

value 密钥的值。

真值表示密钥是加密的，假值表示密钥未加密。

奇偶值，表明控制字的奇偶性。

B.1.5.4.3.2 getParity

原型：public int getParity()

描述：本方法返回控制字的奇偶性。

参数：

无

返回：

控制字的奇偶性

B.1.5.5 类org.ngb.net.cas.controller.CASTEEManager

B.1.5.5.1 描述

原型：public class CASTEEManager

描述：与 TEE 中 TA 通信的接口

B.1.5.5.2 方法

B.1.5.5.2.1 sendCommandToTEE

原型：public byte[] sendCommandToTEE(byte[] teeAppUUID, int commandId, byte[] inputData) throws CADriverException

描述：DCAS 应用选择对应的安全应用，并发送数据给安全应用。

参数：

teeAppUUID TAPP 的 UUID 标识。

commandId 命令类型。

inputData 输入的数据。

返回：

返回的数据

异常处理：

CADriverException 如果跟 TEE 交互驱动过程中有通信错误。

B.1.6 org.ngb.net.cas.event包

org.ngb.net.cas.event 包提供了 DCAS 用于扩展 API 包，TVOS 的 DCAS 应用需实现此包。

Org.ngb.net.cas.event 包概要见表 B.4。

表 B.4　org.ngb.net.cas.event 包概要

接口	
CASEventListener	应被需要接收 CAS 事件的应用实现。
CASAppInfo	提供DCAS应用的信息。
CASEventInfo	提供CASEvent的信息。
类	
CASEventManager	应用使用CASEventManager注册监听器，来获取CAS事件。

B.1.6.1　接口org.ngb.net.cas.event.CASEventListener

B.1.6.1.1　描述

原型：public interface CASEventListener

描述：本接口应被需要接收 CAS 事件的应用实现。CAS events 提供了当前 ServiceContext 的 CA Status 和基本信息。

B.1.6.1.2　方法

B.1.6.1.2.1　receiveCASEvent

原型：public void receiveCASEvent(Object serviceContext,
int appId,
int orgId,
boolean isSuccess,
int caToken)

描述：本方法用于向注册了 CAS 事件监听器的应用传递 CAS 事件。

参数：

serviceContext CAS 事件所属的句柄

appId 用于标识发送事件的 DCAS 应用。这些标识可被应用通过 IXC 和 DCAS 应用通信。在没有 DCAS 应用可以解扰给定码流时，终端软件平台应当使用值 null 作为 appId 的值调用本方法，获取此种 CAS 事件通知的应用应根据自己的设计和实现来处理该情况

orgId 用于标识发送事件的 DCAS 应用。标识 App 所属组织。

IsSuccess 用于指示解扰是否成功的布尔值

caToken 通过 IXC 传回 DCAS 应用的令牌,应用可使用该令牌通过 IXC 向 DCAS 应用查询特定的网络信息

B.1.6.1.2.2　receiveCASOSDEvent

原型：public void receiveCASOSDEvent(Object serviceContext,
int appId,
int orgId,
byte[] msg,
int[] flag)

描述：本方法用于向注册了 CAS 事件监听器的应用传递 CAS 的 OSD 事件。

参数：

serviceContextCAS 事件所属的句柄

appId 用于标识发送事件的 DCAS 应用。这些标识可被应用通过 IXC 和 DCAS 应用通信。在没有 DCAS 应用可以解扰给定码流时，终端软件平台应当使用值 null 作为 casAppId 的值调用本方法，获取此种 CAS 事件通知的应用应根据自己的设计和实现来处理该情况

orgId 用于标识发送事件的 DCAS 应用。标识 App 所属组织。

msg 用于传递 OSD 的内容

flag 用于标识 OSD 的类型

B.1.6.1.2.3 receiveCASFingerEvent

原型：public void receiveCASFingerEvent(Object serviceContext,
int appId,
int orgId,
byte[] msg)

描述：本方法用于向注册了 CAS 事件监听器的应用传递 CAS 的指纹事件。

参数：

serviceContext CAS 事件所属的句柄

appId 用于标识发送事件的 DCAS 应用。这些标识可被应用通过 IXC 和 DCAS 应用通信。在没有 DCAS 应用可以解扰给定码流时，终端软件平台应当使用值 null 作为 casAppId 的值调用本方法，获取此种 CAS 事件通知的应用应根据自己的设计和实现来处理该情况

orgId 用于标识发送事件的 DCAS 应用。标识 App 所属组织。

Msg 用于传递指纹的内容

B.1.6.2 接口org.ngb.net.cas.event.CASAppInfo

B.1.6.2.1 描述

原型：public interface CASAppInfo

描述：本接口提供 DCAS 应用的信息。

B.1.6.2.2 方法

B.1.6.2.2.1 getAID

原型：public int getAID()

描述：本方法返回 DCAS 应用的 application ID。

参数：

无

返回：

DCAS 应用的 application ID

B.1.6.2.2.2 getOID

原型：public int getOID()

描述：本方法返回 DCAS 应用的 organization ID。

参数：

无

返回：

DCAS 应用的 organization ID

B.1.6.3 接口org.ngb.net.cas.event.CASEventInfo

B.1.6.3.1 描述

原型：public interface CASEventInfo

描述：本接口提供 CASEvent 的信息。

B.1.6.3.2 常量域

B.1.6.3.2.1 TYPE_PRESENTATION

```
public static final int TYPE_PRESENTATION = 0x00000001
```

B.1.6.3.2.2 TYPE_RECORDING

```
public static final int TYPE_RECORDING =    0x00000002
```

B.1.6.3.2.3 TYPE_BUFFERING

```
public static final int TYPE_BUFFERING =    0x00000004
```

B.1.6.3.3 方法

B.1.6.3.3.1 getType

原型：public int getType()

描述：本方法返回产生 CAS Event 的操作类型。

参数：

无

返回：

操作类型，可以是本接口中定义的值其中之一或组合

例如 - 本方法返回 0x00000003，即是类型(0x00000001)和(0x00000002)的组合

B.1.6.3.3.2 getNetworkInterface

原型：public org.davic.net.tuning.NetworkInterface getNetworkInterface()

描述：本方法返回和 CAS Event 相关的 NetworkInterface。

参数：

无

返回：

一个 NetworkInterface 对象

B.1.6.3.3.3 getAssociatedService

原型：public java.lang.Object getAssociatedService()
描述：本方法返回 CAS Event 相关联的业务。
参数：
无
返回：
一个 Service 对象

B.1.6.3.3.4 getServiceContext

原型：public java.lang.Object getServiceContext()
描述：本方法返回同 CAS event 相关联的 ServiceContext。
注意，在某些操作中 ServiceContext 无实际意义。本方法返回 null。
参数：
无
返回：
一个 ServiceContext 对象

B.1.6.4 类org.ngb.net.cas.event.CASEventManager

B.1.6.4.1 描述

原型：public class CASEventManager
描述：应用使用 CASEventManager 注册监听器，来获取 CAS 事件。
CAS events 提供了当前的 CA Status 和基本信息。

B.1.6.4.2 方法

B.1.6.4.2.1 getInstance

原型：public static CASEventManager getInstance()
描述：本方法用于取得一个 CASEventManager 实例单体。
参数：
无
返回：
CASEventManager 实例

B.1.6.4.2.2 addListener

原型：public void addListener(CASEventListener aCASEventListener)
描述：本方法用于应用注册一个 CASEventListener。该监听器用于传递所有的的 CAS 事件。
参数：
aCASEventListener 需要注册的 CASEventListener
返回：
无

B.1.6.4.2.3 removeListener

原型：public void removeListener(CASEventListener aCASEventListener)
描述：本方法用于应用取消注册一个 CASEventListener。
参数：
aCASEventListener 已经注册的 CASEventListener
返回：
无

B.1.7 org.ngb.net.cas.detachable包

org.ngb.net.cas.detachable 包提供了 DCAS 可分离安全设备 API，TVOS 需实现此包。
Org.ngb.net.cas.detachable 包概要见表 B.5。

表 B.5 org.ngb.net.cas.detachable 包概要

接口	
DetachableSecurityDevice	用于应用注册可分离安全设备的监听器，来获取设备插拔状态
DetachableSecurityDeviceListener	可分离安全设备的监听器，应被需要监听设备插拔状态的应用实现。

B.1.7.1 接口DetachableSecurityDevice

本接口表示用于控制同可分离安全设备通信的组件（智能卡等）。

B.1.7.1.1 方法

B.1.7.1.1.1 open

原型：public void open() throws CADriverException
描述：本方法用于DCAS应用初始化同可分离安全设备间的会话。
参数：
无
返回：
无
异常处理：
CADriverException 如果发生驱动错误。

B.1.7.1.1.2 close

原型：public void close() throws CADriverException
描述：本方法用于DCAS应用关闭用可分离安全设备间的会话。
参数：
无
返回：
无
异常处理：

CADriverException 如果发生驱动错误。

B.1.7.1.1.3 reset

原型：public byte[] reset() throws CADriverException
描述：本方法用于重置可分离安全设备，并返回数据（智能卡时数据为ATR）。
参数：
无
返回：
一个字节数组，存储设备重置返回的数据
异常处理：
CADriverException 如果发生驱动错误。

B.1.7.1.1.4 sendData

原型：public void sendData(byte [] data) throws CADriverException
描述：本方法用于DCAS应用向可分离安全设备发送数据。
参数：
data 需要发送的数据.（智能卡时为命令APDU）
返回：
无
异常处理：
CADriverException 如果发生驱动错误。

B.1.7.1.1.5 registerListener

原型：public void registerListener(DetachableSecurityDeviceListeneraListener)
描述：本方法用于DCAS应用注册接收可分离安全设备发送数据的监听器
参数：
aListener 要注册的DetachableSecurityDeviceListener
返回：
无

B.1.7.1.1.6 removeListener

原型：public void removeListener()
描述：本方法用于DCAS应用来删除已注册的监听器
参数：
无
返回：
无

B.1.7.2 接口DetachableSecurityDeviceListener

本接口应被DCAS应用实现，用于接收可分离安全设备状态和发送的数据。

B.1.7.2.1 字段

B.1.7.2.1.1 DEVICE_IN

public static final int DEVICE_IN = 1
描述：用于描述可分离安全设备状态：插入（智能卡时表示智能卡插入）。

B.1.7.2.1.2 DEVICE_OUT

public static final int DEVICE_OUT = 2
描述：用于描述可分离安全设备状态：拔出（智能卡时表示智能卡拔出）。

B.1.7.2.1.3 DEVICE_ERROR

public static final int DEVICE_ERROR = 3
描述：用于描述可分离安全设备状态：出错（智能卡时表示智能卡出错）。

B.1.7.2.2 方法

B.1.7.2.2.1 receiveDeviceStatus

原型：public void receiveDeviceStatus(int status)
描述：本方法应被DCAS应用实现，用于接收可分离安全设备状态。
可分离安全设备状态变化时通知 DCAS 应用。
参数：
status 可分离安全设备状态（见字段描述）
返回：
无

B.1.7.2.2.2 receiveData

原型：public void receiveData(byte [] data)
描述：本方法在可分离安全设备向 DCAS 应用发送数据时被调用。
参数：
data 可分离安全设备发送的数据.（智能卡时数据为响应 APDU）
返回：
无

B.2 Javascript应用程序接口

B.2.1 概述

提供HTML5运行环境的终端软件平台上，可以通过DCAS Javascript应用程序接口开发DCAS用户端软件，见表B.6。

表 B.6　DCAS 应用程序 Javascript 接口

类名称	说明
JSDCAS.CASDescriptor	CA 描述对象，这个对象用于表达 PMT 或者 CAT 中的 CA 描述符
JSDCAS.CASEcmEvent	ECM 事件对象，这个对象包含了 ECM 事件的信息
JSDCAS.CASEmmEvent	ECM 事件对象，这个对象包含了 EMM 事件的信息
JSDCAS.CASFilter	过滤器对象，此对象表达了过滤带内或带外 EMM 时，所需要的过滤条件
JSDCAS.CASM	CASM 的全局对象，可以从它访问到 CAS manager 和 controller 对象
JSDCAS.CASModule	CAS 模块对象接口，由 JS DCAS 应用实现，向平台提供的接口
JSDCAS.CASModuleManager	CASModuleManager 对象，此对象提供 JSDCAS 应用需要的接口
JSDCAS.CASPacketEvent	CAS 数据包事件对象，通知 JS DCAS 应用带外 CAS 数据包的事件
JSDCAS.CASSession	CAS Sesion 对象，平台为每个解扰请求都生成一个 CAS Session 对象
JSDCAS.CASStatus	CAS 状态对象，通过这个对象，JSDCAS 向平台传递解扰状态
JSDCAS.TeeController	TEE 控制权对象，JS DCAS 和 TEE 通信的控制器
JSDCAS.TeeRetVal	TEE 返回对象，对象中包含从 TEE 返回的数据，错误等信息

B.2.2　应用接口调用时序

DCAS Javascript应用程序接口调用时序如图B.4所示。

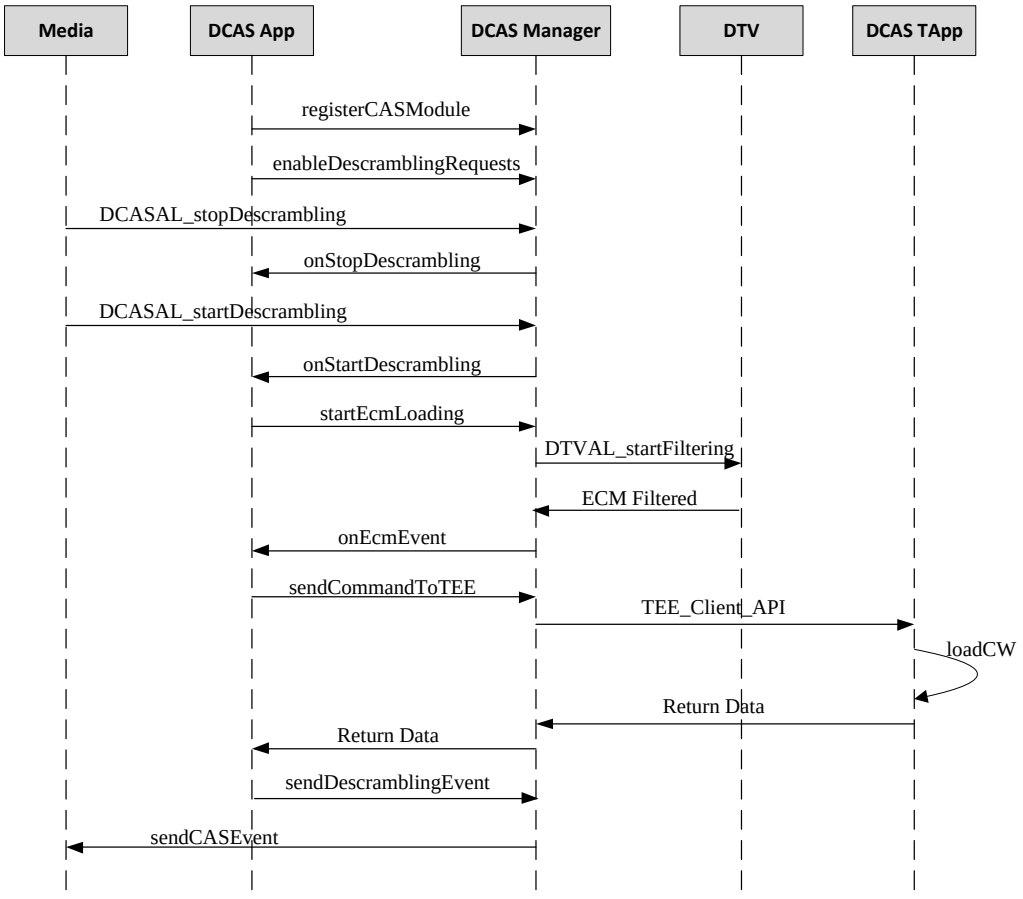

图 B.4　DCAS 应用接口基本调用时序

B.2.3 JSDCAS.CASDescriptor类

这个对象用于表达PMT或者CAT中的CA描述符。

B.2.3.1 getCasId

原型：{number}getCasId()

描述：本方法由终端软件平台提供，返回 CA 描述符中的 CASID。

B.2.3.2 getPid

原型：{number}getPid()

描述：返回 CA 描述符中的 PID。如果描述符来自 CAT，则表示 EMM PID。如果描述符来自 PMT，则表示 ECM PID。

B.2.3.3 getPrivateData

原型：{Uint8Array}getPrivateData()

描述：返回 CA 描述符中的私有数据,私有数据以 Uint8Array 形式返回。

B.2.4 JSDCAS.CASEcmEvent类

这个对象包含了 ECM 事件的信息,ECM 事件是通过 CASModule.onEcmEvent 或者 CASModuleManager.onStartDescrambling 传递给 JS DCAS 应用。它可以表达收到了 ECM 包，或者超时，或者过滤器内部错误。

B.2.4.1 getEcmData

原型：{Uint8Array}getEcmData()

描述：返回完整的 ECM 数据。如果事件是超时或者内部错误，则返回 null。

B.2.4.2 getError

原型：{number}getError()

描述：返回内部 section filter 过滤器的错误。只用于调试。这个方法只能在事件没有提供任何其他信息的时候调用。

B.2.4.3 getTableId

原型：{number} getTableId()

描述：返回 ECM 包的 Table ID,这个方法只能在事件提供 ECM 数据的情况下调用。

B.2.4.4 isTimeout

原型：{boolean} isTimeout()

描述：返回在获取第一个 ECM 包时是否超时。超时的时长可以在 CASModuleManager.enableDescramblingRequests 当中设定，这个方法只能在事件没有提供 ECM 数据的情况下调用。

返回：ture-超时。

False-没有超时。

B.2.5 JSDCAS.CASEmmEvent类

这个对象包含了 EMM 事件的信息,EMM 事件是通过 CASModule.onInBandEmmEvent 传递给 JS DCAS 应用的。收到这个事件可能是因为收到了 CAT，或者任意 EMM 包，或者内部过滤器错误。

B.2.5.1 getEmmData

原型：{Uint8Array}getEmmData()
描述：返回完整的 EMM 数据。如果事件是由 CAT 更新引起，或者由于内部错误引起，则返回 null。

B.2.5.2 getError

原型：{number}getError()
描述：返回内部 section filter 过滤器的错误。只用于调试。这个方法只能在事件没有提供任何其他信息的时候调用。

B.2.5.3 getTableId

原型：{number} getTableId()
描述：返回 EMM 包的 Table ID。

B.2.5.4 isCatUpdateNotification

原型：{boolean}isCatUpdateNotification()
描述：返回事件是否是由 CAT 更新引起的,如果是由 CAT 更新引起的，则 EMM 数据应该是空。
返回：ture-事件由 CAT 更新引起。
false-事件由任意 EMM 包引起，或者由内部错误引起。

B.2.6 JSDCAS.CASFilter类

这个对象用于表达过滤带内或带外 EMM 时，所需要的 section filter 过滤条件。平台应该只在数据包匹配过滤条件的时候才调用 CAS Module。对于那些不符合条件的数据包，应该由平台丢弃，JS DCAS 应用也不应该被调用。CASFilter 对象（或者对象数组）可以通过 CASModuleManager.startCasPacketLoading 和 CASModuleManager.startInbandEmmLoading 设定到平台。过滤条件包括：

a) 以字节为单位的偏移量。偏移量之前的数据将被忽略，不参与比较。
b) 用于和平台收到的数据进行比较的值，以 bitmap 表示。
c) 用于表达哪些位需要参与比较，对于 bitmap 中设定为 0 的位，不需要进行比较。

B.2.6.1 getBitmapMask

原型：{Uint8Array}getBitmapMask()
描述：返回过滤掩码。

B.2.6.2 getBitmapValue

原型：{Uint8Array}getBitmapValue()
描述：返回用于比较的 bitmap 值。

B.2.6.3 getOffset

原型：{number}getOffset()
描述：返回偏移量（单位为字节）。

B.2.7 JSDCAS.CASM类

CASM 是一个全局对象，可以从它访问到所有的 CAS manager 和 controller 对象。

B.2.7.1 getCASModuleManager

原型：{JSDCAS.CASModuleManager}getCASModuleManager()
描述：返回 CAS Module Manager 对象实例。

B.2.7.2 getTeeController

原型：{JSDCAS.TeeController}getTeeController ()
描述：返回 TEE Controller 对象实例。

B.2.8 JSDCAS.CASModule类

CASModule 是一个 CAS 模块对象的接口，应该由 JS DCAS 应用来实现，并注册到平台的 CAS Module Manager 中以便接收解扰请求，ECM，EMM 或者其他 JS DCAS 应用关心的任意元数据。EMM 可以通过带内或者带外方式获得。由端到端的设计以及平台和设备的能力决定。

B.2.8.1 getCasId

原型：{number}getCasId()
描述：返回 CAS 模块的唯一 CAS ID。这个方法必须在调用 CASModuleManager.registerCASModule 之前实现。返回的值是预期要在 CAT 或者 PMT 中 CA 描述符中出现的值。

B.2.8.2 onCasPacketEvent

原型：onCasPacketEvent(casPacketEvent)
描述：平台在收到带外 EMM（或其他带外 CAS 数据包）的时候调用这个方法,参考 CASModuleManager.startCasPacketLoading。
参数：CASPacketEvent casPacketEvent 包含带外 EMM 或其他带外 CAS 数据包的 CASPacketEveng 对象实例。

B.2.8.3 onEcmEvent

原型：onEcmEvent(casSession,ecmEvent)
描述：在平台过滤到了新的 ECM 包后，调用 JS DCAS 应用的这个方法。在 fast 模式下，平台在将 ECM 中的 CW 设定到 K-LAD 之后，调用这个方法。
参数：CASSession casSession-从 CASModule.onStartDescrambling 获得的 CAS Session 对象。
CASEcmEvent ecmEvent-ECM 事件对象。

B.2.8.4 onInbandEmmEvent

原型：onInbandEmmEvent(casSessionForEMM,emmEvent)

描述：平台在收到了 CAT 更新或者带内 EMM 的时候调用这个方法。参考 CASModuleManager.startInbandEmmLoading。

参数：CASSession casSessionForEMM-特殊的 CAS Session，这个 session 是关联到 CAT 所在的 TS，同时也包含了 CAT 中的 CA 描述符(不是 PMT 中的 CA 描述符)。平台应该专门为此创建一个 CAS Session，注意，这个 CAS Session 对象也许只有部分字段有效，注意：如果 CAT 中没有指定 CAS 的 CA 描述符，或者终端离开了当前频点，CAT 更新事件还是会送到 JS DCAS 应用，但是 CAS Session 为 null。

CASEmmEvent emmEvent-包含了 CAT 更新或者 EMM，或者其他错误的 CASEmmEvent 对象实例。

B.2.8.5 onStartDescrambling

原型：onStartDescrambling(casSession,firstEcmEvent)

描述：这个方法由平台在有新的解扰请求时调用。 通常发生在平台开始播放一个新的加扰频道的时候。 JS DCAS 应用只会在调用了 CASModuleManager.enableDescramblingRequests 之后才收到这个解扰请求。在 auto-load 的模式中，平台会自动开始过滤第一个 ECM，并在收到 ECM 后调用这个方法。这种情况下，JS DCAS 应用不必再调用 CASModuleManager.startEcmLoading。如果设备有多个 tuner，这个方法可能会被同时调用多次，每次对应一个播放解扰请求。每个请求都会有不同的 CAS Session。另外，如果需要解扰的 service 的基础流加扰方式不同，这个方法也可能被调用多次。同样，每次调用也都有不同的 CAS Session。

参数：CASSession casSession-平台为特定解扰请求生成的 CAS Session 对象。每个对象有唯一的 session ID，以及关于 service 以及基础流的所有信息。还会包含 PMT 中对应这个 CAS 模块的 CA 描述符。

CASEcmEvent firstEcmEvent-在 auto-load 模式下，平台收到的第一个 ECM 包（或者超时，错误）。如果不是 auto-load 模式，则为空。

B.2.8.6 onStopDescrambling

原型：onStopDescrambling(casSession)

描述：平台在停止播放当前频道的时候调用这个方法，通知 JS DCAS 应用停止解扰。

参数：CASSession casSession-在 CASModule.onStartDescrambling 中获得的 CAS Session。

B.2.9 JSDCAS.CASModuleManager类

JS DCAS 应用通过这个 CAS Module Manager 对象来实现接收解扰请求，ECM，EMM 以及报告 CAS 解扰状态。JS DCAS 应用应该实现 CAS Module 对象，然后注册到 CAS Module Manager。

B.2.9.1 枚举常量

JSDCAS.CASModuleManager.ACTION_ERROR_ACTION_NOT_SUPPORTED
JSDCAS.CASModuleManager.ACTION_ERROR_DRIVER
JSDCAS.CASModuleManager.ACTION_ERROR_INVALID_PARAMETERS
JSDCAS.CASModuleManager.ACTION_ERROR_NETWORK
JSDCAS.CASModuleManager.ACTION_ERROR_SECURITY
JSDCAS.CASModuleManager.ACTION_OK
JSDCAS.CASModuleManager.PROP_ID_BOUQUET

JSDCAS.CASModuleManager.PROP_ID_CAS_VENDOR_ID
JSDCAS.CASModuleManager.PROP_ID_CAS_VERSION
JSDCAS.CASModuleManager.PROP_ID_CHIP_ID
JSDCAS.CASModuleManager.PROP_ID_HSM_ID
JSDCAS.CASModuleManager.PROP_ID_HSM_POSITION_X
JSDCAS.CASModuleManager.PROP_ID_HSM_POSITION_Y
JSDCAS.CASModuleManager.PROP_ID_USER_BITS
JSDCAS.CASModuleManager.PROP_ID_SECURE_BITS
JSDCAS.CASModuleManager.PROP_ID_STB_ACTIVE_STATUS
JSDCAS.CASModuleManager.PROP_ID_ZIPCODE
JSDCAS.CASModuleManager.PROP_TYPE_NUMBER
JSDCAS.CASModuleManager.PROP_TYPE_STRING
JSDCAS.CASModuleManager.PROP_TYPE_UINT8ARRAY

B.2.9.2 方法

B.2.9.2.1 disableDescramblingRequests

原型：{number}disableDescrambingRequests(casModule)

描述：JS DCAS 应用调用这个方法停止接收解扰请求。 这个方法很少有机会被调用。比如在应用需要重新配置解扰请求参数的时候，需要先调用这个方法暂停接收，然后再重新启动接收。或者应用希望关闭自己。重新调用 CASManager.enableDescramblingRequests 可以重新开启接收。

参数：CASModule casModule - CAS 模块实例。

返回：成功 - CASModuleManager.ACTION_OK。

失败 - 返回如下错误值：

CASModuleManager.ACTION_ERROR_INVALID_PARAMETERS - 无效参数

CASModuleManager.ACTION_ERROR_DRIVER - 底层错误

B.2.9.2.2 enableDescramblingRequests

原型：{number} enableDescramblingRequests(casModule,firstEcmTimeout,
autoLoadFirstEcm,isFastMode,ecmTableIds)

描述：JS DCAS 应用通过调用这个方法启动接收解扰请求。通过不同的参数，可以配置平台 CAS Manager 和 CAS 模块之间不同的工作方式。这个方法通常只在 CAS 模块注册后调用，因为通常 JS DCAS 应用不会改变这个工作方式。如果 JS DCAS 应用要改变这个工作方式，需要先调用 CASModuleManager.disableDescramblingRequests，然后重新调用这个方法。

参数：CASModule casModule - CAS 模块实例。

Number firstEcmTimeout - 单位毫秒，平台等待第一个 ECM 的最长时间。如果超时，CAS 模块会通过 onEcmEvent 调用或者 onSTartDescrambling 调用收到 CASEcmEvent。

boolean autoLoadFirstEcm - 指定是否 auto-load 模式。Auto-load 模式是指平台在 JS DCAS 应用调用这个方法后，自动开始过滤第一个 ECM，而不必等待 JS DCAS 应用调用 startEcmLoading。

boolean isFastMode - 快速模式（暂时仅是占位，并没有实际意义）。

Array ecmTableIds - 如果 JS DCAS 应用希望指定 ECM 的 tableID。

返回：成功 - CASModuleManager.ACTION_OK。

失败 - 返回如下错误值：

CASModuleManager.ACTION_ERROR_INVALID_PARAMETERS - 无效参数

CASModuleManager.ACTION_ERROR_ACTION_NOT_SUPPORTED - 平台未实现特定模式

CASModuleManager.ACTION_ERROR_DRIVER - 底层错误

B.2.9.2.3 fetchDataFromCasHeadend

原型：{Uint8Array|number} fetchDataFromCasHeadend(casModule,inputData,casHeURI)

描述：JS DCAS应用通过调用这个方法，经过平台，通过GPRS，或者未来可能的其他手段，从头端获取数据。

参数：CASModule casModule - CAS模块实例。

Uint8Array inputData - 将要发送给头端的数据。

String casHeURI - 头端服务的URI。

返回：成功 - 头端返回的数据。

失败 - 返回如下错误值：

CASModuleManager.ACTION_ERROR_INVALID_PARAMETERS - 无效参数

CASModuleManager.ACTION_ERROR_DRIVER - 底层错误

CASModuleManager.ACTION_ERROR_ACTION_NOT_SUPPORTED - 方法不支持

CASModuleManager.ACTION_ERROR_NETWORK - 网络错误

B.2.9.2.4 registerCASModule

原型：

{number}registerCASModule(vendorId,casModule,networkPriority,applicationContext)

描述：JS DCAS应用通过调用这个方法来将自己注册到平台的CAS Module Manager中。

参数：number vendorId - CAS的Vendor Id。每个CAS厂家都有唯一的特殊ID。

CASModule casModule - 要注册的CAS模块实例。

Number networkPriority - 如果CASModuleManager允许注册超过一个CAS模块， 这个可选参数表达了多个模块之间的优先级，具体的值由运营商决定。 绝对值越大，表示优先级越高，例如3的优先级高于2。 在知道了优先级后，平台在遇到PMT中有多个CA描述符的情况，应该按照优先级，发送解扰请求给拥有最高优先级的CAS模块。如果运营商并没有设定优先级，每个JS DCAS应用必须以0作为参数。这种情况下，哪个CAS模块可以收到解扰请求，是由平台的实现决定的。

*applicationContext - 平台相关的应用参数。通常这个参数是由平台在初始化的时候告诉应用的。这个参数的使用情况是项目相关的。

返回：成功 - CASModuleManager.ACTION_OK。

失败 - 返回如下错误值：

CASModuleManager.ACTION_ERROR_SECURITY - 调用者没有权限访问

CASModuleManager

CASModuleManager.ACTION_ERROR_INVALID_PARAMETERS - 无效的参数

B.2.9.2.5 removeCASModule

原型：{number}removeCASModule(vendorId,casModule,applicationContext)

描述：JS DCAS应用调用这个方法将一个CAS模块从CAS Module Manager中删除。 这个方法很少会被调用到。比如在应用希望换一个CAS ID或者在应用关闭自己之前。

参数：number vendorId - CAS的Vendor Id。每个CAS厂家都有唯一的特殊ID。

CASModule casModule - 要注册的CAS模块实例。

*applicationContext - 平台相关的应用参数。通常这个参数是由平台在初始化的时候告诉应用的。这个参数的使用情况是项目相关的。

返回：成功 - CASModuleManager.ACTION_OK。

失败 - 返回如下错误值：

CASModuleManager.ACTION_ERROR_INVALID_PARAMETERS - 无效参数

CASModuleManager.ACTION_ERROR_DRIVER - 底层错误

CASModuleManager.ACTION_ERROR_SECURITY - 调用者没有权限

B.2.9.2.6 sendCommandToSTB

原型：{number}sendCommandToSTB(casModule,inputData)

描述：数据通道函数，JS DCAS应用通过调用这个方法，把数据发送给DCAS Manager，DCASManager把命令转发给相应模块处理，这些命令是包括OSD、升级触发、指纹、应急广播、收视调查等，这些命令由BOSS发送，DCAS仅负责转发，作为数据通道使用。

参数：CASModule casModule - 要注册的CAS模块实例。

Uint8Array inputData - 将要发送给DCAS管理器的数据。

返回：成功 - CASModuleManager.ACTION_OK。

失败 - 返回如下错误值：

CASModuleManager.ACTION_ERROR_INVALID_PARAMETERS - 无效参数

CASModuleManager.ACTION_ERROR_DRIVER - 底层错误

CASModuleManager.ACTION_ERROR_ACTION_NOT_SUPPORTED - 方法不支持

CASModuleManager.ACTION_ERROR_NETWORK - 网络错误

B.2.9.2.7 sendDataToHeadend

原型：{number}sendDataToHeadend(casModule,inputData)

描述：JS DCAS应用通过调用这个方法，经过平台，通过GPRS，或者未来可能的其他手段，发送数据给头端。

参数：CASModule casModule - 要注册的CAS模块实例。

Uint8Array inputData - 将要发送给头端的数据。

返回：成功 - CASModuleManager.ACTION_OK。

失败 - 返回如下错误值：

CASModuleManager.ACTION_ERROR_INVALID_PARAMETERS - 无效参数

CASModuleManager.ACTION_ERROR_DRIVER - 底层错误

CASModuleManager.ACTION_ERROR_ACTION_NOT_SUPPORTED - 方法不支持

CASModuleManager.ACTION_ERROR_NETWORK - 网络错误

B.2.9.2.8 sendDescramblingEvent

原型：{number}sendDescrambingEvent(casModule,casSession,casStatus)

描述：JS DCAS 应用调用这个方法报告 CAS 状态。CAS 状态包括解扰成功与否。每次状态变化，JS DCAS 都应该报告状态。

参数：CASModule casModule - CAS 模块实例。

CASSession casSession - 从 CASModule.onSTartDescrambling 获得的 CAS Session。

CASStatus casStatus - JS DCAS 应用生成的 CASStatus 对象。

返回：成功 - CASModuleManager.ACTION_OK。

失败 - 返回如下错误值：

CASModuleManager.ACTION_ERROR_INVALID_PARAMETERS - 无效参数

CASModuleManager.ACTION_ERROR_ACTION_NOT_SUPPORTED - 方法不支持

B.2.9.2.9 sendFreeTextOSD

原型：{number}sendFreeTextOSD(casModule,inputData,flags)

描述：JS DCAS 应用通过调用这个方法转发接收到的文本信息给平台 平台可能将这个文本信息转发给 UI 应用或者自己处理。

参数：CASModule casModule - CAS 模块实例。

Uint8Array inputData - 文本信息。

ArrayBuffer flags - 用于显示方式格式等的额外信息，项目相关。

返回：成功 - CASModuleManager.ACTION_OK。

失败 - 返回如下错误值：

CASModuleManager.ACTION_ERROR_INVALID_PARAMETERS - 无效参数

CASModuleManager.ACTION_ERROR_ACTION_NOT_SUPPORTED - 方法不支持

B.2.9.2.10 setCCIBits

原型：{number}setCCIBits(casModule,casSession,cciBits)

描述：设定 CCI(Copy Control Information)数据位。

参数：CASModule casModule - CAS 模块实例。

CASSession casSession - 从 CASModule.onSTartDescrambling 获得的 CAS Session。

Number cciBits - CCI 数据位。

返回：成功 - CASModuleManager.ACTION_OK。

失败 - 返回如下错误值：

CASModuleManager.ACTION_ERROR_INVALID_PARAMETERS - 无效参数

CASModuleManager.ACTION_ERROR_DRIVER - 底层错误

CASModuleManager.ACTION_ERROR_ACTION_NOT_SUPPORTED - 方法不支持

B.2.9.2.11 setData

原型：{number}setData(casModule, propertyId, propertyType, propertyValue)

描述：DCAS APP设置属性值给平台，包含BouquetID、激活状态、CAS信息、北斗信息、ChipID、HSMID、CASVenderID、区域码、CA版本等。

参数：CASModule casModule - CAS模块实例。

Number propertyId - 属性ID，见JSDCAS.CASModuleManager.PROP_ID_xxx。

Number propertyType - 属性类型，见JSDCAS.CASModuleManager.PROP_TYPE_xxx。

Number|string|Uint8Array propertyValue - 属性值

返回：成功 - CASModuleManager.ACTION_OK。

失败 - 返回如下错误值：

CASModuleManager.ACTION_ERROR_INVALID_PARAMETERS - 无效参数

CASModuleManager.ACTION_ERROR_ACTION_NOT_SUPPORTED - 方法不支持

B.2.9.2.12 setPinCode

原型：{number}setPinCode(casModule, pinCode)

描述：通知平台重置 PIN。

参数：CASModule casModule - CAS 模块实例。

Number pinCode - PIN 码。

返回：成功 - CASModuleManager.ACTION_OK。

失败 - 返回如下错误值：

CASModuleManager.ACTION_ERROR_INVALID_PARAMETERS - 无效参数

CASModuleManager.ACTION_ERROR_DRIVER - 底层错误

CASModuleManager.ACTION_ERROR_ACTION_NOT_SUPPORTED - 方法不支持

B.2.9.2.13 setServiceListFilter

原型：{number}setServiceListFilter(casModule,filterData)

描述：设置服务列表过滤条件 过滤条件的定义是平台相关的。

参数：CASModule casModule - CAS 模块实例。

Number filterData - 过滤条件。

返回：成功 - CASModuleManager.ACTION_OK。

失败 - 返回如下错误值：

CASModuleManager.ACTION_ERROR_INVALID_PARAMETERS - 无效参数

CASModuleManager.ACTION_ERROR_ACTION_NOT_SUPPORTED - 方法不支持

B.2.9.2.14 startCasPacketLoading

原型：{number}startCasPacketLoading(casModule,cableModemFilter,sourceURL,casFilter)

描述：JS DCAS 应用调用这个方法启动接收带外 CAS 数据包。CAS 数据包可以是 EMM 或者其他带外元数据。接收方法由设备硬件、平台、网络等决定。对于带有 Cable Modem 的设备，可以通过 ADSG 或者 BDSG 协议接收，也可以通过 IP over Cable 来加入多播地址来接收。IPTV 设备，可以通过以太网 / Wifi 来加入多播地址来接收。也可以通过一个本地的 UDPsocket 来接收。不论哪种情况，JS DCAS 应用都可以在收到 CAS 数据包的时候通过 CASModule.onCasPacketEvent 来处理。注意：平台可能实现为同时从多个不同源接收数据包。这种情况下，这个方法可能会从不同 URL 调用多次。

参数：CASModule casModule - CAS 模块实例。

Number|string cableModemFilter - 在 Cable Modem 和 DSG tunnelD 的情况下，必须提供一个 filter。在非 DSG 的情况下，这个参数应该为 null。

String sourceURL - 如果从 UDP 接收数据包，则需要提供这个参数。从本地 UDP 端口接收：“udp://@127.0.0.1:4444” 或 “udp://@localhost:4444”。

CASFilter|Array casFilter - 过滤条件,可以是一个 CASFilter 数组。

返回：成功 - CASModuleManager.ACTION_OK。

失败 - 返回如下错误值：

CASModuleManager.ACTION_ERROR_INVALID_PARAMETERS - 无效参数

CASModuleManager.ACTION_ERROR_DRIVER - 底层错误

CASModuleManager.ACTION_ERROR_ACTION_NOT_SUPPORTED - 方法不支持

B.2.9.2.15 startEcmLoading

原型：{number}startEcmLoading(casModule,casSession)

描述：JS DCAS 应用通过调用这个方法启动接收特定加扰节目的 ECM 在收到了解扰请求后调用。注意：在 auto-load 模式中，并不需要调用这个方法。

参数：CASModule casModule - CAS 模块实例。

CASSession casSession - 从 CASModule.onSTartDescrambling 获得的 CAS Session。

返回：成功 - CASModuleManager.ACTION_OK。

失败 - 返回如下错误值：

CASModuleManager.ACTION_ERROR_INVALID_PARAMETERS - 无效参数

CASModuleManager.ACTION_ERROR_DRIVER - 底层错误

B.2.9.2.16 startInbandEmmLoading

原型：{number}startInbandEmmLoading(casModule,emmTableIds,casFilter,
includeCatNotifications)

描述：JS DCAS 应用调用这个方法来启动接收带内 EMM。如果 JS DCAS 应用需要，也可以用来接收 CAT。当有 EMM 或者 CAT 更新的时候，JS DCAS 应用通过 CASModule.onInbandEmmEvent 来接收数据。

参数：CASModule casModule - CAS 模块实例。

Array emmTableIds - EMM table id 数组。

CASFilter|Array casFilter - CAS 过滤器。只有符合条件的数据，平台才会通知到应用，传递一个 Filter 数组也是可以的。

boolean includeCatNotifications - 指定是否希望接收 CAT 更新通知。

返回：成功 - CASModuleManager.ACTION_OK。

失败 - 返回如下错误值：

CASModuleManager.ACTION_ERROR_INVALID_PARAMETERS - 无效参数

CASModuleManager.ACTION_ERROR_DRIVER - 底层错误

CASModuleManager.ACTION_ERROR_ACTION_NOT_SUPPORTED - 方法不支持

B.2.9.2.17 stopCasPacketLoading

原型：{number}stopCasPacketLoading(casModule,cableModemFilter,sourceURL)

描述：JS DCAS 应用调用这个方法停止接收带外 CAS 数据包。

参数：CASModule casModule - CAS 模块实例。

Number|string cableModemFilter - Cable Modem 需要。

String sourceURL - 通过 UDP 接收时需要。

返回：成功 - CASModuleManager.ACTION_OK。

失败 - 返回如下错误值：

CASModuleManager.ACTION_ERROR_INVALID_PARAMETERS - 无效参数
CASModuleManager.ACTION_ERROR_DRIVER - 底层错误

B.2.9.2.18 stopEcmLoading

原型：{number}stopEcmLoading(casModule,casSession)
描述：JS DCAS 应用通过调用这个方法停止接收 ECM。 JS DCAS 应用极少有机会调用这个方法。如果希望重新启动 ECM 接收，则需要重新调用 CASManager.startEcmLoading。
参数：CASModule casModule - CAS 模块实例。
CASSession casSession - 从 CASModule.onSTartDescrambling 获得的 CAS Session。
返回：成功 - CASModuleManager.ACTION_OK。
失败 - 返回如下错误值：
CASModuleManager.ACTION_ERROR_INVALID_PARAMETERS - 无效参数
CASModuleManager.ACTION_ERROR_DRIVER - 底层错误

B.2.9.2.19 stopInbandEmmLoading

原型：{number}stopInbandEmmLoading(casModule)
描述：JS DCAS 应用通过调用这个方法停止接收带内 EMM。 这个方法很少需要调用。重新调用 CASManager.startInbandEmmLoading 可以重新开始接收。
参数：CASModule casModule - CAS 模块实例。
返回：成功 - CASModuleManager.ACTION_OK。
失败 - 返回如下错误值：
CASModuleManager.ACTION_ERROR_INVALID_PARAMETERS - 无效参数
CASModuleManager.ACTION_ERROR_DRIVER - 底层错误。

B.2.10 JSDCAS.CASPacketEvent类

B.2.10.1 getCableModemFilter

原型：{number|string}getCableModemFilter()
描述：返回过滤这个包所用的 cableModemFilter。如果没有使用 Cable Modem DSG，则返回空。
返回：ADSG 模式 - 返回 CAS Tunner ID，数字类型。
BDSG 模式 - 返回虚拟 MAC 地址。

B.2.10.2 getPacketData

原型：{uint8Array}getPacketData()
描述：返回数据包中的数据。

B.2.10.3 getPacketHeader

原型：{uint8Array}getPacketHeader()
描述：返回数据包的包头 包头包含了 IP 地址和 UDP 头。

B.2.10.4 getSourceURL

原型：{string}getSourceURL()

描述：返回通过 UDP 接收 CAS 数据包的源地址。
返回：源地址字符串。

B.2.11 JSDCAS.CASSession类

平台为每个解扰请求都生成一个 CAS Session 对象。它包含了唯一的 session ID，以及所有关于播放节目的信息，还包含了 PMT 中与本次解扰相关的 CA 描述符。对每个解扰请求，JS DCAS 应用通过 CASModule.onStartDescrambling 获得 CAS Session。CASSession 对象还可以用于接收 CAT 更新消息的场景，在此场景中，CASSession 对象只会有部分字段有效。

B.2.11.1 GetCasDescriptor

原型：{CASDescriptor}getCasDescriptor()
描述：返回 CA 描述符，可能是来自 PMT，也可能来自 CAT。
返回：CA 描述符对象实例。

B.2.11.2 getChannelNumber

原型：{number}getChannelNumber()
描述：返回频道号。这个方法是可选的，尤其是对于那些不能确定频道号的平台，可以返回 0。
返回：频道号。

B.2.11.3 getNetworkId

原型：{number}getNetworkId()
描述：返回原始网络 ID。这个方法是可选的，如果平台无法获得，可以返回 0。
返回：原始网络 ID 。

B.2.11.4 getOperationType

原型：{number}GetOperationType ()
描述：返回操作类型。。
返回：操作类型值：

CASSession.OPERATION_TYPE_PRESENTATION
CASSession.OPERATION_TYPE_RECORDING
CASSession.OPERATION_TYPE_BUFFERING
CASSession.OPERATION_TYPE_SECOND_DEVICE 。

B.2.11.5 getProgramNumber

原型：{number}getProgramNumber ()
描述：返回节目号。

B.2.11.6 getServiceIdentifier

原型：{number|*}getServiceIdentifier ()
描述：返回正在解扰的服务标识符，此标识符是一个值或者一个对象；

B.2.11.7 getSessionId

原型：{number}getSessionId ()
描述：返回 Session ID 。

B.2.11.8 getStreamPath

原型：*{Uint8Array}* getStreamPath ()
描述：返回 StreamPath 数据。

B.2.11.9 getStreamPIDs

原型：*{Array}* getStreamPIDs()
描述：返回 Stream PIDs 列表 。

B.2.11.10 getStreamTypes

原型：*{Array}* getStreamTypes()
描述：返回 StreamTypes 列表，见 ISO13818-1(表 2-36) 。

B.2.11.11 getTransmitterScrambingMode

原型：{number}getTransmitterScrambingMode ()
描述：返回加扰模式值。

B.2.11.12 getTransportStreamId

原型：{number}getTransportStreamId ()
描述：返回正在解扰的 TS ID 。

B.2.11.13 getTunerId

原型：{number}getTunerId()
描述：返回正在解扰节目使用的 Tuner ID 。

B.2.12 JSDCAS.CASStatus类

JS DCAS 应用通过这个对象，向平台传递解扰状态。每次解扰状态有变化，JS DCAS 应用应该调用 CASModuleManager.sendDescramblingEvent 通知平台。平台接收到这个状态变化后，可以自己处理，也可以转发给 UI 应用。UI 应用可以简单的弹出 OSD 通知用户解扰成功或者失败，也可以通过解析 CASStatus 对象中的附加信息显示具体因为什么原因失败。这些额外的信息格式是项目相关的。如果 JS DCAS 应用没有附加额外信息，UI 应用获得 CASStatus 对象中的 token 也可以使用这个 token 通过 IPC 或者其他平台提供的手段与 JS DCAS 应用通信获得更多信息。

B.2.12.1 状态值列表

JSDCAS.CASStatus.CONTENT_PROBLEM_COMMUNICATION_ERROR
JSDCAS.CASStatus.CONTENT_PROBLEM_GENERAL_ERROR
JSDCAS.CASStatus.CONTENT_PROBLEM_HARDWARE_FAILURE_BEIDOU
JSDCAS.CASStatus.CONTENT_PROBLEM_HARDWARE_FAILURE_HSM
JSDCAS.CASStatus.CONTENT_PROBLEM_HARDWARE_FAILURE_SOC

JSDCAS.CASStatus.CONTENT_PROBLEM_INVALID_CA_PACKET
JSDCAS.CASStatus.CONTENT_PROBLEM_MISSING_KEY
JSDCAS.CASStatus.CONTENT_PROBLEM_NO_CA_PACKET
JSDCAS.CASStatus.CONTENT_PROBLEM_NONE
JSDCAS.CASStatus.CONTENT_PROBLEM_POSITION_NOT_LEGAL_BLOCKING
JSDCAS.CASStatus.CONTENT_PROBLEM_POSITION_NOT_LEGAL_GRACE
JSDCAS.CASStatus.CONTENT_PROBLEM_POSITION_NOT_LEGAL_WARNING
JSDCAS.CASStatus.CONTENT_PROBLEM_POSITION_NOT_READY_BLOCKING
JSDCAS.CASStatus.CONTENT_PROBLEM_POSITION_NOT_READY_WARNING
JSDCAS.CASStatus.CONTENT_PROBLEM_PR_LIMIT_EXCEEDED
JSDCAS.CASStatus.CONTENT_PROBLEM_SERVICE_NOT_AUTHORIZED
JSDCAS.CASStatus.CONTENT_PROBLEM_SUBSCRIBER_NOT_AUTHORIZED
JSDCAS.CASStatus.CONTENT_PROBLEM_TRANSITION_WARNING

B.2.12.2 方法

B.2.12.2.1 getCasToken

原型：{number}getCasToken()

描述：返回 CAS token。 如果平台将 CASStatus 的信息转发给 UI 应用，UI 应用可以使用这个 token 向 JS DCAS 应用发起请求，以获得更详细的状态信息。 请求的方式，是平台决定的，比如 IPC。

B.2.12.2.2 getMajorContentProblem

原型：{number}getMajorContentProblem()

描述：返回不能观看节目的主要错误值。

B.2.12.2.3 getStatusData

原型：{ArrayBuffer}getStatusData()

描述：返回解扰状态扩展数据。通过扩展数据，UI 应用可以显示更详细的关于解扰状态的信息。

返回：返回数据是 ArrayBuffer 类型，如果没有扩展数据可以提供，应该返回 null。

B.2.12.2.4 isSuccess

原型：{boolean}isSuccess()

描述：返回解扰成功与否。

返回：ture-成功，false-失败 。

B.2.13 JSDCAS.TeeController类

JS DCAS 和 TEE 通信的控制器。

sendCommandToTEE

原型：{TeeRetVal}sendCommandToTEE(teeAppUUID,commandId,inputData,applicationContext)

描述：JS DCAS 应用通过这个方法向 TEE 中运行的 TA 发送命令。

参数：

Uint8Array teeAppUUID - TA 的 UUID，16 字节。每个 CA 厂家都有不同的 ID。

number commandId - TEE 通信中的 Command ID。由各个 CA 厂家自己定义。

Uint8Array inputData - 发送给 TA 的数据。

* applicationContext - 应用上下文，平台相关。通常在初始化的时候由平台提供给应用。

返回：

返回 TeeRetVal 对象。这个对象包含从 TA 返回的数据、错误等信息。

B.2.14 JSDCAS.TeeRetVal类

这个对象由 TeeController.sendCommandToTEE 返回。对象中包含从 TEE 返回的数据、错误等信息。

B.2.14.1 返回值列表

JSDCAS.TeeRetVal.TEEC_ERROR_ACCESS_CONFLICT
JSDCAS.TeeRetVal.TEEC_ERROR_ACCESS_DENIED
JSDCAS.TeeRetVal.TEEC_ERROR_BAD_FORMAT
JSDCAS.TeeRetVal.TEEC_ERROR_BAD_PARAMETERS
JSDCAS.TeeRetVal.TEEC_ERROR_BAD_STATE
JSDCAS.TeeRetVal.TEEC_ERROR_BUSY
JSDCAS.TeeRetVal.TEEC_ERROR_CANCEL
JSDCAS.TeeRetVal.TEEC_ERROR_COMMUNICATION
JSDCAS.TeeRetVal.TEEC_ERROR_EXCESS_DATA
JSDCAS.TeeRetVal.TEEC_ERROR_FSYNC_DATA
JSDCAS.TeeRetVal.TEEC_ERROR_GENERIC
JSDCAS.TeeRetVal.TEEC_ERROR_INVALID_CMD
JSDCAS.TeeRetVal.TEEC_ERROR_ITEM_NOT_FOUND
JSDCAS.TeeRetVal.TEEC_ERROR_MAC_INVALID
JSDCAS.TeeRetVal.TEEC_ERROR_NO_DATA
JSDCAS.TeeRetVal.TEEC_ERROR_NOT_IMPLEMENTED
JSDCAS.TeeRetVal.TEEC_ERROR_NOT_SUPPORTED
JSDCAS.TeeRetVal.TEEC_ERROR_OUT_OF_MEMORY
JSDCAS.TeeRetVal.TEEC_ERROR_READ_DATA
JSDCAS.TeeRetVal.TEEC_ERROR_REGISTER_EXIST_SERVICE
JSDCAS.TeeRetVal.TEEC_ERROR_RENAME_OBJECT
JSDCAS.TeeRetVal.TEEC_ERROR_SECURITY
JSDCAS.TeeRetVal.TEEC_ERROR_SEEK_DATA
JSDCAS.TeeRetVal.TEEC_ERROR_SERVICE_NOT_EXIST
JSDCAS.TeeRetVal.TEEC_ERROR_SESSION_MAXIMUM
JSDCAS.TeeRetVal.TEEC_ERROR_SESSION_NOT_EXIST
JSDCAS.TeeRetVal.TEEC_ERROR_SHORT_BUFFER
JSDCAS.TeeRetVal.TEEC_ERROR_TAGET_DEAD_FATAL
JSDCAS.TeeRetVal.TEEC_ERROR_TRUNCATE_OBJECT
JSDCAS.TeeRetVal.TEEC_ERROR_TRUSTED_APP_LOAD_ERROR

JSDCAS.TeeRetVal.TEEC_ERROR_WRITE_DATA
JSDCAS.TeeRetVal.TEEC_ORIGIN_API
JSDCAS.TeeRetVal.TEEC_ORIGIN_COMMS
JSDCAS.TeeRetVal.TEEC_ORIGIN_JS_LAYER
JSDCAS.TeeRetVal.TEEC_ORIGIN_NOT_SPECIFIED
JSDCAS.TeeRetVal.TEEC_ORIGIN_TEE
JSDCAS.TeeRetVal.TEEC_ORIGIN_TRUSTED_APP
JSDCAS.TeeRetVal.TEEC_SUCCESS

B.2.14.2 方法

B.2.14.2.1 getOriginCode

原型：{number}getOriginCode()
描述：返回 origin code。

B.2.14.2.2 getResponseData

原型：{Uint8Array}getResponseData()
描述：得到返回自 TA 的数据。
返回：TA 返回的数据。对于某些命令可以为 null。如果调用或者通信发生错误，也返回 null。

B.2.14.2.3 getReturnCode

原型：{number}getReturnCode()
描述：返回return code。

B.3 安全芯片层级密钥驱动接口

B.3.1 数据类型及结构定义

B.3.1.1 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。
```

B.3.1.2 接口返回值枚举类型

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

B.3.2 接口定义

B.3.2.1 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。

B.3.2.2 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。

B.3.2.3 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。

B.3.2.4 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

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厂商标识描述符字节：

0：VENDOR_ID_DSCR_TAG = 0x05

1：描述符长度 = 2

2-3：CA厂商标识

B.3.2.5 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=AE、2=SM4
CA厂商标识描述符字节：
0：VENDOR_ID_DSCR_TAG = 0x05
1：描述符长度 = 2
2-3：CA厂商标识
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

B.3.2.6 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

B.4 硬件安全模块应用程序接口

B.4.1 数据类型及结构定义

B.4.1.1 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

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

B.4.1.2 接口返回值枚举类型

```
/* 访问HSM成功 */
#define HSM_RESULT_OK 0
/* 应用程序无访问HSM的权限 */
#define HSM_RESULT_ERROR_SECURITY 1
/* 参数校验失败 */
#define HSM_RESULT_ERROR_INVALID_PARAMETERS 2
/* 操作类型不支持 */
#define HSM_RESULT_ERROR_NOT_SUPPORTED 3
/* 长度或偏移量超过范围 */
#define HSM_RESULT_ERROR_OUT_OF_RANGE  4
/* HSM驱动内部错误导致访问失败 */
#define HSM_RESULT_ERROR_DRIVER  5
/* 读写HSM操作失败 */
#define HSM_RESULT_ERROR_IO  6
/* HSM设备忙，稍后再试 */
```

```
#define HSM_RESULT_ERROR_BUSY  7
/* HSM通讯层错误 */
#define HSM_RESULT_ERROR_API_COMMUNICATION  8
/* 应用程序所提供的缓冲区太小，应用应按照函数输出的缓冲区尺寸进行调整 */
#define HSM_RESULT_ERROR_INSUFFICIENT_BUFFER 9
/* 一般性错误 */
#define HSM_RESULT_ERROR_OPERATION_FAILED 10
```

B.4.2 接口定义

B.4.2.1 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，具体失败原因见返回枚举类型定义。

B.4.2.2 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：未激活，1：已激活，2：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，具体失败原因见返回枚举类型定义

B.4.2.3 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：硬件安全模块厂商证书缓冲区长度

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厂商标识，当芯片内部没有CAS厂商标识时，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厂商证书长度

hsmVendorCertificate：硬件安全模块厂商证书内容

返回值：

HSM_RESULT_OK：访问成功

其他：访问失败，具体失败原因见返回枚举类型定义

B.4.2.4 TEE_HSM_GetHsmCapabilities

获得硬件安全模块存储容量及读写能力信息

原型：

```
HSM_Result TEE_HSM_GetHsmCapabilities(uint32_t* secureStorageSize,
                                      uint32_t* publicStorageSize,
                                      uint32_t* maxWriteSecureLength,
                                      uint32_t* maxReadSecureLength,
                                      uint32_t* maxReadPublicLength);
```

输入：

无。

输出：

secureStorageSize: SAC认证存储区的大小

publicStorageSize: 公共存储区的大小

maxWriteSecureLength:写入SAC认证存储区时，单次事务所能写入的最大数据量

maxReadSecureLength: 从SAC认证存储区读出数据时，单次事务所能读出的最大数据量

maxReadPublicLength: 从公共存储读出数据时，单次事务所能读出的最大数据量

返回值：

HSM_RESULT_OK：访问成功

其他：访问失败，具体失败原因见返回枚举类型定义

B.4.2.5 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，具体失败原因见返回枚举类型定义

B.4.2.6 TEE_HSM_GetHsmActivationInfo

从硬件安全模块中读取激活信息中CA厂商专属的数据。

原型：

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

输入：

vendorId：CA厂商标识

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

HSM_RESULT_ERROR_INVALID_PARAMETERS:输入参数错误或CA厂商标识与HSM已有CA厂商标识不匹配

其他：访问失败，具体失败原因见返回枚举类型定义

B.4.2.7 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厂商标识

vendorCertificateLength：CA厂商证书长度

vendorCertificate：CA厂商证书

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10^6

latitude：终端所在位置纬度，取值为实际纬度乘以10^6

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，具体失败原因见返回枚举类型定义

B.4.2.8 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厂商标识

vendorCertificateLength：CA厂商证书长度

vendorCertificate：CA厂商证书

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，具体失败原因见返回枚举类型定义

B.4.2.9 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厂商标识

vendorCertificateLength：CA厂商证书长度

vendorCertificate：CA厂商证书

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：硬件安全模块访问句柄长度，CAS用户端可信应用软件应分配16字节的缓冲区

hsmSacHandle：硬件安全模块访问句柄，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，具体失败原因见返回枚举类型定义

B.4.2.10 TEE_HSM_Read

从硬件安全模块中读取数据

原型：

```
HSM_Result TEE_HSM_Read(unint16_t vendorId,
                        int hsmSacHandleLength,
                        uint8_t* hsmSacHandle,
```

```
                        uint32_t offset,
                        uint32_t* length,
                        unint8_t* data);
```

输入：

vendorId：CA厂商标识

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，具体失败原因见返回枚举类型定义

B.4.2.11 TEE_HSM_Write

写入数据到硬件安全模块

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厂商标识

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，具体失败原因见返回枚举类型定义

B.4.2.12 TEE_HSM_ReadPositionParameters

从硬件安全模块读取存储的所设置的位置信息

原型：

```
HSM_Result TEE_HSM_ReadPositionParameters(unint16_t vendorId,
                                          int hsmSacHandleLength,
                                          uint8_t* hsmSacHandle,
                                          int* longitude,
                                          int* latitude,
                                          uint32_t* radius);
```

输入：

vendorId：CA厂商标识

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10^6

latitude：设置位置的纬度，取值为实际纬度乘以10^6

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，具体失败原因见返回枚举类型定义

B.4.2.13 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，具体失败原因见返回枚举类型定义

B.4.2.14 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,
                                                uint8_t* data);
```

输入：

vendorId：CA厂商标识

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，具体失败原因见返回枚举类型定义

B.4.2.15 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，必须在第一次调用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厂商标识

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：保留，1：保留，2：SM4

输出：

无

返回值：

HSM_RESULT_OK：访问成功

其他：访问失败，具体失败原因见返回枚举类型定义

B.4.2.16 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厂商标识

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：保留，1：保留，2：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，具体失败原因见返回枚举类型定义

B.4.2.17 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);
```

输入：

vendorId：CA厂商标识

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：

无

返回值：

HSM_RESULT_OK：访问成功

其他：访问失败，具体失败原因见返回枚举类型定义

B.5 定位模块（北斗）驱动接口

B.5.1 数据类型及结构定义

B.5.1.1 基本数据类型

```
typedef unsigned int TEE_Result;
typedef unsigned int uint32_t;
```

B.5.1.2 接口返回值枚举类型

```
#define TEE_SUCCESS 0
#define TEE_BEIDOU_NOT_READY 1
```

B.5.2 接口定义

B.5.2.1 TEE_Beidou_GetSoftwareVersion

获取定位模块版本。

原型：

```
TEE_Result TEE_Beidou_GetSoftwareVersion(char * version,uint32_t length);
```

输入：

version: 应用程序分配的缓冲区

length： 缓冲区长度

输出：

version：返回的版本信息字符串，应用程序负责分配和释放此缓冲区。

返回值：

TEE_SUCCESS：获取版本信息成功

TEE_BEIDOU_NOT_READY：定位模块尚未就绪

其他：硬件故障

B.5.2.2 TEE_Beidou_GetPositionParameters

获取定位位置信息及时间信息。

原型：

```
TEE_Result TEE_Beidou_GetPositionParameters(int * longitude,
                                            int * latitude,
                                            uint32_t * timestamp);
```

输入：

longitude：经度缓冲区，应用程序负责分配和释放

latitude：纬度缓冲区，应用程序负责分配和释放

timestamp： 时间信息缓冲区，应用程序负责分配和释放

输出：

longitude：经度信息

latitude：纬度信息

timestamp： 时间信息

返回值：

TEE_SUCCESS：获取位置及时间信息成功

TEE_BEIDOU_NOT_READY：定位模块尚未就绪

其他：硬件故障

B.5.2.3 TEE_Beidou_GetSignalParameters

获取定位卫星信号参数。

原型：

```
TEE_Result TEE_Beidou_GetSignalParameters(uint32_t * numfix,
                                          uint32_t * cn0bds,
                                          uint32_t * cn0gps);
```

输入：

numfix：输出缓冲区，应用程序负责分配和释放

cn0bds：输出缓冲区，应用程序负责分配和释放

cn0gps：输出缓冲区，应用程序负责分配和释放

输出：

Numfix：已经锁定的卫星数量

Cn0bds：北斗信号载噪比

Cn0gps：GPS信号载噪比

返回值：

TEE_SUCCESS：获取卫星信息成功

TEE_BEIDOU_NOT_READY：定位模块尚未就绪

其他：硬件故障

B.5.2.4 TEE_Beidou_ CalculateDistance

计算两点之间距离。

原型：

```
TEE_Result TEE_Beidou_CalculateDistance(int longitudeA,
                                        int latitudeA,
                                        int longitudeB,
                                        int latitudeB);
```

输入：

longitudeA：A点的经度值，其值为实际经度值的10^6倍

latitudeA：A点的纬度值，其值为实际纬度值的10^6倍

longitudeB：B点的经度值，其值为实际经度值的10^6倍

latitudeB：B点的纬度值，其值为实际纬度值的10^6倍

输出：

无

返回值：

两点之间的距离，单位为米。

B.6 其他GP扩展接口

B.6.1 加解密及签名校验接口

B.6.1.1 数据类型及结构定义

B.6.1.1.1 基本数据类型

```
typedef unsigned int TEE_Result;
typedef unsigned int uint32_t;
```

B.6.1.1.2 接口返回值枚举类型

```
#define TEE_SUCCESS 0;
```

B.6.1.2 接口定义

B.6.1.2.1 TEE_SM2_Verify

验证数字签名。

原型：

```
TEE_Result  TEE_SM2_Verify( unsigned char *pub_key,
                            size_t pub_key_len,
                            unsigned char *hash,
                            size_t hash_len,
                            unsigned char *sig,
                            size_t sig_len)
```

输入：

pub_key: 公钥值，包括 1 字节头部及 64 字节密钥数据，共 65 字节

pub_key_len: 公钥长度

hash: 签名所对应的 SM3 散列值，32 字节

hash_len：散列值的长度

sig：签名值，64 字节

sig_len：签名长度

输出：

无

返回值：

TEE_SUCCESS：签名验证成功
其他：失败

B.6.1.2.2 TEE_Perform_SM3

计算SM3散列值。
原型：

```
TEE_Result   TEE_Perform_SM3(unsigned char* dataIn,
                             unsigned int dataInLen,
                             unsigned char* result);
```

输入：

dataIn: 需要计算散列值的原始输入数据

dataInLen: 原始输入数据长度

输出：

result: 计算所得散列值，32字节长
返回值：
TEE_SUCCESS：计算成功
其他：失败

B.6.1.2.3 TEE_SM2_Encrypt

使用公钥进行对数据进行SM2加密。
原型：

```
TEE_Result  TEE_SM2_Encrypt(
                           unsigned char *pub_key,
                            size_t pub_key_len,
                             uint8_t *inputData,
                             uint32_t inputData_size
                            uint8_t *outputData,
                            uint32_t outputData_size)
```

输入：
pub_key: 公钥值，包括1 字节头部及64字节密钥数据，共65字节
pub_key_len:
公钥长度
inputData:
输入数据
inputData_size:输入数据长度
outputData:输出加密结果数据的缓冲区
outputData_size:输出数据长度，输出长度必须是输入长度+96
输出：

outputData: 加密后的输出结果，顺序C1|C3|C2。

返回值：

TEE_SUCCESS：加密成功

其他：失败

B.6.1.2.4 TEE_Perform_CRC

计算CRC校验值。

原型：

```
TEE_Result   TEE_Perform_CRC(int mode,
                             unsigned char* dataIn,
                             unsigned int dataInLen,
                             unsigned char* result);
```

输入：

mode: CRC 计算模式，0=CRC16, 1=CRC32

dataIn: 需要校验值的原始输入数据

dataInLen: 原始输入数据长度

输出：

result: 计算所得校验值，若是CRC16,其结果为2字节，若是CRC32,其值为4字节

返回值：

TEE_SUCCESS：计算成功

其他：失败

B.6.1.2.5 TEE_GenerateRandom

计算CRC校验值。

原型：

```
TEE_Result TEE_GenerateRandom(void* randomBuffer, size_t randomBufferLen);
```

输入：

randomBuffer: 存储随机数的缓冲区

randomBufferLen: 缓冲区长度

输出：

randomBuffer: 存储有随机数的缓冲区

返回值：

TEE_SUCCESS：计算成功

其他：失败

B.6.1.2.6 TEE_SM4_Encrypt

实现SM4加密算法。

原型：

```
TEE_Result    TEE_SM4_Encrypt (int mode,
                               uint8_t *IV,
                               uint8_t *key,
                               uint8_t *inputData,
                               uint8_t *outputData,
                               uint32_t data_size);
```

输入：

mode: 计算模式，0=ECB, 1=CBC

IV: 初始化向量，当模式为CBC时，此为16字节数据，当模式为ECB时，忽略该数据

key: 加密密钥，16字节长

inputData: 输入的待加密数据

outputData:输出加密后的结果的缓冲区

data_size: 输入/输出数据长度，必须为16的整数倍，否则加密失败

输出：

outputData: 加密后的输出结果

返回值：

TEE_SUCCESS：计算成功

其他：失败

B.6.1.2.7 TEE_SM4_Decrypt

实现SM4解密算法。

原型：

```
TEE_Result    TEE_SM4_Decrypt (int mode,
                               uint8_t *IV,
                               uint8_t *key,
                               uint8_t *inputData,
                               uint8_t *outputData,
                               uint32_t data_size);
```

输入：

mode: 计算模式，0=ECB, 1=CBC

IV: 初始化向量，当模式为CBC时，此为16字节数据，当模式为ECB时，忽略该数据

key: 解密密钥，16 字节长

inputData: 输入的待解密数据

outputData:输出解密后的结果的缓冲区

data_size: 输入/输出数据长度，必须为 16 的整数倍，否则解密失败

输出：

outputData: 加密后的输出结果

返回值：

TEE_SUCCESS：计算成功

其他：失败

B.6.2 内存管理接口

B.6.2.1 数据类型及结构定义

B.6.2.1.1 基本数据类型

无

B.6.2.1.2 接口返回值枚举类型

无

B.6.2.2 接口定义

B.6.2.2.1 TEE_MemFill

内存填充。

原型：

```
void TEE_MemFill(void *buffer, uint32_t x, uint32_t size);
```

输入：

buffer: 待填充的内存缓冲区起始地址

x: 待填充的值

size: 待填充的内存缓冲区长度

输出：

无

返回值：

无

B.6.2.2.2 TEE_MemMove

内存内容移动。

原型：

void TEE_MemMove(void *dest, void *src, uint32_t size);

输入：

dest: 目标内存缓冲区起始地址

src: 源内存缓冲区起始地址

size: 待移动的内存缓冲区长度

输出：

无

返回值：

无

B.6.3 其他接口

B.6.3.1 数据类型及结构定义

B.6.3.1.1 基本数据类型

无

B.6.3.1.2 接口返回值枚举类型

无

B.6.3.2 接口定义

B.6.3.2.1 TEE_Printf_Func

打印函数。

原型：

void TEE_Printf_Func(const char * fmt, …);

输入：

fmt: 格式化参数列表

输出：

无

返回值：

无

附 录 C
（规范性附录）
硬件安全模块相关要求

C.1 概述

HSM 是单向可下载条件接收系统的核心功能组件，是一种标准化的安全芯片，可在不同的 CA 厂商中共享，可以用作代替传统条件接收系统终端的智能卡硬件，任何授权的 CA 系统厂商都可以利用这些功能来实现其安全解决方案。

C.2 HSM基础功能

C.2.1 激活

HSM在被CA使用前需要进行激活，非激活状态下HSM的功能将被限制，在收到前端发出的激活指令并完成激活后，HSM的安全功能才能被使用。

激活过程是HSM对CA系统进行验证并初始化的过程，确保授权的CA才能使用该HSM，提高了系统的安全性。

C.2.2 安全认证通道

HSM 与 SoC 之间通过 PairK 建立的安全认证通道，HSM 可对外提供安全相关的服务。

安全认证通道依赖于 HSM 芯片的激活，即只有 HSM 激活后，SAC 才可以建立成功。

C.2.3 CA数据安全存储

HSM 提供通用的安全存储功能，对 CA 私有数据进行安全存储，通过建立安全认证通道实现安全性，即只有 HSM 与 SoC 之间相互验证并建立安全通道后，才可访问 CA 存储区域的数据。

HSM 安全存储功能提供一般化的存储区域，如何分配和使用这些区域，则由用户决定。

C.2.4 层级密钥处理

层级密钥处理模块是由 HSM 提供的安全服务，其主要功能是生成参与终端安全芯片层级密钥运算的数据。层级密钥处理模块支持 3 级层级密钥体系，层级密钥的输出数据被 CREEK 重加密，加密结果作为 HSM 最终输出结果提供给 SoC。

C.2.5 关系描述

激活和安全认证通道是 HSM 两个基础安全功能，HSM 具体服务依赖于两个功能才能实现，这种依赖关系保障了 HSM 的安全性，关系描述见表 C.1。

表 C.1 HSM 服务关系描述

类别	服务名称	是否需要安全认证通道	是否需要激活	服务描述
基本功能	读取公共数据	否	否	读取公共数据
	读取软件版本	否	否	读取 HSM 内部软件版本信息
激活功能	生成激活请求消息	否	否	生成激活请求消息并用 HSM 私钥对消息进行签名
	处理主激活消息	否	否	校验收到的主激活消息并将解析获得的数据保存在 HSM 中
	处理辅助激活消息	否	否	校验收到的辅助激活消息并将解析获得的数据保存在 HSM 中，在此之前，HSM 应该已经正确地收到并处理了主激活消息
	读取最新时间戳	否	否	读取保存在 HSM 中最新的由主激活消息携带的时间戳
	反激活 HSM	否	否	使 HSM 进入未激活状态，并删除所有已经保存的 CA 相关信息
	读取激活信息	否	是	读取芯片激活相关信息，包括由辅助激活消息所携带的 CA 私有数据
	读取位置信息	是	是	读取由辅助激活消息所携带并存储在 HSM 中的位置信息数据
安全存储	读取 SAC 认证存储区数据	是	是	通过安全认证通道读取 SAC 认证存储区域中的数据
	写入读取 SAC 认证存储区数据	是	是	通过安全认证通道将数据写入 SAC 认证存储区域中
	读取公共存储区数据	否	是	读取公共数据区域数据
	写入公共存储区数据	是	是	将数据写入公共数据区域
层级密钥处理	使用层级密钥处理模块功能	是	是	处理输入数据并生成 CW
安全认证通道	建立安全认证通道	否	是	初始化安全认证通道
	关闭安全认证通道	是	是	关闭 SAC

C.3 典型激活流程

C.3.1 总体流程

通过激活流程，HSM 将自己注册到对应 CA 系统的前端，并从前端接收相应的 CA 信息及密钥数据，从而使得 HSM 能够按照对应 CA 系统要求的方式工作。激活流程包含三个基本操作步骤：

a） HSM 生成激活请求消息并发送回前端；

b） HSM 收到并处理前端下发的主激活消息；

c） HSM 收到并处理前端下发的辅助激活消息。

DCAS 用户端软件程序向 HSM 发出请求，作为响应，HSM 将生成激活请求消息并签名。该消息包含一组终端对应的信息，并用 HSM 所持有的个性化私钥进行签名。之后激活请求消息被发往前端进行处理。

HSM 的激活依赖于从前端收到两种不同的消息：主激活消息以及辅助激活消息。在正确的收到并处理这两个消息之前，HSM 都应处于未激活状态，从而不能对外提供安全存储以及加解密相关的功能。

DCAS 用户端软件在收到前端下发的主激活消息后，将其转发给 HSM。主激活消息中包含密钥信息可用来将终端与 HSM 进行配对绑定，同时，该消息还携带有用于生成内容保护控制字的关键密钥信息。

在收到并处理了主激活信息后，HSM 仍处在未激活状态，此时它在继续等待与主激活信息匹配的辅助激活信息。这里的匹配是指两个消息中所携带的时间戳以及 CA 厂商标识必须相同。一旦正确的收到并处理了以上两个匹配的激活消息，HSM 则被认为进入了激活状态，从而可以对外提供核心的安全功能。而在被激活之前，HSM 将会拒绝一切相关安全功能的服务请求。

根据直播星系统的要求，DCAS 系统应该是可更新的，因此 HSM 应可以接收并处理由系统前端多次发送的以上两种激活消息。

是否收到主激活消息并不依赖于终端是否已经发送过激活请求消息。在某些应用场景下，前端可以在未收到终端生成的激活请求消息时，主动发送主激活消息。

C.3.2 读取软件版本

通过调用本功能，可以获取 HSM 中软件的版本信息，通常是以数字的形式返回。

C.3.3 读取公共数据

通过调用本功能，终端客户端程序可以获得存储在 HSM 公共存储区中的数据。

返回数据如下：

——HSM 证书；

——HSMID；

——激活状态；

——是否收到主激活消息的标志。

C.3.4 生成激活请求消息

首先，DCAS 用户端软件获得如下信息：

——Vendor_SysID；

——CA 厂商的证书；

——ChipID；

——通过位置模块获得的终端位置信息；

——通过位置模块获得的当前时间信息。

DCAS 用户端软件将上述信息传递给 HSM。

HSM 进行以下步骤，从而生成激活请求消息：

a) 根据 TA 根证书对 CA 厂商证书进行校验，使用 SM3 哈希算法生成证书摘要信息，使用 SM2 椭圆曲线算法计算数字签名。另外，HSM 会检查以下证书信息，以保证 CA 证书格式及内容符合 C.6 关于证书格式的约定。
——证书版本；
——证书签名算法；
——签发者信息；
——证书过期日期 ---此日期应处在当前日期的未来；
——Subject:OU—对于正式版本的 HSM 芯片，该字段必须为 “PRODUCTION” 。而对于实验调试版本的 HSM，该字段必须为 “TEST”；
——Subject:CN—该字段必须以 “CHINA DTH CA VENDOR CERTIFICATE”字符串开头；
——Subject 公钥信息：公钥算法；
——Subject 公钥信息：公钥数据，其内容应以非压缩格式保存，也就是要以 0x04 开始；
——Subject 公钥信息：证书拥有者域名信息；
——扩展信息：证书签发者标识；
——扩展信息：证书拥有者标识；
——扩展信息：密钥使用约束，该字段必须为”digitalSignature”，即只用于签名；
——扩展信息：基本约束；
——签名算法信息。

b) 当以上任何信息有误时，停止处理过程。只有当以上信息全部校验通过后，HSM 解析出 Vendor_SysID，并转入下一步处理。

c) 用以下信息生成激活请求消息：
——HSMID；
——ChipID；
——Vendor_sysID；
——位置信息；
——当前时间。

d) HSM 使用 HSM 私钥对以上生成的激活请求消息进行签名

e) 最后，HSM 返回所生成的激活请求消息及对应的数字签名。消息包括 C.5.1 中所列出的 0 到 17 字段的内容。

C.3.5 处理主激活消息

当收到主激活消息时，HSM 首先校验消息的签名，校验成功则进行下一步消息内容的解析处理。

需要重点强调的是，一个已经处在激活状态的 HSM 是允许再次收到主激活消息并进行处理的。当此情况发生时，HSM 退回非激活状态，只有收到并正确处理与之匹配的新的辅助激活消息后，HSM 才会被重新激活。当新收到的主激活消息被校验通过后，HSM 会用新的主激活消息所携带的信息覆盖原有的 HSM 已保存的信息，这一操作应为原子操作，以避免 HSM 处于激活状态时，其 NVM 中保存来自不同主激活消息的信息或者来自不匹配的辅助激活消息的信息。

当已被激活的 HSM 收到一个新的合法的主激活消息时，根据其携带的 CAS 厂商标识，其后续处理分为两种情况：与已激活的 CAS 厂商标识不同或者相同。

如果 CAS ID 与之前不同，HSM 会将之前保存的与已激活 CAS ID 对应的所有数据删除，包括安全存储数据以及辅助激活消息。

如果 CAS ID 与之前相同，HSM 将会进入非激活状态，并且删除之前收到的辅助激活消息，于此同时，之前存储在 SAC 认证存储区域中的数据将被保留。

在主激活消息处理过程中，HSM 将收到以下输入信息：

——主激活消息；

——CAS 厂商证书。

之后 HSM 进行如下操作：

a) 使用 TA 根证书对 CA 厂商证书进行校验，使用 SM3 哈希算法生成证书摘要信息，使用 SM2 椭圆曲线算法计算数字签名。

b) 使用 CA 厂商证书校验主激活消息签名，其生成摘要信息使用 SM3 哈希算法，计算数字签名使用 SM2 椭圆曲线算法。

——如果校验失败，则处理过程终止并返回错误；

——如果校验通过，则继续后续处理。

c) 校验主激活消息首字节合法性：

——其中，前半字节为消息版本号，在本文中，其值必须等于 1；

——后半字节为消息类型，对于主激活消息，其值必须等于 1；

——因此，主激活消息的首字节合法值应为 0x11。

d) 比较主激活消息中的当前时间戳与 HSM 中保存的上一次收到消息的时间戳：

——如果当前时间戳大于或等于上一次收到的时间戳，则继续后续处理；

——否则，处理过程终止并返回错误。

e) 检查 CA 厂商证书中“Subject:O”字段 是否与主激活消息中 Vendor_SysID 相等。注意：CA 厂商证书中 CA 厂商 ID 是用 4 个十六进制的字符表示的，而在主激活消息中是用 2 个字节表示的。

f) 使用 HSM 私钥以及主激活消息中的密钥数据，可以解密出 HSM 根密钥 K3_HSM，HSM 首先使用私钥，用 SM2 椭圆曲线算法解密出 16 字节的密钥数据。之后再使用该密钥数据，通过 SM4 算法解密出 16 字节的 HSM 根密钥。

g) 当收到主激活消息时，若 HSM 处在已激活状态，则 HSM 将自己标识为非激活状态，并且把上一次收到并保存的附属激活消息标记为失效。

h) 当收到主激活消息时，若 HSM 处在已激活状态，并且收到的新的 CAS 厂商标识与之前激活 HSM 的 CAS ID 不同，HSM 需将其 SAC 认证存储区域保存的数据全部清除。

i) 以下数据应被 HSM 保存在相应的私有数据区域（此区域不能通过 HSM 的数据访问接口访问）：

———ChipID；

——最新时间戳（从正确处理后的主激活消息中获得）；

——Vendor_sysID；

——HSM 根密钥 K3_HSM。

j) 将最新收到的主激活消息标识为有效（尽管此时 HSM 尚未真正激活）。

k) 将 HSM 中指示是否收到过主激活消息的标志记为“真”

l) 如果以上操作全部成功，则返回成功，否则返回失败。

C.3.6 再激活与反激活

有三种不同的反激活情况。无论哪种情况下，上一次收到的辅助激活消息都将被标记为失效。

a) 当 HSM 收到一个合法有效的反激活消息时，HSM 将回到非激活状态，此时将删除所有之前保留在 HSM 中的数据，包括 SAC 认证存储区域中的数据。而最近一次收到消息的时间戳将被保留。

b) 当 HSM 需要被不同的 CA 厂商激活时。当收到的主激活消息中携带有与已激活 HSM 中保存的不同的 Vendor_SysID 时，HSM 则会删除之前保留的与原有 Vendor_SysID 相关的数据。（包括清除 SAC 认证存储区域的数据，以及上一次收到的辅助激活消息数据），同时 HSM 会将新的主激活消息数据保存起来。

c) 当 HSM 被相同的 CA 厂商再次激活时。收到一个与正在使用的 CA 系统具有相同 Vendor_SysID 的主激活消息时，HSM 会回到非激活状态，同时会更新其最新时间戳以及相应的密钥数据。

C.3.7 辅助激活消息

由于传输数据包尺寸的限制，主激活消息没有足够的空间携带激活 HSM 所需的全部信息。为解决此问题，引入了辅助激活消息。它是激活 HSM 设备过程中由头端发往 HSM 芯片的第二个消息。HSM 只有在接收和正确处理了主激活消息后才可以接收辅助激活消息。处理此消息时，不需要 HSM 与 SoC 之间建立安全认证通道。

当且仅当匹配的主激活消息与辅助激活消息被正确接收和处理后，HSM 才会被迁移到激活状态。HSM 通过这些消息中的 Vendor_SysID 以及时间戳来判断它们是否匹配。

计算辅助激活消息签名的方法与主激活消息不同，其使用 HMAC-SM3 哈希算法，并使用 HSM 根密钥进行签名。

辅助激活消息包含 3 个数据区域：

——位置信息数据 - 10 字节；

——被加密的密钥信息 - 32 字节；

——CA 私有数据 - 71 字节。

当辅助激活消息被接收和正确处理后，以上数据将被保存在 HSM 的专门存储区域中。只有成功建立了安全认证通道，SoC 才能读取位置信息数据，而 CA 私有数据则可以在 SAC 建立前读取。

只有当辅助激活消息被接收和正确处理后，HSM 才能够将位置信息数据及 CA 私有信息写入其存储区域。其他情况将禁止这些数据的写入。

成功调用此处理功能的结果就是将 CA 私有数据以及位置信息数据写入 HSM 中对应的存储区域。

HSM 将进行如下操作：

a) 确认 HSM 是否已经收到过合法的主激活消息

b) 通过以下步骤校验辅助激活消息签名：

——利用 HSM 根密钥 K3_HSM ，计算辅助激活消息的 HMAC 值（计算范围不包括辅助激活消息签名数据本身）；

——比较 HMAC 计算结果与辅助激活消息的签名信息。

c) 检查辅助激活消息的首字节信息：

——首字节前半字节表示辅助激活消息的版本，本文中，此版本必须等于 1；

——首字节后半字节表示消息类型，对于辅助激活消息，此类型必须等于 2；

——因此，首字节必须等于 0x12。

d) 检查辅助激活消息中携带的 Vendor_SysID 以及 SoC 标识是否与主激活消息携带的对应信息相等。

e) 检查辅助激活消息中携带的 HSM 标识与当前 HSM 芯片标识是否相等。

f) 检查辅助激活消息中携带的时间戳与 HSM 最近一次收到的时间戳是否相等(最近一次收到的时间戳是从主激活消息中获得）。

g) 使用 HSM 根密钥用 SM4 算法解密出 CREEK 以及 SAC 配对密钥,并保存在 HSM 中的私有数据区域（此区域不能通过 HSM 的数据访问接口访问）。

h) 将消息中 CA 私有数据写入 HSM 中 NVM 的 CA 私有数据专用区域，并将此数据标记为可用。

i) 将消息中位置信息数据写入 HSM 中 NVM 的位置信息专用区域，并将此数据标记为可用。

j) 最后，将 HSM 的状态标记为激活。

C.3.8 读取激活信息数据

本功能用来读取 HSM 中与激活信息相关的数据：

——CA 私有数据；

——激活后与之配对的 SoC 标识；

——当前激活的 Vendor_SysID。

访问此功能时，不一定要建立 SAC。当 HSM 已经激活时，HSM 将返回以上数据。如果 HSM 尚未激活，则返回错误。

C.3.9 读取位置信息数据

本功能用来读取 HSM 中保存的位置信息。访问此功能时，要求 SAC 必须建立。当 HSM 中有可用位置信息是，则返回位置信息，反之返回错误。

C.3.10 读取最新的时间戳

HSM 应实现读取最新时间戳的功能。最新时间戳是由主激活消息携带并保存在 HSM 中的。访问此功能，不一定要建立 SAC，也不要求 HSM 处在被激活状态。通过调用此功能，客户端程序将可以判断其收到的辅助激活消息是否为匹配的消息，进而判断是否进行下一步处理。

C.3.11 处理反激活消息

运营商要求 HSM 提供一种方式，用以将 HSM 复位到非激活状态，同时将所有之前通过主激活消息和辅助激活消息获得的数据删除。当收到并正确处理一个反激活消息后，HSM 应回到其初始状态，所有数据将复位，一个例外是最新时间戳数据域，它将被更新成反激活消息所携带的最新时间戳。

反激活消息只能在 SAC 建立后进行接收和处理。这意味着如果要处理反激活消息，HSM 必须已经处在激活状态。

要实现此功能，HSM 需要如下输入:

——反激活消息；

——CA 厂商证书。

之后 HSM 将进行如下操作：

a) 根据 TA 根证书对 CA 厂商证书进行校验，使用 SM3 哈希算法生成证书摘要信息，使用 SM2 椭圆曲线算法计算数字签名。

b) 根据 CA 厂商证书校验主激活消息签名，其生成摘要信息使用 SM3 哈希算法，计算数字签名使用 SM2 椭圆曲线算法。

——如果校验失败，则处理过程终止并返回错误；

——如果校验通过，则继续后续处理。

c) 检查反激活消息的首字节信息：
——首字节前半字节表示反激活消息的版本信息，本文规定其值必须是 1；
——首字节后半字节表示消息类型，本文规定其值必须是 3；
——因此反激活消息首字节值必须等于 0x13。
d) 比较反激活消息中的当前时间戳与 HSM 中保存的上一次收到消息的时间戳：
——如果当前时间戳大于或等于上一次收到的时间戳，则继续后续处理；
——否则终止操作，返回错误。
e) 检查 CA 厂商证书中的“Subject:O”字段信息是否等于反激活消息中携带的 Vendor_SysID。注意：CA 厂商证书中 CA 厂商 ID 是用 4 个十六进制的字符表示的，而在反激活消息中是用 2 个字节表示的。
f) 检查反激活消息中携带的 HSM 标识与当前 HSM 芯片标识是否相等。
g) 将 HSM 标记为非激活状态。
h) 将 HSM 中记录是否收到过主激活消息的标志设置为“假”。
i) 用反激活消息中的最新时间戳更新 HSM 中保存的时间戳。
j) 将所有之前通过反激活消息和辅助激活消息获得的数据删除，清除整个 SAC 认证存储区域。HSM 应回到最初始的状态，除了其中的时间戳外，它应该是反激活消息所携带的最新时间。
k) 返回处理结果。

C.4 安全认证通道

C.4.1 总体流程

安全认证通道是建立在 HSM 与 SoC 之间的，用于保护 HSM 与 SoC 之间传输的敏感数据的安全数据通路。

只有 HSM 被激活之后，才可以接收并处理 SoC 发起的 SAC 建立请求。

SAC 的使用分为两个阶段，首先是 SAC 握手阶段，在此阶段 HSM 与 SoC 相互验证对方身份，并协商生成加密用的会话密钥，之后进入数据传输阶段，在此期间，所有在 SAC 内的通讯都将被会话密钥加密保护。

以上两个阶段中，SoC 都是通讯的发起方，而 HSM 只作响应方。

C.4.2 握手阶段

在 SAC 协议握手阶段，SoC 及 HSM 应执行如下操作：

a) SoC 生成一个计数器 i，并随机生成其初始值。这个计数器的值需被于后续协议交互中，通信双方都应保留其所收到的最新的计数器的值，并在构造消息时，使其值自加，即 i++。
b) 为了生成会话密钥，SoC 应提供至少 16 字节长的随机数。
c) SoC 的 TEE 侧的 HSM 驱动验证 HSM 证书链。
d) SoC 应使用 HSM 公钥对握手消息进行加密。
e) HSM 使用其私钥对握手消息进行解密。
f) 生成握手消息时应包含配对密钥信息。
g) 为了生成会话密钥，HSM 应提供至少 16 字节长的随机数。
h) HSM 应根据 SoC 发来的 16 字节随机数，生成一个加密密钥，并用此密钥加密 HSM 自己生成的 16 字节随机数，之后将加密结果返回给 SoC。
i) HSM 及 SoC 根据 HSM 以及 SoC 分别提供的随机数，分别计算得到会话密钥 SK。

C.4.3 数据传输阶段

握手阶段结束后，SoC 及 HSM 开始进行消息通信，每个往来的消息都会被加密保护，加密所用的密钥是根据二者协商出的会话密钥 SK 派生出来的，本标准不定义具体的密钥派生方式。传输流程如图 C.1 所示。

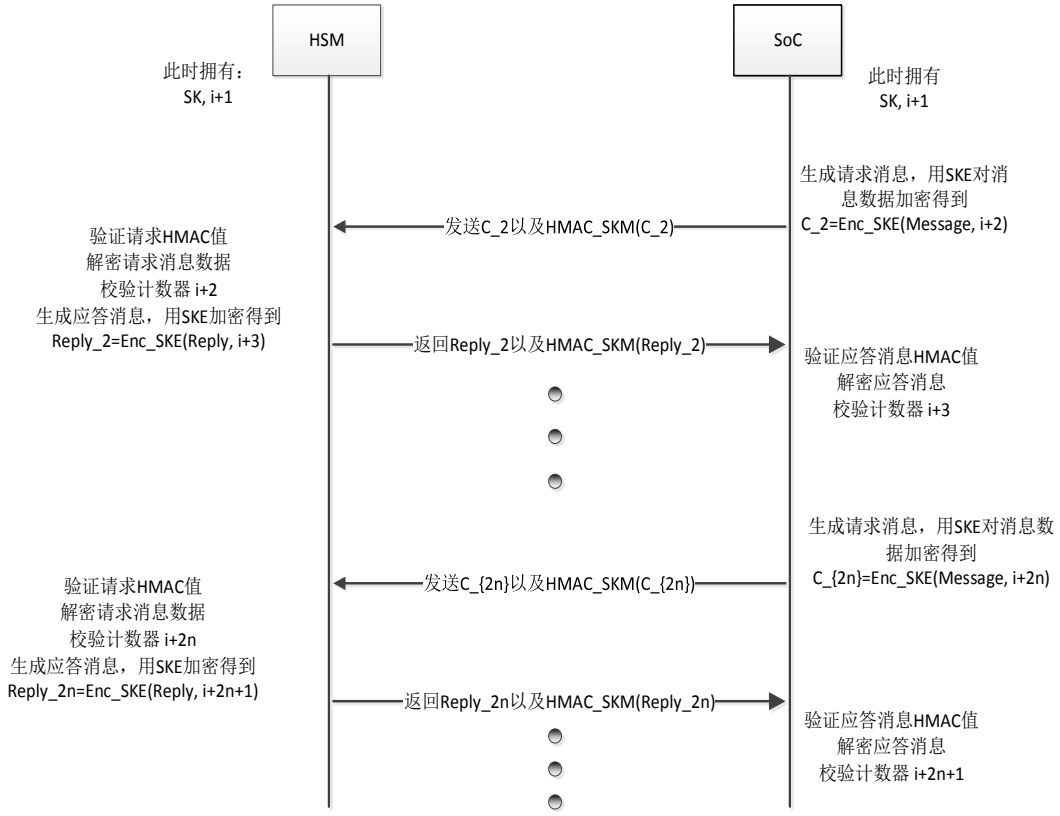

图 C.1 数据传输阶段处理流程

a) HSM 以及 SoC 都应根据会话密钥 SK 派生出 2 个 16 字节长的密钥，分别是加密会话密钥 SKE (Session Key Encryption)和验证会话密钥 SKM (Session Key MAC).

b) 当发送消息时，SoC 或 HSM 都应该：

——使其自身拥有的计数器 i 的值增加 1，即 i++；

——在原始消息后，附上当前的计数器值 i，构成完整的消息包；

——使用加密会话密钥 SKE 加密整个消息；

——使用验证会话密钥 SKM 对整个加密后的消息签名。

c) 当收到消息时，SoC 或 HSM 应该：

——使用验证会话密钥 SKM 对整个数据包进行签名校验；

——使用加密会话密钥 SKE 解密整个消息；

——检查消息中的计数器值是否大于当前接收方保存的计数器值；

——用消息中的计数器值覆盖当前保存的计数器值。

d) 通信双方在每次收到消息时，都应对计数器的值进行检查。

C.5 消息格式

本章定义了一种典型应用场景下的激活格式，实际应用中可根据需求定义。

C.5.1 激活请求消息

表 C.2 为激活请求消息格式。此消息以标准的类型-长度-值（TLV）格式表示。为了适应 HSM 证书的长度以及限制请求消息的整体长度，使用类型字段最高位标识后续长度字段的长度，当最高位为 0 时，后续长度字段用 1 字节表示，即当类型取值范围为 0x00-0x7F 时，长度字段为 1 个字节。当最高位为 1 时，长度字段用 2 字节表示，即当类型取值范围为 0x80-0xFF 时，长度字段为 2 个字节。

表 C.2 激活请求消息格式

字段序号	字段内容	长度 字节	注解
0	消息版本	1	激活请求消息版本 = 0x01
1	类型	1	0x01 - 时间戳
2	长度	1	4 - 长度值
3	时间戳内容	4	采用系统返回的时间戳格式（如北斗系统）
4	类型	1	0x02 - CA 厂商标识 ID
5	长度	1	2 - 长度值
6	CA 厂商标识	2	CA 厂商标识 ID 值
7	类型	1	0x03 - 终端数据
8	长度	1	16 - 长度值
9	Soc 标识	8	ChipID - 8 字节
10	HSM 标识	8	HSMID - 8 字节
11	类型	1	0x04 - 位置信息
12	长度	1	8
13	经度	4	采用位置系统返回的经度值（如北斗系统）
14	纬度	4	采用位置系统返回的纬度值（如北斗系统）
15	类型	1	0x0A - HSM 签名
16	长度	1	64
17	签名内容	64	此签名应被透传至 CA 头端进行校验。该签名是对以上 1~16 字段内容进行计算，通过 HSM 私钥签署获得
18	附加 CA 数据	任意	此数据以 TLV 格式构成，其中类型字段取值范围 0x10-0x3F

C.5.2 激活消息

激活消息由头端发送给终端，用来对 HSM 芯片及 CA 受信应用程序进行初始化。此消息具有固定格式，见表 C.3。

表 C.3　主激活消息格式

字段内容	长度字节	注解
消息首字节	1	前半字节表示消息版本，目前取值为 1 后半字节取值为 1 表示本消息是激活消息
时间戳	4	采用系统的时间戳格式（如北斗系统）
SoC 芯片标识	8	ChipID 芯片标识
HSM 芯片标识	8	HSM ID 芯片标识
CA 厂商标识	2	Vendor_SysID
密钥数据 C1	33	依据此密钥数据，可以派生出用以解密以下 HSM 根密钥的因子。此数据采用压缩格式表示。
被加密的密钥 C2	16	$E_{HSMPubKey}$ (K3_HSM)，加密的 HSM 根密钥
摘要信息 C3	32	SM2 算法的一部分输出结果
数字签名	64	针对消息中以上的内容，由 CA 厂商私钥经 SM2 签名获得

C.5.3　辅助激活消息

表 C.4 为辅助激活消息格式，其具有 168 字节固定长度。

表 C.4　辅助激活消息格式

字段内容	长度字节	注解
消息首字节	1	前半字节表示消息版本，目前取值为 1； 后半字节表示消息类型，取值为 2，表示辅助主激活消息
时间戳	4	采用系统的时间戳格式（如北斗系统）
SoC 芯片标识	8	SoC 芯片标识
HSM 芯片标识	8	HSM 芯片标识
Vendor_SysID	2	Vendor_SysID
经度	4	采用位置系统的经度值（如北斗系统）
纬度	4	采用位置系统的纬度值（如北斗系统）
最大允许距离	2	单位：10 米的倍数 - 最大可达 650 公里 注意：终端中 CA 可信程序在将该值与位置系统返回的距离值做比较时，需要将该值乘以 10，换算成以“米”为单位
加密的密钥信息	32	$E_{KDF(K3\_HSM,48)}$(CREEK, PairK) ，被 HSM 根密钥 K3_HSM 所派生的密钥，通过 SM4-CBC(IV 值为 0)算法加密的重加密密钥以及 SAC 配对密钥。此处以及下述数字签名所用的密钥，是通过 SM2 椭圆曲线公钥密码算法第 3 部分密钥交换协议中定义的密钥派生函数根据 K3_HSM 所产生的 48 字节密钥。此处 SM4-CBC 算法用其前 16 字节做加解密密钥，而 HMAC-SM3 则使用其后 32 字节作为 HMAC 验证密钥。
CA 私有数据	71	CA 私有数据
数字签名	32	通过 HMAC-SM3 算法，经由 HSM 根密钥 K3_HSM 派生的密钥签署获得

C.5.4 反激活消息

表 C.5 为反激活消息格式。

表 C.5 反激活消息格式

字段内容	长度字节	注解
消息首字节	1	前半字节表示消息版本，取值为 1； 后半字节表示消息类型，取值为 3，表示反激活消息
时间戳	4	采用系统的时间戳格式 （如北斗系统）
保留字段	8	保留字段，未使用
HSM 芯片标识	8	HSMID
Vendor_SysID	2	Vendor_SysID
数字签名	64	针对消息中以上的内容，由 CA 厂商私钥经 SM2 签名获得

C.6 证书格式

本章列出所有相关证书的格式，符合 GM/T 0015-2012 的要求，证书格式见表 C.6。

表 C.6 证书格式

	TA 数据认证管理平台根证书	CA 厂商根证书	HSM 厂商根证书	HSM 芯片根证书	注解
证书版本	V3	V3	V3	V3	
证书序列号	生成时的默认值	生成时的默认值	生成时的默认值	生成时的默认值	忽略
证书签名算法	SM3 SM2	SM3 SM2	SM3 SM2	SM3 SM2	OID 为 1.2.156.10197.1.501
证书签发者信息	取值等于 TA 证书中的使用者字段	取值等于 TA 证书中的使用者字段	取值等于 TA 证书中的使用者字段	取值等于 HSM 厂商根证书中的使用者字段	
有效期：不早于	证书签发的日期	证书签发的日期	证书签发的日期	证书签发的日期	当前应用时，忽略此字段
有效期：不晚于	自签发日期起，50 年内有效	自签发日期起，50 年内有效	自签发日期起，50 年内有效	自签发日期起，50 年内有效	当前应用时，忽略此字段
使用者：O 属性	取值为固定字符串“SARFT TRUSTED AUTHORITY”	CA 厂商标识	HSM 厂商名称	HSM 芯片标识	CA 厂商标识 - 4 位十六进制数字 HSM 厂商名称 -长度为 20 个字符以内的字符串 HSM 芯片标识 - 16 位十六进制数字

表 C.6（续）

	TA 数据认证管理平台根证书	CA 厂商根证书	HSM 厂商根证书	HSM 芯片根证书	注解
使用者：OU 属性	“TEST” 或“PRODUCTION”	“TEST” 或“PRODUCTION”	“TEST” 或“PRODUCTION”	“TEST”或“PRODUCTION”	生产环境中，需验证此字段值是否为“PRODUCTION”
使用者：CN 属性	取值为固定字符串“SARFT TRUSTED AUTHORITY MANAGEMENT CERTIFICATE”	取值为字符串“CHINA DTH CA VENDOR CERTIFICATE－<CA Vendor Name>”，其中<CA Vendor Name>为实际 CA 厂商名称	取值为字符串“CHINA DTH HSM VENDOR CERTIFICATE －<HSM vendor name>”，其中<HSM vendor name>为实际 HSM 厂商名称	取值为固定字符串“CHINA DTH HSM DEVICE CERTIFICATE”	CA 厂商名称 －长度为 20 个字符以内的字符串 HSM 厂商名称 －长度为 20 个字符以内的字符串
使用者：C，ST,L 属性	默认值	默认值	默认值	默认值	
使用者公钥信息：公钥算法	EC 公钥	EC 公钥	EC 公钥	EC 公钥	OID 为 1.2.840.10045.2.1
使用者公钥信息：算法参数	SM2	SM2	SM2	SM2	OID 为 1.2.156.10197.1.301
使用者公钥信息：公钥内容	非压缩格式（0x04 开头）	非压缩格式（0x04 开头）	非压缩格式（0x04 开头）	非压缩格式（0x04 开头）	
证书扩展字段：签发机构密钥标识	非关键字段，只需携带 key_id 信息	非关键字段，只需携带 key_id 信息	非关键字段，只需携带 key_id 信息	非关键字段，只需携带 key_id 信息	根据 RFC 5280 中 4.2.1.2 选项(1)计算获得。
证书扩展字段：使用者密钥标识	非关键字段	非关键字段	非关键字段	非关键字段	根据 RFC 5280 中 4.2.1.2 选项(1)计算获得

表 C.6（续）

	TA 数据认证管理平台根证书	CA 厂商根证书	HSM 厂商根证书	HSM 芯片根证书	注解
证书扩展字段：证书密钥用途	关键字段，用于证书签名	关键字段，用于数字签名	关键字段，用于证书签名	关键字段，用于数字签名及密钥加密	
证书扩展字段：基本约束信息	关键字段，取值为 True，路径深度为 1	关键字段，取值为 False	关键字段，取值为 True，路径深度为 0	关键字段，取值为 False	取值为 True 表示该证书拥有者为数字证书认证机构。其中，2/1/0 表示该机构所处证书链位置，0 表示最末级 CA
证书签名算法	SM3 SM2	SM3 SM2	SM3 SM2	SM3 SM2	同证书签名算法
证书签名内容	TA 根证书为自签名				签名内容由 32 字节的 r 值与 32 字节值构成。

参 考 文 献

[1] RFC 5280 X.509 Information technology - Open systems interconnection - The Directory: Public-key and attribute certificate framew

中 华 人 民 共 和 国
广播电影电视行业标准
单向可下载条件接收系统技术规范
GY/T 308－2017
*
国家新闻出版广电总局广播电视规划院出版发行
责任编辑：王佳梅
查询网址：www.abp2003.cn
北京复兴门外大街二号
联系电话：（010）86093424　　86092923
邮政编码：100866