International Telecommunication Union

ITU-T J.1028

TELECOMMUNICATION
STANDARDIZATION SECTOR
OF ITU

(07/2019)

SERIES J: CABLE NETWORKS AND TRANSMISSION OF TELEVISION, SOUND PROGRAMME AND OTHER MULTIMEDIA SIGNALS

Conditional access and protection – Downloadable conditional access system for unidirectional networks

Downloadable conditional access system for unidirectional networks – Terminal system

Recommendation ITU-T J.1028

Recommendation ITU-T J.1028

Downloadable conditional access system for unidirectional networks – Terminal system

Summary

Recommendation ITU-T J.1028 specifies a terminal for a one-way downloadable conditional access system (DCAS) for unidirectional networks. One-way DCAS protects broadcast content/services and controls consumer entitlements like traditional conditional access (CA) systems and enables a terminal, such as a set-top-box (STB), to adapt to a new CA system by downloading and installing the new CA system's client without hardware changing. In particular one-way DCAS can fully work in unidirectional cable TV networks and other unidirectional networks such as satellite TV networks.

History

Edition	Recommendation	Approval	Study Group	Unique ID*
1.0	ITU-T J.1028	2019-07-29	9	11.1002/1000/13974

Keywords

CA, conditional access, downloadable, unidirectional network.

* To access the Recommendation, type the URL http://handle.itu.int/ in the address field of your web browser, followed by the Recommendation's unique ID. For example, http://handle.itu.int/11.1002/1000/11830-en.

FOREWORD

The International Telecommunication Union (ITU) is the United Nations specialized agency in the field of telecommunications, information and communication technologies (ICTs). The ITU Telecommunication Standardization Sector (ITU-T) is a permanent organ of ITU. ITU-T is responsible for studying technical, operating and tariff questions and issuing Recommendations on them with a view to standardizing telecommunications on a worldwide basis.

The World Telecommunication Standardization Assembly (WTSA), which meets every four years, establishes the topics for study by the ITU-T study groups which, in turn, produce Recommendations on these topics.

The approval of ITU-T Recommendations is covered by the procedure laid down in WTSA Resolution 1.

In some areas of information technology which fall within ITU-T's purview, the necessary standards are prepared on a collaborative basis with ISO and IEC.

NOTE

In this Recommendation, the expression "Administration" is used for conciseness to indicate both a telecommunication administration and a recognized operating agency.

Compliance with this Recommendation is voluntary. However, the Recommendation may contain certain mandatory provisions (to ensure, e.g., interoperability or applicability) and compliance with the Recommendation is achieved when all of these mandatory provisions are met. The words "shall" or some other obligatory language such as "must" and the negative equivalents are used to express requirements. The use of such words does not suggest that compliance with the Recommendation is required of any party.

INTELLECTUAL PROPERTY RIGHTS

ITU draws attention to the possibility that the practice or implementation of this Recommendation may involve the use of a claimed Intellectual Property Right. ITU takes no position concerning the evidence, validity or applicability of claimed Intellectual Property Rights, whether asserted by ITU members or others outside of the Recommendation development process.

As of the date of approval of this Recommendation, ITU had not received notice of intellectual property, protected by patents, which may be required to implement this Recommendation. However, implementers are cautioned that this may not represent the latest information and are therefore strongly urged to consult the TSB patent database at http://www.itu.int/ITU-T/ipr/.

Table of Contents

Introduction

This Recommendation is part 3 of a multi-part deliverable covering the terminal of a one-way DCAS specification, as identified below:

Part 1: "Requirements"

Part 2: "System architecture"

Part 3: "Terminal system"

Recommendation ITU-T J.1028

Downloadable conditional access system for unidirectional networks – Terminal system

1 Scope

The object of this Recommendation is to specify the terminal of a one-way downloadable conditional access system for unidirectional networks, including a terminal security chipset, hardware security module (HSM), one-way DCAS client software and related application programming interfaces (APIs). This Recommendation is one in a series of Recommendations, specifying the whole downloadable conditional access system for unidirectional networks. The other parts of the one-way DCAS Recommendations include the requirement for one-way DCAS as defined in [ITU-T J.1026] and the specification of system architecture and related security mechanism for one-way DCAS as defined in [ITU-T J.1027].

2 References

The following ITU-T Recommendations and other references contain provisions which, through reference in this text, constitute provisions of this Recommendation. At the time of publication, the editions indicated were valid. All Recommendations and other references are subject to revision; users of this Recommendation are therefore encouraged to investigate the possibility of applying the most recent edition of the Recommendations and other references listed below. A list of the currently valid ITU-T Recommendations is regularly published. The reference to a document within this Recommendation does not give it, as a stand-alone document, the status of a Recommendation.

[ITU-T J.1026] Recommendation ITU-T J.1026 (2019), *Downloadable conditional access system for unidirectional networks – Requirements*.

[ITU-T J.1027] Recommendation ITU-T J.1027 (2019), *Downloadable conditional access system for unidirectional networks – System architecture.*

[ISO/IEC 10118-3] ISO/IEC 10118-3: 2018, *Information technology – Security techniques – Hash-functions – Part 3: Dedicated hash-functions*.

[ISO/IEC 14888-3] ISO/IEC 14888-3: 2018, *Information technology – Security techniques – Digital signatures with appendix – Part 3: Discrete logarithm based mechanisms*.

3.1 Terms defined elsewhere

This Recommendation uses the following terms defined elsewhere:

3.1.1 descrambling [b-ITU-T J.93]: The processes of reversing the scrambling function (see "scrambling") to yield usable pictures, sound and data services.

3.1.2 entitlement control messages (ECMs) [b-ITU-T J.290]: An ECM is an encrypted message that contains access criteria to various service tiers and a control word (CW).

3.1.3 entitlement management messages (EMMs) [b-ITU-T J.290]: The EMM contains the actual authorization data and shall be sent in a secure method to each CPE device.

3.1.4 scrambling [b-ITU-T J.93]: The process of using an encryption function to render television and data signals unusable to unauthorized parties.

3.2 Terms defined in this Recommendation

This Recommendation defines the following terms:

3.2.1 downloadable conditional access system (DCAS): A conditional access system that supports all the features of legacy CA and provides a CA-neutral mechanism to securely download CA client images and switch CA terminals without changing hardware through either a broadcasting or two-way network.

3.2.2 one-way DCAS: A downloadable conditional access system (DCAS) operated especially in a one-way network.

3.2.3 one-way DCAS App: A one-way downloadable conditional access system (DCAS) application running on the terminal software platform. After a terminal device is deployed in field, this application can be upgraded or replaced through online pushing or other methods.

3.2.4 one-way DCAS trusted App: A one-way downloadable conditional access system (DCAS) trusted application running in the trusted execution environment of a terminal device. After a terminal device is deployed in field, this application can be upgraded or replaced through online pushing or other methods.

3.2.5 one-way DCAS client software: A terminal application implemented by a one-way downloadable conditional access system (DCAS) App and a DCAS trusted App working together on the terminal software platform.

3.2.6 one-way DCAS client software data: Data that needs to be saved or updated when one-way downloadable conditional access system (DCAS) client software runs, which includes CA authorization information, CA private data, positioning information, etc.

3.2.7 one-way DCAS manager: A software module responsible for registering one-way downloadable conditional access system (DCAS) client software, supporting information interaction between a one-way DCAS App and a one-way DCAS trusted App, as well as receiving and forwarding one-way DCAS entitlement control and management messages.

3.2.8 security chipset key de-obfuscation: An algorithm used to de-obfuscate an encrypted security chipset key.

3.2.9 key ladder (KLAD): A structured multi-level key mechanism that ensures secure transport of a control word.

3.2.10 transport stream filtering: A filtering mechanism that is used to extract data matching filter rules from a transport stream.

3.2.11 root key: The key used for the first level of a key ladder.

3.2.12 hash value: The result calculated on any value by using hashing algorithms.

3.2.13 bootloader: The program for initiating hardware and loading software after a receiver boots up.

3.2.14 challenge-response: The process in which one-way downloadable conditional access system (DCAS) client software performs calculations using the key ladder of a terminal security chipset through a one-way DCAS manager.

3.2.15 nonce: Random or repetitive data sent from a one-way downloadable conditional access system (DCAS) headend system for challenge-response.

3.2.16 hardware security module (HSM): A security chipset capable of control word processing, access authorizing and secure storage, etc., which supports hardware security enhancement of a unidirectional receiver.

3.2.17 terminal security chipset: A stream processing chipset with security functions such as secure key deriving and key ladder processing, etc.

3.2.18 terminal software platform: A software platform running on a receiver, integrated with various hardware drivers, having various terminal application APIs, capable of downloading and

running terminal applications according to specified security requirements, and providing a secure execution environment for terminal applications.

3.2.19 secure data management platform: A platform that generates and manages some basic and root information, such as keys and IDs used in downloadable conditional access system (DCAS), including information to DCAS headends, to terminal security chipsets and to hardware security modules (HSMs).

4 Abbreviations and acronyms

This Recommendation uses the following abbreviations and acronyms:

API	Application Programming Interface
App	Application
BFR	Boot From RAM
CA	Conditional Access
CAT	Conditional Access Table
CATA	Conditional Access Trusted Application
CAJS	Conditional Access Javascript
CAS	Conditional Access System
ChipID	Chipset Identification
CPU	Central Processing Unit
CREEK	Crypto-toolkit Re-encryption Key
CSA	Common Scrambling Algorithm
CW	Control Word
DCAS	Downloadable Conditional Access System
DVB	Digital Video Broadcasting
DTH	Direct To Home
ECB	Electronic Code Book
ECM	Entitlement Control Message
ECW	Encrypted Control Word
EMM	Entitlement Management Message
EPG	Electronic Program Guide
ESCK	Encrypted Security Chipset Key
GP	Global Platform
HSM	Hardware Security Module
HSMID	Hardware Security Module Identification
IXC	Inter-Xlet Communication
KDF	Key Derivation Function
KLAD	Key Ladder
MITM	Man-in-the-Middle

NVM	Non-Volatile Memory
OTP	One Time Programmable
PairK	Pairing Key
PID	Packet Identification
SAC	Secure Authenticated Channel
SCK	Security Chipset Key
SCKv	Security Chipset Key Vendor
SDMP	Secure Data Management Platform
Seedv	Seed Vendor
SI	Service Information
SKE	Session Key Encryption
SKM	Session Key MAC
SMK	Secret Mask Key
SoC	System on Chip
TEE	Trusted Execution Environment
TVOS	Television Operating System
Vendor_SysID	Vendor System Identification

5 Conventions

In this Recommendation:

The keywords **"is required to"** indicate a requirement which must be strictly followed and from which no deviation is permitted if conformance to this document is to be claimed.

The keywords **"is recommended"** indicate a requirement which is recommended but which is not absolutely required. Thus this requirement need not be present to claim conformance.

The keywords **"is prohibited from"** indicate a requirement which must be strictly followed and from which no deviation is permitted if conformance to this document is to be claimed.

The keywords **"can optionally"** indicate an optional requirement which is permissible, without implying any sense of being recommended. This term is not intended to imply that the vendor's implementation must provide the option and the feature can be optionally enabled by the network operator/service provider. Rather, it means the vendor may optionally provide the feature and still claim conformance with the specification.

In the body of this document and its annexes, the words *shall*, *shall not*, *should*, and *may* sometimes appear, in which case they are to be interpreted, respectively, as *is required to*, *is prohibited from*, *is recommended*, and *can optionally*. The appearance of such phrases or keywords in an appendix or in material explicitly marked as *informative* are to be interpreted as having no normative intent.

6 Terminal system

6.1 Terminal system architecture

The one-way DCAS terminal system is an essential part of a DCAS system. For an illustration of DCAS architecture please refer to Figure 1 of [ITU-T J.1027].

Figure 1 of the present Recommendation shows the one-way DCAS terminal system architecture comprising a hardware platform (including a terminal security chipset and HSM), a terminal software platform and terminal software/hardware modules such as one-way DCAS client software and other applications. This Recommendation only specifies one-way DCAS related terminal software/hardware modules such as terminal security chipset, HSM, one-way DCAS APIs and one-way DCAS client software. A one-way DCAS terminal system mainly includes a terminal security chipset, HSM, one-way DCAS client software and one-way DCAS APIs of the terminal software platform. A one-way DCAS terminal validates a user's entitlement and descrambles protected services to implement conditional access of services. The terminal software platform can securely download, update and replace one-way DCAS client software.

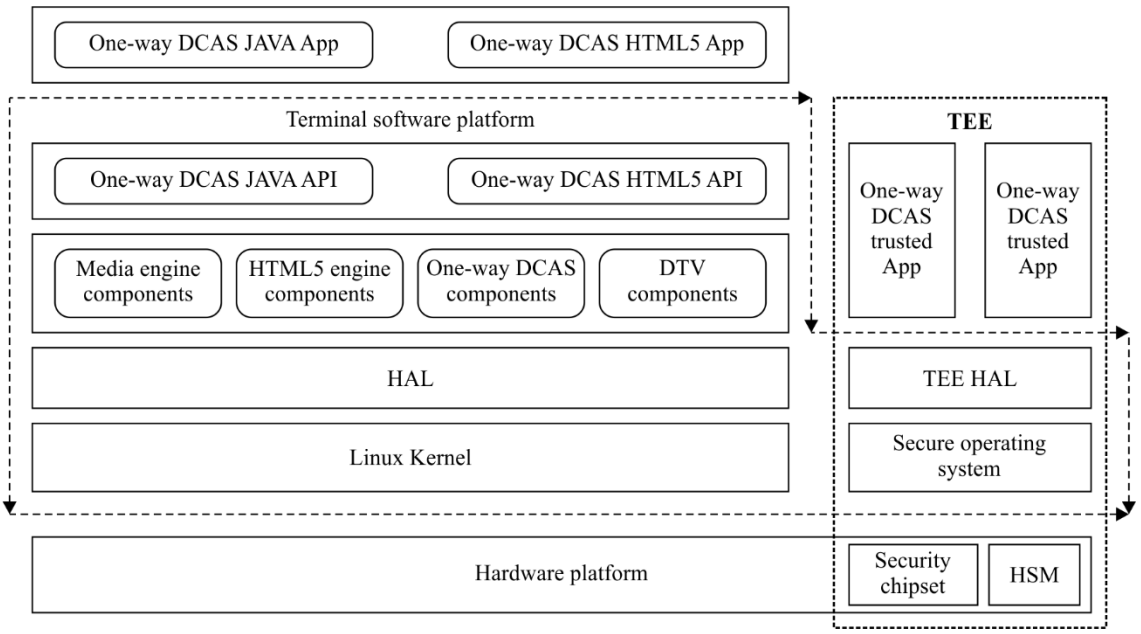

Figure 1 – One-way DCAS terminal system architecture

The terminal security chipset provides the key ladder module and root key generation module, to ensure the security of terminal data transfer and independence of the CA system, see clause 6.3 for details on the terminal security chipset.

The HSM provides hardware-level security enhancement by participating in key ladder processing, access authorization and secure storage, see clause 6.4 for HSM detailed implementation information.

The terminal software platform provides APIs for running one-way DCAS client software.

The terminal software platform can be either a smart TV operation system, or terminal middleware based on operation systems such as Linux.

The one-way DCAS APIs supports the DCAS manager in the management of one-way DCAS client software and supports one-way DCAS client software in processing of CA data such as ECM/EMM, and data interaction with an electronic program guide (EPG) and other applications. See Annex B for the APIs description.

One-way DCAS client software implements analyzing and processing of data such as entitlement control messages (ECMs) and entitlement management messages (EMMs).

One-way DCAS client software consists of DCAS App and DCAS trusted App. One-way DCAS client software functions are implemented by DCAS App and DCAS trusted App working together.

One-way DCAS client software can be downloaded to a terminal software platform and coexists with other applications on the same terminal software platform.

6.2 One-way DCAS APIs

One-way DCAS APIs are used to implement data interaction between the one-way DCAS client software and the terminal software platform. They include general APIs such as Java APIs, JavaScript APIs and APIs for secure environment. See Annex B for implementation details.

The general APIs include:

a) Filtering APIs: One-way DCAS client software invokes the filtering APIs to receive ECMs, EMMs and conditional access tables (CATs).

b) One-way DCAS management APIs: One-way DCAS client software uses DCAS management APIs to register itself to terminal software platform and receive descrambling requests from a DCAS manager in the terminal software platform.

The secure environment APIs include:

a) HSM APIs: One-way DCAS client software invokes HSM APIs to access secure storage areas and generate a secure control word.

b) Key ladder APIs: One-way DCAS client software invokes the key ladder APIs to load encrypted keys to a terminal security chipset to descramble transport streams.

c) Auxiliary information APIs: One-way DCAS client software uses these APIs to obtain auxiliary information, such as a terminal's current positioning data.

d) Other global platform (GP) extension APIs: One-way DCAS client software uses these APIs for encryption/decryption, signature verification, memory management and debug printing, etc.

6.3 Terminal security chipset

6.3.1 Terminal security chipset workflow

Figure 2 shows functions of a terminal security chipset, which includes modules for one time programmable (OTP) areas, root key derivation, key ladder, descrambling and decoding.

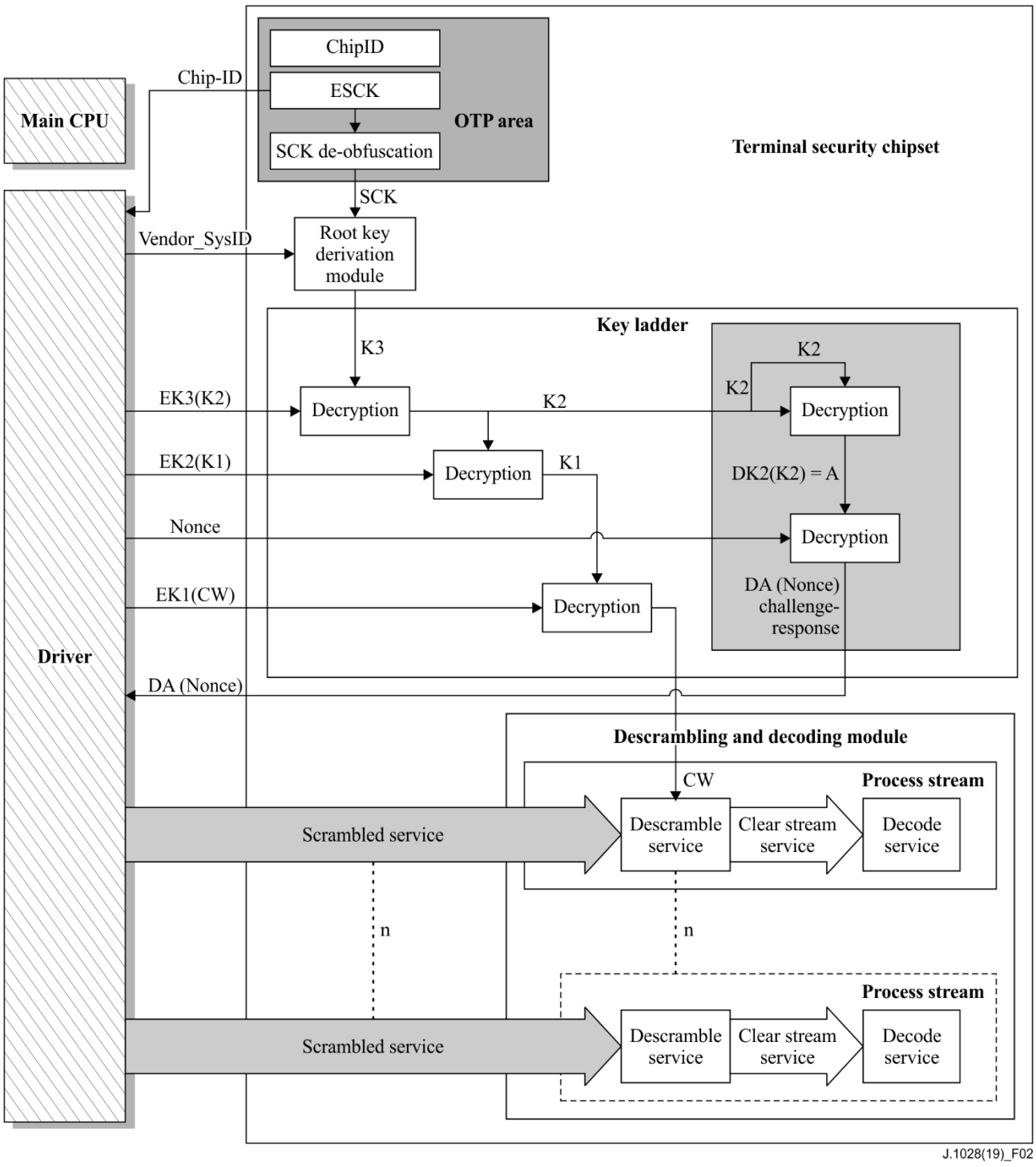

Figure 2 – Terminal security chipset functional diagram

The terminal security chipset uses a root key derivation module to generate root key K3 and uses a key ladder module to ensure secure transfer of control words and other secret keys and the validity of the terminal security chipset.

The Terminal security chipset's functionalities cannot be implemented by a primary central processing unit (CPU).

Terminal security chipset workflow is outlined as follows:

After a security chipset is powered on, the OTP of the security chipset uses a built-in encrypted security chipset key (ESCK) and security chipset key (SCK) de-obfuscation to generate the SCK, which is fed to a root key derivation module to generate a root key K3.

The key ladder module receives the root key K3 and uses it for decryption of keys and challenge-response. Key ladder contains functions such as: decrypting keys level by level with the input encrypted keys; processing challenge information (Nonce) and generating response (DA (Nonce)).

Finally, the decrypted control word (CW) is sent to descrambling and decoding modules for descrambling and decoding services. The challenge-response is a function in a two-way system for the headend to certificate the system on chip (SOC) chip.

6.3.2 Root key derivation module

The root key derivation module consists of a group of hardware logical modules. It uses an embedded derivation mechanism together with input parameters for the derivation of root keys. All CA vendors use this derivation mechanism to generate different root keys. This method can circumvent the vulnerability of using a singular root key.

Figure 3 shows the functions of the root key derivation module, which include: an SCK preliminary manipulation function, a vendor separation function and a final root key derivation function. It derives a specific root key for a CA vendor according to the input SCK and Vendor_SysID.

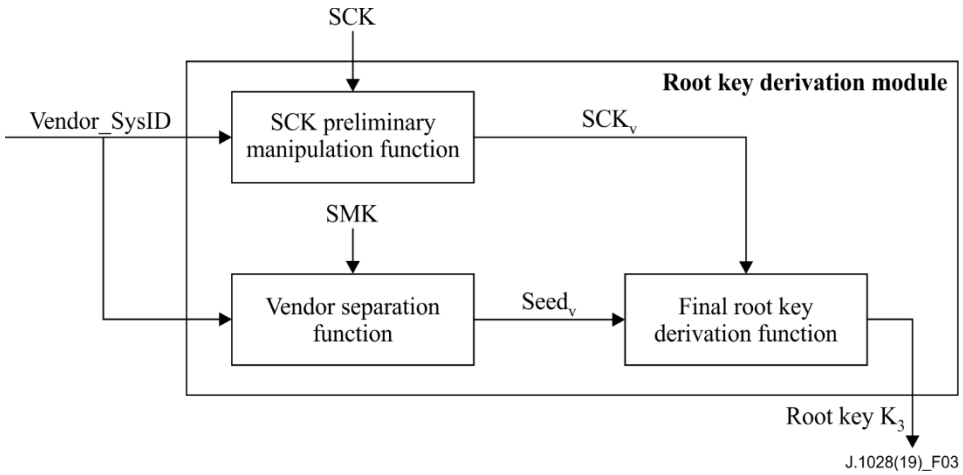

Figure 3 – Root derivation module functional diagram

The preliminary SCK manipulation function generates SCKv based on the SCK and input Vendor_SysID. The function must use a cipher algorithm with a certain level of security strength which ensures the SCK cannot be retrieved when Vendor_SysID and SCKv are known. In addition, the function must be provided by the terminal security chipset vendor and is certified by a secure data management platform (SDMP).

The vendor separation function generates Seedv by using Vendor_SysID and SMK as input. The function must use a cipher algorithm with a certain level of security strength which ensures the SMK cannot be retrieved when Vendor_SysID and Seedv are known. In addition, the function must be provided by the terminal security chipset vendor and is certified by a SDMP.

The final derivation function of the root key derives root key K3 based on the input of SCKv and Seedv. In the execution process, the final derivation function of root key must use a one-way function to ensure the other input parameter cannot be retrieved when K3 and any other input parameter are known. For example, Seedv cannot be retrieved when K3 and SCKv are known. The final derivation function of the root key must be provided by the terminal security chipset vendor and is certified by the SDMP.

Vendor_SysID: 2 bytes, used for identifying CA system and assigned by the SDMP.

– SCKv: 16 bytes

– SMK: 16 bytes, gate-level data provided by chipset vendor

– Seedv: 16 bytes

– Terminal security chipset can support various final root key derivation functions at the same time.

6.3.3 Key ladder

6.3.3.1 The 3-level key mechanism

Figure 4 is a functional diagram of the key ladder module.

The terminal security chipset specified by this Recommendation shall support the 3-level key ladder mechanism. For the mechanism of a key ladder with more levels, the security requirements and technical details are out of the scope of this Recommendation.

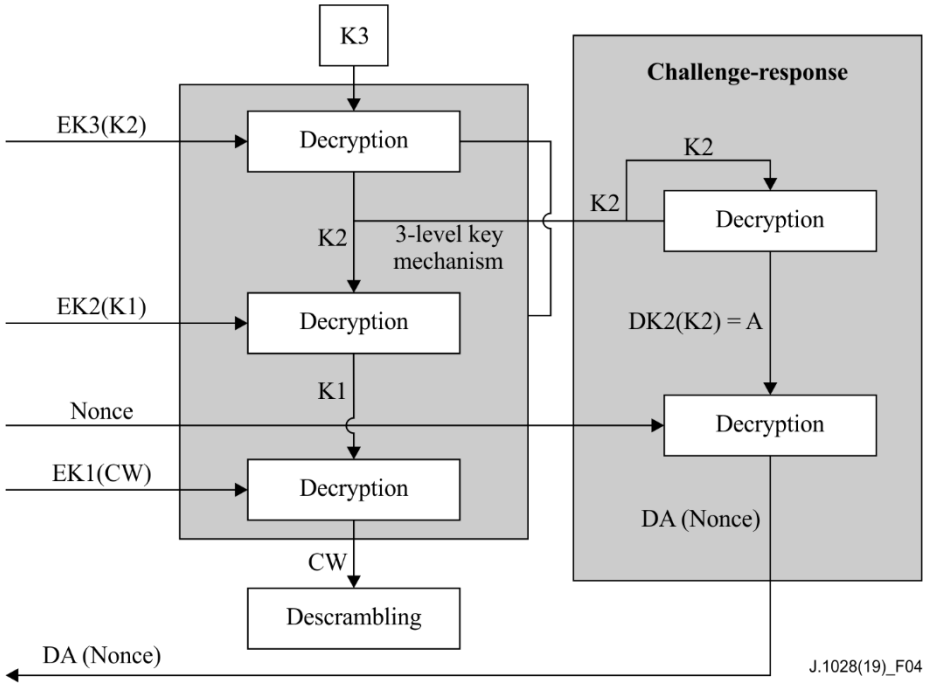

Figure 4 – Key ladder module functional diagram

The 3-level key ladder mechanism ensures secure transfer of the CW in the terminal.

The 3-level key ladder mechanism uses the root key K3 obtained from the root key derivation module, to decrypt EK3 (K2), EK2 (K1), EK1 (CW) to obtain the CW required by descrambling; at the same time, K2 works with challenge information (Nonce) to generate the response (DA (Nonce)).

The terminal security chipset shall decrypt scrambled services following the procedure below:

a) Shall receive encrypted EK3(K2), use K3 to decrypt it and generate K2;

b) Shall receive EK2(K1), use K2 to decrypt it and generate K1;

c) Shall receive EK1(CW), use K1 to decrypt it and generate CW;

d) CW is used for decrypt scrambled services.

EK3(K2) represents key K2 encrypted with key K3.

EK2(K1) represents key K1 encrypted with key K2.

EK1(CW) represents CW encrypted with key K1.

K3 is derived root key, 16 bytes.

K2 is the key used to decrypt K1, 16 bytes.

K1 is the key used to decrypt CW, 16 bytes.

CW is key used to descramble services, 8 or 16 bytes

The key ladder mechanism should support the SM4 algorithm defined in [b-GB/T 32907] with 128-bits keys and data blocks in the Electronic Code Book (ECB) mode.

6.3.3.2 Challenge response

The terminal security chipset shall support a challenge-response mechanism, which can be used in some security functions.

The challenge-response mechanism shall comply with the following procedure:

a) Terminal security chipset shall receive Vendor_SysID, EK3(K2) and Nonce by using the driver API.

b) Terminal security chipset shall use derived K3 to decrypt EK3(K2) to get K2.

c) Terminal security chipset shall decrypt K2 using K2 itself to generate DK2(K2), denoted as A.

d) Terminal security chipset shall decrypt Nonce using A, to generate DA(Nonce).

e) Terminal security chipset returns DA(Nonce) by using the driver API.

DK2(K2) represents the process of decrypting K2 using K2.

A denotes the result of DK2(K2), 16 bytes.

Nonce denotes challenge data, 16 bytes.

DA(Nonce) denotes the result from decrypting Nonce with A as the key.

6.3.4 OTP area

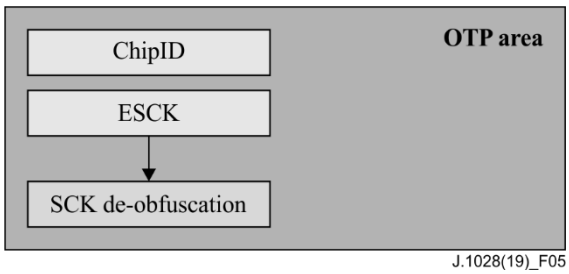

Figure 5 – OTP area functional diagram

Figure 5 is a functional diagram of the OTP area, which is used to store information such as ChipID, ESCK and SCK de-obfuscation. The logical circuit of SCK de-obfuscation reads the ESCK from the OTP area, de-obfuscates the ESCK to SCK and provides the SCK to the root key derivation module.

The SCK is the security information required for deriving the root key. The SCK should not be stored in the OTP area as plain text.

a) SCK is the terminal security chipset key, which is unique per chipset and is generated by the SDMP, 16 bytes.

b) ESCK is the encrypted SCK provided by the SDMP. It is stored in the OTP area with the same length as the SCK.

c) ChipID is an 8-byte public identifier of the terminal security chipset. It includes information such as chipset vendor, type, as well as a 4-byte unique global identifier

assigned by the secure data management platform. The ChipID format is described in Table 1.

Table 1 – ChipID data format

Data name	Length (Bit)	Data type
Chip Vendor ID	8	Uimsbf
Chip Type	12	Uimsbf
Reserved	12	Uimsbf
Chipset SN	32	Uimsbf

Chip Vendor ID: unique ID for chipset vendor, 8 bits.

Chip Type: ID for a chipset model manufactured by a chipset vendor. Assigned by SDMP, 12 bits.

Reserved: 12 bits.

Chipset SN, a 32-bit serial number of a chipset manufactured by a chipset vendor. It is globally unique for any chipset regardless of whether the chipset vendor or type is the same.

6.4 Hardware security module

6.4.1 HSM architecture

The HSM is a core functional component in the one-way DCAS system for unidirectional networks. It participates in the decryption of the CW and can be used as a replacement for the hardware smart card of the traditional CA terminal systems; different from a smart card which is a proprietary hardware unit of CA vendors, the HSM is a standard security component that can be shared among different CA vendors.

Figure 6 shows the HSM basic architecture. The HSM contains modules such as the key ladder processing module, algorithm tools and secure storages, etc. It establishes a secure authenticated channel (SAC) with the terminal security chipset to ensure the security of data transfer.

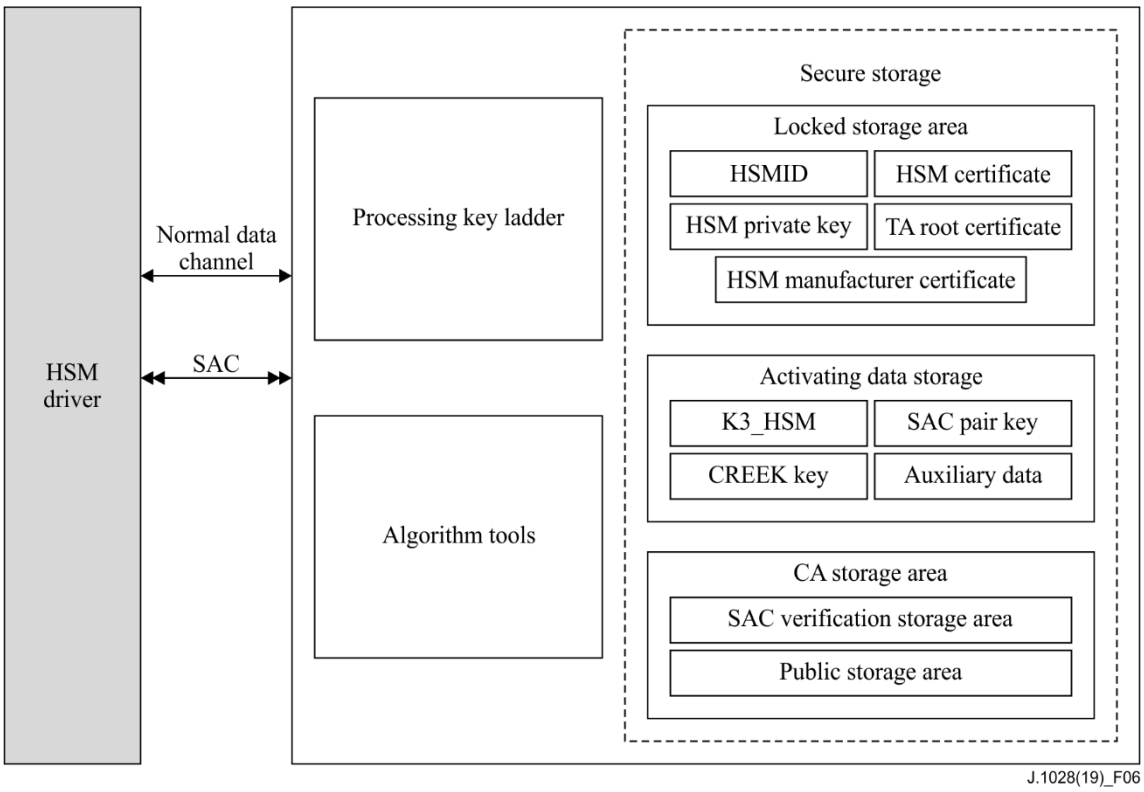

Figure 6 – HSM basic architecture

6.4.2 HSM activation

The HSM needs to be activated before being used by any conditional access system (CAS). Functions of an inactivated HSM are limited and cannot be fully used until the HSM receives activation messages and completes the activation. Figure 7 shows the basic procedure of the activation.

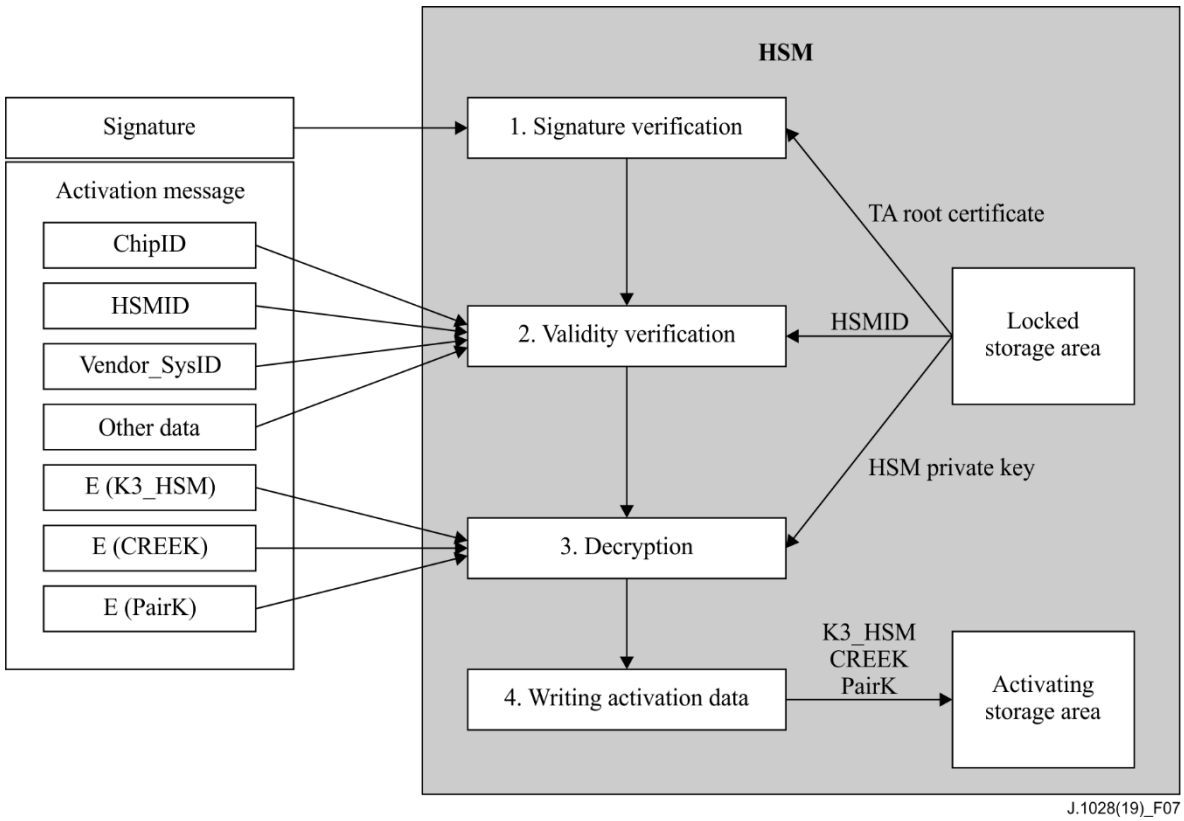

Figure 7 – Basic HSM activation procedure

After receiving the activation messages, the HSM first verifies the signature of the activation messages, then validates other data such as ChipID, HSMID, Vendor_sysID, version number and timestamp. After the validation, the HSM decrypts the encrypted keys such as K3_HSM, and stores the keys in the activating storage area.

See Annex C for detailed information on the HSM activation procedure and requirements.

6.4.3 Key ladder processing module

Key ladder processing is the core function of a HSM, it uses K3_HSM and a crypto-toolkit re-encryption key (CREEK) to decrypt and re-encrypt the key ladder data sent from the headend to generate key ladder data for control word decryption of the terminal security chipset.

A typical HSM key ladder mechanism uses a 3-level symmetric encryption/decryption algorithm, which can be accessed via the HSM driver API running in a trusted execution environment (TEE) environment. The output result is returned via SAC to the DCAS trusted App running in a TEE environment and finally sent to the terminal security chipset. The processing procedure of a HSM is shown in Figure 8.

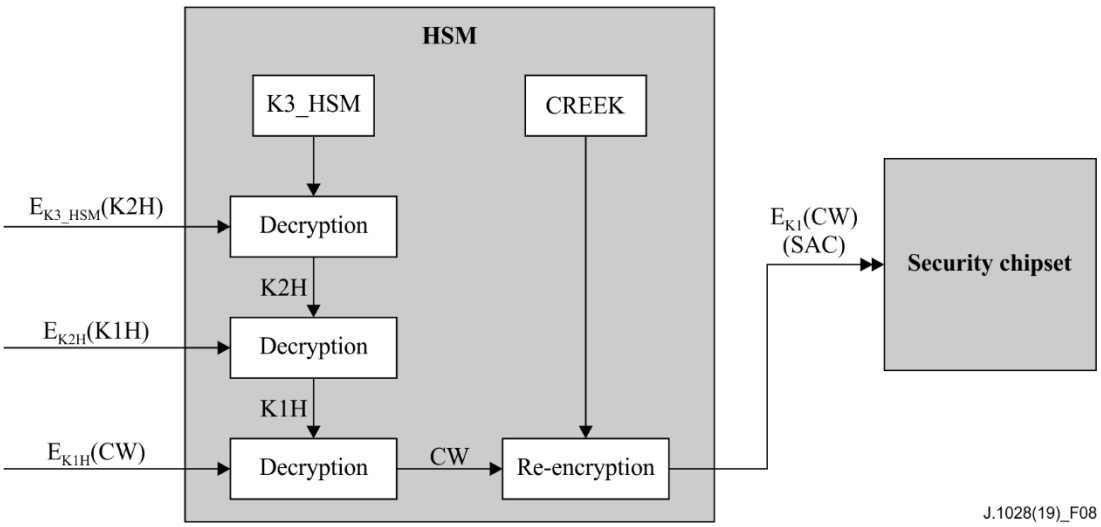

Figure 8 – HSM key ladder processing procedure

The 3-level key ladder mechanism ensures secure transfer of the CW in the terminal. Operations related to the key ladder processing module must be invoked via the SAC, which means the key ladder processing module is not available before the SAC is established.

The 3-level key ladder mechanism uses root key K3_HSM obtained from a locked storage area, to decrypt EK3_HSM(K2H), EK2H(K1H) and EK1H(CW) one by one to get the required CW, and re-encrypt the CW with CREEK.

The terminal security chipset shall decrypt according to the following procedure:

a) Shall receive encrypted data E_{K3_HSM}(K2H), decrypt it with K3_HSM, resulting in K2H;

b) Shall receive encrypted data E_{K2H}(K1H), decrypt it with K2H, resulting in K1H;

c) Shall receive encrypted data E_{K1H}(CW), decrypt it with K1H, resulting in CW;

d) Encrypt CW with CREEK, resulting in EK1(CW) for descrambling services.

E_{K3_HSM}(K2H) denotes key K2H encrypted with key K3_HSM.

E_{K2H}(K1H) denotes K1H encrypted with key K2H.

$_{K1H}$(CW) denotes CW encrypted with key K1H

K3_HSM denotes HSM Root key sent during activation, 16 bytes.

K2H is the key for decrypting K1H, 16 bytes.

K1H is the key for decrypting CW, 16 bytes.

CREEK is the key for reencryption of the CW, 16 bytes.

The key ladder mechnism should support the SM4 algorithm defined in [b-GB/T 32907] and use a 128-bit data block in ECB mode.

6.4.4 Algorithm tools

The HSM is built in with sharable cryptographic tools for encryption/decryption, signing and verification, which can be utilized by CA vendors for development and implementation of their security solutions, so as to enhance the security on a shared platform.

The detailed implementation of HSM algorithm tools is not defined in this Recommendation.

6.4.5 Secure storage

The HSM secure storage area consists of a locked area, an activating storage area and a CA storage area.

6.4.5.1 Locked storage area

Some areas within the non-volatile memory (NVM) of the HSM have locking mechanism that, once written, data in this area will be locked and the locked area becomes read-only and cannot be modified or deleted anymore.

Data stored in the locked area includes: the TA root certificate, HSMID, HSM device certificate, HSM vendor certificate and HSM private key.

The TA root certificate is a root certificate issued by the SDMP, signed by itself. The HSM vendor certificate is a subordinate certificate issued by the TA root certificate, the HSM chipset certificate is a subordinate certificate issued by the HSM vendor certificate. The HSM chipset certificate contains the HSM public key. The HSM private key is stored in the locked storage area, as described in Table 2. See Annex C for certificate formats.

Table 2 – Data in locked storage

Key name	Length (Byte)	Type	Write method	Function description
HSM private key	32	SM2 private key	Production serialization	For signing and encrypting
HSM certificate	Variable length	SM2 public key and signature	Production serialization	For authentication and verification of HSM
TA root certificate	Variable length	SM2 public key and signature	Hard coded	For verification of other certificates

HSMID is an 8-byte public identifier of the HSM, which includes information such as HSM vendor and type, and a 4-byte globally unique identifier of a chipset. The HSMID format is described in Table 3.

Table 3 – HSMID data format

Data name	Length (Bit)	Data type
HSM Chip Manufacturer ID	8	uimsbf
HSM chip type	6	uimsbf
China Commercial Cryptographic Algorithm Indicator	1	uimsbf
HSM Chipset Vendor Specific Data	5	uimsbf
Reserved	12	uimsbf
HSM Chipset ID	32	uimsbf

HSM Chipset Vendor ID: Unique ID for a chipset vendor, 8 bits.

HSM chipset type: Model ID for a chipset manufactured by a chipset vendor, 6 bits.

China Commercial Cryptographic (CCC) Algorithm Indicator: To indicate if the CCC Algorithm is supported, 1 for supported, 0 for not supported. 1 bit.

HSM vendor specific data: 5 bits, content and usage are defined by each HSM vendor.

Reserved bits: All '0', 12 bits.

HSM chipset ID: a 32-bit serial number of a HSM chipset manufactured by an HSM manufacturer. This ID is globally unique to any HSM chipset, regardless of whether the chipset vendor or type is same.

6.4.5.2 Activating storage area

The activating storage is the area where activation data and activation-related data are stored. The data includes: K3_HSM, CREEK, PairK and auxiliary data, etc, as shown in Table 4.

Table 4 – Activating storage area

Key name	Length (Byte)	Type	Write method	Function description
SAC Pair Key (PairK)	16	Symmetric key	HSM activation	Used for establishing SAC
Cryptographic Engine Root Key (K3_HSM)	16	Symmetric key	HSM activation	Used for key ladder computation
Key for Reencryption of CW (CREEK)	16	Symmetric key	HSM activation	Used for reencryption of CW

6.4.5.3 CA storage area

The CA storage area is the area where CA private data is stored. It is divided into 2 parts: SAC authentication storage area and public storage area.

Accessing of SAC authentication storage area such as readings and writings must be done via SAC. It supports writing and reading of any data at a specified offset address.

Writings in public storage area must be done via SAC, whereas readings can be done without using SAC.

6.4.6 Secure authenticated channel (SAC)

SAC is a securely authenticated data channel established between the HSM and SoC. It can only be used in a TEE environment. The establishment of SAC relies on the activation of the HSM chipset, which means, only after HSM is activated can the SAC be successfully established. There are two stages in the using of SAC: Handshake and Communication.

See Annex C for details of SAC.

6.5 Security implementation mechanism (SIM)

6.5.1 The SIM of a terminal security chipset

OTP area

The OTP area of a terminal security chipset is used to store information such as ChipID, ESCK and BL_KEY0, etc. It shall comply with the following requirements:

a) Content written in OTP area cannot be changed;

b) The SCK is necessary security information required for root key derivation, a clear SCK should not be directly stored in OTP area;

c) The security information in the OTP area shall be able to be tested and verified by the terminal security chipset to see if it has been tampered. If the security information is tampered, the one-way DCAS function modules shall stop working immediately.

SCK de-obfuscation function

The SCK de-obfuscation function of a terminal security chipset shall comply with the following requirements:

a) Shall be a one-way function cryptographically secure enough;

b) Shall be implemented by internal hardware logical circuits, external software cannot intercept SCK or ESCK;

c) SCK de-obfuscation function shall only be used by the root key derivation module.

Root key derivation module

The root key derivation module of the terminal security chipset consists of sub modules such as the SCK preliminary processing function, the vendor separation function, and the final root key derivation function. The root key derivation module shall comply with the following requirements:

a) The root key derivation module shall be implemented by using hardware logical circuits or independent secure operation unit, working independently with dedicated calculation and storage resources. Any other units cannot interfere with the logic, execution, and result of the derivation module;

b) Any intermediate result from the running of the root key derivation module shall not be output or read to external modules.

c) The SCK preliminary processing function shall use a one-way function with a certain level of security strength, to ensure the SCK cannot be de-obfuscated in case Vendor_SysID and SCKv are known.

d) The vendor separation function shall use a one-way function with a certain level of security strength, to ensure SMK cannot be de-obfuscated in case Vendor_SysID and Seedv are known.

e) The final root key derivation function shall use a one-way function with a certain level of security strength, so that when root key K3 and any input parameter are known, another input parameter cannot be retrieved. For example, Seedv cannot be retrieved when K3 and SCKv are known.

f) The SMK shall not be read or tampered by external modules and its length shall be at least 128 bits;

g) The terminal security chipset can support a variety of final root key derivation functions at the same time.

Key ladder

The key ladder module of a terminal security chipset consists of a multi-level key mechanism and challenge-response mechanism. The key ladder module shall comply with the following requirements:

a) Must be implemented using hardware logical circuits or independent secure operation unit, and work independently with dedicated calculation and storage resources.

b) The logic, execution and results of the key ladder module cannot be interfered with or used by any other modules except the descrambling module that can get the CW from the key ladder module.

c) Any intermediate result from running of the key ladder shall not be output to or read out by an external module.

d) Any key in key ladder cannot be exposed in the challenge-response process.

6.5.2 The SIM of HSM

General SIM

The HSM shall comply with the following requirements:

a) HSM shall have a security mechanism to prevent HSM software from being tampered;

b) HSM shall have anti physical attack mechanisms such as voltage pulse detection and internal shielding, etc.

Secure storage

HSM secure storage shall comply with the following requirements:

a) Shall have anti-theft function for the HSM private key so that the HSM private key cannot be disclosed;

a) Shall have data anti-tamper mechanism. Data shall not be modified once written in locked storage areas of the HSM;

c) Shall have a security detection mechanism, to detect if data stored in locked storage is tampered and shall stop working immediately once data tampering is detected;

d) Shall have a data security protection mechanism during deactivation. When a deactivation command is received, all data in the SAC authenticated storage area shall be erased prior to moving to the deactivate state;

e) Shall be resistant to electronic pulse interference, reading and writing of storage area shall be able to resist electronic pulse interference.

SAC

The SAC of a HSM shall comply with the following requirements:

a) The random number used for establishing the SAC shall not be exposed as plain text outside the TEE of SoC;

b) Shall have a security mechanism to check handshake and data transfer. During handshake and data transfer, the HSM shall not respond if any error is detected from messages it receives;

c) Shall have a security design against replay attack and man-in-the-middle (MITM) attack.

Serialization

HSM serialization shall comply with the following requirements:

a) During serialization, the HSM private key should not be stored outside the HSM as plain text during serialization;

b) After serialization, the HSM private key should not be stored outside the HSM in any form.

Annex A

Security mechanism of one-way DCAS client software downloading and bootloading

(This annex forms an integral part of this Recommendation.)

A.1 Basic principles of chain of trust

The security mechanism of DCAS client software bootloading is based on a bottom-to-top chain of trust.

This chain of trust is established by using the digital signature technique. From bottom to top it includes: terminal security chipset, bootloader, terminal software platform and DCAS client software.

The security mechanism of DCAS client software bootloading requires that every link in the chain of trust must perform signature verification in a bottom-to-top order. Only if the signature verification at the current link passes can the security verification of next link be started. Only if the signature verification at all links passes can DCAS client software be launched.

In the entire chain of trust:

a) The OTP area of the terminal security chipset shall be preset with a verification key for bootloader.

b) Every link in the chain of trust shall be preset with a verification key for verifying software of the next link.

c) Software of every link of the chain of trust shall be digitally signed by a private key corresponding to the verification key.

d) Software of every link shall have its digital signature.

e) Software of the current link shall complete the security verification of the software of the next link first, before the software of the next link can be launched.

   The downloading, loading and running of DCAS client software shall comply with the mechanism of chain of trust described above at any time.

A.2 Bootup signature verification

The CPU shall execute certain secure codes to check that the bootloader in the Flash that is already signed. The bootloader needs to verify the signature of the terminal software platform and the terminal software platform needs to verify signed DCAS client software.

Figure A.1 shows the verification process of a bootloader.

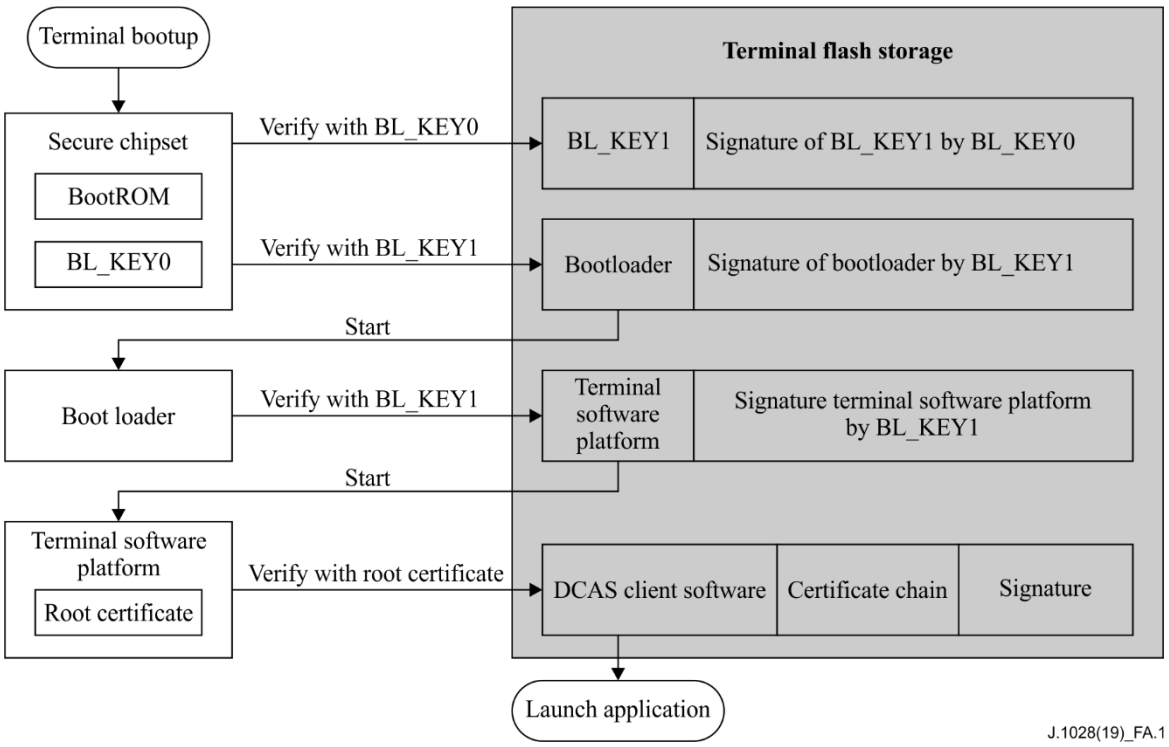

Figure A.1 – Verification of bootloader

A public key assigned by the SDMP, denoted as BL_KEY0, shall be embedded in the secure area of the terminal security chipset during SoC production. The SDMP shall under no circumstances leak BL_KEY0's private key. The BL_KEY0 in Figure A.1 is the public key.

During bootup, the code of the BootRom in the secure area of the terminal security chipset executes to read an additional public key (denoted by BL_KEY1) and signature of BL_KEY1 from the terminal flash storage, and use BL_KEY0 to verify BL_KEY1's signature. The BL_KEY1 in Figure A.1 is the public key.

After BL_KEY1 is successfully verified, BootRom shall verify the bootloader using BL_KEY1.

The bootloader shall use BL_KEY1 to verify the signature of the terminal software platform. The terminal software platform shall use its embedded root certificate to verify DCAS client software.

The algorithms used by all the signatures mentioned above include SM3 hash and SM2 public key cryptographic algorithm.

A.3 Downloading and replacing DCAS client software

The downloading and replacing of the DCAS client software including DCAS App and DCAS trusted App are implemented by the loader function of the bootloader or the application manager in the terminal software platform. The bootloader is used for the downloading and replacing of the whole terminal software image including terminal software platform and applications software, and the application manager is only used for the downloading and replacing of the applications software. The downloading and replacing procedure is as follows:

a) Headend system sends information of starting DCAS downloading and replacement to the terminal;

b) Bootloader or application manager downloads the image or DCAS client software;

c) Bootloader or application manager performs signature verification over the downloaded image or DCAS client software;

d) Bootoader or application manager replaces the image or DCAS client software with the verified new one;

e) The terminal reboots;

f) After replacement, the headend system notifies DCAS client software of HSM reactivation. See clause C.3.6 for details;

g) After the HSM is reactivated, the new DCAS client software shall be able to work with the HSM. Thus the downloading and replacing of DCAS client software are complete.

A.4 Key management

See Table A.1 for description of keys for the bootloader.

Table A.1 – Description of keys for the bootloader

Key name	Key owner	Signed by	Used to sign	Note
BL_KEY0	SDMP	N/A	BL_KEY1	Public key is embedded to chipset
BL_KEY1	Operator	BL_KEY0	Bootloader terminal software platform.	

For BL_KEY0

The SDMP shall manage its internal database for BL_KEY0, and shall be responsible for distributing BL_KEY0 to chipset vendors, which will be embedded to the designated chipset.

For BL_KEY1

BL_KEY1 is owned by the operator. The key could be managed by either the operator or a third trusted authority, who signs the terminal software.

The SDMP shall provide the method and procedure for signing BL_KEY1 with BL_KEY0. The owner of BL_KEY1 shall provide detailed information to the SDMP, including chipset model and public key of BL_KEY1. The SDMP shall sign BL_KEY1's public key with BL_KEY0's private key, and return the result together with BL_KEY1's public key to BL_KEY1's owner, so that BL_KEY1's public key and signed result can be preset into terminal device.

A.5 Security requirements of the bootloader

The security requirements of the bootloader focus on the bootup and download process. The bootloader in Flash should be copied to the RAM before boot, and the boot from RAM (BFR) bootloader shall comply with the chain of trust mechanism to verify the software signature during boot and loading.

For the bootup process

The bootloader shall not launch subsequent component software until their signatures are verified. Bootloader shall verify the signature again every time the subsequent software restarts. Only the bootloader stored in the terminal's flash can be executed. The signature verifications and executions of all software shall be performed in RAM.

For the download

The bootloader shall write downloaded software and its signature to Flash only after the signature of the downloaded software is verified in RAM. After software is successfully stored, the terminal shall perform a complete reboot. If the downloaded software exceeds the maximum size of flash allocated by the bootloader, the bootloader shall reject the software and reboot. Downgrades of upgradable software should be avoided.

A.6 Performance requirements of bootloader and terminal security chipset

To ensure the terminal user experience, a terminal security chipset shall be used to provide hardware acceleration for the hash algorithm.

Annex B

One-way DCAS APIs

(This annex forms an integral part of this Recommendation.)

B.1 Java APIs

The standard Java virtue machine solution has been widely used in the industry for downloading and executing applications. Figure B.1 illustrates such the runtime environment of the CAS client software.

DCAS client software is an Xlet application running on a terminal software platform that supports Java runtime environment.

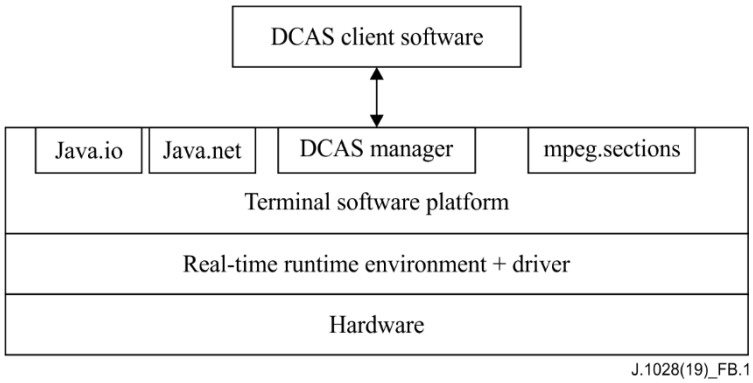

Figure B.1 – DCAS Java APIs

B.1.1 APIs type

B.1.1.1 APIs for CAS manager

A DCAS module manager (DCAS manager) has been defined for the terminal software platform to manage requests for descrambling services. The DCAS manager includes upper layer APIs and bottom layer APIs of the terminal software platform, and extension application APIs.

B.1.1.1.1 The upper layer APIs of terminal software platform

A CA module manager is defined by the upper-level APIs of the terminal software platform, to manage requests for descrambling services (which means to descramble video/audio streams). A DCAS client software must register the CA module in the CA module manager in order to receive the descramble request from the terminal software platform on a terminal device.

DCAS client software requires the terminal software platform to implement the upper-layer APIs of DCAS terminal software platform.

B.1.1.1.2 The bottom-layer APIs of terminal software platform

Except for the existing Java APIs, the DCAS client software requires a collective of Java APIs that are implemented on the terminal software platform, including the extension APIs required for accessing the terminal security chipset

B.1.1.1.3 Extension application APIs

By extending DCAS APIs, the CAS management module on a terminal software platform can perform basic CA information communication with Java Apps without being limited by using inter-xlet communication (IXC) between Java application and DCAS application.

The DCAS client software requires DCAS extension APIs to be implemented by terminal software platform for DCAS applications.

B.1.1.1.4 Detachable security device APIs

DCAS applications can communicate with detachable security devices via this APIs.

B.1.1.2 Network APIs

DCAS client software can use Java network APIs to access network resources, such as interconnection with a headend CA server.

DCAS client software requires the terminal software platform to implement the existing Java network API according to the definition in Java.net.

B.1.1.3 MPEG section filter APIs

DCAS client software uses MPEG section filter APIs to load the MPEG section for CA. CA related data includes ECM, EMM and CAT table.

The DCAS client software requires the terminal software platform to implement the MPEG section filter APIs according to definitions in org.davic.mpeg.sections, org.davic.mpeg.TransportStream and org.davic.net.tuning.NetworkInterface.

B.1.1.4 Non-volatile storage APIs

DCAS client software can use the existing Java APIs to access terminal storage, including storing data in non-volatile storage.

DCAS client software requires the terminal software platform to implement non-volatile storage APIs.

DCAS client software can store data by using a designated directory in the file system of a non-volatile storage. The terminal software platform needs to provide proper functions to read the name of root path of the file system.

B.1.2 APIs invoking sequence

This clause describes two scenarios where DCAS APIs is used: CASModule registration and channel switching. Figure B.2 shows CASModule registration and Figure B.3 shows channel switching.

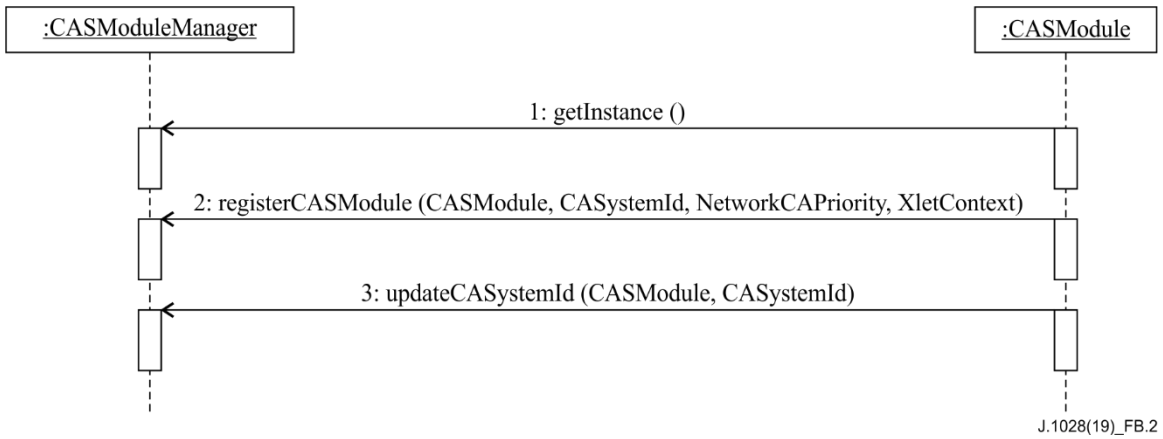

Figure B.2 – CA Module registration in CASModuleManager

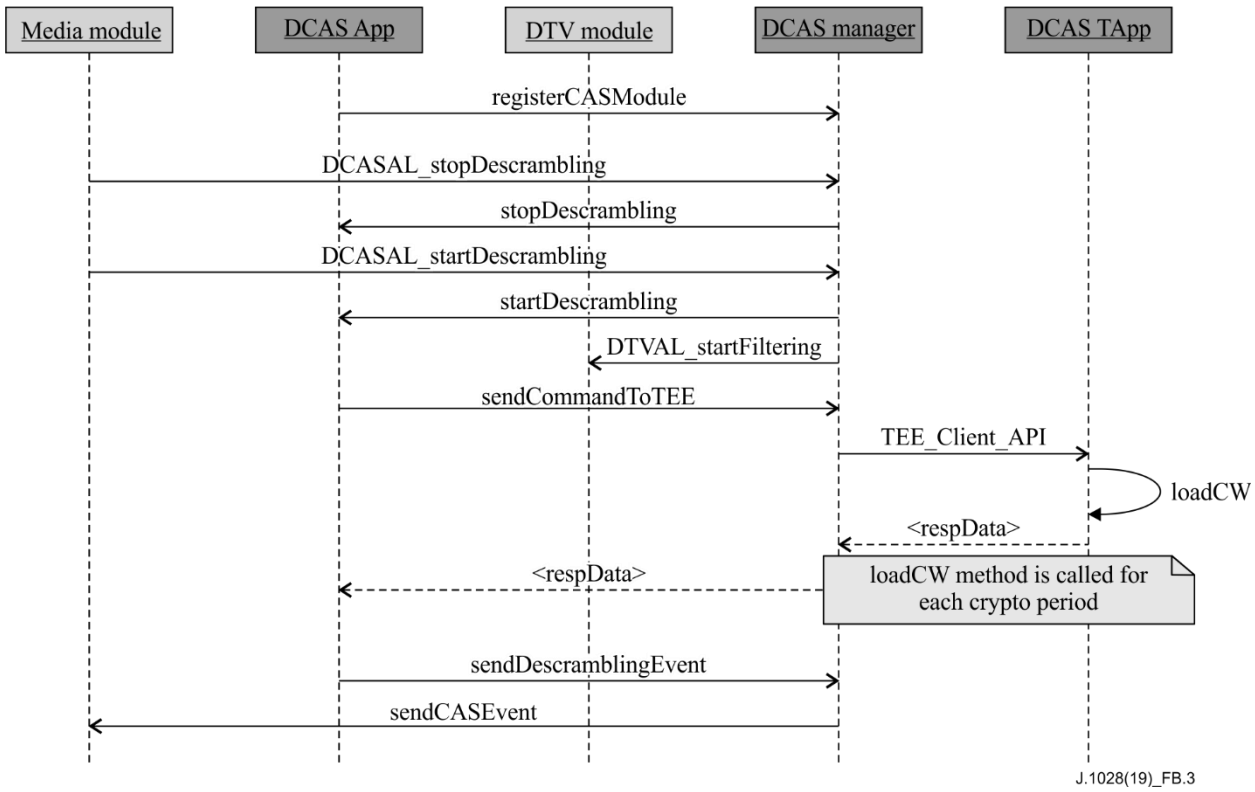

Figure B.3 – Channel selection

B.1.3 APIs description

Table B.1 lists the names of APIs.

Table B.1 – APIs

APIs name	Package name
Terminal software platform upper layer APIs	org.ngb.net.cas.module
Terminal software platform bottom layer APIs	org.ngb.net.cas.controller
Extension Application APIs	org.ngb.net.cas.event
Detachable Security Device APIs	org.ngb.net.cas.detachable
Network APIs	java.net
Section Filter APIs	org.davic.mpeg.sections
Non-volatile Storage APIs	java.io

B.1.4 Package org.ngb.net.cas.module

The package org.ngb.net.cas.module provides the upper layer APIs of the DCAS terminal software platform, which needs to be implemented in television operating system (TVOS).

Refer to Table B.2 for the overview of the package org.ngb.net.cas.module

Table B.2 – Overview of package org.ngb.net.cas.module

Interface	
CASModule	To denote the CASModule object that requests descrambling of a group of elementary streams.
CASDataUtils	To obtain and set CA information, as well as read and write DCAS data.
CADescriptor	Provide information of CA descriptor. The PMT of a given service may provide CA descriptors. CAT may also have CA descriptors.
CAServiceComponentInfo	For extracting componnet information of a CA service, for example, ECM PID and DescreamblerContext for loading CW.
CASPacketListener	DCAS application uses this interface to receive out of band CAS Packets (e.g. EMMs).
CASSession	Provide information about a CAS session.
CAStatus	DCAS application sends CAStatus every time the descrambling status in DescramblerContext changes. This status is usded to indicate if descrambling is successful. If any component fails to be descrambled, this status must report the failure of the request on descrambling the entire service. When receiving a new CAStatus, terminal software platform shall use the CAS event described in this section to notify other applications.
CATListener	Required by DCAS application to filter in-band EMM using CA descriptor in CAT.
CATNotifier	Used by DCAS application to register the listener for receiving CAT update notification.
Class	
CASModuleManager	Used to register all CASModules implemented by all DCAS applications.
CASPermission	Any DCAS application must get CASPermission to access CASModuleManager. This mechanism ensures only network operator authorised DCAS can use DCAS APIs.

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

B.1.4.1.1 Methods

B.1.4.1.1.1 startDescrambling

Prototype:

public void startDescrambling (CASSession,
CAServiceComponentInfo[]casci)

Description:

This method is invoked by terminal software platform to request CASModule to descramble a group of elementary streams in a given session.

DCAS application gets the relevant Network Interface object from CAS session, and gets the TransportStream object from the NetworkInterface object, which will be used for ECM section filtering via org.davic.mpeg.sectionsAPI.

Parameters:

casSession: Session for the descrambling request.

casci: CA service component information array. This array can be used to get ECM PID, as well as for the DescramblerContext object for loading CW in DCAS.

Returns:

None.

B.1.4.1.1.2 updateDescrambling

Prototype:

public void updateDescrambling (CASSession casSession,
CAServiceComponentInfo[]casci)

Description:

invoked by terminal software platform to update the descrambling component list in CASModule.

According to request, CASModule will start descrambling components added to the array and stop descrambling removed components.

There will be no change to components after update.

Note, this method is rarely invoked. It usually happens due to change of PMT in a session.

Terminal software platform can also invoke this method to notify CAModule when there is any change to a CAS session, e.g. the session type.

Parameters:

casSession: Session for the descrambling request.

casci: CA service component information array. This array can be used to get ECM PID, as well as for the DescramblerContext object for loading CW in DCAS.

Returns:

None.

B.1.4.1.1.3 stopDescrambling

Prototype:

public void stopDescrambling (CASSession casSession)

Description:

Invoked by terminal software platform to request CASModule to stop descrambling all components in a given session.

Parameters:

casSession Session for the descrambling request.

Returns:

None.

B.1.4.1.1.4 getCAInfo

Prototype:

public String getCAInfo (int cmdId,

String data)

Description:

Invoked by terminal software platform to get CA information.

Parameters:

cmdId: ID for command, can be extended according to actual project requirements.

data: Inquiry parameter.

Returns:

CA information data.

B.1.4.1.1.5 setCAInfo

Prototype:

public int setCAInfo (int cmdId,

String data)

Description:

Invoked by terminal software platform to set CA information.

Parameter:

cmdId: Unique ID for command, can be extended according to actual project's requirements.

Data: CA information data

Returns:

Return 0 if the set is successful

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

B.1.4.2.1 Description

This interface is used to represent general API for getting CA information.

B.1.4.2.2 Methods

B.1.4.2.2.1 getCAInfo

Prototype:

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

Description:

Terminal software platform gets CA information from DCAS application in response to user operations.

Parameters:

casId: Identifier of a CAS vendor

cmdId: Unique ID for command, can be extended according to actual project requirements

data: inquiry data

Returns:

CA information data.

B.1.4.2.2.2 setCAInfo

Prototype:

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

Description:

Terminal software platform sets CA information for DCAS application in response to user operations.

Parameters:

casId: Identifier of a CAS vendor

cmdId: Unique ID for command, can be extended according to actual project requirements

data: CA information data

Returns:

Return 0 if set successfully

B.1.4.2.2.3 getData

Prototype:

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

Description:

Terminal software platform gets data from DCAS manager in response to user operations.

Parameters:

casId: Identifier of a CAS vendor

cmdId: Unique ID for command, can be extended according to actual project requirements

type: Data type reference

Returns:

Data obtained

B.1.4.2.2.4 setData

Prototype:

```
public int setData ( int casId,
                     int cmdId,
                     int type,
                     String data )
```

Description:

Terminal software platform sets CA information for DCAS application in response to user operations.

Parameters:

casId: Identifier of a CAS vendor.

cmdId: Unique ID for command, can be extended according to actual project requirements.

Data: DCAS data

Type: Data type

Returns:

Return 0 if set successfully

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

B.1.4.3.1 Description

This interface provides information of CA descriptor, which may present in the PMT of a given service. CA descriptor may also appear in CAT.

B.1.4.3.2 Methods

B.1.4.3.2.1 getCASystemId

Prototype:

public int getCASystemId ( )

Description:

This method returns the CASystemId of inside CA descriptor.

Parameters:

None.

Returns:

CASystemId.

B.1.4.3.2.2 getPid

Prototype:

public int getPid ( )

Description:

This method returns PID (ECM PID or EMM PID) in CA descriptor.

Parameters:

None.

Returns:

PID value.

B.1.4.3.2.3 getPrivateData

Prototype:

public byte[] getPrivateData ( )

Description:

This method returns private data array in CA descriptor.

Parameters:

None.

Returns:

privateData array.

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

B.1.4.4.1 Description

This interface is used to extract information of certain CA service components, such as ECM PID and DescramblerContext for loading CW.

B.1.4.4.2 Methods

B.1.4.4.2.1 getDescramblerContext

Prototype:

public DescramblerContext getDescramblerContext ( )

Description:

This method returns the DescramblerContext object used by DCAS application for loading CW. In the cycle of PMT components, when the same CA descriptor (with same ECMPID and private data) appears multiple times there should be only one DescramblerContext object.

Parameters:

None.

Returns:

DescramblerContext object.

B.1.4.4.2.2 getCADescriptor

Prototype:

public CADescriptor getCADescriptor ( )

Description:

This method returns the CA descriptor related to service components. CADescriptor instance is generated by CA information in PMT.

Parameters:

None.

Returns:

A CADescriptor object.

B.1.4.4.2.3 getComponentStreamPIDs

Prototype:

public int[] getComponentStreamPIDs ( )

Description:

This method returns an elementary stream array described in PMT, the order of array elements shall be consistent with that of array elements returned by getComponentStreamType.

Parameters:

None.

Returns:

ES PID array.

B.1.4.4.2.4 getComponentStreamTypes

Prototype:

public int[] getComponentStreamTypes ( )

Description:

This method returns stream type array in PMT, stream types shall comply with MPEG standard. The order of array elements shall be in consistence with that of the array elements returned by getComponentStreamPID.

Parameters:

None.

Returns:

Stream Type Array.

B.1.4.4.2.5 getServiceIdentifiers

Prototype:

public int[] getServiceIdentifiers ( )

Description:

This method returns the service identifier array associated with an object, the form of service identifier is determined by the actual usage scenario.

Parameters:

None.

Returns:

ServiceID array.

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

B.1.4.5.1 Description

DCAS application uses this interface to receive out-of-band CAS Packets (e.g., EMMs). According to given CA system ID, DCAS application uses the method registerCasPacketListener provided by class CASModuleManager to register this listener. CA system ID is denoted as casId. Receiving of CAS packet depends on the implementations of terminal software platform.

B.1.4.5.2 Methods

B.1.4.5.2.1 casPacketArrived

Prototype:

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

Description:

DCAS application uses the registerd listener to get CAS packets.

Parameters:

casId CA: CA system ID

casPacketData: CAS packet data

casPacketHeader: Terminal software platform dependent CAS packet header.

Returns:

None.

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

B.1.4.6.1 Description

This interface provides information for CAS session.

B.1.4.6.2 Constants – Session Types

B.1.4.6.2.1 TYPE_PRESENTATION

public static final int TYPE_PRESENTATION = 0x00000001

B.1.4.6.2.2 TYPE_RECORDING

public static final int TYPE_RECORDING = 0x00000002

B.1.4.6.2.3 TYPE_BUFFERING

public static final int TYPE_BUFFERING = 0x00000004

B.1.4.6.3 Methods

B.1.4.6.3.1 getType

Prototype:

public int getType ( )

Description:

This method returns the session type of a session.

Parameters

None.

Returns:

Session type, which can be one or a combination of values defined in this interface.

For example – a result of 0x00000003 returned of this method is a combination of type (0x00000001) and (0x00000002)

B.1.4.6.3.2 getNetworkInterface

Prototype:

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

Description:

This method returns the NetworkInterface associated with CAS session. DCAS application can get NetworkInterface object from a CAS session. Using NetworkInterface, DCAS application can get object TransportStream, for invoking the org.davic.mpeg.sections APIs to perform ECM Section filtering.

Paramerters:

None.

Returns:

The NetworkInterface object.

B.1.4.6.3.3 getAssociatedService

Prototype:

public java.lang.Object getAssociatedService ( )

Description:

This method returns the service associated with a CAS session.

Parameters:

None.

Returns:

A Service object.

B.1.4.6.3.4 getServiceContext

Prototype:

public java.lang.Object getServiceContext ( )

Description:

This method returns ServiceContext associated with a CAS session.

NOTE – This method returns null in some cases where ServiceContext has no actual meaning.

Parameters:

None.

Returns:

ServiceContext object.

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

B.1.4.7.1 Description

Prototype:

public interface CAStatus

Description:

DCAS application uses this interface when invoking the sendDescramblingEvent method in CASModuleManager. DCAS application sends CAStatus every time the descrambling status in DescramblerContext changes. This status is used to indicate success or failure of a descrambling. If any one of the descrambling components fails to be descrambled, this status must report the failure of the descrambling request on the entire service. When terminal software platform receives a new CAStatus, it should notify other applications the CAS event described in this section. Detailed usages of this interface are described in the Extension Application Interface clause.

B.1.4.7.2 Methods

B.1.4.7.2.1 isSuccess

Prototype:

public boolean isSuccess ( )

Description:

This method returns status of a descrambling request.

Parameters:

None.

Returns:

Return true if descrambling succeeds, or false if descrambling fails.

B.1.4.7.2.2 getCAToken

Prototype:

public int getCAToken ( )

Description:

This method returns parameters with which other applications to inquire network information from DCAS application via IXC.

Parameters:

None.

Returns:

CA token.

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

DCAS application requires this interface to filter in-band EMM using the CA descriptor in CAT. DCAS application requires registerCATListener defined in CATNotifier to register the listener.

B.1.4.8.1 Methods

B.1.4.8.1.1 catUpdate

Prototype:

public void catUpdate (CADescriptor desc,
org.davic.net.tuning.NetworkInterface ni)

Description:

This interface is used for notifying DCAS application of CAT update at specified network interface. DCAS application can get the TransportStream object with the NetworkInterface object.

EMM section filtering can be achieved by TransportStream object by calling.

org.davic.mpeg.sections APIs. Client Software Platform will notify any CAT update to the CAT listener registered with the corresponding casId.

NOTE – If CAT is no longer to be filtered (after being successfully filtered); or CA descriptor is deleted from CAT, terminal software platform should invoke as catUpdate(null, theNetworkInterface).

Parameters:

Desc: The CA descriptor. DCAS application use the CASDescriptor object to get EMM PID.

ni: Updated NetworkInterface where CAT is transported.

Returns:

None.

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

B.1.4.9.1 Description

Prototype:

public interface CATNotifier.

Description:

DCAS application uses this interface to register the listener for CAT update notification.

DCAS application uses CAT information to filter in-band EMM.

B.1.4.9.2 Methods

B.1.4.9.2.1 registerCATListener

Prototype:

public void registerCATListener (int casId,
CATListener catListener)

Description:

DCAS application invokes this method to register a CATListener.

Parameters:

casId: CA system ID

catListener: CATListerner object for registration.

Returns:

None.

B.1.4.9.2.2 unregisterCATListener

Prototype:

public void unregisterCATListener (CATListener catListener)

Description:

DCAS application invokes this method to unregister a CATListener.

Parameters:

catListener: CATListener that needs to be unregistered.

Returns:

None.

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

B.1.4.10.1 Description

Used to register all CASModules implemented by DCAS applications.

B.1.4.10.2 Methods

B.1.4.10.2.1 getInstance

Prototype:

public static CASModuleManager getInstance ( ) throws java.lang.SecurityException.

Description:

This method is used to get a CASModuleManager instance.

Parameters:

None.

Returns:

CASModuleManager Instance.

Exception Handling:

java.lang.SecurityException – Throw this exception when security policy is enforced but no org.ngb.net.ca.module.CASPermission has been assigned to the invoker.

B.1.4.10.2.2 registerCASmodule

Prototype:

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

Description:

This method is used by DCAS application for registering a CASModule on terminal software platform.

Parameters:

caModule: CASModule that needs to be registered.

caSystemId: caSystemId managed by the CASmodule.

networkCAPriority: used when registering more than one CASModules in CASModuleManager. Operator can decide whether this parameter is optional for each CASModule. When the priority policy is enforced, operator needs to specify priority for every CASModule. Terminal software platform should send descrambling request to the registered CASModule which has the hightest priority and of which the managed caSystemId mapping to the CA descriptor can be found in PMT. When the priority policy is disabled, DCAS application should set this parameter to 0. The method for selecting CASModule will be determined by the Terminal software platform.

context: context of DCAS application that needs to register a CASModule. Used for terminal software platform in deciding to identify of a DCAS application.

Returns:

None.

Exception Handling:

java.lang.IllegalArgumentException – Throw this exception if the specified CASModule instance has been registered.

B.1.4.10.2.3 updateCASystemId

Prototype:

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

Description:

This method is used by DCAS application to update CASystemId in a CASModule on application platform.

Parameters:

aModule : CASModule to be updated

caSystemId : new caSystemId to be associated to the CASModule.

Returns:

None.

Exception handling:

java.lang.IllegalArgumentException: If given CASModule instance has not been registered.

B.1.4.10.2.4 sendDescramblingEvent

Prototype:

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

Description:

This method is used by DCAS App to return a CAStatus to terminal software platform. Every time a DescramblerContext changes as any scrambled element in a service changes, DCAS App must send CAStatus to indicate if the descrambling is successful. If any element fails to descramble, CAStatus must notify the descrambling failure of the entire service.

When receiving a new CAStatus, terminal software platform should continue to send the information to related applications using CAS Event defined in extension Application APIs.

Parameters:

aModule: Associated CASModule.

casSession: Session for descrambling request.

aCAStatus: CAStatus to be sent.

Returns:

None.

Exception Handling:

java.lang.IllegalArgumentException: if the given CASModule instance has not been registered.

B.1.4.10.2.5 unregisterCASModule

Prototype:

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

Description:

This method is used by DCAS application to unregister a CASModule from terminal software platform.

Parameters:

aModule: CASModule to be unregistered.

Returns:

None.

Exception Handling:

java.lang.IllegalArgumentException: if the given CASModule instance has not been registered.

B.1.4.10.2.6 getChipControllers

Prototype:

public ChipController[] getChipControllers ( )

Description:

This method is used by DCAS application to request available chip controller list from terminal software platform. This method returns one chip controller for each terminal security chipset. Most terminals support only one chip controller, in which case the returned array only contains one element.

Parameters:

None.

Returns:

A chip controller array.

B.1.4.10.2.7 setCurrentController

Prototype:

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

Description:

This method is used to set up default chip controller according to given CAModule for descrambling. Not required to specify it in CASModuleManager if this method is not invoked.

Parameters:

aModule: Associated CASModule.

aChipController: Default chip controller in use.

Returns:

None.

Exception Handling:

java.lang.IllegalArgumentException: If the given CASModule instance has not been registered.

B.1.4.10.2.8 setCCIBits

Prototype:

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

Description:

This method is used for setting up CCI (Copy Control Information) bits required for copy control of a service. The definition of CCI bits is assigned, and will be explained and executed by terminal software platform.

Parameters:

aModule: Associated CASModule.

casSession: Session for descrambling request.

cciBits: CCI bits value used by current service.

Returns:

None.

B.1.4.10.2.9 setServiceListFilter

Prototype:

public void setServiceListFilter (int filterData)

Description:

This method is used to provide terminal software platform parameters for filtering service list. Definition of service list parameter is assigned and executed by terminal software platform.

Parameters:

filterData: filterData service list filter parameters.

Returns:

None.

B.1.4.10.2.10 registerCASPacketListener

Prototype:

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

Description:

This method is used by DCAS application to register a CAPacketListener. The CAPacketListener is invoked by terminal software platform to transport CAS data packet (e.g., EMM) to DCAS application. CA system ID is denoted by the casID parameter. The retrieving of CAS data packet is implemented by terminal software platform itself.

Parameters:

casId: CasystemID

casPacketListener: CASPacketListener to be registered

Returns:

None.

Exception Handling:

java.lang.IllegalArgumentException: If a listener has been registered with the given casID.

B.1.4.10.2.11 unregisterCASPacketListener

Prototype:

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

Description:

This method is used by DCA application to unregister a CASPacketListener.

Parameters:

casPacketListener: The listener to be unregistered.

Returns:

None.

Exception Handling:

java.lang.IllegalArgumentException: If the given CASPacketListener has not been registered.

B.1.4.10.2.12 getDetachableSecurityDevices

Prototype:

public DetachableSecurityDevice[] getDetachableSecurityDevices ( )

Description:

This method is used by DCAS application to get the object handle of a detachable device (e.g., smart card).

Parameters:

None.

Returns:

A DetachableSecurityDevice object.

B.1.4.10.2.13 receiveOsdMsg

Prototype:

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

Description:

Display OSD message, the meaning of its parameters are project-specific.

Parameters:

msg: OSD message content, it can also contain descriptive information other than text content.

flags: OSD type indicator.

Returns:

None.

B.1.4.10.2.14 showFingerMsg

Prototype:

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

Description:

Display fingerprint message, the meaning of its parameters are project-specific.

Parameters:

aModule: Associated CASModuel.

casSession: Session for descrambling request.

msg: Fingerprint information, NULL for disabling fingerprint display.

Returns:

None.

B.1.4.10.2.15 receiveTuningAlert

Prototype:

public void receiveTuningAlert (int[] serviceIdentifiers,

int[] flags)

Description:

Emergency broadcast. In some projects the parameters of an emergency broadcast are not sent by CA system, in which case this function is not required to be implemented.

Parameters:

serviceIdentifiers: A group of values for identifying emergency broadcast channel parameters. Meaning of the values are defined in actual project.

Flags: Parameters used to denote the type of emergency broadcast.

Returns:

None.

B.1.4.10.2.16 getCATNotifier

Prototype:

public CATNotifier getCATNotifier ( )

Description:

This method is used by DCAS application to get the CATNotifier object. DCAS application can register to the CAT notifier with a listener for getting CAT update notification.

Parameters:

None.

Returns:

The CAT Notifier object.

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

B.1.4.11.1 Description

Prototype:

public class CASPermission extends java.security.BasicPermission.

Description:

Any DCAS application must get CASPermission before accessing CASModuleManager.

This mechanism is used to ensure only the DCAS App authorized by network operator is able to use DCAS APIs.

B.1.4.11.2 Methods

B.1.4.11.2.1 CASPermission

Prototype:

public CASPermission (String name)

Description:

Create a new CASPermission. If the Name string is not to be used and it should be set to NULL.

Parameters:

Name: Name of the CASPermission.

Returns:

None.

B.1.4.11.2.2 CASPermission

Prototype:

public CASPermission (String name,

String actions)

Description:

Create a new CASPermission. If the Name string is not to be used and it should be set to NULL. If the actions string is not to be used and it should be set to NULL. This constructor is used by the java.security.Policy object to instantiate a new Permission object.

Parameters:

Name: Name of the CASPermission.
Actions: Action list.

Returns:

None.

B.1.5 Package org.ngb.net.cas.controller

The org.ngb.net.cas.controller package provides the bottom APIs for DCAS terminal software platform, which needs to be implemented in TVOS.

See Table B.3 for the overview of org.ngb.net.cas.controller package.

Table B.3 – Overview of org.ngb.net.cas.controller package

Interface	
DescramblerContext	Component for controlling the descrambling function of terminal security chipset. Multiple DescramblerContext instances can be used to descramble multiple streams using different keys.
ChipController	Component used to control the execution of terminal security chipset.
Class	
Key	A fundmental cryptographic key, used to describe the cryptography used by K-LAD and the output parameters of the cipher function.
CWKey	Descrambling key, a.k.a. control word.

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

B.1.5.1.1 Description

Prototype:

public interface DescramblerContext.

Description:

Component used to control the descrambling of terminal security chipset. Multiple DescramblerContext instances can be used to descramble multiple streams using different keys.

B.1.5.1.2 Methods

B.1.5.1.2.1 loadCW

Prototype:

public void loadCW (int Vendor_SysID,

CWKey cwKey,

Key[] levelKeys,

int schemeId)

throws CADriverException

Description:

This method is used to load CW to descrambler on terminal software platform, and load required keys into terminal security chipset.

A descrambler channel is a logical collective of all streams descrambled by a single CW.

It depends on DescreamblerContext to use the scrambler channel.

Besides, DCAS App should notify terminal software platform that current CW is invalid (e.g., due to unauthorized entitlements), and terminal software platform should stop the corresponding descrambling. In this case, DCAS App will provide a null CWKey.

Parameters:

Vendor_SysID: This value is used to identify CA vendor and support root key derivation in controller. The root key of terminal security chipset is derived from this value.

cwKey: control word. If control word is plaintext, the levelKeys parameter is ignored. If cwKey is null, no valid control word is provided by DCAS App.

levelKeys: Multi-level keys to be set to terminal security chipset. The index of key array is the same as its absolute position in terminal security chipset. In an array an element with value Null indicates that no key should be loaded to the corresponding place in terminal security chipset.

So: levelKey[0] is Key 1 (encrypted by Key 2); levelKey[1] is Key 2 (encrypted by Key3); levelKey[2] is not to be used.

schemeId: This schemeId is used to specify cryptography for terminal security chipset. A list of scheme values are defined in the ChipController interface. This value is ignored if controller only supports one scheme.

Returns:

None.

Exception Handling:

CADriverException: If key loading fails.

B.1.5.1.2.2 overrideChipController

Prototype:

public void overrideChipController (ChipController aChipController)

throws CADriverException.

Description:

This method is used by DCAS App to request terminal software platform to override default terminal security chipset key ladder (can be done by invoking the setCurrentController method of CASModuleManager). If this method is not invoked, terminal security chipset will use the default controller. This method is only used in terminal security chipset systems where multiple terminal security chipsets are implemented.

Parameters:

aChipController: Controller to be overridden.

Returns:

None.

Exception Handling:

CADriverException: If operation fails.

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

B.1.5.2.1 Description

Prototype:

public interface ChipController.

Description:

component used to control the execution of terminal security chipset.

B.1.5.2.2 Constants

B.1.5.2.2.1 SCHEME_TDES

public static final int SCHEME_TDES=0

Description: Used to indicate TDES should be used by terminal security chipset.

B.1.5.2.2.2 SCHEME_AES

public static final int SCHEME_AES=1

Description: Used to indicate AES should be used by terminal security chipset.

B.1.5.2.2.3 PROCESSING_MODE_REGULAR

public static final int PROCESSING_MODE_REGULAR=0

Description: Used to indicate that terminal security chipset does not need additional processing in the challenge-response algorithm.

B.1.5.2.2.4 PROCESSING_MODE_POST_PROCESSING

Public static final int PROCESSING_MODE_POST_PROCESSING=1

Description: Used to indicate that terminal security chipset needs post processing in the challenge-response algorithm.

B.1.5.2.3 Methods

B.1.5.2.3.1 getPublicId

Prototype:

public byte[] getPublicId ( ) throws CADriverException.

Description:

This method returns the public ID of terminal security chipset.

Parameters:

None.

Returns:

The publicId of terminal security chipset.

Exception Handling:

CADriverException: If communication error occurs when accessing the driver of terminal security chipset.

B.1.5.2.3.2 getChipType

Prototype:

public byte[] getChipType ( ) throws CADriverException.

Description:

This method returns the type ID of terminal security chipset.

Parameters:

None.

Returns:

Terminal security chipset type.

Exception Handling:

CADriverException: if communication error occurs when accessing the driver of terminal security chipset.

B.1.5.2.3.3 getChipControllerProperty

Prototype:

public java.lang.String getChipControllerProperty (java.lang.String propertyName)
throws CADriverException

Description:

This method returns the value of the corresponding property according to the property name provided for terminal security chipset. This function is reserved in this interface, and can be used to read properties that will be added to the controller in the future. No property name is defined at present.

Parameters:

propertyName: Property name.

Returns:

Property value.

Exception Handling:

CADriverException: if communication error occurs when accessing the driver of terminal security chipset.

B.1.5.2.3.4 authenticate

Prototype:

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

Description:

This method is used to authenticate the key ladder mechanism in terminal security chipset. Terminal security chipset should calculate authentication information according to the input challenge information.

Parameters:

Vendor_SysID: This value is used to identify CA vendor, which is used to support root key derivation in controller. The root key of terminal security chipset is derived from this value.

challenge: Challenge information, nonce.

levelKeys: level keys required by key ladder. The index of key array is equal to the absolute position in terminal security chipset. In an array an element with value Null indicates that no key should be loaded to the corresponding place in terminal security chipset. That is, levelKey[0] is Null; levelKey[1] is Key 2 (encrypted by Key 3); levelKey[2] is not used.

schemeId: This schemeId is used to specify the cipher algorithm of terminal security chipset (such as AES and TDES). The ChipController interface defines the list of scheme. If the controller supports only one scheme, then the value will be ignored.

processingMode: it is used to specify whether additional post-processing in the calculation of the response needs to be implemented. If the controller only supports no post-processing mode, then the parameter will be ignored.

Returns:

Response calculated by terminal security chipset.

Exception Handling:

CADriverException: If communication error occurs when accessing the driver of terminal security chipset.

B.1.5.2.3.5 encryptData

Prototype:

```
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
```

Description:

This method invokes chip functions to encrypt data in memory.

Parameters:

Vendor_SysID: this parameter is used to identify CA vendor. Security chipset uses this value to derive root key.

cwKey: control word for encryption. If control word is not encrypted, subsequent levelKeys will be ignored.

levelKeys: The index of key element in the array is equal to the absolute position in key ladder. Null element in array indicates no key need to be set in the corresponding position in the key ladder.

schemeId: Cipher algorithm used by key ladder. If the chipset supports only one algorithm, then the parameter will be ignored.

encryptionId: Algorithm of data encryption/decryption, if the chipset supports only one algorithm, then the parameter will be ignored.

src: source data array.

srcPos: starting position of source data array.

dest: Destination data array.

destPos: starting position of destination data array.

length: length of data to be processed, in byte.

Exception Handling:

CADriverException: Throw the CADriverException exception when key ladder communication error occurs.

B.1.5.2.3.6 decryptData

Prototype:

```
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
```

Description:

This method invokes chipset functions to decrypt data in memory.

Parameters:

Vendor_SysID: This parameter is used to identify CA vendor. Security chipset uses this value to derive root key.

cwKey: Control word for decryption. If control word is not encrypted, subsequent levelKeys will be ignored.

levelKeys: The index of key element in the array is equal to the absolute position in key ladder. Null element in array indicates no key need to be set in the corresponding position in the key ladder.

schemeId: Cipher algorithm used by key ladder. If the chipset supports only one algorithm, then the parameter will be ignored.

encryptionId: Algorithm of data encryption/decryption. If the chipset supports only one algorithm, then the parameter will be ignored.

Src: source data array.

srcPos: starting position of source data array.

dest: destination data array.

destPos: starting position of destination data array.

length: data size to be processed, in byte.

Returns:

None.

Exception Handling:

CADriverException: Throw the CADriverException exception when key ladder communication error occurs.

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

B.1.5.3.1 Description

Prototype:

public class Key

Description:

It denotes a basic cipher key used to describe the cryptography for K-LAD and output parameters of cipher functions.

B.1.5.3.2 Methods

B.1.5.3.2.1 Key

Prototype:

public Key (byte[] value,

boolean encrypted)

Parameters:

Value: The key value.

Encrypted: Tag to indicate if a key is encrypted. True means key has been encrypted, false means key is plaintext.

B.1.5.3.2.2 getKeyValue

Prototype:

public byte[] getKeyValue ( )

Description:

This method returns the key value.

Parameters:

None.

Returns:

The key value.

B.1.5.3.2.3 isEncrypted

Prototype:

public boolean isEncrypted ( )

Description:

When this method returns true, it means key is encrypted, whereas false means key is not encrypted.

Parameter:

None.

Returns:

True means key is encrypted, false means key is not encrypted.

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

B.1.5.4.1 Description

Prototype:

public class CWKey extends Key.

Description:

This class denotes descrambling key or control word.

B.1.5.4.2 Constant

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 Methods

B.1.5.4.3.1 CWKey

Prototype:

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

Description:

value: value of key.

encrypted: true indicates key is encrypted, false indicates key is not encrypted.

parity: indicates the parity of control word.

B.1.5.4.3.2 getParity

Prototype:

public int getParity ( )

Description:

This method returns the parity of control word.

Parameters:

None.

Returns:

The parity of control word.

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

B.1.5.5.1 Description

Prototype:

public class CASTEEManager.

Description:

The interface for communicating with TA in TEE.

B.1.5.5.2 Methods

B.1.5.5.2.1 sendCommandToTEE

Prototype:

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

Description:

DCAS App selects a dedicated security application, and send data to it.

Parameters:

teeAppUUID: UUID of the selected TAPP.

commandId: Type of command.

inputData: Data input.

Returns:

Data returned.

Exception Handling:

CADriverException: throws CADriverException if communication error occurs in the interaction with TEE driver.

B.1.6 Package org.ngb.net.cas.event

The org.ngb.net.cas.event package provides extension APIs package for DCAS. It's required to implement this package by DCAS of TVOS.

See Table B.4 for overview of Org.ngb.net.cas.event package.

Table B.4 – Overview of org.ngb.net.cas.event package

Interface	
CASEventListener	Shall be implemented by the application that needs to receive CAS event.
CASAppInfo	Provides information of DCAS App
CASEventInfo	Provides information of CASEvent
Class	
CASEventManager	CASEventManager should be used to register listener to get CAS event.

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

B.1.6.1.1 Description

Prototype:

public interface CASEventListener.

Description:

This interface should be implemented by the application that needs to receive CAS events. CAS events provide CA Status and basic information of the current ServiceContext.

B.1.6.1.2 Methods

B.1.6.1.2.1 receiveCASEvent

Prototype:

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

Description:

This method is used to transfer CAS envent to App that has registered CAS event listener.

Parameter:

serviceContext: Handle to which CAS event belongs.

appId: Used to identify the DCAS App that sends events. The Id can be used by App to communicate with DCAS App via IXC. When no DCAS App is available to descramble a given stream, terminal software platform should use NULL as the value of appId to invoke this method. Application for receiving notifications about such CAS event should handle such case according to its design and implementation.

orgId: orgId is used to identify the organization of DCAS App that sends events.

isSuccess: Bolean value used to indicate descrambling success/failure.

caToken: Token sent back to DCAS via IXC. Applications can use this token to search specific network information via IXC.

B.1.6.1.2.2 receiveCASOSDEvent

Prototype:

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

Description:

This method is used to transfer OSD event of CAS to application that has registered CAS event listener.

Parameters:

serviceContextCAS: Handle to which CAS OSD event belongs.

appId: Used to identify the DCAS App that send events. These ID can be used by App to communicate with DCAS App via IXC. When no DCAS App is available to descramble given stream, terminal software platform should use NULL as the value of casAppId to invoke this method. Application for receiving notfications about such CAS event should handle such case according to its design and implementation.

orgId: orgId is used to identify the organization of DCAS App that sends events.

msg: Used to transfer OSD content.

flag: Used to identify OSD type.

B.1.6.1.2.3 receiveCASFingerEvent

Prototype:

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

Description:

This method is used to transfer CAS fingerprint event to application that has registered CAS event listener.

Parameters:

serviceContext: Handle to which CAS Fingerprint event belongs.

appId: Used to identify the DCAS App that send events. These ID can be used by App to communicate with DCAS App via IXC. When no DCAS App is available to descramble given stream, terminal software platform should use NULL as the value of casAppId to invoke this method. Application for receiving notifications about such CAS event should handle such case according to its design and implementation.

orgId: orgId is used to identify the organization of DCAS App that send events.

msg: Used to transfer the fingerprint data.

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

B.1.6.2.1 Description

Prototype:

public interface CASAppInfo

Description:

This interface provides information of DCAS application.

B.1.6.2.2 Methods

B.1.6.2.2.1 getAID

Prototype:

public int getAID()

Description:

This method returns the application ID of DCAS application.

Parameters:

None.

Returns:

The application ID of DCAS application.

B.1.6.2.2.2 getOID

Prototype:

public int getOID ( )

Description:

This method returns the organization ID of DCAS application.

Parameters:

None.

Returns:

The organization ID of DCAS application.

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

B.1.6.3.1 Description

Prototype:

public interface CASEventInfo.

Description:

This interface provides information of CASEvent.

B.1.6.3.2 Constant

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 Methods

B.1.6.3.3.1 getType

Prototype:

public int getType()

Description:

This method returns the type of the operation that produces the CAS Event.

Parameters:

None.

Returns:

Operation type, can be one or combination of the values defined in this interface.

For example – A returned value 0x00000003 is combination of type (0x00000001) and (0x00000002)

B.1.6.3.3.2 getNetworkInterface

Prototype:

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

Description:

This method returns the NetworkInterface related to CAS Event.

Parameters:

None.

Returns:

A NetworkInterface object.

B.1.6.3.3.3 getAssociatedService

Prototype:

public java.lang.Object getAssociatedService ( )

Description:

This method returns the associated service to the CAS Event.

Parameters:

None.

Returns:

A Service object.

B.1.6.3.3.4 getServiceContext

Prototype:

public java.lang.Object getServiceContext ( )

Description:

This method returns associated ServiceContext to the CAS Event.

Please note that in some cases, ServiceContext may not have actual meaning, and this method thus returns null.

Parameters:

None.

Returns:

A ServiceContext object.

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

B.1.6.4.1 Description

Prototype:

public class CASEventManager

Description:

Application uses CASEventManager to register listener to receive CAS event.

CA event provides current CA Status and basic information.

B.1.6.4.2 Methods

B.1.6.4.2.1 getInstance

Prototype:

public static CASEventManager getInstance ( )

Description:

This method is used to get a CASEventEManager instance. Singleton.

Parameters:

None.

Returns:

The CASEventManager instance.

B.1.6.4.2.2 addListener

Prototype:

public void addListener (CASEventListener aCASEventListener)

Description:

This method is used by application to register a CASEventListener for transferring all CAS events.

Parameters:

aCASEventListener: CASEventListener to be registered

Returns:

None.

B.1.6.4.2.3 removeListener

Prototype:

public void removeListener (CASEventListener aCASEventListener)

Description:

This method is used to unregister a CASEventListener.

Parameters:

aCASEventListener: CASEventListener a registered CASEventListener.

Returns:

None.

B.1.7 Package org.ngb.net.cas.detachable

The org.ngb.net.cas.detachable package provides DCAS detachable security device APIs. TVOS needs to implement this package.

See Table B.5 for the overview of Org.ngb.net.cas.detachable package.

Table B.5 – Overview of Org.ngb.net.cas.detachable package

Interface	
DetachableSecurityDevice	Used by application to register the listener for detachable security device to get the plugging status of a device.
DetachableSecurityDeviceListener	The listener for detachable security device status. It should be implemented by the application that needs to listen to the plugging status of a device.

B.1.7.1 Interface DetachableSecurityDevice

B.1.7.1.1 Description

This interface denotes the components used to control communications with detachable security devices (e.g., smart card).

B.1.7.1.2 Methods

B.1.7.1.2.1 open

Prototype:

public void open ( ) throws CADriverException

Description:

This method is used by DCAS App to initiate session with detachable security devices.

Parameters:

None.

Returns:

None.

Exception Handling:

CADriverException: If driver error occurs.

B.1.7.1.2.2 close

Prototype:

public void close ( ) throws CADriverException

Description:

This method is used by DCAS application to close session with detachable security devices.

Parameters:

None.

Returns:

None.

Exception Handling:

CADriverException: If driver error occurs.

B.1.7.1.2.3 reset

Prototype:

public byte[] reset ( ) throws CADriverException

Description:

This method is used to reset detachable security device and return data (ATR in the case of smart card).

Parameters:

None.

Returns:

A byte array to store the data returned after device reset.

Exception Handling:

CADriverException: If driver error occurs.

B.1.7.1.2.4 sendData

Prototype:

public void sendData (byte [] data) throws CADriverException

Description:

This method is used by DCAS App to send data to detachable security device.

Parameter:

data: Data to be sent (the command APDU in the case of smart card)

Returns:

None.

Exception Handling:

CADriverException: if driver error occurs.

B.1.7.1.2.5 registerListener

Prototype:

public void registerListener (DetachableSecurityDeviceListener aListener)

Description:

This method is used by DCAS to register the listener for receiving data sent by detachable security device.

Parameters:

aListener: DetachableSecurityDeviceListener to be registered.

Returns:

None.

B.1.7.1.2.6 removeListener

Prototype:

public void removeListener ( )

Description:

This method is used by DCAS App to remove a registered listener.

Parameters:

None.

Returns:

None.

B.1.7.2 Interface DetachableSecurityDeviceListener

B.1.7.2.1 Description

This method should be implemented by DCAS App to receive the status of detachable security device and data sent by it.

B.1.7.2.2 Fields

B.1.7.2.2.1 DEVICE_IN

public static final int DEVICE_IN = 1

Description: used to describe status of detachable security device: Inserted (indicates smart card is inserted in the case of smart card)

B.1.7.2.2.2 DEVICE_OUT

public static final int DEVICE_OUT = 2

Description: used to describe status of detachable security device: unplugged (indicates smart card is unplugged in the case of smart card)

B.1.7.2.2.3 DEVICE_ERROR

public static final int DEVICE_ERROR = 3

Description: used to describe status of detachable security device: ERROR (indicates smart card error in the case of smart card)

B.1.7.2.3 Methods

B.1.7.2.3.1 receiveDeviceStatus

Prototype:

public void receiveDeviceStatus (int status)

Description:

This method should be implemented by DCAS App to receive status of detachable security device.

Notify DCAS App when status of detachable security device changes.

Parameters:

status: Status of detachable security device (See description of the field).

Returns:

None.

B.1.7.2.3.2 receiveData

Prototype:

public void receiveData (byte [] data)

Description:

This method is invoked when detachable security device sends data to DCAS App.

Parameters:

data: Data sent by detachable security device (response APDU in the case of smart card)

Returns:

None.

B.2 Javascript APIs

B.2.1 Overview

DCAS Client Software can be developed based on DCAS Javascript APIs, in order to run in a client software platform that supports HTML5 execution environment. See Table B.6 for the overview of DCAS Javascipt interface.

Table B.6 – DCAS App Javascript interface overview

Class name	Description
JSDCAS.CASDescriptor	The CA descriptor object represents a CAS Descriptor which appears either in the PMT (Program Map Table) for a selected MPEG service, or in the CAT (Conditional Access Table).
JSDCAS.CASEcmEvent	The ECM event object contains ECM event.
JSDCAS.CASEmmEvent	The EMM Event object contains EMM event.
JSDCAS.CASFilter	The CAS Filter object represents filter criteria that the JSDCAS application defines when requesting the platform to filter OOB EMMs or inband EMMs.

Table B.6 – DCAS App Javascript interface overview

Class name	Description
JSDCAS.CASM	This is the CASM global object by which CAS manager object and controller object can be accessed.
JSDCAS.CASModule	An interface of a CAS Module object that the JSDCAS application should implement and export by registering in the platform CAS Module Manager, in order to receive descrambling requests, ECMs, EMMs, and any other metadata that the specific JSDCAS application requires.
JSDCAS.CASModuleManager	CAS Module Manager utilized by the JSDCAS application to receive descrambling requests, ECMs, EMMs, as well as report CAS descrambling status.
JSDCAS.CASPacketEvent	The CAS Packet Event object notify JS DCAS application about any CAS packet event.
JSDCAS.CASSession	The object represents a CAS Session that is generated by the platform for a specific descrambling request.
JSDCAS.CASStatus	This object is created by the JSDCAS application for reporting a CAS Status to the platform CAS manager.
JSDCAS.TeeController	The TEE Controller object is used by the JSDCAS application to communicate with the TEE (Trusted Execution Environment).
JSDCAS.TeeRetVal	The TEE Ret object is returned from TEE containing data or error.

B.2.2 APIs calling sequence

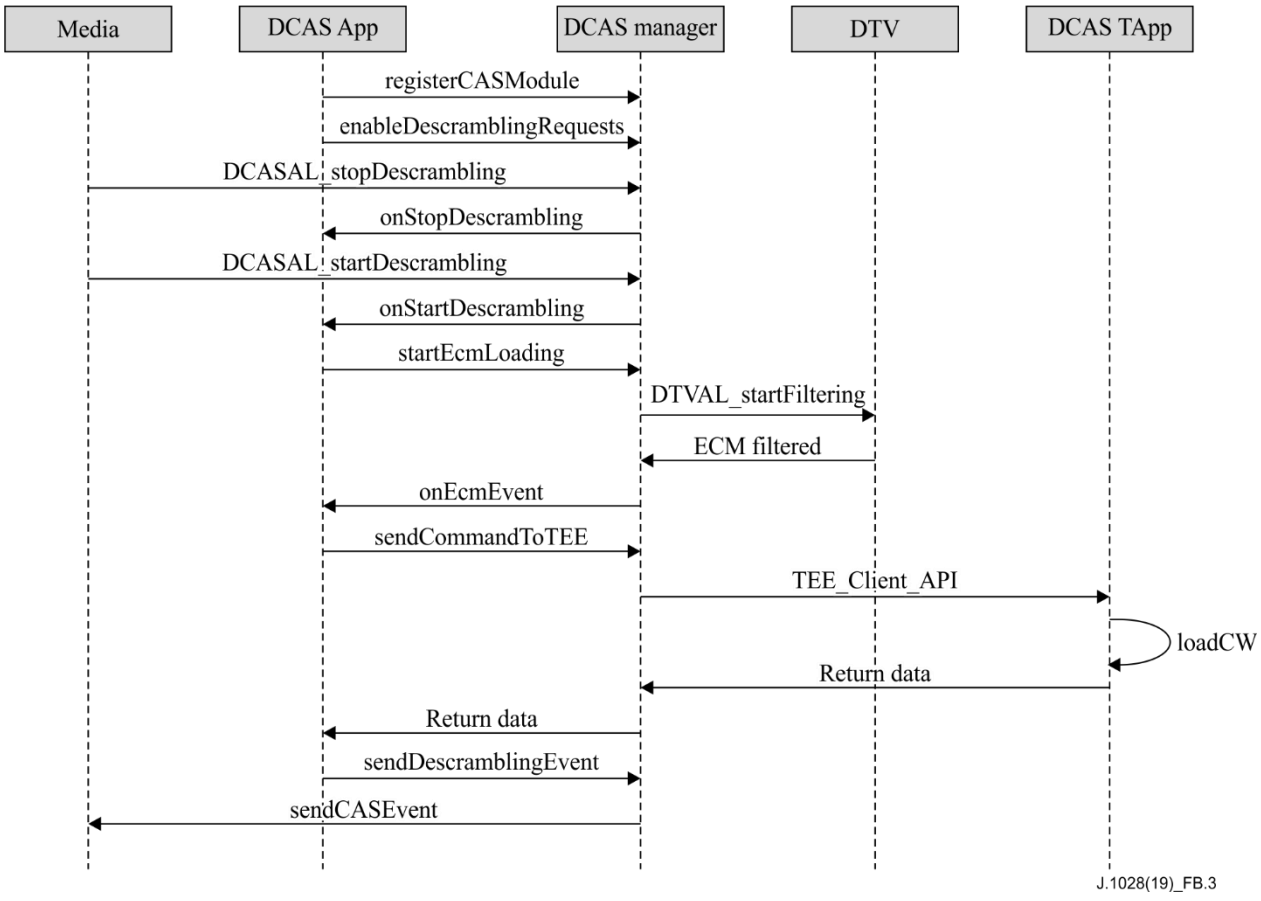

Figure B.4 – Basic calling sequence of DCAS APIs

B.2.3 Class JSDCAS.CASDescriptor

This object is used to represent CA descriptors in PMT or CAT.

B.2.3.1 getCasId

Prototype:

{number}getCasId ( )

Description:

This method is provided by terminal software platform and returns the CAS ID that appears in the CAS Descriptor.

B.2.3.2 getPid

Prototype:

{number}getPid ( )

Description:

This method returns the PID that appears in the CAS Descriptor. The PID can be either ECM PID (if the CAS descriptor appeared in the PMT), or EMM PID (if the CAS descriptor appeared in the CAT).

B.2.3.3 getPrivateData

Prototype:

{Uint8Array}getPrivateData ( )

Description:

This method returns the private data that appears in the CAS Descriptor. The private data is returned in the format of Uint8Array.

B.2.4 Class JSDCAS.CASEcmEvent

This class contains the information that can be received within an ECM event. An ECM event is passed to the JSDCAS application via the method CASModule.onEcmEvent or via the method CASModuleManager.onStartDescrambling when auto-load first ECM feature is enabled. The event can represent ECM packet arrivals, or timeout, or notifications of internal errors in the section filtering process in the device.

B.2.4.1 getEcmData

Prototype:

{Uint8Array}getEcmData ( )

Description:

Returns complete ECM data, or Null for a timeout event or internal error.

B.2.4.2 getError

Prototype:

{number}getError ( )

Description:

This method returns the error value reported by the section filtering process. Used only for debug. This method can only be invoked when the event does not provide any other information.

B.2.4.3 getTableId

Prototype:

{number} getTableId ( )

Description:

This method returns the Table ID of the ECM packet that received. This method can only be invoked when the event can provide ECM data.

B.2.4.4 isTimeout

Prototype:

{boolean} isTimeout ( )

Description:

Returns the value to indicate if there is a timeout for receiving the first ECM packet. The timeout duration can be set via CASModuleManager.enableDescramblingRequests API. This method can only be invoked when event does not provide ECM data.

Returns:

Ture – There is timeout.

False – No timeout.

B.2.5 Class JSDCAS.CASEmmEvent

This class contains the information of EMM event. An EMM event is passed to the JSDCAS application via the method CASModule.onInbandEmmEvent.

The event may represent a CAT arrival on the tuned transport stream, or for notification of inband EMM packet arrival, or for notification of internal error in the section filtering process in the device.

B.2.5.1 getEmmData

Prototype:

{Uint8Array}getEmmData ( )

Description:

This method returns full EMM data. If the event is notified by CAT update or internal error, this method returns null.

B.2.5.2 getError

Prototype:

{number}getError ( )

Description:

This method returns the error value reported by the section filtering process. Used only for debug. This method should be called only when the event does not provide any other information.

B.2.5.3 getTableId

Prototype:

{number} getTableId ( )

Description:

Return the Table ID of the EMM packet.

B.2.5.4 isCatUpdateNotification

Prototype:

{boolean}isCatUpdateNotification ( )

Description:

This method indicates whether the EMM event received is due to CAT update or not. In case of CAT update, the EMM data shall be null.

Returns:

True – when there is CAT update.

False – the event is notified by EMM arrival, or an internal error.

B.2.6 Class JSDCAS.CASFilter

The CAS Filter object represents filter criteria that the JSDCAS application defines when requesting the platform to filter OOB EMMs or inband EMMs. The platform should invoke the CAS Module only when packets match the filtering rules. Any packet that does not match the filter rules should be discarded by the platform, and the application framework should not be invoked. A CASFilter (or an array of CAS filters) can be set in the methods CASModuleManager.startCasPacketLoading and CASModuleManager.startInbandEmmLoading.

The filter criteria includes:

a) An offset (in byte) from the beginning of the packet. All the bytes before the offset are always ignored and are not part of the comparison rules.

b) A bitmap, used to compare with the coming packet.

c) A bitmap mask representing which bits locations should be included during the comparison in b). For every bit that is set to '0' in this mask, this specific bit location should be ignored during the comparison in b) above.

B.2.6.1 getBitmapMask

Prototype:

{Uint8Array}getBitmapMask ( )

Description:

Returns the bitmap mask.

B.2.6.2 getBitmapValue

Prototype:

{Uint8Array}getBitmapValue ( )

Description:

Returns the bitmap value for comparison.

B.2.6.3 getOffset

Prototype:

{number}getOffset ( )

Description:

Returns offset (in bytes).

B.2.7 Class JSDCAS.CASM

CASM is a global object that can be used to access all CAS manager and controller objects.

B.2.7.1 getCASModuleManager

Prototype:

{JSDCAS.CASModuleManager}getCASModuleManager ( )

Description:

Returns an object instance of the CAS Module Manager.

B.2.7.2 getTeeController

Prototype:

{JSDCAS.TeeController}getTeeController ( )

Description:

Returns an object instance of the TEE Controller.

B.2.8 Class JSDCAS.CASModule

An interface of a CAS Module object that the JSDCAS application should implement and register in the platform CAS Module Manager, in order to receive descrambling requests, ECMs, EMMs, and any other metadata a specific JSDCAS application requires.

B.2.8.1 getCasId

Prototype:

{number}getCasId ( )

Description:

This method returns the unique CAS ID associated with this CAS module. The JSDCAS application must implement this method to return a valid CAS ID, before calling CASModuleManager.registerCASModule. This value is expected to appear in the CA descriptors within PMT for scrambled services as well as appear in the CA descriptor within CAT.

B.2.8.2 onCasPacketEvent

Prototype:

onCasPacketEvent (casPacketEvent)

Description:

This method is called by the CAS Module Manager platform to notify the JSDCAS application when an out-of-band EMM or any other out-of-band CAS packet is received by the device. See the method CASModuleManager.startCasPacketLoading for more descriptions.

Parameters:

CASPacketEvent casPacketEvent: This parameter is a CASPacketEvent instance that contains the received OOB EMMs or other out-of-band CAS packets.

B.2.8.3 onEcmEvent

Prototype:

onEcmEvent (casSession,
ecmEvent)

Description:

This method is called by the platform CAS Manager to notify the JSDCAS application when a new ECM is filtered. In fast mode, the platform will invoke this method after setting the CW carried in ECM in K-LAD.

Parameters:

CASSession casSession-
The CAS Session object returned by CASModule.onStartDescrambling.

CASEcmEvent ecmEvent – a CASEcmEvent instance that contains ECM.

B.2.8.4 onInbandEmmEvent

Prototype:

onInbandEmmEvent (casSessionForEMM,
emmEvent)

Description:

This method is called by the platform CAS Manager to notify the JSDCAS application when a new inband EMM is filtered or when a CAT is filtered and updated on the tuned transport stream.

Parameters:

CASSession casSessionForEMM: A special CAS Session object representing the tuner and the transport stream in which the CAT is found, and including the CAS Descriptor of the CAT (instead of CAS Descriptor of the PMT).

The platform should create a dedicated CAS Session (with different session ID) for this purpose. Note that in this case, the CAS Session object may be filled only partially and may not contain all service information within.

NOTE – If the CAS descriptor with the matching CAS ID is removed from the CAT on that specific transport stream, or the STB/device is tuned out of the transport stream, there shall still be a CAT update notification but the CAS Session shall be null.

CASEmmEvent emmEvent: CASEmmEvent object that contains the EMM, or the CAT notification, or any error.

B.2.8.5 onStartDescrambling

Prototype:

onStartDescrambling (casSession,
firstEcmEvent)

Description:

This method is called by the platform CAS Manager to invoke the JSDCAS application with a new descrambling request. It is usually called when platform tunes and starts to descramble a new channel. The JSDCAS application can receive this descrambling requests only after it called CASModuleManager.enableDescramblingRequests. In the case of auto-load where the First ECM feature is enabled (in CASModuleManager.enableDescramblingRequests), the platform automatically starts filtering for the first ECM, and invokes this method with a valid CASECMEvent as the second parameter. In this case, the JSDCAS application should not call CASModuleManager.startEcmLoading explicitly. This method can be called several times simultaneously if there are several tuners in the device and each one may be used to tune to different service (or even if it is the same service). Each tuning triggers its own descrambling request with a corresponding CAS session. Another case that this method can be called several times simultaneously if the stream components of the service are not scrambled in the exact same way, whether they use different ECMs, or different private

data in their respective CAS descriptors in the PMT. Each request will have its own CAS Session.

Parameters:

CASSession casSession: CAS Session objects generated by the platform for specific descrambling request. Each object has unique session ID and all information about service and elementary streams, as well as the CA descriptor for this CAS ID in PMT.

CASEcmEvent firstEcmEvent: The first ECM packet (or timeout or error) the platform receives in auto-load mode. Set it as Null when it is not in auto-load mode.

B.2.8.6 onStopDescrambling

Prototype:

onStopDescrambling (casSession)

Description:

This method is called by the platform CAS Manager to notify the JSDCAS application to stop an ongoing descrambling session. This usually happens when the device tunes out of the scrambled channel, before it tunes to a new channel.

Parameters:

CASSession casSession-The CAS Session object returned by CASModule.onStartDescram bling.

B.2.9 Class JSDCAS.CASModuleManager

This is the platform CAS Module Manager utilized by the JSDCAS application to receive descrambling requests, ECMs, EMMs, as well as to report CAS descrambling statuses. The JSDCAS application shall implement a CAS Module object and register it as a listener in the platform CAS Module Manager.

B.2.9.1 Enums

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 Methods

B.2.9.2.1 disableDescramblingRequests

Prototype:

{number}disableDescrambingRequests (casModule)

Description:

This method is called by the JSDCAS application to stop receiving descrambling requests via the CAS Module. It is called in rare cases when JSDCAS wants temporarily not to receive requests, or wants to re-configure the work mode parameters, or before manual shutdown.

A call to CASManager.enableDescramblingRequests will renew the reception of descrambling requests.

Parameters:

CASModule casModule – Instance of CAS module.

Returns:

Success – CASModuleManager.ACTION_OK.

Failure – Returns the following error values:

CASModuleManager.ACTION_ERROR_INVALID_PARAMETERS - Invalid parameters.

CASModuleManager.ACTION_ERROR_DRIVER - Driver error.

B.2.9.2.2 enableDescramblingRequests

Prototype:

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

Description:

This method is called by the JSDCAS application to start receiving descrambling requests. Via this method, several parameters can be configured that set the mode of work between the platform CAS Manager and the specific CAS Module. This method is usually called only once (after the CAS Module has been registered), as it is assumed that a specific JSDCAS application does not change the mode of work later. For calling this method again with different configuration, the JSDCAS application should call CASModuleManager.disableDescramblingRequests firstly, to discard the current configuration.

Parameters:

CASModule casModule – Instance of CAS module.

Number firstEcmTimeout – The maximum length of time (in millisecond) the platform waits for the first ECM. If it times out, CAS module will invoke onEcmEvent or onSTartDescrambling to receive CASEcmEvent.

boolean autoLoadFirstEcm – To specify if auto-load mode is to be used. In auto-load mode, after invoking this method, the platform automatically filters the first ECM without having to wait for JS DCAS App to invoke startEcmLoading.

boolean isFastMode – Fast mode (as placeholder, no actual meaning)

Array ecmTableIds – If JS DCAS App needs to specify the tableID for ECM.

Returns:

Success – CASModuleManager.ACTION_OK.

Failure – Return the following error values:

CASModuleManager.ACTION_ERROR_INVALID_PARAMETERS - Invalid parameters.

CASModuleManager.ACTION_ERROR_ACTION_NOT_SUPPORTED – specific mode not supported.

CASModuleManager.ACTION_ERROR_DRIVER – Driver error.

B.2.9.2.3 fetchDataFromCasHeadend

Prototype:

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

Description:

By invoking this method, JS DCAS Application fetches data from headend via the platform, GPRS, or other possible methods in the future.

Parameters:

CASModule casModule-Instance of CAS module.

Uint8Array inputData – Data to be sent to headend.

String casHeURI – URI of headend server.

Returns:

Success – Data returned from headend.

Failure – Returns the following error values:

CASModuleManager.ACTION_ERROR_INVALID_PARAMETERS – Invalid parameters.

CASModuleManager.ACTION_ERROR_DRIVER – Driver error.

CASModuleManager.ACTION_ERROR_ACTION_NOT_SUPPORTED – method not supported.

CASModuleManager.ACTION_ERROR_NETWORK – network error.

B.2.9.2.4 registerCASModule

Prototype:

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

Description:

JS DCAS App registers itself to the platform CAS Module Manager by using this method.

Parameters:

number vendorId – Vendor Id of CAS. Each CAS vendor has a unique specific ID.

CASModule casModule – CAS module instance to be registered.

Number networkPriority – If more than one CASModule is registered within the CASModuleManager, this value indicates the priority of CASModule. The value is defined by operator. A higher value means higher priority (e.g., 3 is higher priority than 2).

If priority is enforced by the network operator, the platform shall send descrambling request to the CASModule with the highest priority.

If priority is not enforced by the network operator, each JSDCAS application must pass a zero value for this parameter. In this case, which CASModule receives the descrambling request is depends on the platform's implementation.

* applicationContext – An additional platform-specific application parameter. It is usually passed to the application from the platform during initialization time. The use of this parameter is project-specific.

Returns:

Success – CASModuleManager.ACTION_OK.

Failure – Returns the following error values:

CASModuleManager.ACTION_ERROR_SECURITY – The caller application is not permitted to access this function in the CAS Module Manager.

CASModuleManager.ACTION_ERROR_INVALID_PARAMETERS – Invalid parameter.

B.2.9.2.5 removeCASModule

Prototype:

{number}removeCASModule (vendorId,
casModule,
applicationContext)

Description:

This method is used for removing a registered CAS Module from the platform CAS Module Manager. It is called in rare cases when the JSDCAS application wants to change casId, or before manual shutdown.

Parameter:

number vendorId – Vendor Id of CAS. Each CAS vendor has a unique specific ID.

CASModule casModule – Instance of CAS module to be unregistered

* applicationContext – An additional platform-specific application parameter. It is usually passed to the application from the platform during initialization time. The use of this parameter is project-specific.

Returns:

Sueccess – CASModuleManager.ACTION_OK.

Failure – Returns the following error values:

CASModuleManager.ACTION_ERROR_INVALID_PARAMETERS - Invalid parameter.

CASModuleManager.ACTION_ERROR_DRIVER - Driver error.

CASModuleManager.ACTION_ERROR_SECURITY - Permission denied.

B.2.9.2.6 sendCommandToSTB

Prototype:

{number}sendCommandToSTB (casModule,
inputData)

Description:

Data channel function invoked by JS DCAS App to send data to DCAS Manager. DCAS Manager forwards commands to corresponding modules to process. The commands including OSD, upgrade trigger, fingerprint, emergency broadcast and audience survey, etc, which are sent by BOSS. DCAS as data channel is only responsible for redistributing the commands.

Parameters:

CASModule casModule – Instance of CAS module registered.

Uint8Array inputData – Data to be sent to DCAS manager.

Returns:

Success – CASModuleManager.ACTION_OK.

Failure – Returns the following error values:

CASModuleManager.ACTION_ERROR_INVALID_PARAMETERS – Invalid parameter.

CASModuleManager.ACTION_ERROR_DRIVER – Driver error.

CASModuleManager.ACTION_ERROR_ACTION_NOT_SUPPORTED – Method not supported.

CASModuleManager.ACTION_ERROR_NETWORK – Network error.

B.2.9.2.7 sendDataToHeadend

Prototype:

{number}sendDataToHeadend (casModule,
inputData)

Description:

By invoking this method, JS DCAS sends data to headend via platform, GPRS or other possible methods in the future.

Parameters:

CASModule casModule – Instance of CAS module registered.

Uint8Array inputData – Data to be sent to headend.

Returns:

success – CASModuleManager.ACTION_OK.

Failure – Returns the following error values:

CASModuleManager.ACTION_ERROR_INVALID_PARAMETERS – Invalid parameter.

CASModuleManager.ACTION_ERROR_DRIVER – Driver error.

CASModuleManager.ACTION_ERROR_ACTION_NOT_SUPPORTED – Method not supported.

CASModuleManager.ACTION_ERROR_NETWORK – Network error.

B.2.9.2.8 sendDescramblingEvent

Prototype:

{number}sendDescrambingEvent (casModule,
casSession,
casStatus)

Description:

This method is called by the JSDCAS application to report a CAS Status to the platform CAS Manager. The JSDCAS application shall send a CAS Status to the platform CAS Manager each time the descrambling status is changed within the session.

Parameters:

CASModule casModule – Instance of CAS module registered

CASSession casSession – CAS Session obtained from CASModule.onSTartDescrambling.

CASStatus casStatus – CASStatus object generated by JS DCAS App.

Returns:

Success – CASModuleManager.ACTION_OK.

Failure – Returns the following error values:

CASModuleManager.ACTION_ERROR_INVALID_PARAMETERS - Invalid parameters.

CASModuleManager.ACTION_ERROR_ACTION_NOT_SUPPORTED - Method not supported.

B.2.9.2.9 sendFreeTextOSD

Prototype:

{number}sendFreeTextOSD (casModule,
inputData,
flags)

Description:

This method is called by JSDCAS application to pass broadcasted free text to Middleware. The Middleware may pass the text to a UI application or choose to handle the OSD by itself, depends on the project requirements.

Parameters:

CASModule casModule – Instance of CAS module registered.

Uint8Array inputData – Text information

ArrayBuffer flags – Additional information indicating display method or format, etc. Project specific.

Returns:

Success – CASModuleManager.ACTION_OK.

Failure – Returns the following error values:

CASModuleManager.ACTION_ERROR_INVALID_PARAMETERS - Invalid parameters.

CASModuleManager.ACTION_ERROR_ACTION_NOT_SUPPORTED - Method not supported.

B.2.9.2.10 setCCIBits

Prototype:

{number}setCCIBits (casModule,
casSession,
cciBits)

Description:

Set data bits of CCI (Copy Control Information)

Parameters:

CASModule casModule – Instance of CAS module registered.

CASSession casSession – CAS Session obtained from CASModule.onSTartDescrambling.

Number cciBits – CCI bits.

Returns:

Success – CASModuleManager.ACTION_OK.

Failure – Returns the following error values:

CASModuleManager.ACTION_ERROR_INVALID_PARAMETERS - Invalid parameters.

CASModuleManager.ACTION_ERROR_DRIVER - Driver error.

CASModuleManager.ACTION_ERROR_ACTION_NOT_SUPPORTED - Method not supported.

B.2.9.2.11 setData

Prototype:

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

Description:

For DCAS App to set platform properties including BouquetID, activation status, CAS information, Beidou information, ChipID, HSMID, CASVendorID, region code and CA version, etc.

Parameters:

CASModule casModule - Instance of CAS module registered.

Number propertyId - Property ID, see JSDCAS.CASModuleManager.PROP_ID_xxx.

Number propertyType -
Property Type, see JSDCAS.CASModuleManager.PROP_TYPE_xxx.

Number|string|Uint8Array propertyValue -Property Value.

Returns:

Success - CASModuleManager.ACTION_OK.

Failure - Returns the following error values:

CASModuleManager.ACTION_ERROR_INVALID_PARAMETERS - Invalid parameters.

CASModuleManager.ACTION_ERROR_ACTION_NOT_SUPPORTED - Method not supported.

B.2.9.2.12 setPinCode

Prototype:

{number}setPinCode (casModule,
pinCode)

Description:

Set pin code to platform.

Parameter:

CASModule casModule - instance of CAS module

Number pinCode - PIN code to be set.

Returns:

Success - CASModuleManager.ACTION_OK.

Failure - Returns the following error value:

CASModuleManager.ACTION_ERROR_INVALID_PARAMETERS - invalid parameters.

CASModuleManager.ACTION_ERROR_DRIVER - Driver error.

CASModuleManager.ACTION_ERROR_ACTION_NOT_SUPPORTED - Method not supported.

B.2.9.2.13 setServiceListFilter

Prototype:

{number}setServiceListFilter (casModule,
filterData)

Description:

Set filtering criteria for service list. Definition of the filter criteria is platform specific.

Parameters:

CASModule casModule - Instance of CAS module.

Number filterData - Filtering criteria.

Returns:

Success - CASModuleManager.ACTION_OK.

Failure - Returns the following error values:

CASModuleManager.ACTION_ERROR_INVALID_PARAMETERS - Invalid parameters.

CASModuleManager.ACTION_ERROR_ACTION_NOT_SUPPORTED - Method not supported.

B.2.9.2.14 startCasPacketLoading

Prototype:

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

Description:

This method is called by JS DCAS application to start receiving CAS packets from out of band. CAS packets can be EMMs or any other out-of-band metadata required by the JSDCAS application. The mechanism for receiving of the packets depends on the device hardware, the platform, and the network environment.

In devices which includes a Cable Modem, the reception of the CAS packets can be done via ADSG or BDSG protocol, or by using the implementation of IP over Cable to join a multicast IP address. Other devices like IPTVs can connect to Ethernet/Wi-Fi and can also join a multicast IP address to receive CAS packets. There is also an option to tell the platform to open a UDP datagram socket locally (on localHost 127.0.0.1) and bind it to specific local port.

In all cases – when new CAS packet arrives, the JSDCAS application can handle it via CASModule.onCasPacketEvent.

NOTE – Some platform implementations may support receiving of packets from more than one source simultaneously. In these cases, this function may be called more than once with different URLs or with different CAS Tunnel IDs.

Parameters:

CASModule casModule - Instance of CAS module.

Number|string cableModemFilter - In case of Cable Modem and DSG tunnel, a filter must be provided: In case DSG is not used, this parameter should be null.

String sourceURL - When opening a local UDP datagram socket, the string value of the URL should be as the following example:

"udp://@127.0.0.1:4444" or "udp://@localhost:4444" where 4444 is the local port that the socket should be bind to.

CASFilter|Array casFilter - Filter criteria, can be a CASFilter array.

Returns:

Success - CASModuleManager.ACTION_OK.

Failure - Returns the following error values:

CASModuleManager.ACTION_ERROR_INVALID_PARAMETERS - Invalid parameters.

CASModuleManager.ACTION_ERROR_DRIVER - Driver error.

CASModuleManager.ACTION_ERROR_ACTION_NOT_SUPPORTED - Method not supported.

B.2.9.2.15 startEcmLoading

Prototype:

{number}startEcmLoading (casModule,
casSession)

Description:

This method is called by JSDCAS application to start receiving ECMs for a specific scrambled service. It is called after receiving descrambling request via the CAS Module with a valid CASSession object that is generated by the platform CAS Manager.

NOTE – In the case that auto-load first ECM feature is enabled, calling this method is not required. For more information about setting the auto-load first ECM feature, see the method CASModuleManager.enableDescramblingRequests.

Parameters:

CASModule casModule - Instance of CAS module.

CASSession casSession - CAS Session obtained from CASModule.onSTartDescrambling.

Returns:

Success - CASModuleManager.ACTION_OK.

Failure - Returns the following error values:

CASModuleManager.ACTION_ERROR_INVALID_PARAMETERS - Invalid parameters.

CASModuleManager.ACTION_ERROR_DRIVER - Driver error.

B.2.9.2.16 startInbandEmmLoading

Prototype:

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

Description:

This method can be called by JSDCAS application either to start receiving inband EMM, or to receive CAT as necessary. JSDCAS application receives data via CASModule.onInbandEmmEvent when there is EMM or CAT update.

Parameters:

CASModule casModule - Instance of CAS module.

Array emmTableIds - EMM table ID array.

CASFilter|Array casFilter - The platform will notify application only for data that matches the criteria. It is also possible to pass a Filter array.

boolean includeCatNotifications - Specifies whether the JSDCAS application also wishes to receive CAT update notification.

Returns:

Success - CASModuleManager.ACTION_OK.

Failure - Return the following error values:

CASModuleManager.ACTION_ERROR_INVALID_PARAMETERS -Invalid parameters.

CASModuleManager.ACTION_ERROR_DRIVER -Driver error.

CASModuleManager.ACTION_ERROR_ACTION_NOT_SUPPORTED -Method not supported.

B.2.9.2.17 stopCasPacketLoading

Prototype:

{number}stopCasPacketLoading (casModule,
cableModemFilter,
sourceURL)

Description:

JS DCAS App invokes this method to stop receiving out-of-band CAS data packet.

Parameters:

CASModule casModule - Instance of CAS module.

Number|string cableModemFilter - Required by Cable Modem.

String sourceURL - Required when receiving by UDP.

Returns:

Success - CASModuleManager.ACTION_OK.

Failure - Return the following error values:

CASModuleManager.ACTION_ERROR_INVALID_PARAMETERS -Invalid parameters.

CASModuleManager.ACTION_ERROR_DRIVER -Driver error.

B.2.9.2.18 stopEcmLoading

Prototype:

{number}stopEcmLoading (casModule,

casSession)

Description:

This method is called by JSDCAS application to temporarily stop receiving ECMs. JS DCAS App rarely invokes this method. CASManager.startEcmLoading needs to be re-invoked to resume ECM receiving.

Parameters:

CASModule casModule - Instance of CAS module.

CASSession casSession - CAS Session obtained from CASModule.onSTartDescrambling.

Returns:

Success - CASModuleManager.ACTION_OK.

Failure - Returns the following values:

CASModuleManager.ACTION_ERROR_INVALID_PARAMETERS - Invalid parameters.

CASModuleManager.ACTION_ERROR_DRIVER - Driver error.

B.2.9.2.19 stopInbandEmmLoading

Prototype:

{number}stopInbandEmmLoading (casModule)

Description:

This method is called by JS DCAS App to stop receiving in band EMM. JS DCAS App rarely invokes this method. CASManager.startInbandEmmLoading needs to be re-invoked to resume EMM receiving.

Parameters:

CASModule casModule - Instance of CAS module.

Returns:

Success - CASModuleManager.ACTION_OK.

Failure - Returns the following error values:

CASModuleManager.ACTION_ERROR_INVALID_PARAMETERS - Invalid parameters.

CASModuleManager.ACTION_ERROR_DRIVER - Driver error.

B.2.10 Class JSDCAS.CASPacketEvent

B.2.10.1 getCableModemFilter

Prototype:

{number|string}getCableModemFilter ( )

Description:

Returns the cableModemFilter used for filtering packet. Returns Null if Cable Modem DSG is not used.

Returns:

ADSG mode - returns CAS Tuner ID, numeric.

BDSG mode - Returns virtual MAC address.

B.2.10.2 getPacketData

Prototype:

{uint8Array}getPacketData ( )

Description:

Returns packet data.

B.2.10.3 getPacketHeader

Prototype:

{uint8Array}getPacketHeader ( )

Description:

Returns data packet header, which includes IP address and UDP header.

B.2.10.4 getSourceURL

Prototype:

{string}getSourceURL ( )

Description:

Returns the source URL for receiving CAS data packet by UDP.

Returns:

Source URL string.

B.2.11 Class JSDCAS.CASSession

For every descrambling request, the platform generates a CAS Session object, which contains a unique session ID and all information about the playing of the program, as well as the CA descriptor in the PMT related to the descrambling. For each descrambling request, JS DCAS App uses CASModule.onStartDescrambling to obtain CAS Session. The CASSession object can also be used in receiving CAT update message, in which case only some fields of the CASSession object is valid.

B.2.11.1 GetCasDescriptor

Prototype:

{CASDescriptor}getCasDescriptor ( )

Description:

Returns CA descriptor, which can either be from PMT or CAT.

Returns:

Instance of CA descriptor.

B.2.11.2 getChannelNumber

Prototype:

{number}getChannelNumber ( )

Description:

Returns channel number. This method is optional, especially for platforms without defined channel numbers (can return 0)

Returns:

Channel number

B.2.11.3 getNetworkId

Prototype:

{number}getNetworkId ( )

Description:

Returns original network ID. This method is optional. The platform can return 0 if unable to obtain it.

Returns:

original network ID

B.2.11.4 getOperationType

Prototype:

{number}GetOperationType ( )

Description:

Returns operation type.

Returns:

Value for operation type

CASSession.OPERATION_TYPE_PRESENTATION

CASSession.OPERATION_TYPE_RECORDING

CASSession.OPERATION_TYPE_BUFFERING

CASSession.OPERATION_TYPE_SECOND_DEVICE.

B.2.11.5 getProgramNumber

prototype:

{number}getProgramNumber ( )

Description:

Returns program number

B.2.11.6 getServiceIdentifier

Prototype:

{number|*}getServiceIdentifier ( )

Description:

Returns the identifier of the service being descrambled. This identifier can be a value or an object.

B.2.11.7 getSessionId

Prototype:

{number}getSessionId ( )

Description:

Returns Session ID

B.2.11.8 getStreamPath

Prototype:

{Uint8Array} getStreamPath ( )

Description:

Returns the StreamPath data

B.2.11.9 getStreamPIDs

Prototype:

{Array} getStreamPIDs ( )

Description:

Returns the Stream PIDs list.

B.2.11.10 getStreamTypes

Prototype:

{Array} getStreamTypes ( )

Description:

Returns the StreamTypes list, see [ISO 13818] (Table 2-36).

B.2.11.11 getTransmitterScrambingMode

Prototype:

{number}getTransmitterScrambingMode ( )

Description:

Returns value for descrambling mode.

B.2.11.12 getTransportStreamId

Prototype:

{number}getTransportStreamId ( )

Description:

Returns TS ID of the service being descrambled.

B.2.11.13 getTunerId

Prototype:

{number}getTunerId ( )

Description:

Returns the Tuner ID of the service being descrambled.

B.2.12 Class JSDCAS.CASStatus

JS DCAS App uses this object to report descrambling status to the platform. Every time the descrambling status changes, JS DCAS App should invoke CASModuleManager.sendDescrambling Event to notify the platform. After receiving the status change, the platform can either handle by itself or forward it to UI application. UI application can notify user of descrambling success or failure simply by pop-up OSD, or display details about why descrambling fails, by analyzing additional information in the CASStatsus object. Format of additional information is project specific. If there is no additional information added by JS DCAS App, the UI application can also use the token obtained from the CASStatus object to obtain more information by communicating with JS DCAS App using IPC or other methods provided by the platform.

B.2.12.1 Status Value List

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 Methods

B.2.12.2.1 getCasToken

Prototype:

{number}getCasToken ( )

Description:

Returns CAS token. If the platform forwards information of CASStatus to UI application, UI application can use this token to request JS DCAS application for detailed status information. The request method such as IPC, is decided by the platform.

B.2.12.2.2 getMajorContentProblem

Prototype:

{number}getMajorContentProblem ( )

Description:

Returns a value representing the reason for not being able to view program.

B.2.12.2.3 getStatusData

Prototype:

{ArrayBuffer}getStatusData ( )

Description:

This method returns extended status data that the JSDCAS application attached to the CAS status object. With this additional status data, the UI application can show more detailed information regarding descrambling status to user.

Returns:

Returned data can be in the ArrayBuffer type. Shall return null if no extended data to provide.

B.2.12.2.4 isSuccess

Prototype:

{boolean}isSuccess ( )

Description:

Returns descrambling status, which can be either success or failure.

Returns:

True - Success. False - Failure.

B.2.13 Class JSDCAS.TeeController

Controller of communication between JS DCAS and TEE.

B.2.13.1 Methods

B.2.13.1.1 sendCommandToTEE

Prototype:

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

Description:

JS DCAS App uses this method to send command to TA running in TEE.

Parameters:

Uint8Array teeAppUUID - The UUID of TA, 16 bytes. Every CA vendor has different ID.

number commandId - Command ID in TEE communication, defined by each CA vendor itself.

Uint8Array inputData - Data sent to TA

* applicationContext - Application context which is platform specific. Usually provided to application by the platform during initiation.

Returns:

Returns the TeeRetVal object. This object contains information returned from TA, such as data or error, etc.

B.2.14 Class JSDCAS.TeeRetVal

This object is returned by TeeController.sendCommandToTEE and contains information returned from TEE, such as data and error, etc.

B.2.14.1 Returned Value List

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 Method

B.2.14.2.1 getOriginCode

prototype:

{number}getOriginCode ( )

Description:

Returns origin code.

B.2.14.2.2 getResponseData

prototype:

{Uint8Array}getResponseData ( )

Description:

To get data returned from TA

Returns:

Data returned from TA, which can be Null for some commands. Returns Null if error occurs in invocation or communication.

B.2.14.2.3 getReturnCode

Prototype:

{number}getReturnCode ( )

Description:

Returns to retrieve return code.

B.3 HSM driver APIs

B.3.1 Data types and structures

B.3.1.1 Basic data types

HSM return value type:

typedef unsigned int HSM_Result;

Basic data types:

typedef unsigned int uint32_t;

typedef unsigned short uint16_t;

typedef unsigned char unit8_t;

B.3.1.2 Enums returned

/* HSM Access Success*/

#define HSM_RESULT_OK 0

/*Application not permitted to access the HSM */

#define HSM_RESULT_ERROR_SECURITY 1

/* Invalid parameters passed */

#define HSM_RESULT_ERROR_INVALID_PARAMETERS 2

/* Operation not supported */

#define HSM_RESULT_ERROR_NOT_SUPPORTED 3

/* Length or offset out of range */

#define HSM_RESULT_ERROR_OUT_OF_RANGE 4

/* Error in the driver that controls the HSM */

#define HSM_RESULT_ERROR_DRIVER 5

/* specific READ or WRITE ERROR when trying to process the request */

#define HSM_RESULT_ERROR_IO 6

/* Device or driver is busy, application should try again later*/

#define HSM_RESULT_ERROR_BUSY 7

/* HSM communication error */

#define HSM_RESULT_ERROR_API_COMMUNICATION 8

/* Insufficient buffer from application, correct length returned */

#define HSM_RESULT_ERROR_INSUFFICIENT_BUFFER 9

/* General error when trying to process the request */

#define HSM_RESULT_ERROR_OPERATION_FAILED 10

B.3.2 APIs definitions

B.3.2.1 TEE_HSM_GetSoftwareVersion

To read out software version from HSM.

The recommended version string is as: starting with a fixed prefix of at least 8 characters, then followed by a version varaible string, for example, "DCAS HSM Version: 34.9.2b".

Prototype:

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

Inputs:

versionLength: size of the version buffer

Outputs:

versionLength: the real length of the output version string after calling this API.

Version: version string, defined by each HSM vendor.

Return values:

HSM_RESULT_OK: operation success

Other values: operation failure. Refer to Enums defined above.

B.3.2.2 TEE_HSM_GetHsmGeneralInfo

To read out general information from HSM.

Prototype:

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

Inputs:

hsmIdLength: Buffer size for HSMID

Outputs:

hsmStatus: activation status of HSM, 0: inactivated, 1: activated, 2: interim

hsmIdLength: the actual size of the HSMID retrieved

hsmId: HSMID retrieved by this operation

Returns:

HSM_RESULT_OK:Operation succeeds

Other values: Operation fails. Refer to the definitions of Enums above for detailed reason for failure.

B.3.2.3 TEE_HSM_GetHsmDiagnosticInfo

To read out the diagnostic information from HSM

Prototype:

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

Inputs:

activeChipIdLength: the buffer size for storing the paired SoC ID with which the HSM is activated

hsmDeviceCertificateLength: the buffer size for storing HSM device certificate

hsmVendorCertificateLength: the buffer size for storing CAS vendor certificate

Outputs:

activeChipIdLength: the actual size of the paired SoC ID

activeChipId: the SoC ID, when there is no SoC ID stored, the output should be all 0s.

activeCasVendorId: the actual Vendor_SysID with which HSM is activated. Returns all 0s if there is no CAS vendor ID in chipset.

hsmDeviceCertificateLength: the actual size of the HSM device certificate retrieved

hsmDeviceCertificate: HSM device certificate data retrieved

hsmVendorCertificateLength: the actual size of the HSM vendor certificate retrieved

hsmVendorCertificate: HSM vendor certificate data retrieved

Returns:

HSM_RESULT_OK: operation succeeds

Other values: Operation fails. Refer to the definitions of Enums above for detailed reason for failure.

B.3.2.4 TEE_HSM_GetHsmCapabilities

To get the capability information of HSM, such as storage volumn and max read/write data length

Prototype:

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

Inputs:

N/A.

Outputs:

secureStorageSize: SAC secure storage size of the HSM

publicStorageSize: public storage size of the HSM

maxWriteSecureLength: the max data size of each writing operation into SAC secure storage

maxReadSecureLength: the max data size of each reading operation from SAC secure storage

maxReadPublicLength: the max data size of each reading operation from public storage

Returns:

HSM_RESULT_OK: Operation succeeds

Other values: Operation fails. Refer to the definitions of Enums above for detailed reason for failure.

B.3.2.5 TEE_HSM_GetHsmLastTimeStamp

To get the last timestamp of the very recent activation

Prototype:

HSM_Result TEE_HSM_GetHsmLastTimeStamp (uint32_t* timestamp);

Inputs:

N/A

Outputs:

Timestamp: the last timestamp of the very recent acceptable activation message, return all zero when the HSM has not been activated before.

Returns:

HSM_RESULT_OK: operation succeeds

Other values: operation failure. Refer to the definitions of Enums above for detailed reason for failure.

B.3.2.6 TEE_HSM_GetHsmActivationInfo

To read CAS proprietary data that stored in HSM when activated.

Prototype:

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

Inputs:

vendorId: Vendor_SysID

casPropDataLength: the buffer size for storing CAS proprietary data

Outputs:

casPropDataLength: the actual size of the CAS proprietary data retrieved

HSM_RESULT_OK:

HSM_RESULT_ERROR_INVALID_PARAMETERS: Error in input parameters, or Vendor_SysId does not match the existing Vendor SysId of HSM

Other values: Operation fails. Refer to the definitions of Enums above for detailed reason for failure.

B.3.2.7 TEE_HSM_GenerateActivationRequest

To generate activation request message

Prototype:

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);

Inputs:

VendorId: Vendor_SysID

vendorCertificateLength: CA vendor certificate size

vendorCertificate: CA vendor certificate

chipIdLength: SoC ID size

chipId: SoC ID

longitude: the longitude of terminal device, of which the value is the real longitude multiplied by 10^6

latitude: the latitude of the terminal device, of which the value is the real latitude value multiplied by 10^6

timestamp: System time (e.g., Beidou time), stands for the time elapsing since 00:00:00 of 1970-1-1 ActivationRequestLength the buffer size for storing activation request message

Outputs:

activationRequestLength: the actual size of the activation request message generated

activationRequest: the activation request message generated

Returns:

HSM_RESULT_OK: Operation succeeds

Other values: Operation fails. Refer to the definitions of Enums above for detailed reason for failure.

B.3.2.8 TEE_HSM_SetMessage

To process the received primary activation message or auxiliary data message, and store the results inside HSM.

Prototype:

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

Inputs:

vendorId: Vendor_SysID

vendorCertificateLength: CAS vendor certificate length

vendorCertificate: CAS vendor certificate

messageLength: Length of Activation message

message: body of activation message

Outputs:

N/A

Returns:

HSM_RESULT_OK: Operation succeeds

Other values: operation fails. Refer to the definitions of Enums above for detailed reason for failure.

B.3.2.9 TEE_HSM_OpenSac

To establish SAC with HSM.

Prototype:

```
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 );
```

Inputs:

vendorId: Vendor_SysID

vendorCertificateLength: CAS vendor certificate length

vendorCertificate: CAS vendor certificate

chipIdLength: SoC ID length

chipId: SoC ID

PairKLength: Pairing key length

PairK: Pairing Key

randomNonceLength: Nonce length

randomNonce: Nonce

Outputs:

hsmHandleLength: The size of the handle for accessing HSM, DCAS trusted App should allocate a 16-byte buffer for storing the handle

hsmSacHandle: the handle for accessing HSM, DCAS trusted App accesses HSM with the handle.

Returns:

HSM_RESULT_OK: Operation succeeds

Other values: Operation fails. Refer to the definitions of Enums above for detailed reason for failure.

B.3.2.10 TEE_HSM_Read

To read data from HSM

Prototype:

```
HSM_Result TEE_HSM_Read ( unint16_t vendorId,
                          int hsmSacHandleLength,
                          uint8_t* hsmSacHandle,
                          uint32_t offset,
                          uint32_t* length,
                          unint8_t* data );
```

Inputs:

vendorId: Vendor_SysID

hsmHandleLength: the size of the handle for accessing HSM

hsmSacHandle: the handle for accessing HSM

offset:the position from where the HSM data is read

length: data length to be read out

Outputs:

Length: actual data length read out

Data: data read out

Returns:

HSM_RESULT_OK: Operation succeeds

Other values: Operation fails. Refer to the definitions of Enums above for detailed reason for failure.

B.3.2.11 TEE_HSM_Write

To write data into HSM.

Prototype:

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

Inputs:

vendorId: Vendor_SysID

hsmHandleLength: the size of the handle for accessing HSM

hsmSacHandle: the handle for accessing HSM

offset: the position from where the HSM data is to be written

length: the length of data to be written

data: data to be written

Outputs:

length: actual length of the data written

Returns:

HSM_RESULT_OK: Operation succeeds

Other values: Operation fails. Refer to the definitions of Enums above for detailed reason for failure.

B.3.2.12 TEE_HSM_ReadPositionParameters

To get the set position parameters from HSM

Prototype:

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

Inputs:

vendorId: Vendor_SysID

hsmHandleLength: the size of the handle for accessing HSM

hsmSacHandle: the handle for accessing HSM

Outputs:

longitude: longitude set in the terminal device, of which the value is the real longitude multiplied by 10^6

latitude the latitude set in the terminal device, of which the value is the real latitude multiplied by 10^6

radius: the radius set in the terminal device, its unit is ten meters

Returns:

HSM_RESULT_OK: Operation succeeds

Other values: Operation fails. Refer to the definitions of Enums above for detailed reason for failure.

B.3.2.13 TEE_HSM_ReadPublicSecureStorage

To read data from public secure storage of HSM

Prototype:

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

Inputs:

offset: the position from where the HSM data is to be read out

length: data length to be read

Outputs:

length: actual data length read

data: data read out

Returns:

HSM_RESULT_OK: Operation succeeds

Other values: Operation fails. Refer to the definitions of Enums above for detailed reason for failure.

B.3.2.14 TEE_HSM_WritePublicSecureStorage

To write data into public secure storage of HSM

Prototype:

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

Inputs:

vendorId: Vendor_SysID

hsmHandleLength: the size of the handle for accessing HSM

hsmSacHandle: the handle for accessing HSM

offset: the position from which the HSM data is to be written

length: the length of data to be written

data: data to be written

Outputs:

Length: the actual length of the data written

Returns:

HSM_RESULT_OK: Operation succeeds

Other values: Operation fails. Refer to the definitions of Enums above for detailed reason for failure.

B.3.2.15 TEE_HSM_ChangeCwEncryptionScheme

To set the re-encryption algorithm in HSM, which should be consistent with the algorithm used in SoC key ladder.

This API should be invoked only when the default algorithm of HSM is not supported by SoC key ladder.

If this API is not invoked, then HSM use the algorithm set by TEE_HSM_GenerateCW API to re-encrypt the CW.

Prototype:

```
HSM_Result TEE_HSM_ChangeCwEncryptionScheme (  uint16_t vendorId,
```

int hsmSacHandleLength,
uint8_t* hsmSacHandle,
uint16_t socSchemeId);

Inputs:

vendorId: Vendor_SysID

hsmHandleLength: the size of the handle for accessing HSM

hsmSacHandle: the handle for accessing HSM

socSchemeId: the re-encryption algorithm ID set in HSM, 0: reserved, 1: reserved, 2: SM4

Outputs:

N/A

Returns:

HSM_RESULT_OK: Operation succeeds

Other values: Operation fails. Refer to the definitions of Enums above for detailed reason for failure.

B.3.2.16 TEE_HSM_GenerateCW

To generate re-encrypted CW by HSM, which will be used by SoC key ladder

Prototype:

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);

Inputs:

vendorId:Vendor_SysID

hsmHandleLength: the size of the handle for accessing HSM

hsmSacHandle: the handle for accessing HSM

schemeId: the algorithm used in HSM Key ladder, 0: reserved, 1: reserved, 2: SM4

keyL2Length: the length of the 2nd level key in HSM key ladder

keyL2: 2nd level key

keyL1Length: the length of the 1st level key in HSM key ladder

keyL1: 1st level key

keyL0Length: the length of the 0-level key in HSM key ladder

keyL0: 0-level key

CWLength: the buffer size for storing CW

Outputs:

CWLength: actual size of the CW

CW: re-encrypted CW generated

Returns:

HSM_RESULT_OK: Operation succeeds

Other values: Operation fails. Refer to the definitions of Enums above for detailed reason for failure.

B.3.2.17 TEE_HSM_CloseSac

To close the SAC between SoC and HSM

Prototype:

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

Inputs:

vendorId: Vendor_SysID

hsmHandleLength: the size of the handle for accessing HSM

hsmSacHandle: the handle for accessing HSM

Outputs:

N/A

Returns:

HSM_RESULT_OK: Operation succeeds

Other values: Operation fails. Refer to the definitions of Enums above for detailed reason for failure.

B.4 Positioning module APIs (Beidou)

B.4.1 Data types and structures

B.4.1.1 Basic data types

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

B.4.1.2 Enums teturned

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

B.4.2 APIs definitions

B.4.2.1 TEE_Beidou_GetSoftwareVersion

To get positioning module software version

Prototype:

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

Inputs:

version: buffer allocated by DCAS trusted APP for storing version information

length: buffer size

Outputs:

Version: The version information string returned, this buffer should be released by DCAS trusted APP

Returns:

TEE_SUCCESS: Get version information successfully

TEE_BEIDOU_NOT_READY: Positioning module is not ready yet

Other values: Hardware failure

B.4.2.2 TEE_Beidou_GetPositionParameters

To get positioning information and time

Prototype:

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

Inputs:

Longitude: buffer for storing longitude parameter, should be allocated and released by the caller

latitude: buffer for storing latitude parameter, should be allocated and released by the caller

timestamp: buffer for storing time parameter, should be allocated and released by the caller

Outputs:

Longitude: longitude

Latitude: latitude

Timestamp: timestamp

Returns:

TEE_SUCCESS: Get position and time successfully

TEE_BEIDOU_NOT_READY: Positioning module is not ready yet

Other values: Hardware failure

B.4.2.3 TEE_Beidou_GetSignalParameters

To get signal parameters of the positioning satellite

Prototype:

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

Inputs:

Numfix: buffer for storing satellite numbers, should be allocated and released by the caller

cn0bds: buffer for storing carrier-to-noise ratio of Beidou, should be allocated and released by the caller

cn0gps: buffer for storing carrier-to-noise ratio of GPS, should be allocated and released by the caller

Outputs:

Numfix: satellite numbers

Cn0bds: carrier-to-noise ratio of Beidou

Cn0gps: carrier-to-noise ratio of GPS

Returns:

TEE_SUCCESS: Get signal parameters successfully

TEE_BEIDOU_NOT_READY: Positioning module is not ready yet

Other values: Hardware failure

B.4.2.4 TEE_Beidou_ CalculateDistance

To compute the distance between 2 points on the earth

Prototype:

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

Inputs:

longitudeA: longitude of point A, its value is the actual longitude multiplied by 10^6

latitudeA: latitude of point A, its value is the actual latitude multiplied by 10^6

longitudeB: longitude of point B, its value is the actual longitude multiplied by 10^6

latitudeB: latitude of point B, its value is the actual latitude multiplied by 10^6

Outputs:

N/A

Returns:

The distance between the 2 points, in meter.

B.5 Other GP extension APIs

B.5.1 Cryptography and signature verification APIs

B.5.1.1 Data types and structures

B.5.1.1.1 Basic data types

typedef unsigned int TEE_Result;

typedef unsigned int uint32_t;

B.5.1.1.2 Enums returned

#define TEE_SUCCESS 0;

B.5.1.2 APIs definitions

B.5.1.2.1 TEE_SM2_Verify

To verify SM2 signature.

Prototype:

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)

Inputs:

pub_key: SM2 Public key consisting of 1 byte header and 64-byte public key data, totally 65 bytes

pub_key_len: SM2 Public key size

hash: SM3 hash value of the content, 32 bytes

hash_len: Hash value size

sig:Signature, 64 bytes

sig_len: Length of signature

Outputs:

N/A

Returns:

TEE_SUCCESS: Signature verification success

Other values: signature verification failure

B.5.1.2.2 TEE_Perform_SM3

To compute SM3 hash.

Prototype:

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

Inputs:

dataIn: content data over which the SM3 hash is to be computed

dataInLen: content size

Outputs:

result: hash value retrieved, 32 bytes

Returns:

TEE_SUCCESS: computation successful

Other values: computation failure.

B.5.1.2.3 TEE_SM2_Encrypt

To encrypt data with SM2 algorithm and public key.

Prototype:

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)

Inputs:

pub_key: SM2 Public key, consisting of a 1-byte header and 64-byte public key data, totally 65 bytes

pub_key_len: Length of SM2 public key

inputData: input data to be encrypted

inputData_size: size of the input data

outputData:data buffer to store the encrypted data

outputData_size: data buffer to store the size of the encrypted data, it should be the input data size plus 96 bytes

Outputs:

outputData: encrypted data stored in the outputData buffer, it is a concatenation of C1|C3|C2, where C1 is the EC point randomly generated, C2 is the encrypted message, C3 is the hash value related to C1 and C2.

Returns:

TEE_SUCCESS: Encryption success

Other values: Encryption failure.

B.5.1.2.4 TEE_Perform_CRC

To compute the CRC value.

Prototype:

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

Inputs:

mode: CRC computing mode, 0=CRC16, 1=CRC

dataIn: the data over which the CRC will be computed

dataInLen: Length of input data

Outputs:

result: the CRC result computed, it is of 2 bytes for CRC16, 4 bytes for CRC32

Returns:

TEE_SUCCESS: CRC computation success

Other values: computation failure

B.5.1.2.5 TEE_GenerateRandom

To generate a random number.

Prototype:

TEE_Result TEE_GenerateRandom (void* randomBuffer,
size_t randomBufferLen);

Inputs:

randomBuffer: data buffer for storing the random number

randomBufferLen: data buffer length

Outputs:

randomBuffer: the data buffer with a random number stored

Returns:

TEE_SUCCESS: Generation success

Others: Generation failure

B.5.1.2.6 TEE_SM4_Encrypt

To encrypt data with SM4 algorithm.

Prototype:

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

Inputs:

mode: encryption mode, 0=ECB, 1=CBC

IV: Initialization vector. It is a 16-byte data in CBC mode, and should be ignored in ECB mode

key: SM4 key, 16 bytes

inputData: content to be encrypted

outputData: data buffer for storing the encrypted output

data_size: data size for both input and output data, it should be any multiple of 16 byptes, otherwise the encryption will fail

Outputs:

outputData: Data encrypted

Returns:

TEE_SUCCESS: encryption success

Others: failure

B.5.1.2.7 TEE_SM4_Decrypt

To decrypt data with SM4 algorithm

Prototype:

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

Inputs:

mode: decryption mode, 0=ECB, 1=CBC

IV: Initialization vector. It is a 16-byte data in CBC mode, and should be ignored in ECB mode.

key: SM4 key, 16 bytes

inputData: content to be decrypted

outputData: data buffer to store the decrypted output

data_size: data size for both input and output data. It should be multiple of 16 bytes, otherwise the decryption will fail.

Outputs:

outputData: data decrypted

Returns:

TEE_SUCCESS: Decryption success

Others: failure.

B.5.2 Memory Management APIs

B.5.2.1 Data Types and Structures

B.5.2.1.1 Basic Data Types

N/A

B.5.2.1.2 Enums returned

N/A

B.5.2.2 APIs definitions

B.5.2.2.1 TEE_MemFill

Fill a memory space with a specified value.

Prototype:

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

Inputs:

buffer: the starting address of the memory space to be filled

x: specified value for the filling

size: size of the memory space to be filled

Outputs:

N/A

Returns:

N/A

B.5.2.2.2 TEE_MemMove

Move data from one place to another in memroy.

Protoytpe:

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

Inputs:

dest: the starting address of the destination memory space

src: the starting address of the source memory space

size: size of the data to be moved

Outputs:

N/A

Returns:

N/A

B.5.3 Miscellaneous APIs

B.5.3.1 Data Types and Structures

B.5.3.1.1 Basic data types

N/A

B.5.3.1.2 Enums returned

N/A

B.5.3.2 APIs definitions

B.5.3.2.1 TEE_Printf_Func

Print logs.

Prototype:

void TEE_Printf_Func(const char * fmt, …);

Inputs:

fmt: format list for the printing

Outputs:

N/A

Returns:

N/A

B.6 Security Chipset Key Ladder Driver APIs

B.6.1 Data types and structures

B.6.1.1 Basic data types

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.6.1.2 Enums returned

typedef enum

{

TEE_KLAD_OK,

TEE_KLAD_FAIL,

TEE_KLAD_UNMATCH_CHAN,

}TEE_KLAD_STATUS

B.6.2 APIs definitions

B.6.2.1 TEE_KLAD_Init

Initialize Key Ladder.

Prototype:

TEE_KLAD_STATUS TEE_KLAD_Init(void);

Inputs:

N/A;

Outputs:

N/A.

B.6.2.2 TEE_KLAD_Delnit

Deinitialize Key Ladder.

Prototype:

TEE_KLAD_STATUS TEE_KLAD_Delnit(void);

Inputs:

N/A;

Outputs:

N/A.

B.6.2.3 TEE_KLAD_GetChipId

Read security chipset's ChipId.

Prototype:

TEE_KLAD_STATUS TEE_KLAD_GetChipId(TEE_KLAD_BYTE* chipid);

Inputs:

N/A;

Outputs:

Security chipset's ChipId, 8 bytes buffer, allocated and freed by the application which calls this interface.

B.6.2.4 TEE_KLAD_GetResponseToChallenge

Compute the response according to the challenge.

Prototype:

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
);
```

Inputs:

Nonce: challenge data

NonceLength: length of challenge data

keyDescriptorsLength: length of key descriptors

keyDescriptors: key descriptors

Outputs:

response: the computing result of challenge

responseLength: the length of the result

The descriptors in key descriptors is as following:

Key ladder descriptor's definition in byte:

0: ENCRYPTION_KEY_DSCR_TAG = 0x03

1: descriptor's length

2: level of key ladder, for challenge-response computing, set 2

3: length of the key used by key ladder

4-n: encrypted keys for key ladder

Algorithm descriptor's definition in byte:

0: ENCRYPTION_SCHEME_DSCR_TAG = 0x04

1: descriptor's length

2-3: enums value of algorithm: 0=3DES, 1=AES, 2=SM4

CA vendor ID's descriptor's definition in byte:

0: VENDOR_ID_DSCR_TAG = 0x05

1: descriptor's length = 2

2-3: CA vendor ID

B.6.2.5 TEE_KLAD_SetDescrambler

Set descrambling parameters and keys to invoke descrambling.

Prototype:

```
TEE_KLAD_STATUS TEE_KLAD_SetDescrambler
(
    int                 streamPathLength,
    TEE_KLAD_BYTE       *streamPath;
    int                 numberOfStreamPids,
    TEE_KLAD_BYTE       *streamPids,
    Int                 OddkeyDescriptorsLength,
    TEE_KLAD_BYTE       *OddkeyDescriptor,
    Int                 EvenkeyDescriptorLength,
```

TEE_KLAD_BYTE *EvenkeyDescriptor

);

Inputs:

streamPathLength: length of stream path for descrambling

streamPath: stream path for descrambling

numberOfStreamPids: pid numbers of descrambling stream program

streamPids: pids of descrambling stream program

OddkeyDescriptorsLength: length of odd key descriptor

OddkeyDescriptor: odd key descriptor

EvenkeyDescriptorLength: length of even key descriptor

EvenkeyDescriptor: even key descriptor

The descriptor in odd key descriptor and even key descriptor is as following:

Clear CW descriptor's definition in byte:

0: CLEAR_CW_DSCR_TAG = 0x01

1: descriptor's length

2-n: clear CW;

Encrypted CW descriptor's definition in byte:

0: ENCRYPTED_CW_DSCR_TAG = 0x02

1: descriptor's length

2-n: encrypted CW. To decrypt to get CW, additional descriptors will be provided.

Key ladder descriptor's definition in byte:

0: ENCRYPTED_KEY_DSCR_TAG = 0x03

1: descriptor's length

2: key level, 0 for CW, 1, 2… for other keys

3: key length

4-n: encrypted key

Key encryption algorithm descriptor's definition in byte:

0: ENCRYPTION_SCHEME_DSCR_TAG = 0x04

1: descriptor's length

2-3: enums value of algorithm: 0=3DES, 1=AES, 2=SM4

CA vendor ID's descriptor's definition in byte:

0: VENDOR_ID_DSCR_TAG = 0x05

1: descriptor's length = 2

2-3: CA vendor ID

Descrambling algorithm descriptor's definition in byte:

0: DESCRAMBLING_ALGORITHM_DSCR_TAG = 0x07

1: descriptor's length

2-3: enums value of algorithm: 0=DVB-CSA2, 1=CSA3

Outputs:

N/A

B.6.2.6 TEE_KLAD_StopDescrambler

Stop descrambling.

Prototype:

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

Inputs:

streamPathLength: length of stream path for descrambling

streamPath: stream path for descrambling

numberOfStreamPids: pid numbers of descrambling stream program

streamPids: pids of descrambling stream program

Outputs:

N/A

Annex C

HSM functional specification

(This annex forms an integral part of this Recommendation.)

C1 Overview

The HSM is the key component for the downloadable conditional access system for unidirectional networks (such as China DTH DCAS). It is a standardized security chipset intended to replace many of the functionalities of the traditional smartcard of the client device. Any authorized CA vendor can use the HSM to implement its security solutions.

C.2 HSM basic functionalities

C.2.1 Activation

Before fully functioning, the HSM chip should be activated. Functions of a not-activated HSM are limited, after receiving the activation message sent by a headend, the HSM is activated and secure functionalities are enabled.

Activation is a HSM process to verify and initialize the CA system, so that only an authorized CA can use the HSM to enhance the security of the DCAS system.

C.2.2 Secure authenticated channel

In order to provide secure services, HSM should establish a secure authenticated channel (SAC) with SoC by using a PairK.

Establishment of the SAC depends upon the activation of the HSM.

C.2.3 CA secure storage

The HSM provides a generic secure storage mechanism to the SoC, such as CA data storage. The security of the storage is established through the SAC: Only if the SoC and HSM authenticate each other and establish the SAC, can the SoC read or write to the secure storage of the HSM.

The storage is provided as a general block to the client. The client is responsible for managing the allocation and usage.

C.2.4 Key ladder process

One of the secure services provided by the HSM is the key ladder process, which is to compute the data used for the key ladder of SoC, the terminal secure chip. The HSM key ladder supports a 3-level key hierarchy, the output of the key ladder will be re-encrypted with CREEK and the encrypted result will be the final output to SoC.

C.2.5 Dependencies on SAC and activation

As activation and SAC are two basic secure services, other HSM services may depend on them. The dependencies which assure the security of the HSM are summarized in Table C.1.

Table C.1 – Dependencies on SAC and activation

Service class	Service name	SAC required	Activation required	Service description
Basic information	Get Public Data	N	N	Read data from Public area of HSM storage
	Get SW Version	N	N	Read out HSM SW version
Activation/Deactivation	Generate Activation Request Message	N	N	Generate the Activation Request Message and sign it with HSM private key
	Set Primary Activation Message	N	N	Verify Primary Activation Message and analyze it to get data, and save the data to HSM
	Set Auxiliary Data Message	N	N	Verify Auxiliary Data Message and analyze it to get data, and save the data to HSM. Primary Activation Message must have been received
	Read Last Timestamp	N	N	Read the last timestamp received in a valid Primary Activation Message
	Deactivate HSM	N	N	Deactivate the HSM and erase all CA Vendor data
	Read Activation Data	N	Y	Read data related to activation including CA Proprietary Data received in Auxiliary Data Message
	Read Location Data	Y	Y	Read location data received in Auxiliary Data Message
Secure Storage	Read Secure Data	Y	Y	Read data via SAC
	Write Secure Data	Y	Y	Write data via SAC
	Read Public Secure Data	N	Y	Read from the Public Secure Storage area
	Write Public Secure Data	Y	Y	Write to the Public Secure Storage area
Key Ladder Process	Crypto Toolkit	Y	Y	Process input and generate the re-encrypted CW
SAC	Open SAC	N	Y	Initialize SAC
	Close SAC	Y	Y	Close the SAC

C.3 Typical activation flow

C.3.1 General overview

At activation, the HSM can register itself into a specific CA headend and retrieve dedicated CA information and keys. The HSM can then work with the corresponding CAS. Activation flow includes 3 basic operations:

a) HSM generates activation request message and delivers to headend;

b) HSM receives and processes the primary activation message from headend;

c) HSM receives and processes the auxiliary activation message from headend.

DCAS client software makes request to the HSM, as the response, the Activation Request Message will be generated and signed by the HSM. It includes a group of information of the client device, and is signed by the private key serialized inside the HSM. The Activation Request Message is then passed to the headend for further processing.

There is only one moment when the one-way DCAS needs two-way communication, which is the time that the activation request message is delivered to headend.

Activation of the HSM is depends on two distinct messages received: the Primary Activation Message, and the Auxiliary Data Message. Until a valid pair has been received, the HSM is not activated and does not provide secure storage or cryptographic services.

The Primary Activation Message is sent by the CA headend to the STB and DCAS client software then passes it to the HSM. The Primary Activation Message contains the time stamp, SoC ID, HSM ID, Vendor_SysID, and critical key material to be used for CW processing.

After the Primary Activation Message has been received, validated and processed, the HSM is still not active, the HSM should wait for a matching Auxiliary Data Message. A "matching" Auxiliary Data Message is the one with an identical timestamp to the Primary Activation message, and the same Vendor ID. The Auxiliary Activation Message contains critical key material for 'pairing' the HSM to the host STB, as well as location information which is used by the main CA application on the STB. Once a valid pair of messages has been received and processed, the HSM is activated and begins to provide its essential security services. Prior to the receipt of a valid pair, requests to use these services are denied by the HSM.

As the China DTH DCAS system is designed to be renewable, the pair of Activation Messages may be received more than once.

Receiving the Primary Activation Message is not dependent on having previously issued an Activation Request Message. There are use-cases where a Primary Activation Message is sent from the headend without an Activation Request Message being generated.

C.3.2 Get Software Version

This function shall return a string indicating the HSM SW version to DCAS client software.

C.3.3 Get Public Data

This function is intended to allow the DCAS client software to read data stored in public storage of the HSM.

The function shall return the following data:

– HSM certificate

– HSM ID

– Activation status (True/False)

– Primary Activation Message received (True/False)

C.3.4 Generate Activation Request Message

The HSM receives the following parameters from the DCAS Client Software:

– CA Vendor Certificate (contains the Vendor_SysID)

– SoC ID

– Location data (from Beidou)

– Timestamp (from Beidou)

The HSM then performs the following operations:

a) Use the trusted authority root public key to check the validity of the signature of CA vendor certificate using the SM3 and SM2 algorithm. Additionally, the following fields of the CA vendor certificate shall be checked to ensure they adhere to the guidelines specified in Annex C.6 "Certificate formats":
   – Version
   – Signature algorithm
   – Issuer
   – Validity Not After – must be a date that is later than the timestamp
   – Subject:OU – must equal "production" in a production HSM. In a debug HSM, this field must equal "test".
   – Subject:CN – must start with the phrase "CHINA DTH CA VENDOR CERTIFICATE"
   – Subject Public Key Info: Subject public key algorithm
   – Subject Public Key Info: Subject's public key (in particular that the key is uncompressed, i.e., starts with 0x04).
   – Subject Public Key Info: Subject's domain
   – Extension: Authority/issuer key identifier
   – Extension: Subject key identifier
   – Extension: Key Usage. This must be "digitalSignature" only.
   – Extension: Basic constraints
   – Signature Algorithm

b) If invalid, stop and produce an error. If valid, extract the Vendor_SysID from the certificate and continue.

c) Compile the Activation Request Message with the following fields:
   – HSM ID
   – SoC ID – from input
   – Vendor_SysID – extracted above
   – Location data – from input
   – Timestamp – from input

d) HSM signs the activation request message using the HSM private key.

e) HSM returns the Activation Request Message and the signature. The value returned by the HSM shall consist of fields 0-17 of the Activation Request Message as specified in Annex C.5.1.

C.3.5 Set Primary Activation Message

Upon receiving the Primary Activation Message, the HSM checks the signature on the message and then proceeds to process its contents.

It is important to note that it is perfectly valid for an already "activated" HSM to receive a new Primary Activation Message. In this case, the HSM reverts to a non-activated state until the new matching Auxiliary Data Message has been received and validated. If the received Primary Activation Message is valid, the update of all data related to the Primary Activation Message shall be in an atomic fashion, such that at no point may the HSM be in a valid "active" state with activation data from two different Primary Activation Messages, or with data from a mismatched Primary Activation Message and Auxiliary Data Message.

When an activated HSM receives a new valid Primary Activation Message, depending on the Vendor_SysID it contains, there are two cases: the received Vendor_SysID differs from the currently active one, or the received Vendor_SysID is identical to the currently active one.

If the Vendor_SysID is different, the HSM must erase all data associated with the previous CA vendor, including the data in the secure store as well as the Auxiliary Data Message. If the Vendor_SysID is identical to the currently active one, the HSM becomes inactive and the Auxiliary Data Message is erased, but data in SAC authentication area must remain intact.

In this function, the HSM receives the following inputs:

– Primary Activation Message

– CAS vendor certificate

The HSM then performs the following operations:

a) Check the validity of the CA Vendor Certificate using the trusted authority root key using the SM3 and SM2 algorithm. The CA vendor certificate shall be checked in the same manner as in clause C.3.4 "Generate Activation Request Message".

b) Use the public key in the CA vendor certificate to check the signature of the Primary Activation Message using the SM3 and SM2 algorithm.
   – If invalid, stop and return an error.
   – If valid, continue.

c) Check the validity of the first byte of the Primary Activation Message:
   – The first 4 bits is the Primary Activation Message version; in this document, the only valid value is 1.
   – The last 4 bits is the Message Type. For the Primary Activation Message, this must be equal to 1.
   – Thus, the first byte of the Primary Activation Message must equal 0x11.

d) Check the timestamp in the current Primary Activation Message against the stored Last Valid Timestamp.
   – If the current timestamp is later than or equal to the last valid timestamp, continue.
   – If not, return an error.

e) Check that the field "Subject:O" in the CA Vendor Certificate is equal to the Vendor_sysID field in the received Primary Activation Message. Note that the Vendor_sysID in the certificate is four hex characters, while the Vendor_sysID in the Primary Activation Message is two bytes.

f) Use the HSM SM2 private key and the key material field from the Primary Activation Message to decrypt the encrypted key in the Primary Activation Message and obtain 16-byte HSM root key (K3_HSM).

g) If the HSM was in an "Activated" state when the command was received, mark the HSM as "non-activated" and mark any data from the Auxiliary Data Message as invalid.

h) If the HSM was in an "Activated" state when the command was received and the received Vendor_sysID differs from the currently active Vendor_sysID, the contents of the Secure Storage shall be erased.

i) The following values shall be written to a private area of storage (This area cannot be read via the secure storage mechanism):
   – SoC ID
   – Last Valid Timestamp <- timestamp from Primary Activation Message
   – Vendor_SysID
   – K3_HSM

j) Mark the newly received data from the Primary Activation Message as valid (though the HSM is not yet activated).

k) Set the Received Primary Activation Message flag to "True".

l) If the operation completes successfully, return a success; otherwise, return an error.

C.3.6 Re-activate and deactivate

There are three distinct types of deactivation. In all three types of deactivation, the data received in the previous Auxiliary Data Message becomes invalid.

a) Deactivation of an HSM chip occurs whenever a valid Deactivation Message is received. This results in the HSM erasing all data (including the secure storage) except for the Last Valid Timestamp.

b) Activation to another CA Vendor. This occurs when an active HSM receives a valid Primary Activation Message with a Vendor_sysID that differs from the currently active Vendor_sysID. This results in the HSM erasing all data from the old CAS Vendor (including Secure Store, and any data received in the Auxiliary Data Message) and storing the data received in the new Primary Activation Message.

c) Reactivation to the same CA Vendor occurs when an active HSM receives a new Primary Activation Message with the same Vendor_sysID as the one currently in use. The result of receiving the new Primary Activation Message is to deactivate the HSM, update the last valid timestamp, and update the key materials.

C.3.7 Auxiliary Data Message

Due to limitations on the size of transport packets, the Primary Activation Message is not large enough to contain the entire data required for device activation. To solve this issue, a second message, known as the Auxiliary Data Message, can be sent from the headend to the HSM. This message may only be received once the HSM has received the Primary Activation Message. The command to send this message to the HSM shall be available without the SAC.

Only once matching pairs of the Primary Activation Message and the Auxiliary Data Message have been received, is the HSM considered to be "activated". The HSM shall use the Vendor_sysID, SOC ID and the timestamp in the two messages to determine if they are matched pairs.

The signature on the Auxiliary Data Message differs from that on the Primary Activation Message, which is based on the algorithm HMAC-SM3 using the derived key from the K3_HSM received in the Primary Activation Message.

The Auxiliary Data Message that contains 3 subfields:

– Location Data – 10 bytes
– Encrypted keys – 32 bytes
– CA Proprietary Data – 71 bytes

When the Auxiliary Data Message is received, the data fields are stored to dedicated storage area of HSM. The Location Data can be read out only under the protection of the SAC, while the CA Proprietary Data can be read out without SAC establishment.

There shall be no means to write either the location data or keys or the CA proprietary data other than through the receipt of a properly signed Auxiliary Data Message.

A successful calling of this function will lead to the result that the location data, keys and CA proprietary data get written to a dedicated Data fields.

The Set Auxiliary Data function receives the following input:

– Auxiliary Data Message

The HSM then performs the following actions:

a) Check that the HSM has received a valid Primary Activation Message.

b) Check the signature on the Auxiliary Data Message by:
   – Compute the HMAC on the entire Auxiliary Data Message (excluding the signature field) using the derived key from K3_HSM.
   – Compare the computed HMAC to the signature field in the received Auxiliary Data Message.

c) Check the validity of the first byte of the Auxiliary Data Message:
   – The first 4 bits is the Auxiliary Data Message Version; in this document, the only valid value is 1.
   – The last 4 bits is the Message Type. For the Auxiliary Data Message, this must be equal to 2.
   – Thus, the first byte of the Auxiliary Data Message must equal 0x12.

d) Check that the Vendor_sysID and SoC ID in the Auxiliary Data Message matches the Vendor_sysID and SoC ID received in the Primary Activation Message.

e) Check that the HSM ID in the Auxiliary Data Message is equal to the HSM ID of the given chip.

f) Check that the timestamp in the Auxiliary Data Message is equal to the Last Activation Timestamp (received in the Primary Activation Message).

g) Decrypt CREEK and SAC Pair key by using SM4 algorithm and a key derived from K3_HSM, and store the decrypted keys into HSM.

h) Write the payload of the CA Proprietary Data to the dedicated CA Proprietary Data field in HSM NVM. Mark the CA Proprietary Data Field as Valid.

i) Write the payload of the Location Data to the dedicated Location Data Field in HSM NVM. Mark the Location Data Field as Valid.

j) Finally, mark the HSM as "Activated".

C.3.8 Read Activation Data

The function to read the following data fields related to activation shall be provided:

– CA Proprietary Data

– Activated SoC ID

– Activated Vendor_SysID

It shall be possible to access this function without the SAC. If the HSM is activated, the HSM shall return the content of the fields. If not valid, the HSM shall return an error.

C.3.9 Read Location Data

The function to read the Location Data shall be provided. It shall be available only within the SAC. If the Location Data is marked as Valid, the HSM shall return the Location Data; otherwise, the HSM shall return an error.

C.3.10 Read Last Valid Timestamp

The function to read the last valid timestamp received in a Primary Activation Message shall be provided. It shall be available outside of the SAC and does not require the HSM to be activated. The function is used by the DCAS Client Software to determine whether or not it possesses a matching Auxiliary Data Message.

C.3.11 Deactivation Message

The China DTH DCAS operator has requested that there be a method for sending a message to the HSM which will reset the HSM to an inactive state, and delete any data received through the Primary Activation or Auxiliary Data Messages. After receiving a valid deactivation message, the HSM shall return to its initial state, with the exception of the Last Timestamp field, which will be updated to the timestamp received in the Deactivation Message.

The Deactivation Message can be processed only under the protection of the SAC. Thus, the HSM must be activated in order to process the Deactivation Message.

In this function, the HSM receives the following inputs:

– Deactivation Message

– CAS Vendor Certificate

Upon receiving the Deactivation message, HSM shall:

a) Check the validity of the CA Vendor Certificate using the Trusted Authority Root Key using the SM3 and SM2 algorithm. The CA Vendor Certificate shall be checked in the same manner as in section C3.4 "Generate Activation Request Message".

b) Use the public key in the CA Vendor Certificate to check the signature of the Deactivation Message using the SM3 and SM2 algorithm.
   – If invalid, stop and return an error.
   – If valid, continue.

c) Check the validity of the first byte of the Deactivation Message:
   – The first 4 bits is the Deactivation Message Version; in this document, the only valid value is 1.
   – The last 4 bits is the Message Type. For the Deactivation Message, this must be equal to 3.
   – Thus, the first byte of the Deactivation Message must equal 0x13.

d) Check the timestamp in the current Deactivation Message against the stored Last Valid Timestamp.
   – If the current timestamp is later than or equal to the Last Valid Timestamp, continue.
   – If not, return an error.

e) Check that the field "Subject:O" in the CA Vendor Certificate is equal to the Vendor_sysID field in the received Deactivation Message. Note that the Vendor_sysID in the certificate is four hex characters, while the Vendor_sysID in the Deactivation Message is two bytes.

f) Check that the HSM ID in the Deactivation Request Message equals the HSM ID of the given HSM.

g) Mark the HSM as inactive.

h) Set the Received Primary Activation Message flag to "False".

i) Update the Last Valid Timestamp to the values received in the Deactivation Message.

j) Delete all data received in the Primary Activation Message, the Auxiliary Data Message, and the entire Secure Storage. The HSM should be identical to an initial HSM except for the value of the Last Valid Timestamp.

k) Return success or failure.

C.4 Secure authenticated channel (SAC)

C.4.1 Overview

The secure authenticated channel (SAC) is a secure data channel established between HSM and SoC, to protect all sensitive data and operations between HSM and SoC.

SoC can only initiate the establishment of SAC after HSM being activated.

The SAC between the HSM and the SoC comprises two stages:

– First, the SAC establishment stage (A.K.A. the handshake) is used to authenticate both parties and negotiate a session key.

– Then begins the communication, steady-state operational stage, where all communication is protected with the session key.

In both stages, the SoC is always the initiator of communication, and the HSM only responds.

C.4.2 Handshake

During the handshake stage of the SAC protocol, SOC and HSM shall have the abilities to do:

a) Both SOC and HSM shall generate an at least 16-bytes random number (RN) to be used in the computation of the session key.

b) PairK is used to the security of communication and/or computation during the SAC establishment

c) SoC creates a counter which is initialized with a random number i. It will be used for the subsequent SAC setup steps. Both SoC and HSM should store the latest value received during exchanges, and increase it by 1 when constructing handshake messages.

d) Both HSM and SOC shall compute the session key SK based on the RNs from both the SOC and HSM

C.4.3 Communication

After the handshake stage has ended, the SoC and the HSM can communicate with individual messages, each of which is encrypted using derivatives of the session key, SK. The way of derivation is out of scope of this standard. The communication flowchart is shown as Figure C.1.

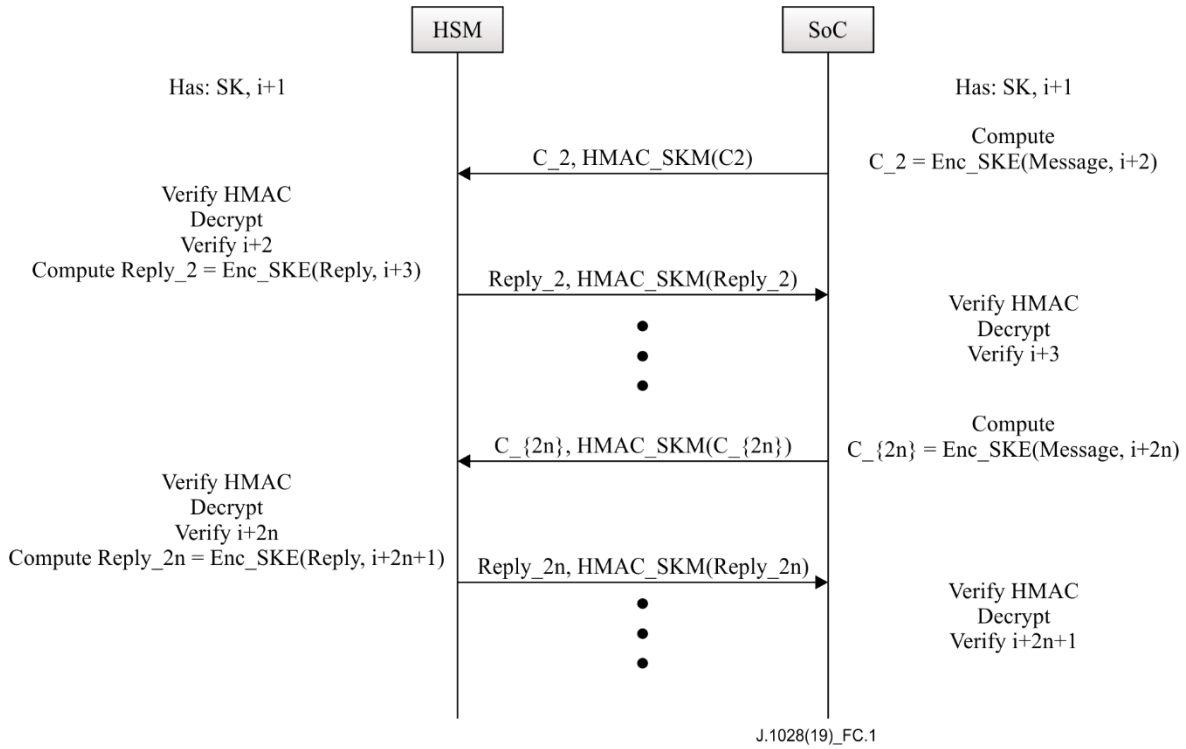

Figure C.1 – Process of the communication stage

a) Both the HSM and the SoC shall derive two 16-byte keys from the session key, called the session key encryption (SKE) and the (session key MAC) SKM.

b) When sending a message, the SoC/HSM shall:
   - Increment the counter.
   - Take the message and append the current value of the counter to form a packet.
   - Encrypt the packet with SKE.
   - Sign the packet with SKM.

c) When receiving a packet, the SoC/HSM shall:
   - Verify the packet's signature using SKM.
   - Decrypt the packet using SKE.
   - Verify that the decrypted counter is larger than the current value of the counter.
   - Update the counter to the value received in the packet.

d) Both SoC and HSM should check the counter upon each message received.

C.5 Message formats

In this clause, the relevant activation message formats are defined, which are subject to change when required.

C.5.1 Activation Request Message

Table C.2 describes the Activation Request Message. This message is contained in a standard tag-length-value descriptor string. In order to adapt to the length of the HSM certificate and limit the total length of the message, the highest bit of the "tag" field indicates the "length" field is 1 or 2 bytes. Thus, the tags 0x00-0x7F have the "length" field of 1 byte, while tags 0x80-0xFF have the length field of 2 bytes.

Table C.2 – Activation Request Message format

Field number	Field	Length (bytes)	Comment
0	Message Version	1	Message Version = 0x01
1	Tag	1	0x01 – Timestamp
2	Length	1	4
3	Timestamp	4	Beidou Format
4	Tag	1	0x02 – Vendor_SysID
5	Length	1	2
6	Vendor_SysID	2	Vendor_SysID
7	Tag	1	0x03 – STB Data
8	Length	1	16
9	SoC ID	8	SoC ID – 8 bytes
10	HSM ID	8	HSM ID – 8 bytes
11	Tag	1	0x04 – Position data tag
12	Length	1	8
13	Longitude	4	As returned by Beidou
14	Latitude	4	As returned by Beidou
15	Tag	1	0x0A – HSM signature
16	Length	1	64
17	Signature	64	This shall be passed unchanged to the headend for signature checking – it is an SM2 signature on all bytes in fields 0-16, using HSM Private Key
18	Additional CA Data	arbitrary	Must be in TLV format with tags from 0x10-0x3F

C.5.2 Primary Activation Message

The Primary Activation Message is sent by the headend to the terminal to initialize the HSM and DCAS Trusted APP. The message has a fixed format and is described in Table C.3.

Table C.3 – Primary Activation Message format

Field	Length (bytes)	Comment
Header Byte	1	First 4 bits – Version=1; Last 4 bits – Message Type = 1 (Primary Activation Message)
Time stamp	4	In Beidou format
SoC ID	8	SoC ID
HSM ID	8	HSM ID
Vendor_SysID	2	Vendor_SysID
Key Material C1	33	Compressed Key Material for deriving key data for decrypting HSM root key K3_HSM
Encrypted Keys C2	16	$E_{HSMPubKey}$ (K3_HSM)

Table C.3 – Primary Activation Message format

Field	Length (bytes)	Comment
Digest Data C3	32	An input parameter of SM2 decryption
Signature	64	Signed by CA Vendor Private Key using SM2 over all previous bytes

C.5.3 Auxiliary Data Message

Table C.4 describes the Auxiliary Data Message. It is of a fixed length of 168 bytes.

Table C.4 – Auxilary Data Message format

Grouping	Field	Length (bytes)	Comment
	Header Byte	1	First 4 bits – Version=1; Last 4 bits – Message Type = 2 (Auxiliary Data Message)
Header	Timestamp	4	Beidou Format
	SoC ID	8	SoC ID
	HSM ID	8	HSM ID
	Vendor_SysID	2	Vendor_SysID
Location Data	Longitude	4	In Beidou format
	Latitude	4	In Beidou format
	Maximum distance	2	In tens of meters – up to 650 km NOTE – DCAS trusted APP multiplies this value by 10 when comparing it with the distance between the point (longitude and latitude) in location data and the point returned by Beidou.
Encrypted Keys	Encrypted keys	32	$E_{KDF(K3\_HSM,48)}$ (CREEK, PairK), CREEK and PairK encrypted with a key derived from K3_HSM with SM4-CBC (IV=0). The KDF is described in part 3 of SM2 standard. The first 16 bytes of the 48-byte derived key is used for SM4 above, and the last 32 bytes of the 48-byte derived key is used for HMAC-SM3 below.
CA Proprietary Data	CA Proprietary Data	71	CA proprietary data
Signature	Signature	32	Signed by HMAC-SM3 using a key derived from K3_HSM

C.5.4 Deactivation Message

Table C.5 describes the Deactivation Message.

Table C.5 – Deactivation Message format

Field	Length (bytes)	Comment
Header Byte	1	First 4 bits- Version=1; Last 4 bits- Message Type = 3 (Deactivation Message)
Time stamp	4	In Beidou format
Reserved	8	Reserved
HSM ID	8	HSM ID
Vendor_SysID	2	Vendor_SysID
Signature	64	Signed by CA Vendor Private Key using SM2 over all previous bytes

C.6 Certificate formats

Table C.6 describes the formats of all certificates used in DCAS system, which should be compliant with [b-GM/T 0015].

Table C.6 – Certificate formats

	TA root certificate	CA vendor root certificate	HSM vendor root certificate	HSM device dertificate	Comments
Version	V3	V3	V3	V3	
Serial number	Default value when issuing	Default value when issuing	Default value when issuing	Default value when issuing	Ignored
Signature algorithm	SM3 SM2	SM3 SM2	SM3 SM2	SM3 SM2	OID is 1.2.156.10197.1.501
Issuer	Same as TA Subject field	Same as TA Subject field	Same as TA Subject field	Same as HSM vendor Subject field	
Validity: Not before	Certificate Issuing Date	Certificate Issuing Date	Certificate Issuing Date	Certificate Issuing Date	Currently ignored. In future, we might want to check this field to identify old devices.
Validity: Not after	Up to 50 years after issuing	Up to 50 years after issuing	Up to 50 years after issuing	Up to 50 years after issuing	Ignored
Subject: O	"SARFT TRUSTED AUTHORITY"	<Vendor_SysID>	<HSM vendor name>	<HSM device ID>	Vendor_SysID – 4 hex digits. HSM vendor name - up to 20 characters. HSM device ID – 16 hex digits
Subject: OU	"TEST" or "PRODUCTION"	"TEST" or "PRODUCTION"	"TEST" or "PRODUCTION"	"TEST" or "PRODUCTION"	Production system has to verify that value of this field is "PRODUCTION"

Table C.6 – Certificate formats

	TA root certificate	CA vendor root certificate	HSM vendor root certificate	HSM device dertificate	Comments
Subject: CN	"SARFT TRUSTED AUTHORITY MANAGEMENT CERTIFICATE"	"CHINA DTH CA VENDOR CERTIFICATE - <CA Vendor Name>"	"CHINA DTH HSM VENDOR CERTIFICATE - <HSM vendor name>"	"CHINA DTH HSM DEVICE CERTIFICATE"	CA vendor name – up to 20 characters. HSM vendor name – up to 20 characters.
Subject: C, ST, L	Default	Default	Default	Default	
Subject Public Key Info: Subject Public Key Algorithm	EC public key	EC public key	EC public key	EC public key	OID is 1.2.840.10045.2.1
Subject Public Key Info: Subject Public Key Algorithm Parameters	SM2 sm2p256v1	SM2 sm2p256v1	SM2 sm2p256v1	SM2 sm2p256v1	OID is 1.2.156.10197.1.301
Subject Public Key Info: Subject's Public Key	Uncompressed form (starts with 0x04)	Uncompressed form (starts with 0x04)	Uncompressed form (starts with 0x04)	Uncompressed form (starts with 0x04)	
Extension: Authority Key Identifier	Non-critical, key_id only	Non-critical, key_id only	Non-critical, key_id only	Non-critical, key_id only	Calculated according to option (1) of clause 4.2.1.2 in RFC 5280
Extension: Subject Key Identifier	Non-critical	Non-critical	Non-critical	Non-critical	Calculated according to option (1) of clause 4.2.1.2 in RFC 5280
Extension: Key Usage	Critical, KeyCertSign	Critical, digitalSignature	Critical, KeyCertSign	Critical, digitalSignature + keyEncipherment	
Extension: Basic Constraints	Critical, True, path len = 1	Critical, False	Critical, True, path len = 0	Critical, False	True = It is a CA authority. 2/1/0 = pathlen (level of subsequent CA authorities in chain)
Signature Algorithm	SM3 SM2	SM3 SM2	SM3 SM2	SM3 SM2	Same as certificate Signature Algorithm field

Table C.6 – Certificate formats

	TA root certificate	CA vendor root certificate	HSM vendor root certificate	HSM device dertificate	Comments
Signature	(Note: Self-signed)				First 'r' and then 's'. Each is 32 bytes.

Bibliography

[b-ITU-T J.93] Recommendation ITU-T J.93 (1998), *Requirements for conditional access in the secondary distribution of digital television on cable television systems.*

[b-ITU-T J.290] Recommendation ITU-T J.290 (2006), *Next generation set-top box core architecture.*

[b-GB/T 32907] GB/T 32907 (2016), *Information security technology-SM4 block cipher algorithm.*
http://openstd.samr.gov.cn/bzgk/gb/newGbInfo?hcno=7803DE42D3BC5E80B0C3E5D8E873D56A

[b-GM/T 0015] GM/T 0015 (2012), *Digital certificate format based on SM2 algorithm*.
http://www.gmbz.org.cn/main/viewfile/20180110020301917 80.html

SERIES OF ITU-T RECOMMENDATIONS

Series A	Organization of the work of ITU-T
Series D	Tariff and accounting principles and international telecommunication/ICT economic and policy issues
Series E	Overall network operation, telephone service, service operation and human factors
Series F	Non-telephone telecommunication services
Series G	Transmission systems and media, digital systems and networks
Series H	Audiovisual and multimedia systems
Series I	Integrated services digital network
Series J	**Cable networks and transmission of television, sound programme and other multimedia signals**
Series K	Protection against interference
Series L	Environment and ICTs, climate change, e-waste, energy efficiency; construction, installation and protection of cables and other elements of outside plant
Series M	Telecommunication management, including TMN and network maintenance
Series N	Maintenance: international sound programme and television transmission circuits
Series O	Specifications of measuring equipment
Series P	Telephone transmission quality, telephone installations, local line networks
Series Q	Switching and signalling, and associated measurements and tests
Series R	Telegraph transmission
Series S	Telegraph services terminal equipment
Series T	Terminals for telematic services
Series U	Telegraph switching
Series V	Data communication over the telephone network
Series X	Data networks, open system communications and security
Series Y	Global information infrastructure, Internet protocol aspects, next-generation networks, Internet of Things and smart cities
Series Z	Languages and general software aspects for telecommunication systems

Printed in Switzerland
Geneva, 2019