International Telecommunication Union

ITU-T

TELECOMMUNICATION
STANDARDIZATION SECTOR
OF ITU

X.1194

(04/2012)

SERIES X: DATA NETWORKS, OPEN SYSTEM COMMUNICATIONS AND SECURITY

Secure applications and services – IPTV security

Algorithm selection scheme for service and content protection descrambling

Recommendation ITU-T X.1194

ITU-T X-SERIES RECOMMENDATIONS

DATA NETWORKS, OPEN SYSTEM COMMUNICATIONS AND SECURITY

PUBLIC DATA NETWORKS	X.1–X.199
OPEN SYSTEMS INTERCONNECTION	X.200–X.299
INTERWORKING BETWEEN NETWORKS	X.300–X.399
MESSAGE HANDLING SYSTEMS	X.400–X.499
DIRECTORY	X.500–X.599
OSI NETWORKING AND SYSTEM ASPECTS	X.600–X.699
OSI MANAGEMENT	X.700–X.799
SECURITY	X.800–X.849
OSI APPLICATIONS	X.850–X.899
OPEN DISTRIBUTED PROCESSING	X.900–X.999
INFORMATION AND NETWORK SECURITY	
General security aspects	X.1000–X.1029
Network security	X.1030–X.1049
Security management	X.1050–X.1069
Telebiometrics	X.1080–X.1099
SECURE APPLICATIONS AND SERVICES	
Multicast security	X.1100–X.1109
Home network security	X.1110–X.1119
Mobile security	X.1120–X.1139
Web security	X.1140–X.1149
Security protocols	X.1150–X.1159
Peer-to-peer security	X.1160–X.1169
Networked ID security	X.1170–X.1179
IPTV security	**X.1180–X.1199**
CYBERSPACE SECURITY	
Cybersecurity	X.1200–X.1229
Countering spam	X.1230–X.1249
Identity management	X.1250–X.1279
SECURE APPLICATIONS AND SERVICES	
Emergency communications	X.1300–X.1309
Ubiquitous sensor network security	X.1310–X.1339
CYBERSECURITY INFORMATION EXCHANGE	
Overview of cybersecurity	X.1500–X.1519
Vulnerability/state exchange	X.1520–X.1539
Event/incident/heuristics exchange	X.1540–X.1549
Exchange of policies	X.1550–X.1559
Heuristics and information request	X.1560–X.1569
Identification and discovery	X.1570–X.1579
Assured exchange	X.1580–X.1589

For further details, please refer to the list of ITU-T Recommendations.

Recommendation ITU-T X.1194

Algorithm selection scheme for service and content protection descrambling

Summary

Recommendation ITU-T X.1194 develops an algorithm selection standard for descrambling in various terminal devices. This Recommendation provides the general service and content protection (SCP) architecture, security requirements and algorithm selection scheme (ASS). In particular, the algorithm selection scheme consists of the SCP control client function, ASS descrambler/demuxer control function and descrambler authentication function.

History

Edition	Recommendation	Approval	Study Group
1.0	ITU-T X.1194	2012-04-13	17

Keywords

Conditional access, descrambling, multiple descrambling, SCP client, service protection, terminal device.

Table of Contents

Page

Recommendation ITU-T X.1194

Algorithm selection scheme for service and content protection descrambling

1 Scope

Recommendation ITU-T X.1194 develops a set of algorithm selection functions from the existing descrambling algorithms to share terminal devices between service providers and security providers. The scope includes algorithm selection schemes (ASSs) and signalling for selection and interoperability issues and does not include any other schemes of Recommendation ITU-T X.1191.

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 X.800]	Recommendation ITU-T X.800 (1991), *Security architecture for Open Systems Interconnection for CCITT applications*.
[ITU-T X.1191]	Recommendation ITU-T X.1191 (2009), *Functional requirements and architecture for IPTV security aspects*.

3 Definitions

3.1 Terms defined elsewhere

This Recommendation uses the following terms defined elsewhere:

3.1.1 access control [ITU-T X.800]: The prevention of unauthorized use of a resource, including the prevention of use of a resource in an unauthorized manner.

3.1.2 authentication [ITU-T X.800]: See data origin authentication, and peer-entity authentication.

3.1.3 authorization [ITU-T X.800]: The granting of rights, which includes the granting of access based on access rights.

3.1.4 conditional access [ITU-T X.1191]: The function served by a conditional access system; often used as an abbreviation for conditional access system.

3.1.5 conditional access system [ITU-T X.1191]: A component of a Service and Content Protection system the purpose of which is to prevent unauthorized (unentitled) access to a service or to content.

3.1.6 confidentiality [ITU-T X.800]: The property that information is not made available or disclosed to unauthorized individuals, entities, or processes.

3.1.7 data origin authentication [ITU-T X.800]: The corroboration that the source of data received is as claimed.

3.1.8 digital signature [ITU-T X.800]: Data appended to, or a cryptographic transformation (see cryptography) of a data unit that allows a recipient of the data unit to prove the source and integrity of the data unit and protect against forgery e.g., by the recipient.

3.1.9 integrity [ITU-T X.800]: The property that data has not been altered or destroyed in an unauthorized manner.

3.1.10 key [ITU-T X.800]: A sequence of symbols that controls the operations of encipherment and decipherment.

3.1.11 key management [ITU-T X.800]: The generation, storage, distribution, deletion, archiving and application of keys in accordance with a security policy.

3.1.12 peer-entity authentication [ITU-T X.800]: The corroboration that a peer entity in an association in the one claimed.

3.1.13 scrambling algorithm [ITU-T X.1191]: An algorithm used in a scrambling (encryption) or descrambling (decryption) process.

3.1.14 service and content protection [ITU-T X.1191]: A combination of service protection and content protection, or a system or implementation thereof.

3.1.15 service protection [ITU-T X.1191]: Ensuring that an end user can only acquire a service, and, by extension, the content contained therein, that they are entitled to receive.

3.2 Terms defined in this Recommendation

This Recommendation defines the following terms:

3.2.1 demuxer: Device that takes an input signal and selects one of the output-channels; the selected output channel is connected to the input signal.

3.2.2 descrambler: Device that transposes or decrypts encoded messages or video streams for the terminal or subscribers with privilege.

3.2.3 process: Instance of a computer program that is being executed; contains the program code and its current status.

3.2.4 virtual machine: Software implementation of a machine that can execute programs just like a physical machine; supports separated operating systems.

4 Abbreviations and acronyms

This Recommendation uses the following abbreviations and acronyms:

A/V	Audio/Video
AAA	Authentication, Authorization, and Accounting
ASS	Algorithm Selection Scheme
CA	Conditional Access
CAS	Conditional Access System
CAT	Conditional Access Table
CBC	Cipher Block Chaining
CW	Control Word
DTCP	Digital Transmission Content Protection
ECM	Entitlement Control Message
EMM	Entitlement Management Message

HMAC	Hash-based Message Authentication Code
IDSA	IIF Default Scrambling Algorithms
IIF	IPTV Interoperability Forum
IPTV	Internet Protocol Television
MK	Master Key
MMS	Multi-Mode Service
MPEG	Moving Picture Experts Group
MPEG-2 TS	MPEG-2 Transport Stream
OTID	One-Time Identifier
PAT	Program Association Table
PES	Packetized Elementary Stream
PGP	Pretty Good Privacy
PID	Program Identifier
PKI	Public Key Infrastructure
PMT	Program Map Table
RAL	Resource Abstraction Layer
SAC	Secure Authenticated Channel
SCP	Service and Content Protection
SIM	Subscriber Identity Module
SP	Service Protection
SSL	Secure Socket Layer
SVM	Secure Virtual Machine
TD	Terminal Device
TK	Terminal's SAC Key
TPS	Triple Play Service
TV	Television
USB	Universal Serial Bus
VM	Virtual Machine
VoD	Video on Demand
VPN	Virtual Private Network

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 Recommendation 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 Introduction

In general, two or more SCPs are deployed on a single terminal device. The content acquired via one SCP system (e.g., from a network) can be accessed via another SCP residing on the same device according to the granted rights. There is a need to support SCP interoperability among multiple security systems that use different security mechanisms to support service and content protection interoperability, thereby maintaining transparency for users.

The object of this Recommendation is to develop a set of functions of algorithm selection schemes from the existing algorithms for content descrambling. This includes the algorithm selection scheme, SCP function, RAL function, interoperability support function and message format.

6.1 Internet Protocol television (IPTV) general architecture and content protection architecture

The general security architecture for IPTV is depicted in Figure 1 below. This general architecture is divided into two primary areas: one considered in-scope for the purpose of considering interoperability based on this Recommendation, and another considered out-of-scope. The first area encompasses the end-user, network provider, and service provider domains, whereas the second area covers the content provider domain.

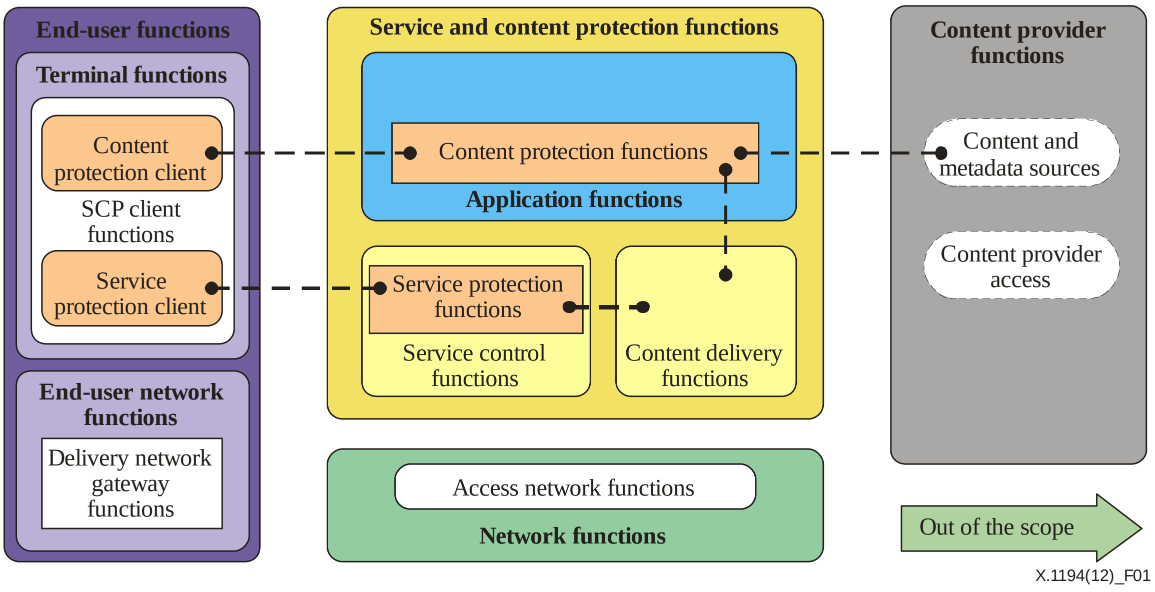

End-user functions
Terminal functions
Content protection client
SCP client functions
Service protection client
End-user network functions
Delivery network gateway functions
Service and content protection functions
Content protection functions
Application functions
Service protection functions
Service control functions
Content delivery functions
Access network functions
Network functions
Content provider functions
Content and metadata sources
Content provider access
Out of the scope
X.1194(12)_F01

Figure 1 – IPTV general security architecture

6.2 Service protection architecture

The service protection architecture for IPTV is depicted in Figure 2 below. The primary functions of the service protection architecture include:

- Authentication and authorization

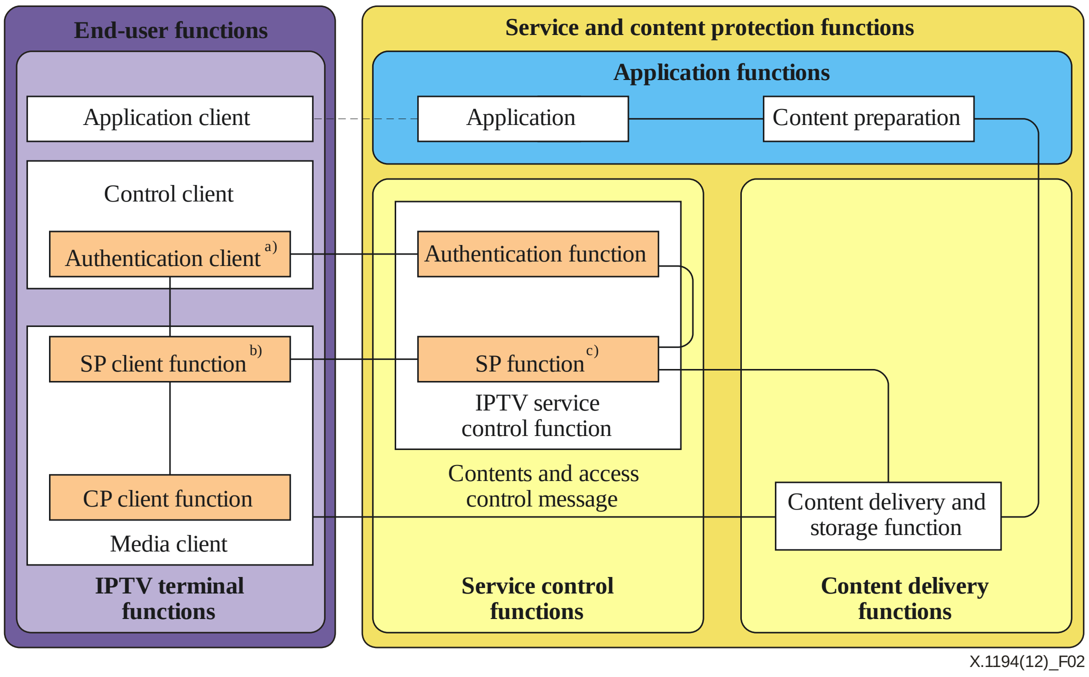
  For managed services involving protected content, it is typically the case wherein the end user (subscriber) must be authenticated and – subsequent to successful authentication – authorized to access service(s) and content therein.

  Depending on the circumstances, authentication and authorization functions may be performed separately on the IPTV terminal device and the end user(s). In other cases, additional devices in the end-user premises such as delivery network gateway and other end-user devices may require authentication before service access is authorized.

  The combination of authentication and authorization can be considered to effect positive access control on the terminal device and the end user for purposes of service and content acquisition prior to use.

- Control signalling and content interchange encryption

  To limit the acquisition of and access to services and content, both control message information and content itself are encrypted (scrambled). Different (strength) levels of encryption may be applied to various control signalling and content interchange paths in accordance with the requirements of a concrete IPTV system or the service providers.



[a] Authentication: It identifies a subscriber name and ID with the assigned privilege.
[b] SP client: A client program for service protection; SP is a shortened form for "service protection".
[c] SP function: A service access control function that protects a service from unauthorized access.

Figure 2 – IPTV service protection architecture

A service protection system will in most – if not all – cases, contain mechanisms that can – or do – perform encryption (scrambling) and decryption (descrambling) of both service control signalling traffic and content traffic. Typically, two-way service control traffic will be encrypted in both directions: from server to client and from client to server. On the other hand, content streams will typically be encrypted only from server (service provider) to client (terminal device). Nevertheless, there are usage scenarios wherein a content stream may be uploaded from client to server, in which case such content may be encrypted on a terminal device for uploading purposes (to ensure, for example, that only an authenticated, authorized service provider may access the uploaded content).

6.3 General software architecture for terminal devices

Normally, a terminal device has hardware and software logic to receive and play IPTV contents. The software supports two major functions, e.g., middleware and security. The IPTV middleware supports several functions, including the remote controller, data broadcasting, and media related processing. Another major component is the security function, including the security data session filter and key generation. In the terminal device, the SCP client is only one component of the SCP system.

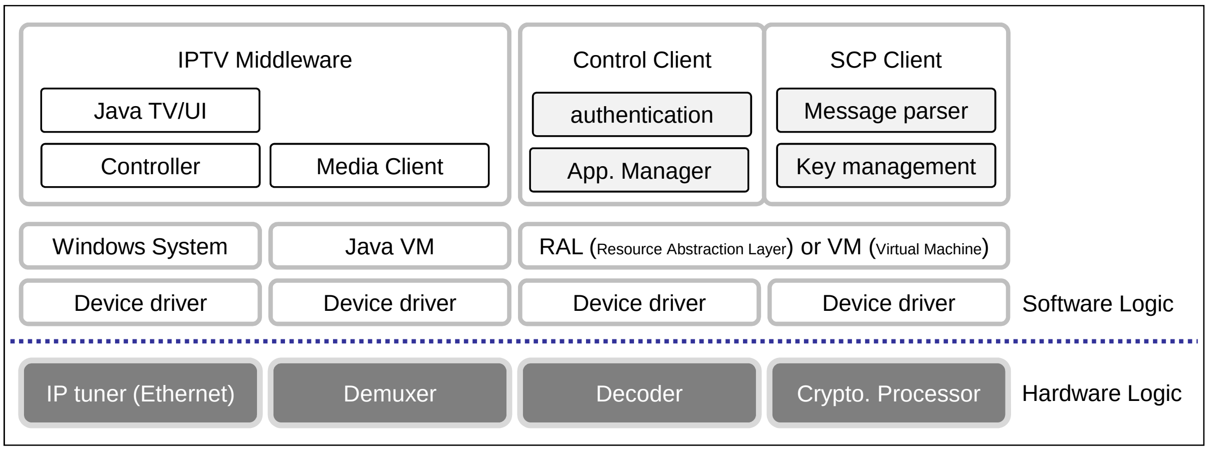


Figure 3 – General software architecture for terminal devices

In Figure 3, the SCP client, one of its security modules, operates based on the resource abstraction layer (RAL) or the virtual machine (VM). A software SCP client can be implemented by native code or application code. Native code means that it does not support portability but is pre-installed on the terminal device. Application code means that the SCP client code runs on a virtual machine.

The Resource abstraction layer (RAL) is an abstraction layer implemented in software between the physical hardware resource and the software that runs on that device. Thus, an SCP client takes unique APIs for hardware access and independent execution. As an isolated duplicate of a real machine, the virtual machine (VM) can provide an instruction set to support the SCP client.

An SCP client can be interoperable among several devices whenever a device's platform is the same. However, in other cases the SCP client is not replaceable.

6.4 Requirement for algorithm selection mechanism for service protection [ITU-T X.1191]

- R 6.1.4-04: The IPTV architecture is required to support mechanisms that allow IPTV services delivery from a third-party provider, as defined in clause 3.
- R 6.3.3-04: The IPTV architecture is required to support the capability to update and query the SCP system concerning scrambling algorithms for IPTV, and any other operator-selected scrambling algorithms.

- R 6.3.3-13: The IPTV architecture is required to support a mechanism to securely retrieve the SCP parameters (e.g., configuration, status) from IPTV terminal device.
- R 6.3.3-14: The IPTV architecture is required to support a mechanism to securely update the SCP parameters (e.g., configuration) of the IPTV terminal device.
- R 6.3.3-16: The IPTV architecture is prohibited from precluding support for the installation and operation of multiple service protection solutions without any hardware replacement except removable devices (such as USB dongle and SIM cards).
- R 6.3.3-19: The IPTV architecture is prohibited from precluding support for a mechanism for the selection of a SCP system from the available SCP systems without any hardware replacement except removable devices.
- R 6.3.3-20: The IPTV architecture is prohibited from precluding support for secure downloading of a SCP system. The specific downloading can optionally depend on specific service protection requirements.
- OR 6.3.2.1-01: Scrambling algorithms for IPTV can optionally apply cryptographic algorithms of different security strength depending on content value.

7 Algorithm selection scheme

The algorithm selection scheme (ASS) provides a common descrambling function with the support of a descrambler control function and a pre-defined ASS descrambling algorithm. In addition, the scrambler provides messages to indicate which algorithm is used. The message depends on the delivery system, such as MPEG-2 TS. In this Recommendation, simple, reasonable message format is provided, and adaptation to other delivery systems is required.

7.1 Overview

Traditional service protection functions are designed with the hardware-based approach. The software-based approach is less secure because the terminal device only has a one-way communication channel. Note, however, that IPTV terminal devices are always online and that they use a two-way communication channel. This means that we can provide a more secure mechanism without hardware support such as security processor.

To remove the hardware dependency, the descrambling function should be separated from the service protection functions. The separated descrambling function is defined by the existing common cryptographic algorithms.

7.1.1 ASS role model for the service protection of linear IPTV

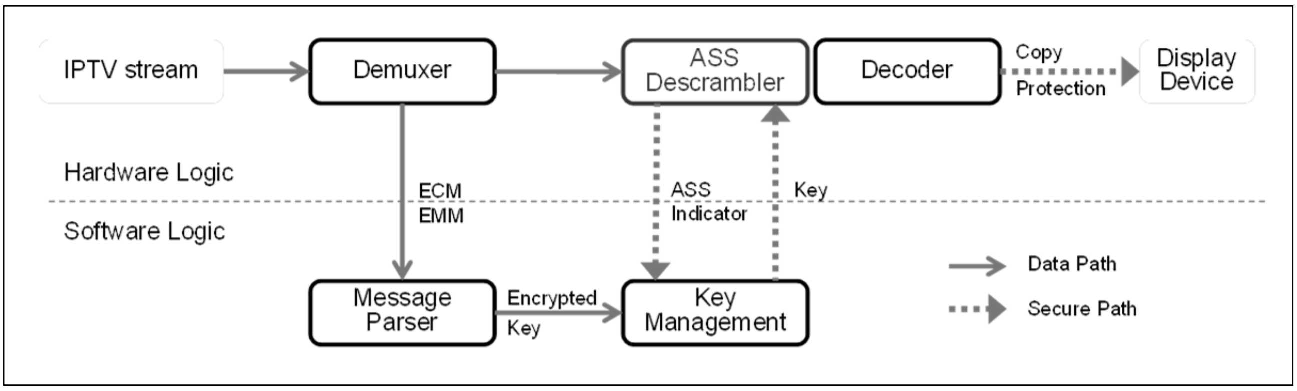


Figure 4 – Key management and ASS descrambler

The relation between an ASS descrambler and a key management function is shown in Figure 4. The key management framework in [b-ITU-T X.1193] generates several keys, and the content decryption key is securely delivered to the ASS descrambler through a secure, authenticated channel.

In Figure 4, the original IPTV stream, which includes linear TV and VoD contents, is delivered to the demuxer (or IP tuner); the demuxer then decodes the received contents and parses the system information defined in the delivery system MPEG-2 TS, because the IPTV stream is encoded with MPEG-TS. In this phase, we extract several stream channels, such as the video stream, audio stream, data stream, and SCP data stream. The information concerning each stream channel is defined in the system information, indicating which packet is which data stream.

After demuxing, the IPTV stream is divided into video/audio stream and data stream; video/audio streams (or only video stream) are scrambled with a key. The ASS descrambler receives a descrambling key from the key management function and descrambles video/audio contents. Decrypted plain video/audio stream is moved to the decoder, which decodes the MPEG contents; the uncompressed video contents are then sent to the display device. Generally, the descrambler and decoder are strongly coupled to minimize the possibility of attack because the data path among these two parties is not protected. This is why those components are implemented with single hardware or single software logic.

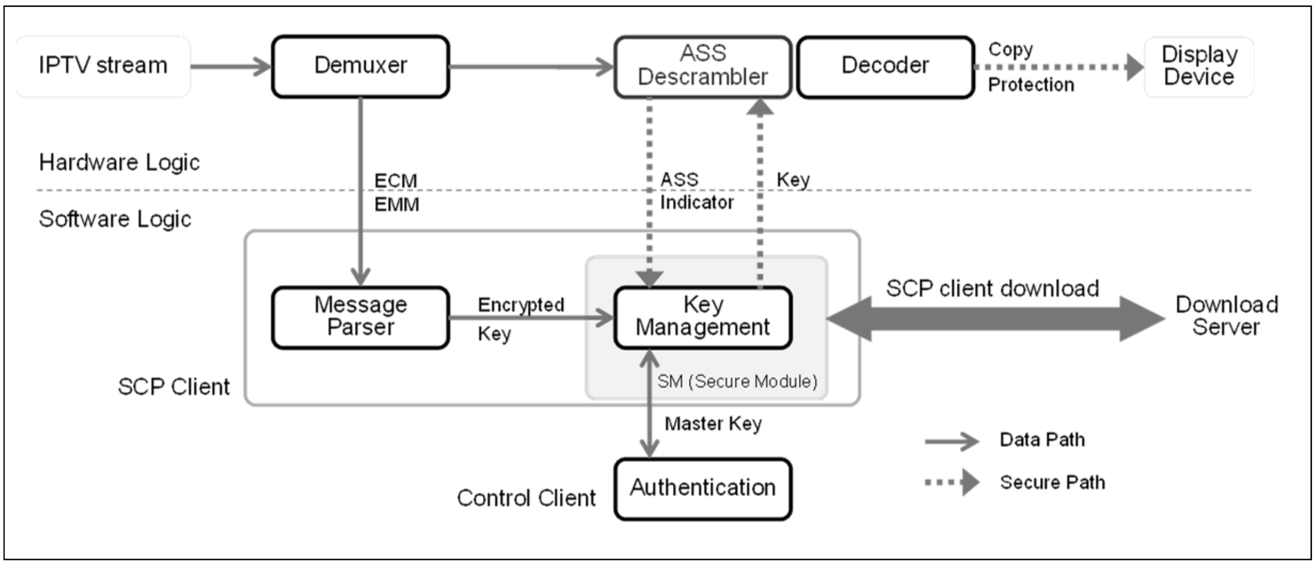


Figure 5 – Downloadable SCP and ASS descrambler

Figure 5 depicts the role of the ASS descrambler in the case of a downloadable SCP. The thick arrow indicates SCP client code download. The thin arrow is content flow, and the dotted arrows are secure communication channels. Each component has the following functions:

– IPTV stream: Integrated contents that include audio, video and data stream.

– Demuxer: This divides the input stream into individual output streams. The audio and video streams are transferred to the ASS descrambler if it is scrambled, and the data stream is sent to the SCP client. Otherwise, the audio and video streams are transferred to the decoder without the descrambling phase.

– SCP client: To decrypt all encrypted streams, the SCP client decodes the data stream and receives SCP-related data, which includes the descrambling algorithm and the decryption key. The SCP client sends those data to the ASS descrambler through a secure communication channel. The SCP client has two functional logics: one is the message parser, which decodes error and key update messages for the ASS descrambler; the other is the secure module (SM), which generates decryption keys within a time period and sends the contents decryption key to an ASS descrambler. The SM provides a key management function, and it is defined in [b-ITU-T X.1193].
– Control client: As a conceptual function for service protection, it includes the client function and the server side function for SCP client management, which involves downloading an SCP client and authenticating the terminal device. After authentication, the SCP client and authentication server can share the master key to make a secure communication channel.
– ASS descrambler: This decrypts all scrambled or encrypted contents. The decryption key is received from the SCP client and the control client. The main role is real-time descrambling and self-protection from several attacks.
– Decoder: This receives the clean audio and video stream. This stream is encoded with a specified algorithm such as MPEG-2 [b-ISO/IEC 13818-1] or ITU-T H.264 [b-ITU-T H.264]. Therefore, the decoder refers to the MPEG media decoder. After decoding, a clean audio and video stream is re-encrypted by a copy protection mechanism such as DTCP between the decoder and display device.
– Display device: Physical display device that supports a copy protection mechanism to receive the protected audio and video data.

Generally, the demuxer, descrambler, and decoder functions are implemented in hardware form, because the system requires real-time processing and no interference from a legacy application.

Figure 6 illustrates the related system components and message flow between components in the terminal device.

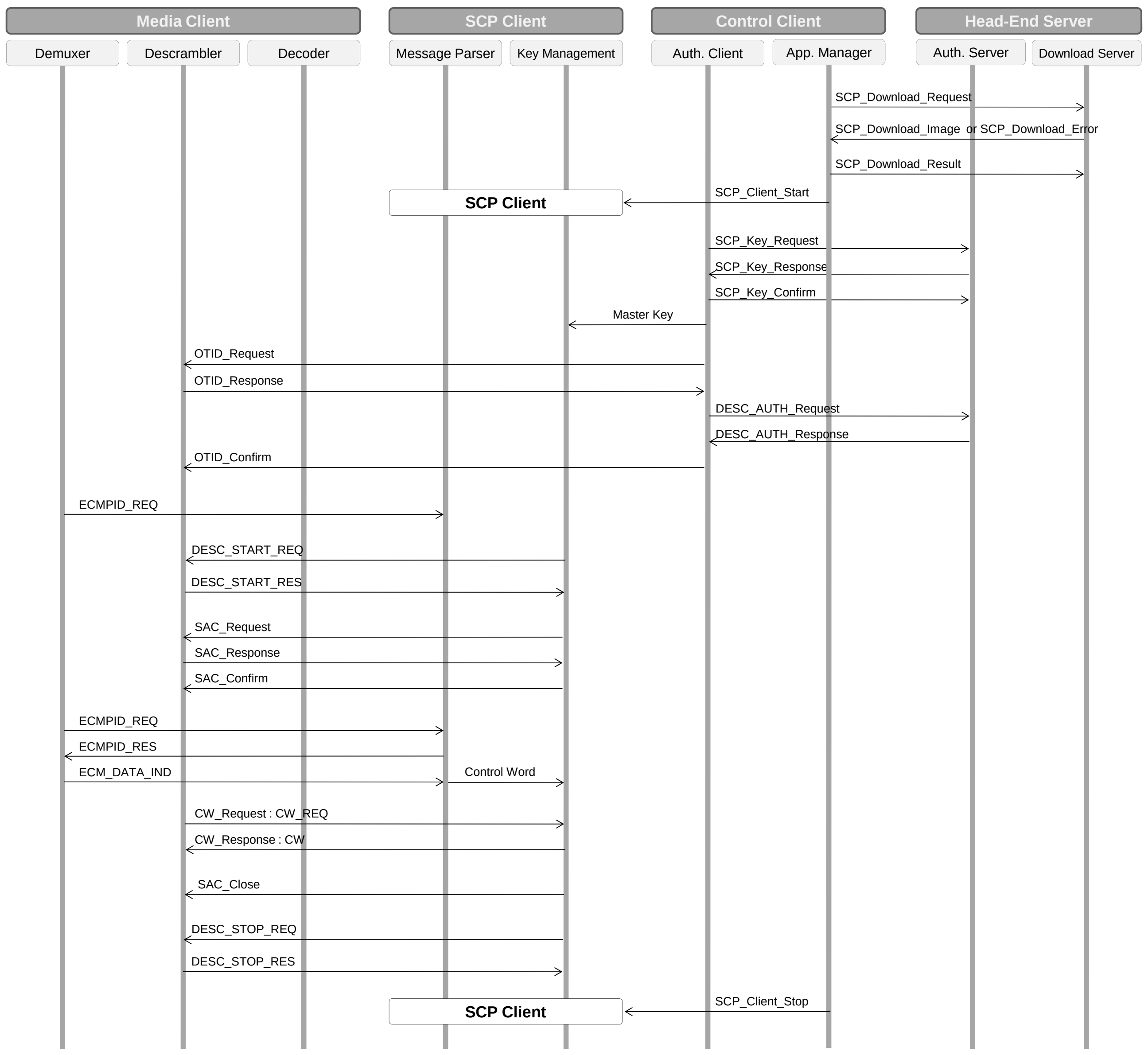


Figure 6 – Message flow of ASS

All messages are divided into three groups: the first group is the SCP control client, which includes SCP downloading and master key distribution; the second group is descrambler control that includes SAC (Secure Authenticated Channel) establishment; the third group is demuxer control.

7.2 SCP control client function

The SCP control client has two functions. One function is the secure downloading and binding of the SCP client to an embedded security module such as a trusted platform, a security kernel, or a hardware security chip. It also controls the life cycle of an SCP client, such as start, pause, stop, and terminate.

The other function is the key distribution protocol. This key is a master key that is used to generate a secure communication channel or authenticate the terminal device as one of the provisioning functions. In the case of the ASS, the main role of the control client is control and delivery of the decryption key to the ASS descrambler.

Therefore, this clause describes some general SCP client functions, SCP client downloading, master key distribution and service protection functions.

7.2.1 Download protocol for the SCP client

In this clause, the procedure for secure SCP client download is defined. SCP clients are developed by the security provider, but delivery is performed by a service provider; hence the need for protection of the SCP client code. A symmetric algorithm is used to ensure confidentiality. HMAC (hash-based message authentication code) is used for integrity.

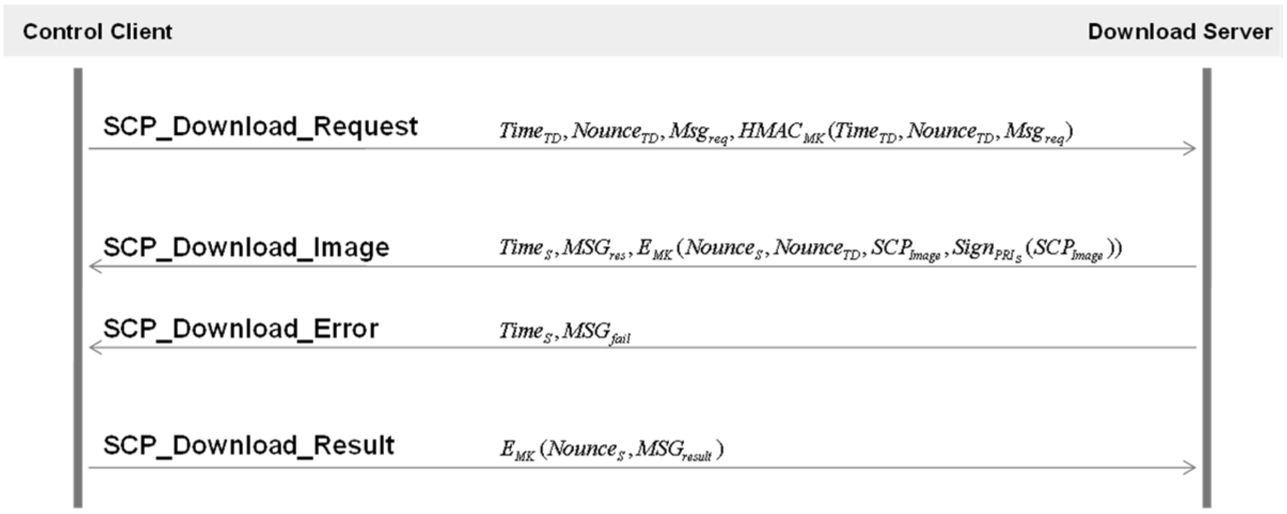

Control Client
Download Server
SCP_Download_Request
$Time_{TD}, Nounce_{TD}, Msg_{req}, HMAC_{MK}(Time_{TD}, Nounce_{TD}, Msg_{req})$
SCP_Download_Image
$Time_S, MSG_{res}, E_{MK}(Nounce_S, Nounce_{TD}, SCP_{Image}, Sign_{PRI_S}(SCP_{Image}))$
SCP_Download_Error
$Time_S, MSG_{fail}$
SCP_Download_Result
$E_{MK}(Nounce_S, MSG_{result})$

Figure 7 – SCP client download protocol

Figure 7 shows the SCP client download protocol, which consists of four message types. Basically, it has three steps like a 3-way handshake, with an error control step. In this protocol, we use four messages: CA_Download_Request, CA_Download_Image, CA_Download_Error and CA_Download_Result.

EMK(…) is an encryption of AES-128-CBC mode with master key MK. This key is delivered by the key distribution protocol in clause 7.2.2. The details are described in steps P1, P2-1, P2-2, and P3.

P1. SCP_Download_Request: $Time_{TD}$, $Nounce_{TD}$, MSG_{req}, $HMAC_{MK}$(...)

– $Time_{STD}$: Terminal device time to avoid a replay attack; it has 14 octets (year/month/day/hour/minute/seconds "YYYY/MM/DD/HH/MM/SS").

– $Nounce_{STD}$: As the random number to avoid a pattern of message data, it is used to generate different signature and encryption data with the same input data. To avoid conflict of hash function, we use 40-octet (320 bit) random number.

– MSG_{req}: This is a download request message. The following item, Request_Message, is not fixed for authentication. This message usually delivers information on the terminal device or subscribers such as product model, serial number, and subscriber id. This message has the following items:

{	
Message_type	2*8
Message_length	4*8
Request_Message	Variable
}	

– *HMAC$_{MK}$*(...): To guarantee the integrity of download requests, HMAC of a request message is added to the original message. In this message, HMAC is [b-IETF RFC 2104]. Figure 8 shows the details of operation.

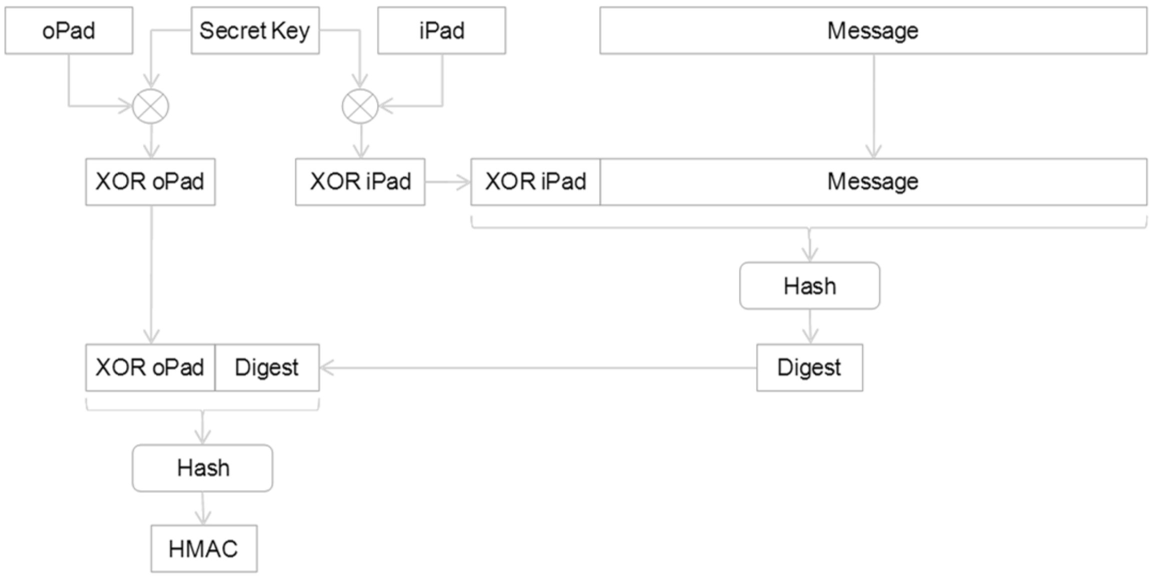


Figure 8 – HMAC operation

P2-1. SCP_Download_Image: Time$_S$, MSG$_{res}$, E$_{MK}$, (Nounce$_S$, Nounce$_{TD}$, SCP$_{Client}$, Sign$_{PRIS}$ (SCP$_{client}$))

– *Time$_S$*: To protect a terminal device from a malicious attacker, server-side time information is added to the message. The time information has 14 octets ("YYYY/MM/DD/HH/MM/SS").

– *MSG$_{res}$*:The response message of CA_Download_Request has the following items:

```
{
Message_type        2*8
Message_length                                                          4*8
Response_Message                                                          Variable
}
```

– *E$_{MK}$* (*Nounce$_S$*, *Nounce$_{TD}$*, *SCP$_{Client}$*, *Sign$_{PRIS}$* (*SCP$_{client}$*)): For secure download, the SCP client (*SCP$_{client}$*), service-side nounce *Nounce$_S$*, terminal nounce *Nounce$_{TD}$*, and signature of the SCP client are encrypted by the master key (MK). Before the SCP client distribution, the registered SCP client must be verified.

P2-2. SCP_Download_Error: *Time$_s$*, *MSG$_{fail}$*

– *Time$_s$*: To protect a terminal device from malicious attackers, server-side time information is added to the message. The time information has 14 octets ("YYYY/MM/DD/HH/MM/SS").

– MSG_{fail}: The error response message of CA_Download_Request has the following items (in case of error, the system has no action):

```
{
Message_type     2*8
Message_length                                                     4*8
Error_code                                                         4*8
Error_Message                                                    Variable
}
```

P3. SCP_Download_Result: $E_{MK}(Nouce_S, MSG_{result})$

– $E_{MK}(Nouce_S, MSG_{result})$: After downloading, the terminal device sends a confirm message. If there are abnormal program stop or server-side errors, the P2 message CA_Download_Error is sent; otherwise, the terminal device sends a result message (MSG_{result}) to the head-end server.

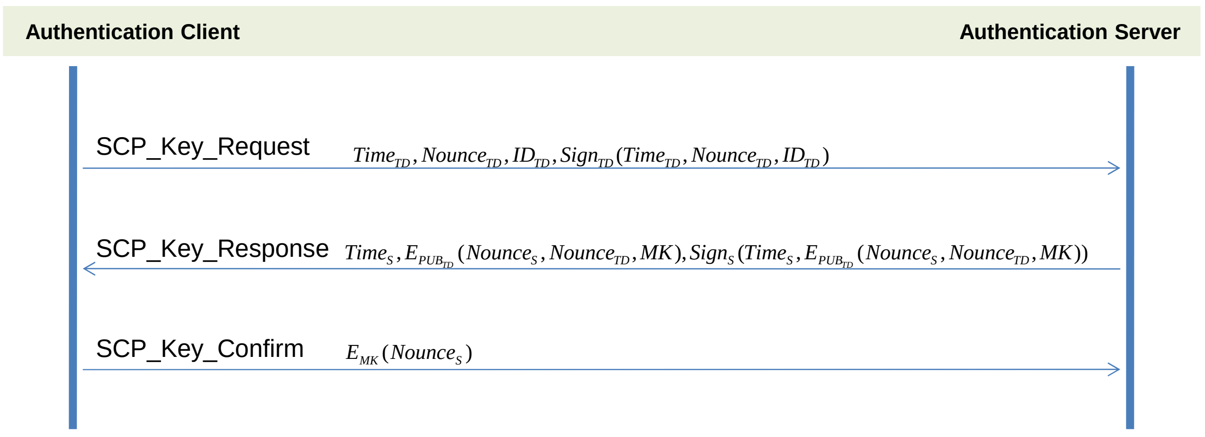
```
{
Message_type     2*8
Message_length                                                     4*8
Message_Code                                                     4*8
Result_Message                                                   Variable
}
```

7.2.2 Master key distribution protocol

Key distribution protocol is one of the provisioning functions. In the phase of provisioning, the terminal device is authenticated, and the master key is distributed. In this clause, we provide a key distribution protocol, and this key will be used in the authentication of an ASS descrambler.

This key distribution protocol will be activated when the system is booting up or in case of failure caused by an incurred or an invalid master key. As the basement key for all secure communications, this key is called the master key (MK), and its lifetime is one day.

The key distribution protocol has three phases and three messages: SCP_Key_Request, SCP_Key_Response, and SCP_Key_Confirm.



Authentication Client
Authentication Server
SCP_Key_Request
$Time_{TD}, Nounce_{TD}, ID_{TD}, Sign_{TD}(Time_{TD}, Nounce_{TD}, ID_{TD})$
SCP_Key_Response
$Time_S, E_{PUB_{TD}}(Nounce_S, Nounce_{TD}, MK), Sign_S(Time_S, E_{PUB_{TD}}(Nounce_S, Nounce_{TD}, MK))$
SCP_Key_Confirm
$E_{MK}(Nounce_S)$

Figure 9 – Key distribution protocol

Figure 9 shows the key distribution protocol. It consists of Request, Response and Confirm. The message format and its explanation are as follows:

P1. SCP_Key_Request: $Time_{TD}, Nounce_{TD}, ID_{TD}, Sign_{TD}(Time_{TD}, Nounce_{TD}, ID_{TD})$

– $Time_{TD}$: Time information to avoid a replay attack; it has 14 octets (year/month/day/hour/minute/seconds "YYYY/MM/DD/HH/MM/SS").

– $Nounce_{TD}$: As the random number to avoid a pattern of message data, it is used to generate different signature and encryption data with the same input data. To avoid conflict of hash function, we use 40-octet (320 bit) random number as the minimum size for collision-free hash.

– ID_{TD}: As the identifier of a terminal device, it is the unique number or string to detect a clone device. The ID management mechanism will be applied by the manufacturer of the terminal device. (16-octet string)

– $Sign_{TD}(Time_{TD}, Nounce_{TD}, ID_{TD})$: To enable the non-repudiation of the key request message, we add a digital signature with the private key of the terminal device. The key request message $Time_{TD}, Nounce_{TD}, ID_{TD}$ is hashed with SHA-1 as in Figure 10. The result value – 160-bit message digest – is encrypted with the private key of a terminal device to generate a digital signature.

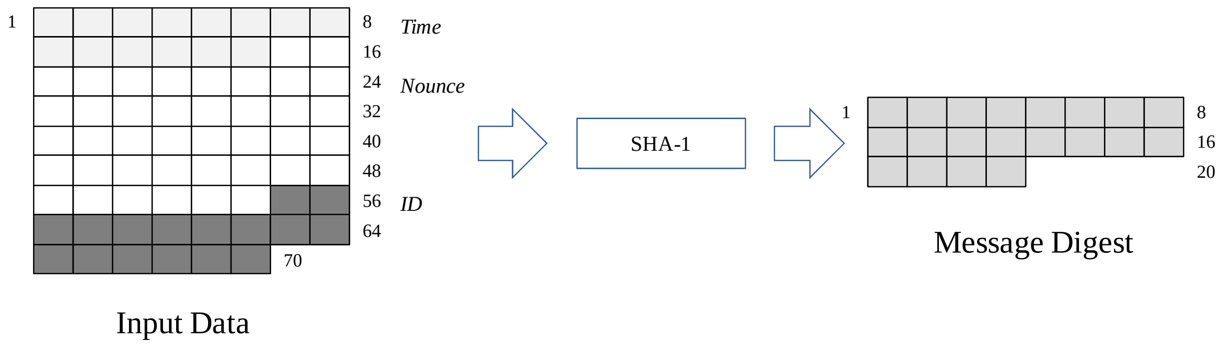


Figure 10 – A message digest for the signature $Sign_{TD}(Time_{TD}, Nounce_{TD}, ID_{TD})$

P2. SCP_Key_Response: $Time_S, E_{PUB_{TD}}(Nounce_S, Nounce_{TD}, MK), Sign_S(...)$

– $Time_S$: Time information to avoid a replay attack of the malicious server; this response message is generated by the server system, and a malicious server can also use this message again later, i.e., a replay attack. Therefore, terminal devices need to identify that the message is valid or not by checking the time information. The terminal device receives this response message and takes action. Note, however, that a malicious server can also send an invalid message to cause failure in a terminal device. It has a 14-octet string ("YYYY/MM/DD/HH/MM/SS").

– $E_{PUB_{TD}}(Nounce_S, Nounce_{TD}, MK)$: In the response message, the request nounce ($Nounce_{TD}$), server-generated nounce ($Nounce_S$), and new distribution master key (MK) are encrypted by the public key of the terminal device.

– $Sign_S(...)$: The part $E_{PUB_{TD}}(Nounce_S, Nounce_{TD}, MK)$ is encrypted by the public key of the terminal device. Therefore, anyone can generate it with a fraud master key (MK). To guarantee message integrity and originality, we add a digital signature. The server-side nounce ($Nounce_S$) value also guarantees message originality.

P3. SCP_Key_Confirm: $E_{MK}(Nounce_S)$

– $E_{MK}(Nounce_S)$: Before the activation of the distribution key, the terminal device sends a confirm message. It encrypts the server-generated nounce ($Nounce_S$) with the distributed master key (MK). In the phase of P1 and P2, the server nounce ($Nounce_S$) protects the terminal device from the man-in-the-middle attack.

7.2.3 SCP protocols and messages

Since SCP protocols are designed and implemented by individual security providers, this clause provides the general architecture and some messages including the Entitlement Control Message (ECM).

In the linear TV service, the key hierarchy has four layers: Device Certificate → Master Key (MK) → Distribution Keys → Control Word. An SCP client manages only the distribution keys that do or do not include a control word. The device certificate is related to terminal provisioning, whereas the master key (MK) is related to the key distribution protocol.

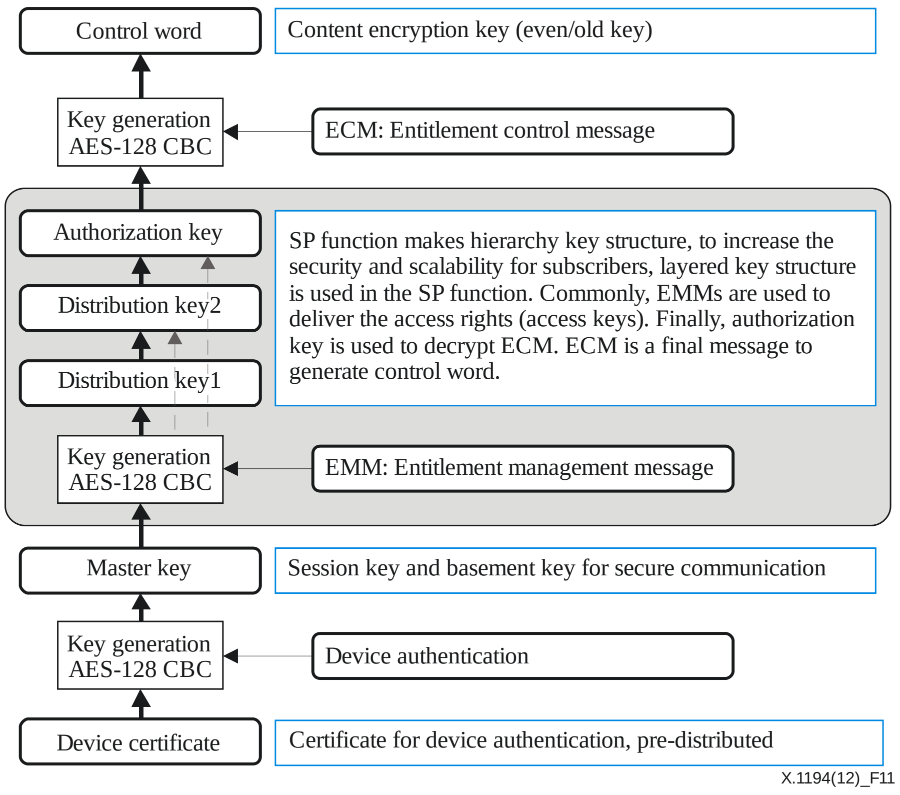

Control word
Content encryption key (even/old key)
Key generation AES-128 CBC
ECM: Entitlement control message
Authorization key
Distribution key2
Distribution key1
SP function makes hierarchy key structure, to increase the security and scalability for subscribers, layered key structure is used in the SP function. Commonly, EMMs are used to deliver the access rights (access keys). Finally, authorization key is used to decrypt ECM. ECM is a final message to generate control word.
Key generation AES-128 CBC
EMM: Entitlement management message
Master key
Session key and basement key for secure communication
Key generation AES-128 CBC
Device authentication
Device certificate
Certificate for device authentication, pre-distributed
X.1194(12)_F11

Figure 11 – Key hierarchy of an SCP system

Figure 11 shows the key hierarchy of an SCP system. In linear TV, there are several kinds of program packages and group-base services available. Therefore, a service protection mechanism requires subscriber grouping and its control. Those functions are usually performed by the Entitlement Management Message (EMM), which is controlled by individual SCP providers.

The key hierarchy is Device Certificate→ Master Key → Distribution Keys → Authorization Key → Control Word. The action step is device authentication, EMM processing and ECM processing.

The first step is device authentication and making of the master key (MK). The second step is decoding the EMM and making distribution and authorization keys. The third step is decoding the ECM and making a Control Word, which is an 8- or 16-octet secret key.

The second step is performed by the SCP client, not by a system. Note, however, that the first and third steps are performed by the system. In some cases, the second step has two or three more internal layers depending on the subscriber size or number of program packages available. The reference action of the first and third steps is as follows:

– First step – authentication, master key: This is defined in clause 7.2.1 of the master key distribution protocol. The key distribution protocol includes the identification of the terminal device wherein the authentication issue involves the protection of the identifier, or how to use the identifier efficiently.

– Third step – content descrambler: Descramblers are generally designed and implemented by hardware logic. In fact, the descrambler needs very fast processing power and has a simple working pattern, i.e., decoding of the ECM even if the ECM is extracted from the data stream by the demuxer and real-time decryption of the audio/video (A/V) stream. Recent IPTV terminals have embedded cryptographic modules within the main board for the descrambler.

- ECM

The Entitlement Control Message (ECM) is delivered with the media stream to reduce the number of messages, indicating the access rights of the media stream. The service protection function encrypts the A/V stream with a symmetric key. The key is very weak because it uses 1.5 gigabyte an hour. The key should be changed by periods to reduce the number of encryptions used. In cryptography, the increase in encryption number also increases the attack possibility of the symmetric key. Currently, all subscriber keys have an update period of 10 seconds; hence the need for simpler, more efficient mechanism than peer-to-peer key distribution. In the broadcasting system, access control of the service is performed by EMM, which defines the subscriber groups and broadcasts the entitlement message to the group member to distribute an update key. The ECM data format has the following items:

```
{
ECM_Version                                 1*8
ECM_Length                                  2*8
Scrambling_Algorithm                        1*8
Scrambling_Mode                             1*8
E(AK, Odd_CW)                               16*8
E(AK, Even_CW)                              16*8
Control_Data                                3*8
HMAC(AK, …)                                         20*8
}
```

In MPEG-2 TS, PID of ECM is defined in PMT, and real ECM data is extracted from the MPEG stream with the specified ECM PID. Information on ECM is located in the program map table (PMT) instead of the conditional access table (CAT) because the PMT indicates each stream program Identifier (PID) as well as whether it is scrambled or not. However, the system does not use the ECM data without EMM and CAT. EMM and CAT are defined by the SCP providers, and they include the group control logic.

- MPEG-2 TS level key synchronization

MPEG-2 TS delivers several channels together in a single stream, which includes audio, video, and data stream. In the functional aspects of security, MPEG-2 TS provides the control logic of key synchronization. It is a bit – transport_scrambling_control – in an MPEG-2 TS packet as in Figure 11.

Scrambling has two types. One is TS-level scrambling wherein the scrambling server encrypts only audio and video stream except data stream. The other one is PES-level scrambling wherein the scrambling server encrypts the payload of the PES packet instead of the TS packet; therefore, every data field should be finished within one packetized elementary stream (PES) packet. In this Recommendation, we use transport stream (TS)-level scrambling; the data stream including PAT and CAT is not encrypted.

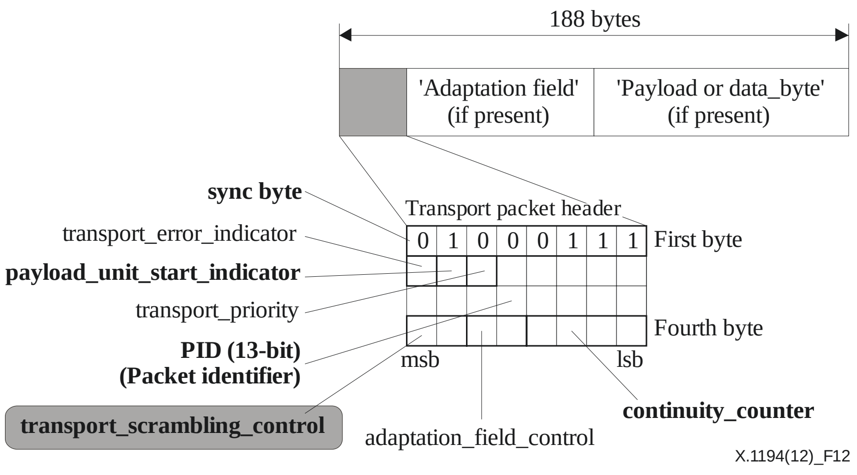

188 bytes
'Adaptation field' (if present)
'Payload or data_byte' (if present)
Transport packet header
0 1 0 0 0 1 1 1
First byte
Fourth byte
msb
lsb
sync byte
transport_error_indicator
payload_unit_start_indicator
transport_priority
PID (13-bit) (Packet identifier)
transport_scrambling_control
adaptation_field_control
continuity_counter
X.1194(12)_F12

Figure 12 – Transport packet header [b-ISO/IEC 13818-1]

In MPEG-2 TS streams, the transport_scrambling_control bit is placed in every TS packet. It is the 24th and 25th bits of the transport packet header as shown in Figure 12. These 2-bits have four states. In ISO/IEC 13818-1, "00" is "not scrambled"; others are "User-defined". Here, we define transport_scrambling_control as in Table 1.

Table 1 – Transport_scrambling_control

value	*Means*	*etc.*
00	No scrambled (Clear)	
01	Reserved	
10	Scrambled with Even Key	
11	Scrambled with Odd Key	

The table defines "10" as scrambled with even key and "11" as scrambled with odd key. This means that there is a switching condition. The time of data switch – from "11" to "10" or from "10" to "11" – indicates that the encryption key is updated.

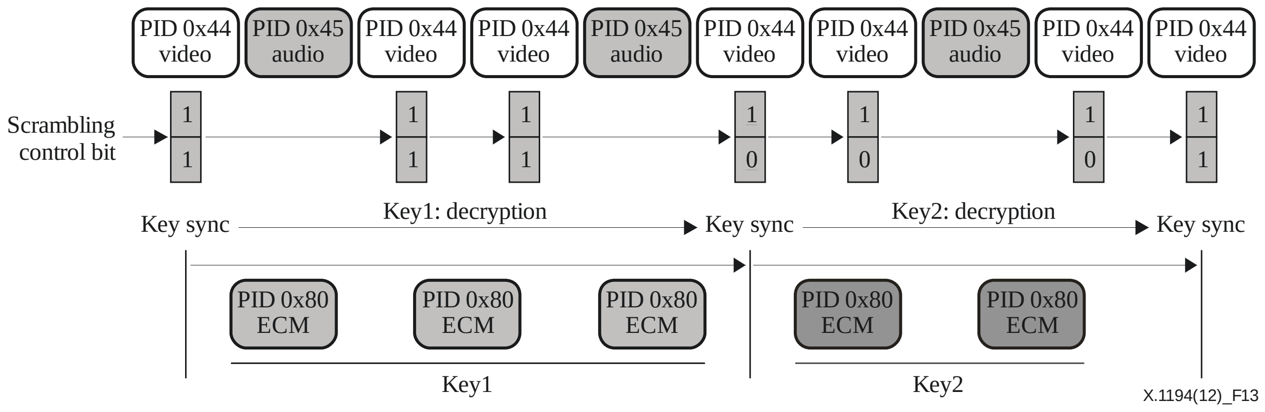


Figure 13 – Control word synchronization

Figure 13 shows how to update an encryption key. There are three streams: PID 0x80 is ECM stream, PID 0x44 is video, and PID 0x45 is audio stream. To recover a key error and to verify the current key, the ECM stream includes the current key and the next key. In Figure 13, the first packet has "11" and uses key1. The sixth packet has "10", and key2 is applied after the sixth packet.

8 Authentication functions for descrambling algorithm selection

In descrambling algorithm selection, the descrambler should be authenticated by the descrambler authentication server. In this clause, we define the ASS descrambler authentication function to identify the ASS descrambler, including third-party ASS descrambler. The third-party descrambler means that every descrambler can be replaced with a new one.

8.1 ASS descrambler authentication function

Descrambler authentication is performed during booting up time; the related components are descrambler, control client, and authentication server. In this Recommendation, we consider the embedded descrambler and third-party descrambler. The authentication of the descrambler is performed by the authentication logic of the control client.

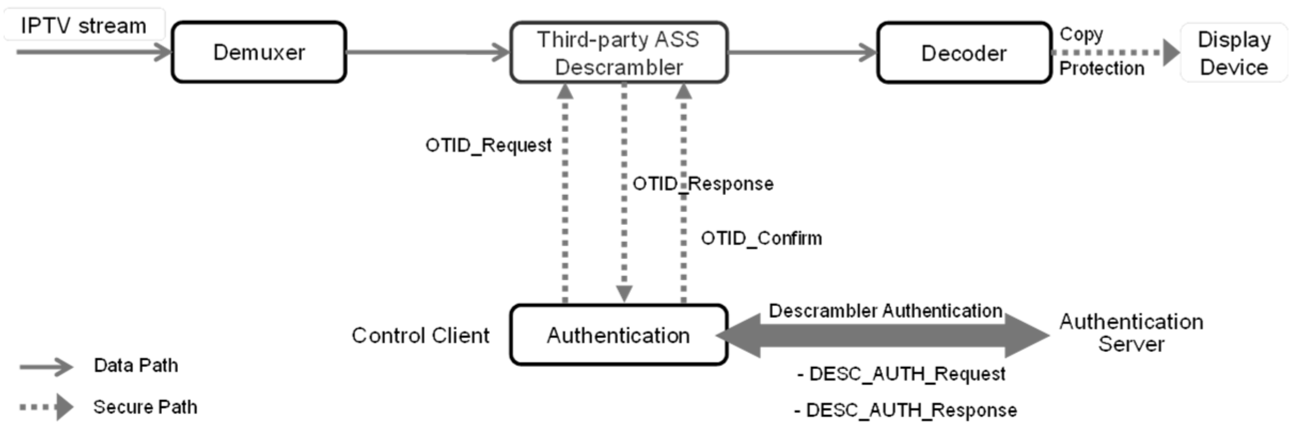


Figure 14 – Authentication procedure for descrambler

First, the authentication logic sends an OTID_Request message to the ASS descrambler; the ASS descrambler then generates a one-time ID for authentication. OTID is based on the time provided from the authentication logic in the control client.

Second, the ASS descrambler returns a message OTID_Response, which includes OTID for authentication. The authentication logic in the control client then requests an authentication procedure to the authentication server for the embedded descrambler and the third-party descrambler.

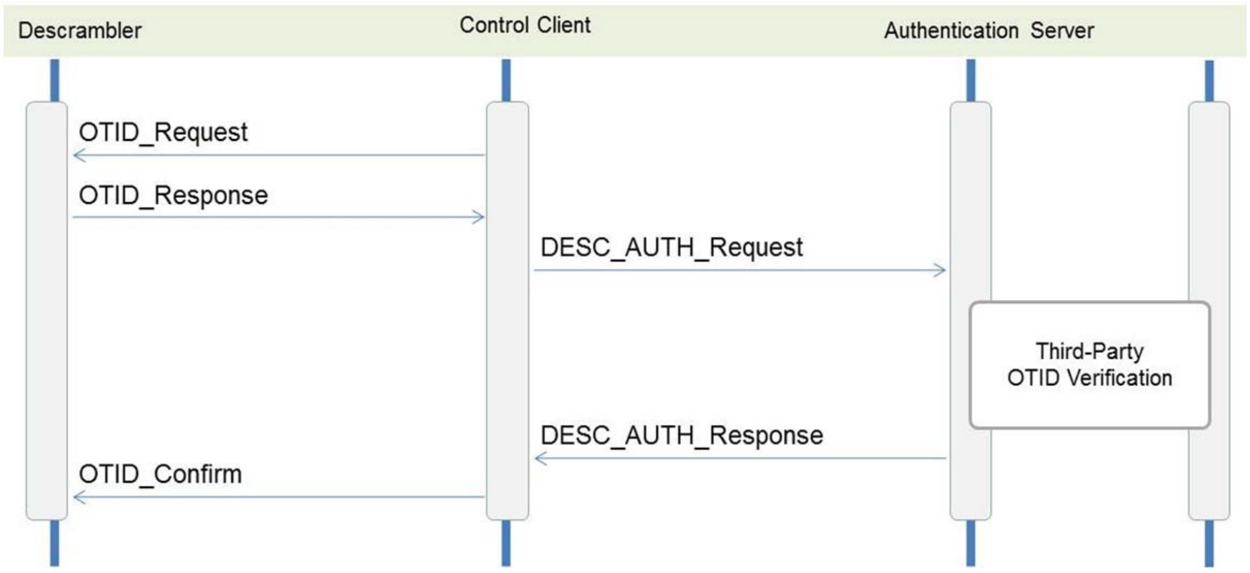


Figure 15 – Detailed authentication protocol for descrambler

Descrambler authentication is based on OTID of the descrambler; it is a dynamic ID for the one-time authentication of the descrambler. Initially, OTID is assigned to each descrambler. After authentication, ID is changed every time.

OTID is provided by the third-party verification server – which is also assigned the initial OTID – to the descrambler. In Figure 15, Third-Party OTID Verification means that the OTID assigner and the verification server have the same operation modules or use the same algorithm.

In the ASS descrambler authentication function, the descrambler should provide an identity for authentication; note, however, that the identity can be subject to eavesdropping by the control client. Thus, the identity should be made by one-time concepts.

In this clause, we provide a concept of OTID as well as how to use it in this Recommendation; the applied system can make various versions for OTID, but the basic operation should be aligned with the following procedures:

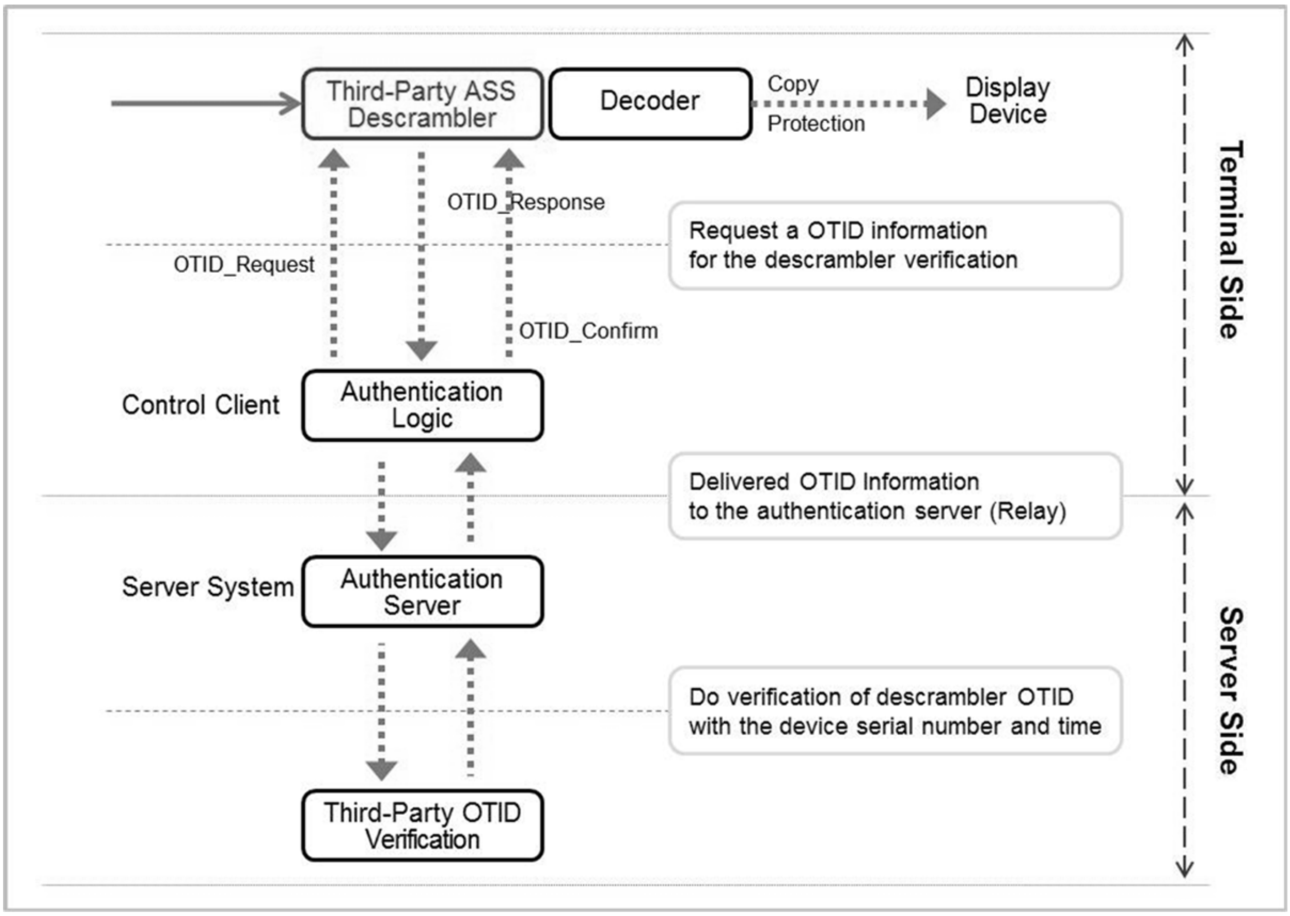

Third-Party ASS Descrambler
Decoder
Copy Protection
Display Device
OTID_Response
OTID_Request
OTID_Confirm
Request a OTID information for the descrambler verification
Control Client
Authentication Logic
Delivered OTID Information to the authentication server (Relay)
Server System
Authentication Server
Do verification of descrambler OTID with the device serial number and time
Third-Party OTID Verification
Terminal Side
Server Side

Figure 16 – OTID and its verification procedures

In Figure 16, there are four parties involved, and there are different protocols between Descrambler and Authentication Login in the Control Client, Authentication Logic and Authentication Server, and Authentication Server and Third-Party OTID Verification (server).

At first, the descrambler generates its own OTID for the verification of originality. The protocol has three messages, and the control client (including authentication logic) can be a malicious one.

• OTID request/confirm phase: 3-way handshake for OTID exchange

P1. OTID_Request:

$$Time_{CC}, Nounce_{CC}, MSG_{req}$$

– $Time_{CC}$: time information of control client has a 14-octet string format ("YYYY/MM/DD/HH/MM/SS").

– $Nounce_{CC}$: Random number is 40 octets (320bit).

– MSG_{req}: Request message is 8-octet string ("_MSG_REQ").

P2. OTID_Response:

$$Time_{DESC}, OTID, K_{pri}^{OTID}(Serial\ Number), Hash(S)$$
$$S = Time_{DESC}, Time_{CC}, Nounce_{CC}, OTID_{Initial}, K_{pri}^{OTID}(Serial\ Number)$$

– $Time_{DESC}$: time information of descrambler has a 14 octet string format ("YYYY/MM/DD/HH/MM/SS").

– $OTID$: This is a one-time ID generated by the descrambler. The details of the generation logic depend on the security provider's policy. It is beyond the scope of this Recommendation, but the size of $OTID$ is fixed at 16 octets.

– $K_{pri}^{OTID}(Serial\ Number)$: This is a fixed value of the descrambler ID; the serial number is encrypted by the private key of the $OTID$ server and is pre-generated and installed in the factory.

– $Hash(Time_{DESC}, Time_{CC}, Nounce_{CC}, OTID_{Initial}, K_{pri}^{OTID}(Serial\ Number))$: This is a hash value of $Time$, $Nounce$, $OTID_{Initial}$ and $K_{pri}^{OTID}(Serial\ Number)$ and used to prevent a time-replay attack.

* $Time_{DESC}$ is used for the prevention of replay attack of the control client. $Time_{CC}$ and $Nounce_{CC}$ are used to prevent the replay attack of the descrambler.

P3. OTID_Confirm:

$$\boxed{Nouce_{CC}, MSG_{confirm}}$$

– $Nounce_{CC}$: random number with 40 octets (320bit).

– $MSG_{confirm}$: Confirm message is 8-octet string ("_MSG_CON").

The authentication server does not have a role in the verification of the descrambler but can check whether the terminal device has been requested to verify its originality or not. In other words, the core logic of authentication is a duty of the OTID server or OTID provider, but flow control and status monitoring is a role of the authentication server.

Note, however, that the authentication logic encrypts all required information to the authentication server to relay it to the OTID verification server.

• OTID replay phase: control client (authentication logic) and authentication server

P1. DESC_AUTH_Request

$$\boxed{\begin{array}{l} E_{MK}(Time_{CC}, Nounce_{CC}, Time_{DESC}, OTID, K_{pri}^{OID}(Serial\ Numer), Hash(S), MSG_{req}) \\ S = Time_{DESC}, Time_{CC}, Nounce_{CC}, OTID_{Initial}, K_{pri}^{OID}(Serial\ Numer) \end{array}}$$

– $E_{MK}(...)$: This message is encrypted by master key MK, which is already shared between the control client and the authentication server. For details, see 7.2.2 Master key distribution protocol.

– $Nounce_{CC}$: This is the random number of the control client. When the control client attempts a replay attack, $Nounce_{CC}$ can be used to generate $OTID$ in the descrambler; therefore, it is also delivered to the authentication server.

– $OTID$: This is $OTID$ from the descrambler (16 octets).

– $K_{pri}^{OTID}(Serial\ Number)$: This is a fixed value of the descrambler ID. The serial number is encrypted by the private key of the $OTID$ server and is pre-generated and installed in the factory. It is used to identify the descrambler for verification.

– $Hash(S)$: This is a hash value to verify that the requester already knows the secret information $OTID_{Initial}$ with $Time_{DESC}, Time_{CC}, Nounce_{CC}$, and $K_{pri}^{OTID}(Serial\ Number)$.

– MSG_{req}: The request message is an 8-octet string ("AUTH_REQ") or can be used for the additional information of $OTID$ verification.

P2. DESC_AUTH_Response

$$\boxed{E_{MK}(Time_{CC}, MSG_{res})}$$

– $E_{MK}(...)$: This message is encrypted by master key MK, which is already shared between the control client and the authentication server.

– $Time_{CC}$: Time information of the control client has a 14-octet string format ("YYYY/MM/DD/HH/MM/SS").

– MSG_{res}: Confirm message is an 8-octet string ("RES_OK__" or "RES_FAIL"). It depends on the authentication result of the head-end system.

8.2 Generation and verification procedures for one-time ID

With one-time ID (OTID), the authentication procedures are different because some verification processes are added to the traditional authentication procedure. In traditional authentication, two parties are involved; in contrast, this has three parties for third-party verification as in Figure 16. The authentication logic and authentication server are considered as one party because they simply relay the required data for verification.

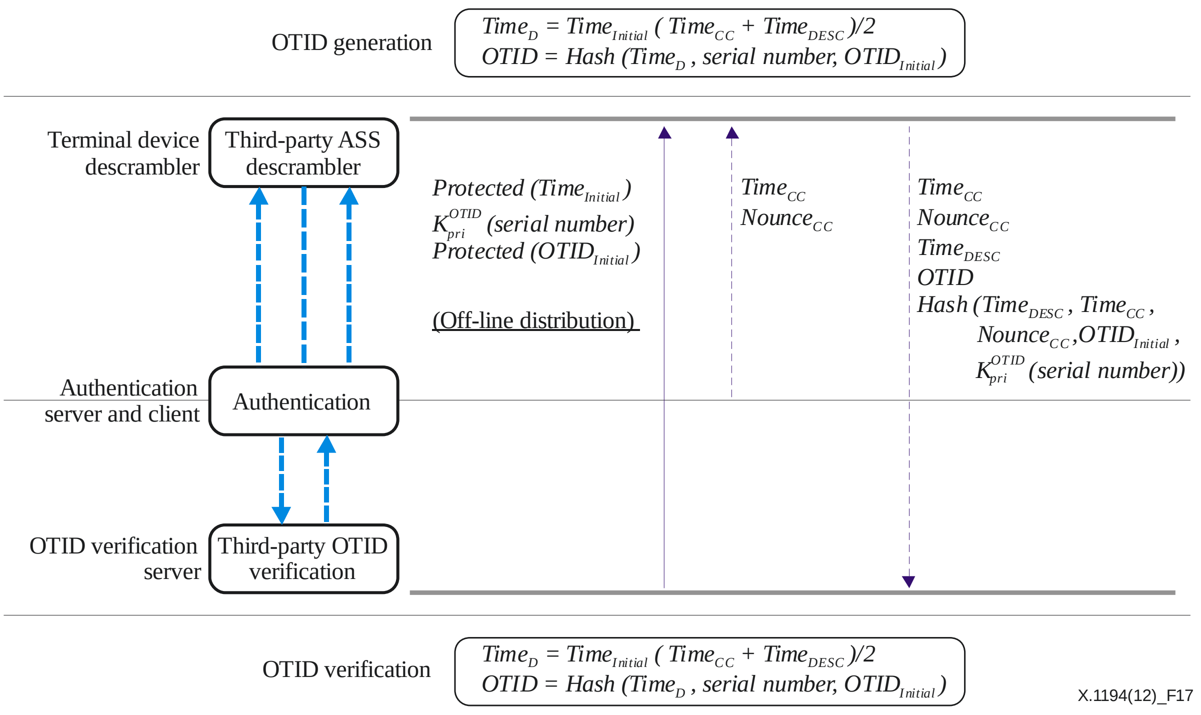

OTID generation
$Time_D = Time_{Initial}\ (Time_{CC} + Time_{DESC})/2$
$OTID = Hash\ (Time_D, serial\ number, OTID_{Initial})$
Terminal device descrambler
Third-party ASS descrambler
Authentication server and client
Authentication
OTID verification server
Third-party OTID verification
$Protected\ (Time_{Initial})$
$K_{pri}^{OTID}\ (serial\ number)$
$Protected\ (OTID_{Initial})$
(Off-line distribution)
$Time_{CC}$
$Nounce_{CC}$
$Time_{CC}$
$Nounce_{CC}$
$Time_{DESC}$
$OTID$
$Hash\ (Time_{DESC}, Time_{CC}, Nounce_{CC}, OTID_{Initial}, K_{pri}^{OTID}\ (serial\ number))$
OTID verification
$Time_D = Time_{Initial}\ (Time_{CC} + Time_{DESC})/2$
$OTID = Hash\ (Time_D, serial\ number, OTID_{Initial})$
X.1194(12)_F17

Figure 17 – OTID generation and verification

Figure 17 shows the OTID generation and verification protocol. In this protocol, secure information is $OTID_{Initial}$ and $Time_{Initial}$. The control client can capture all information except $OTID_{Initial}$ and $Time_{Initial}$. The secret information is delivered to the descrambler in the factory; this means that the secret information is delivered via offline distribution.

Every time the identity of the descrambler needs to be verified, a request is made (OTID_Request) to the control client with $Time_{CC}$ and $Nounce_{CC}$. The control client then sends the relevant data to the authentication server, which in turn sends it to the verification server. In this case, if the service provider accepts multiple OTID providers, then the relevant messages (OTID_Response, DESC_AUTH_Request) should include the verification server address.

$$\boxed{OTID = Hash(Time_{Initial} \quad (Time_{CC} + Time_{DESC})/2, Serial\ Number, OTID_{Initial})}$$

OTID has a one-time concept; therefore, we used time difference and $OTID_{Initial}$ for the hash function. The time difference is $TimeD = Time_{Initial} - (Time_{CC} + Time_{DESC})/2$ instead of ($TimeD = Time_{DESC} - Time_{CC}$) because the variation is too small.

This illustrates an example of OTID generation and verification procedures because it could be adding an industrial variation. For example, Figure 17 uses the time difference between the control client and initial time, but we can change it with the serial counter, a general One-time Password concept.

9 Control functions for descrambling algorithm selection

The descrambler is located in the secure area of the terminal device, and access is highly restricted. In this clause, we define the control function of the descrambler and the demuxer together, because the demuxer is very strongly coupled with the descrambler's operation.

9.1 ASS descrambler control functions

The media client consists of demuxer, descrambler and decoder. Its main functions are implemented by the manufacturers of the terminal device. To guarantee a descrambler's security, we should authenticate the descrambler or the entire media client. For convenience, we only authenticate the descrambler in this Recommendation.

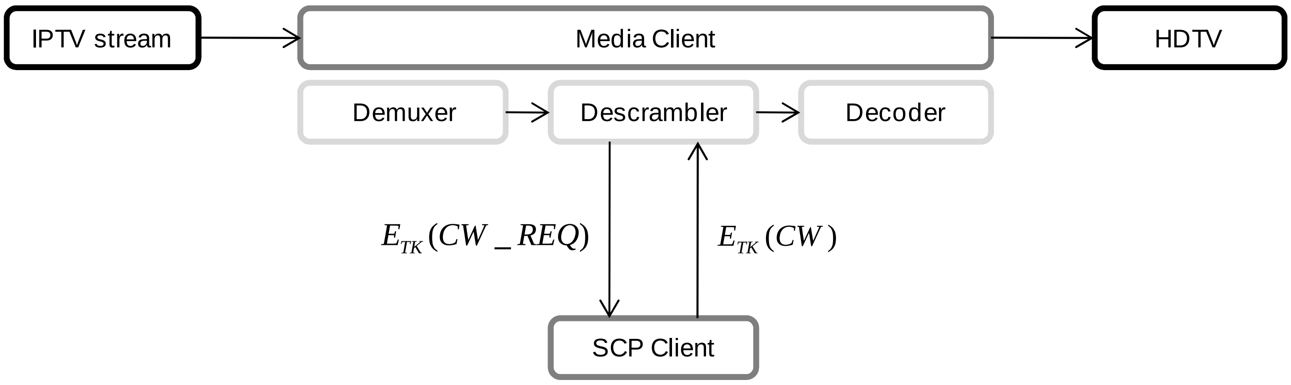


Figure 18 – Relation between descrambler and SCP client

Figure 18 shows the relation between the descrambler and the SCP client. All communication messages are encrypted by the terminal key (TK) (terminal's SAC key). The descrambler and SCP client share the TK, which is pre-distributed before the system starts. Basically, the SCP client and descrambler use only CW_REQ and CW in running states.

The descrambler's role is to ① take encrypted contents from the IPTV stream, ② decrypt it, and ③ send clean data to the decoder. The SCP client decodes the system information and generates proper keys. The periodically generated key is sent to the descrambler if there is a need to update a decryption key.

An additional role of the descrambler is the prevention of illegal export between the decoder and the descrambler. Therefore, the decoder and the descrambler have to be coupled tightly.

The secure authenticated channel (SAC) protocol has three steps as described in Figure 19; the first step is to establish a secure authenticated channel between the descrambler and the SCP client. The second step is to deliver the encryption key, and the third step is closing.

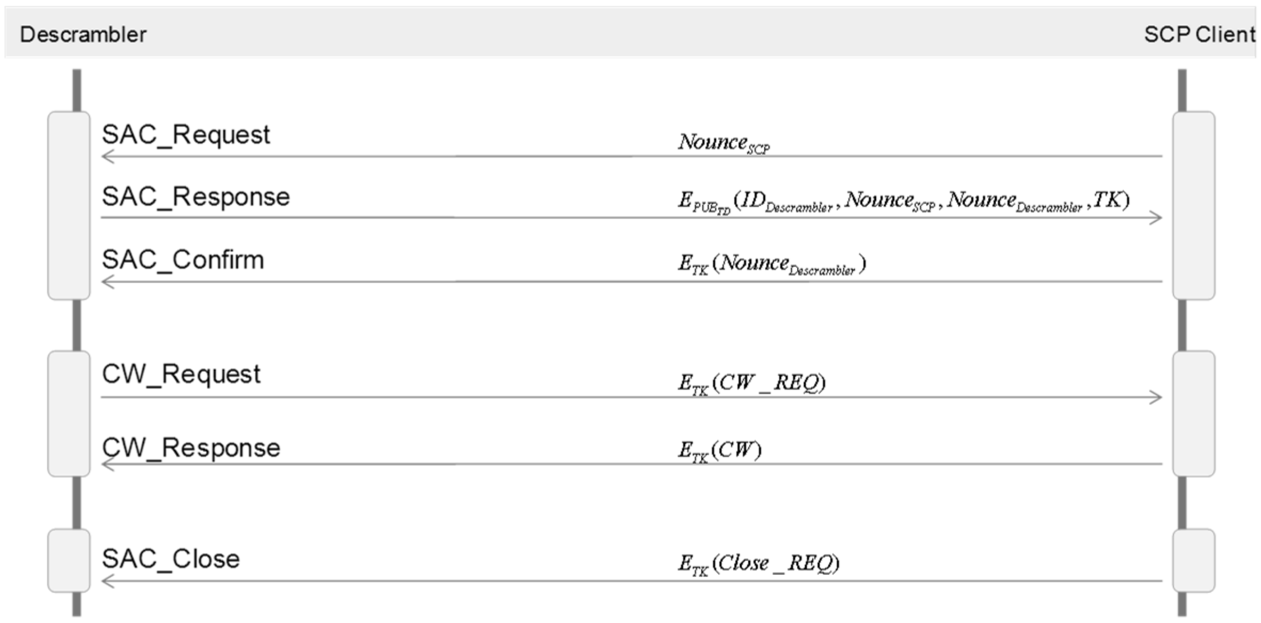


Figure 19 – SAC protocol

After booting up, the descrambler is ready to receive the SAC_Request message. When the SCP client starts, it sends SAC_Request to the descrambler. If the descrambler is not ready at that time, the descrambler is initialized and should be in running state. After SAC_Request/SAC_Response/SAC_Confirm, the descrambler sends a CW_Request message to the SCP client upon receiving scrambled content, and the SCP client sends a CW_Response message including the generated CW (decryption key). The details of $E_{TK}(CW\_REQ)$ and $E_{TK}(CW)$ are illustrated in Annex A.1 Content key delivery messages.

In SAC_Request, the SCP client sends its nounce value, and the descrambler sends the SAC_Response message encrypted with the public key of terminal device PUB_{TD}. The message contains the descrambler's ID (later, it is OTID), descrambler's nounce, and SCP client's nounce and TK. After receiving the descrambler's ID, the SCP client verifies it. Finally, the SCP client sends the SAC_Confirm message with the descrambler's nounce ($Nounce_{Descrambler}$), and it is encrypted by TK, $E_{TK}(Nounce_{Descrambler})$.

For systemic operation, the SCP client needs control of the descrambler. The control messages between the SCP client and the descrambler are defined in the next sub-clause. Control means management of the ASS descrambler's states.

9.1.1 Data format for ASS descrambler control

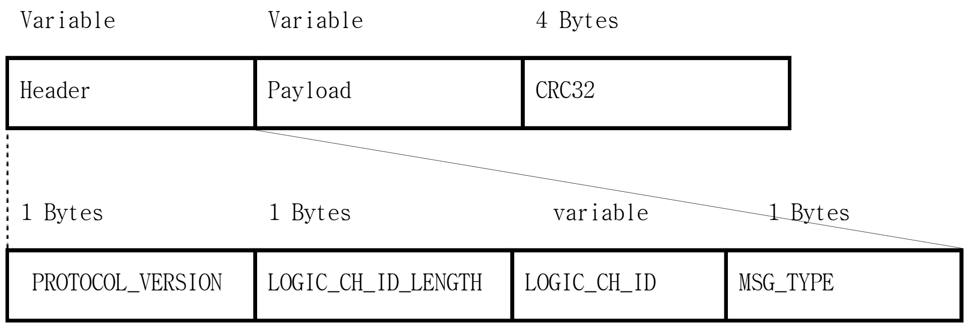


Figure 20 – Data format between descrambler and SCP client

The control message of the descrambler has a different message format from the SAC protocol, which defines how to make a secure channel and how to deliver key information from the SCP client to the descrambler. Therefore, the control message of the descrambler provides a control function of start and stop. The message has three parts: header, payload, and CRC32. The message structure is shown in Table 2.

Table 2 – Message format between descrambler and SCP client

Item	*Size(octets)*	*Description*
VERSION	1	Version
LOGIC_CH_ID_LENGTH	1	LOGIC_CH_ID data size
LOGIC_CH_ID	Variable	Logical Channel ID
MSG_TYPE	1	Message Type
PAYLOAD	Variable	Data
CRC	4	CRC32

9.1.2 Messages for the ASS descrambler control

The SCP client controls the ASS descrambler with a control message. After SAC connection, the SCP client sends a start message to the ASS descrambler. If the descrambler needs to be stopped, the SCP client sends a stop message. In case of descrambler error, the SCP client stops and restarts the descrambler. Functionally, the descrambler is not destroyed but simply stops the operation for the re-synchronization of the encryption key. Many error cases are caused by key synchronization problems that usually occur when the communication channel has delay or loss.

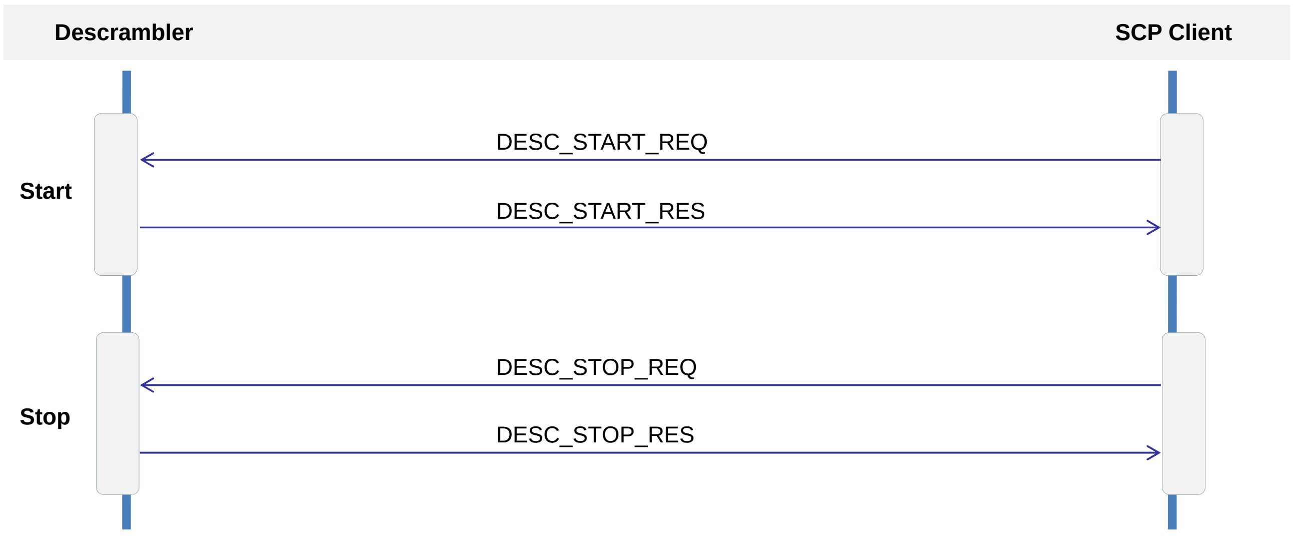


Figure 21 – ASS descrambler control protocol

The ASS descrambler control has two actions: start and stop. Additionally, the error message represents the reason for the error with DESC_ERROR_IND and UNDEFINED_ERROR in Table 3.

Table 3 – Message types between descrambler and SCP client

	Message Name	*Message Type*
ERROR	UNDEFINED_ERROR	0xd0
	DESC_ERROR_IND	0xa1
CONTROL	DESC_START_REQ	0xa2
	DESC_START_RES	0xa3
	DESC_STOP_REQ	0xa4
	DESC_STOP_RES	0xa5

UNDEFINED_ERROR and DESC_ERROR_IND messages have a payload field. Note, however, that other messages have no payload because DESC_START_REQ, DESC_START_RES, DESC_STOP_REQ, and DESC_STOP_RES are a simple signal to indicate what to do in the descrambler.

UNDEFINED_ERROR is not defined here and is used for the private analysis of error such as internal errors or hardware errors. Therefore, the message payload should include detailed messages to aid in understanding. The details of the DESC_ERROR_IND message are defined in Table 4. The LOGIC_CH_HD field is included to support MPEG-2 TS MMS (Multi-Mode Service).

Table 4 – DESC_ERROR_IND message

Item	*Description*
Template	DESC_ERROR_IND { size [bit] [type] PROTOCOL_VERSION 1*8 [INT] LOGIC_CH_ID_LENGTH 1*8 [INT] LOGIC_CH_ID variable[CHARSTRING] MSG_TYPE 1*8 [INT] PAYLOAD_SIZE 2*8 [INT] ERR_NUM 1*8 [INT] ERR_DESC_LENGTH 1*8 [INT] ERR_DESC variable[CHARSTRING] TIME_STAMP 4*8 [INT] CRC32 4*8 [INT] }
Constraints	PAYLOAD_SIZE: Octet size from ERR_NUM to TIME_STAMP ERR_NUM: Error type ERR_DESC_LENGTH: Octet size of ERR_DESC ERR_DESC: Error description TIME_STAMP: time
Effect	SCP client can receive a message, whenever a failure has occurred
Return	

9.2 ASS demuxer control functions

When the terminal device is booting up, the demuxer is also started automatically. The control message of the demuxer is only one "ECM PID" (Packet Identifier), which is included in "ECMPID_RES". Others are a request message to the SCP client. Upon receiving PMT, PAT and CAT, the demuxer sends "ECMPID_REQ" to the SCP client. The SCP client then extracts "ECM PID" from PMT, PAT, and CAT and sends back "ECM PID" to the demuxer, which then sets up the MPEG-2 TS section filter with the specified ECM_PID. When taking ECM data, the demuxer sends "ECM_DATA_IND" to the SCP client.

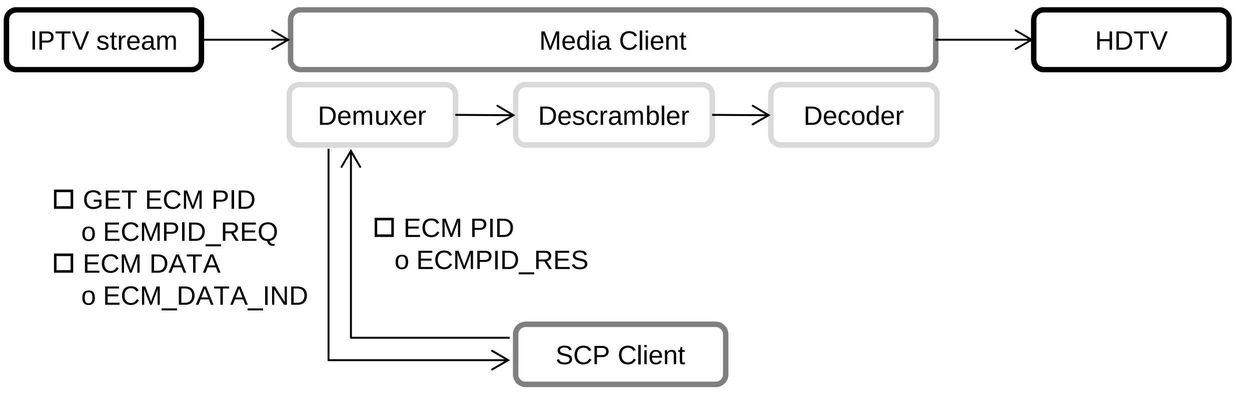


Figure 22 – Relation between demuxer and SCP client (message parser)

9.2.1 Data format for ASS demuxer control

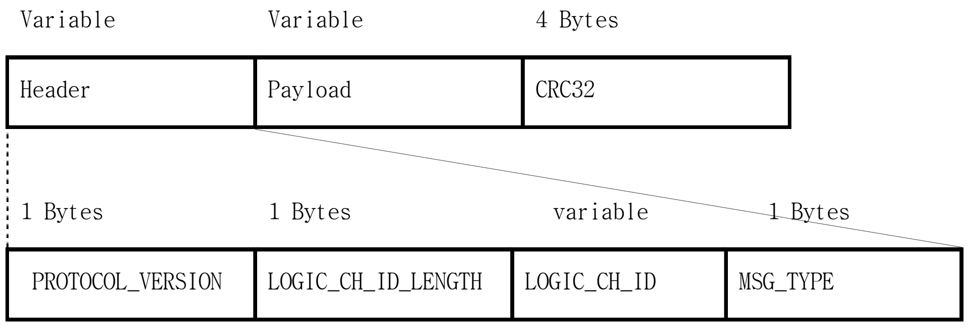


Figure 23 – Data format between the demuxer and the SCP client

The control message of the demuxer is related to EMM, ECM, CW, and error messages. This Recommendation includes ECM and CW but not EMM because it is under the control of the security provider. Therefore, DEMUX_GET_ECMPID_REQ, DEMUX_GET_ECMPID_RES, and DEMUX_ECM_DATA_IND are defined in this Recommendation but not REQUEST_EMM and GET_EMM. The sequence of the protocol is shown in Figure 24.

9.2.2 Messages for ASS demuxer control

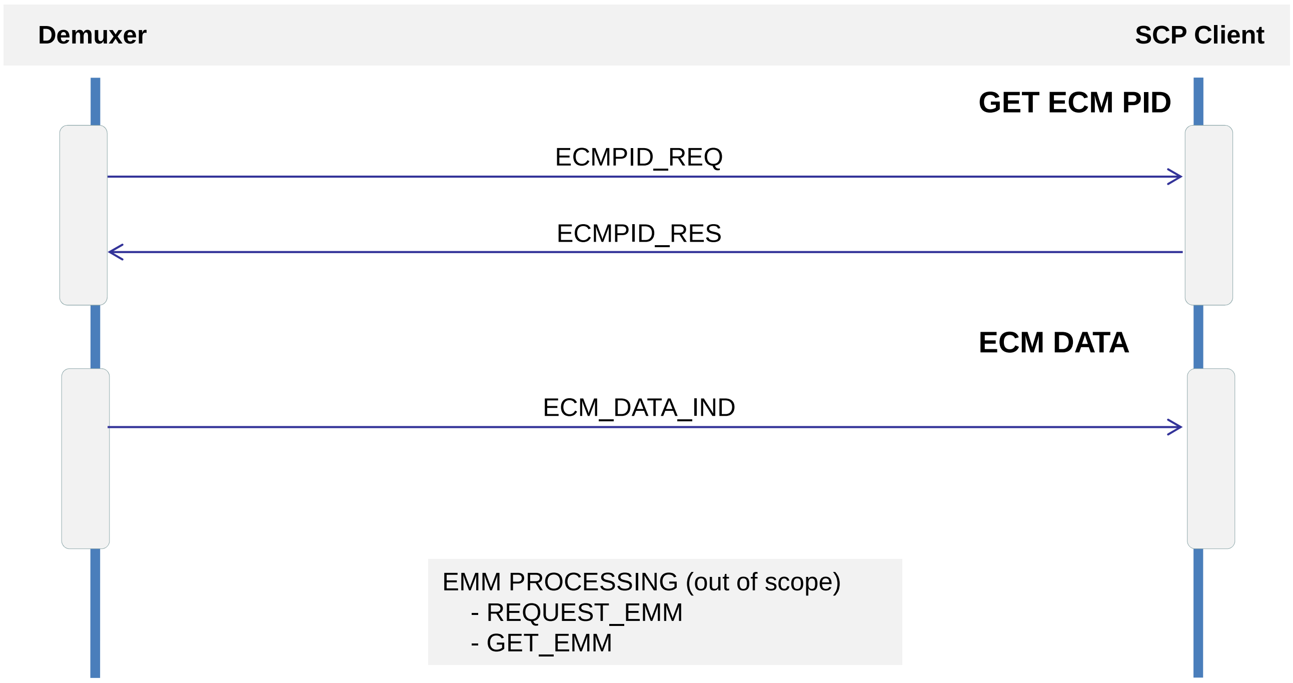


Figure 24 – ASS demuxer control protocol

First, the demuxer addresses a request for an ECM PID to the SCP client; the request message "ECMPID_REQ" includes CAT, PAT, and PMT. The SCP client decodes those tables and takes an ECM PID. The SCP client sends the ECM PID to the demuxer in the form of "ECMPID_RES". For fast connection, the SCP client can use a pre-defined PID instead of "ECMPID_REQ" filtering.

Table 5 – Demuxer control message

	Message Name	*Message Type*
ERROR	UNDEFINED_ERROR	0xd0
	DEMUX_ERROR_IND	0xc1
CONTROL	ECMPID_REQ	0xc2
	ECMPID_RES	0xc3
	ECM_DATA_IND	0xc4

Table 5 defines the demuxer control messages. Some messages are control messages such as ECMPID_REQ, ECMPID_RES, and ECM_DATA_IND. Other messages are related to errors such as DEMUX_ERROR_IND and UNDEFINED_ERROR.

- DEMUX_ERROR_IND

UNDEFINED_ERROR is not defined here and is used for the private analysis of error such as internal errors or hardware errors. Therefore, the message payload should include detailed messages to aid in understanding. The details of the DEMUX_ERROR_IND message are defined in Table 6. The LOGIC-CH_HD field is included to support MPEG-2 TS MMS (Multi-Mode Service).

Table 6 – DEMUX_ERROR_IND message

Item	*Description*
Template	DEMUX_ERROR_IND { size[bit] [type] PROTOCOL_VERSION 1*8 [INT] LOGIC_CH_ID_LENGTH 1*8 [INT] LOGIC_CH_ID variable[CHARSTRING] MSG_TYPE 1*8 [INT] PAYLOAD_SIZE 2*8 [INT] ERR_NUM 1*8 [INT] ERR_DEMUX_LENGTH 1*8 [INT] ERR_DEMUX variable[CHARSTRING] TIME_STAMP 4*8 [INT] CRC32 4*8 [INT] }
Constraints	PAYLOAD_SIZE: Octet size from ERR_NUM to TIME_STAMP ERR_NUM: Error type 0x00: Terminal device is not registered 0x01: Terminal device is not started 0x02: There is no ECM data with the request PID 0x03: There is no ECM PID with the request PMT and CAT ERR_DEMUX_LENGTH: Octet size of ERR_DESC ERR_DEMUX: Error description TIME_STAMP: time
Effect	SCP client can take this message, whenever there has been a failure
Return	

- ECMPID_REQ

The ECMPID_REQ message is defined in Table 7 ECMPID_REQ message. It includes system information for the SCP client, such as PMT and CAT. PMT is required to extract program information related to ECMs in PMT and EMMs in CAT.

Table 7 – ECMPID_REQ message

Item	*Description*
Template	ECMPID_REQ { size[bit] [type] PROTOCOL_VERSION 1*8 [INT] LOGIC_CH_ID_LENGTH 1*8 [INT] LOGIC_CH_ID variable[CHARSTRING] MSG_TYPE 1*8 [INT] PAYLOAD_SIZE 2*8 [INT] PMT_SIZE 1*8 [INT] PMT variable[CHARSTRING] CAT_SIZE 1*8 [INT] CAT variable[CHARSTRING] CRC32 4*8 [INT] }
Constraints	PAYLOAD_SIZE: Octet size from PMT_SIZE to CAT PMT_SIZE: Octet size of PMT PMT: MPEG-2 TS Program Map Table contains information about programs CAT_SIZE: Octet size of CAT CAT: MPEG-2 TS Conditional Access Table provides association with EMM stream.
Effect	SCP client can take this message, whenever contents are scrambled
Return	DEMUX_GET_ECMPID_RES DEMUX_ERROR_IND

EMM and ECM are found in the conditional access (CA) descriptor, which is placed in the private section of PMT and CAT. The detail field of the CA descriptor is defined in [b-ISO/IEC 13818-1], and the brief data set is illustrated in Figure 25.

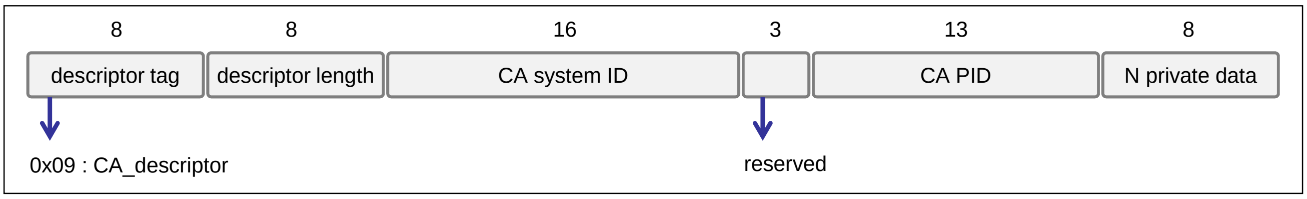


Figure 25 – CA descriptors in PMT and CAT [b-ISO/IEC 13818-1]

There are several descriptors in PMT and CAT, but the CA descriptor has a descriptor tag of 0x09. When a CA descriptor is found in the PMT, the CA_PID indicates packets containing program-related access control information such as ECMs. The CA descriptor is also found in CAT; CA_PID then points to packets containing system-wide and/or access control management information such as EMMs. Therefore, the SCP client takes ECM_PID and EMM_PID and makes the "ECMPID_RES" message.

- ECMPID_RES

The ECMPID_RES message is a response message to ECMPID_REQ; its detailed data format is defined in Table 8 ECMPID_RES message. Response messages are ERROR or ECM/EMM PID. In the case of ERROR, it is DEMUX_ERROR_IND message; otherwise, it is ECMPID_RES message. This message includes ECM PID and EMM PID.

Table 8 – ECMPID_RES message

Item	***Description***
Template	ECMPID_RES { size[bit] [type] PROTOCOL_VERSION 1*8 [INT] LOGIC_CH_ID_LENGTH 1*8 [INT] LOGIC_CH_ID variable[CHARSTRING] MSG_TYPE 1*8 [INT] PAYLOAD_SIZE 2*8 [INT] ECM_PID 2*8 [INT] EMM_PID 2*8 [INT] (optional) CRC32 4*8 [INT] }
Constraints	PAYLOAD_SIZE: Octet size from ECM_PID to EMM_PID ECM_PID: program ID for ECM EMM_PID: program ID for EMM
Effect	SCP client can take this message, whenever contents are scrambled
Return	DEMUX_GET_DATA_IND DEMUX_ERROR_IND

- DATA_IND

The DATA_IND message includes the ECM data defined in clause 7.2.3 service protection protocol and message.

Table 9 – DATA_IND message

Item	*Description*

Template	
	```
	DATA_IND
	{                                           size[bit] [type]
	        PROTOCOL_VERSION                    1*8     [INT]
	        LOGIC_CH_ID_LENGTH                  1*8     [INT]
	        LOGIC_CH_ID                         variable[CHARSTRING]
	        MSG_TYPE                            1*8     [INT]

	        PAYLOAD_SIZE                        2*8     [INT]
	        ECM                                 60*8    [INT]

	        CRC32                               4*8     [INT]
	}
	```

Constraints	PAYLOAD_SIZE: Octet size of ECM ECM: Entitlement Control Message
Effect	SCP client receives ECM data whenever demuxer extracts ECM data from MPEG-2 TS
Return	

Annex A

Descrambling algorithm and key delivery message

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

In the SCP system, content encryption is simple but the key delivery and related messages are very complex because it is updated with a time period. In this clause, the CW_REQ and CW message for updating the descrambling key is described. In clause 7.2.3, the ECM indicates which algorithm is used for contents descrambling.

A.1 Contents key delivery messages

In Figure 13, CW_Request and CW_Response messages are used for key delivery and its synchronization. It is performed by *transport_scrambling_control* in Figure 12 whose key is used for the scrambled packet. Note, however, that key synchronization focuses on which key will be used in the next time period. The time is the same as a life cycle of the encryption key. Usually, the key life cycle is 10 seconds. Therefore, the descrambler waits for a new key, which is delivered from the SCP client within the next 10 seconds.

The CW_Request message is used to indicate that the descrambler needs a new decryption key (control word). The descrambler needs a new decryption key in the following two cases:

1) The descrambler has just started; the descrambler does not have a decryption key.
2) The descrambler has an error in decryption; the current decryption is not valid.

Key synchronization messages have two actions: CW_REQ and CW. The additional message types between the descrambler and the SCP client are defined in Table A.1.

Table A.1 – Control word (CW) messages between descrambler and SCP client

Message Name	***Message Type***
CW_REQ	0xb0
CW	0xb1

• CW_REQ

The CW_Request message includes the encrypted CW_REQ, which is defined in Table A.2 below. CW_REQ provides a program ID and a random number of SCP.

Table A.2 – CW_REQ message

Item	*Description*

Template	
	```
	CW_REQ
	{                                   size [bit] [type]
	        PROTOCOL_VERSION            1*8     [INT]
	        LOGIC_CH_ID_LENGTH          1*8     [INT]
	        LOGIC_CH_ID                 variable[CHARSTRING]
	        MSG_TYPE                    1*8     [INT]

	        PAYLOAD_SIZE                2*8     [INT]
	        PROGRAM_ID                  2*8     [INT]
	        RANDOM_SCP                  40*8    [INT]

	        CRC32                       4*8     [INT]
	}
	```

Constraints	PAYLOAD_SIZE: Octet size from PROGRAM_ID to RANDOM_SCP PROGRAM_ID: program ID of target stream (16 bit) RANDOM_SCP: SCP random number in the SAC_Request message (320 bit)
Effect	SCP client can take a message whenever the descrambler needs a key
Return	CW

The CW_Request message is identified with $E_{TK}(CW\_REQ)$ and CW_REQ, with "ASS indicator" in Figure 4. Similarly, the message CW_Response is identified with $E_{TK}(CW)$ and CW, with "key" in Figure 4 and "control word" in Figure 11.

- CW

The CW_Response message includes the encrypted CW defined in Table A.3 below. The CW provides the current encryption key and the next encryption key when synchronization fails.

Table A.3 – CW message

Item	*Description*

Template	
	```
	CW
	{                                         size [bit] [type]
	        PROTOCOL_VERSION          1*8     [INT]
	        LOGIC_CH_ID_LENGTH        1*8     [INT]
	        LOGIC_CH_ID               variable[CHARSTRING]
	        MSG_TYPE                  1*8     [INT]

	        PAYLOAD_SIZE              2*8     [INT]
	        CW1_SIZE                  1*8     [INT]
	        CW1                       variable[CHARSTRING]
	        CW2_SIZE                  1*8     [INT]
	        CW2                       variable[CHARSTRING]

	        CRC32                     4*8     [INT]
	}
	```

Constraints	PAYLOAD_SIZE: Octet size of CW CW1_SIZE: size of CW1 CW1: current encryption key CW2_SIZE: size of CW2 CW2: the next encryption key
Effect	ASS descrambler takes two CW, current and next key.
Return	

The data CW1 and CW2 are identified with even/odd control word in MPEG-2 TS. If the *transport_scrambling_control* bit is "11", the odd key is CW1 and the even key is CW2; if it is "10", the even key is CW1 and the odd key is CW2.

A.2 ASS descrambling algorithm

In the ECM message, Scrambling_Algorithm is assigned 8bit. It is the default scrambling algorithm for content encryption. Table A.4 below shows the algorithm, mode, and size. It consists of well-known block cipher algorithms such as TDEA, AES, SEED [b-ISO/IEC 18033-3], DVB CSA/CSA3 [b-ETSI TS 101 699], and reserved algorithms, which are used for the new algorithm or local standard.

Table A.4 – ASS descrambling algorithms

Algorithm	*Mode*	*Size*	*Message Name*	*Message Type*
TDEA	ECB	168	3-DES ECB_168	0xa1
	CBC	168	3-DES CBC_168	0xa2
AES	ECB	128	AES_ECB_128	0xb1
		192	AES_ECB_192	0xb2
		256	AES_ECB_256	0xb3
	CBC	128	AES_CBC_128	0xb6
		192	AES_CBC_192	0xb7
		256	AES_CBC_256	0xb8
DVB	CSA	64	DVB-CSA	0xc1
	CSA3	–	DVB-CSA3	0xc6
SEED	ECB	128	SEED_ECB_128	0xd1
	CBC	128	SEED_CBC_128	0xd6
Reserved	ECB	128	LEA-ECB_128	0xe1
		256	LEA_ECB_256	0xe2
	CBC	128	LEA-CBC_128	0xe6
		256	LEA_CBC_256	0xe7

TDEA, AES, and DVB-CSA are well-used in the service protection area, whereas DVB-CSA3 and SEED are newly deployed in the hardware chip in broadcasting terminal devices. Table A.4 shows descrambling algorithms and does not preclude any other algorithms.

Appendix I

Descrambling-related standards

(This appendix does not form an integral part of this Recommendation.)

I.1 ATIS IIF default scrambling algorithm (IDSA)

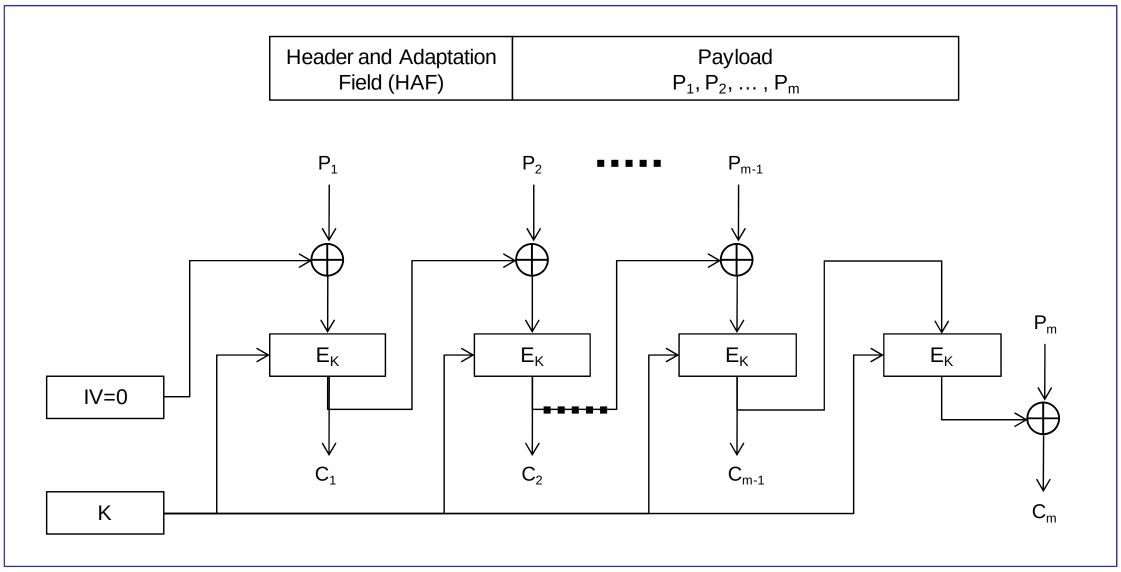


Figure I.1 – Scrambling of MPEG-2 transport stream packet

- Scrambling of MPEG-2 transport stream packets

Figure I.1 illustrates the scrambling of an MPEG-2 transport stream packet. In Figure I.1, Ek denotes the AES encryption algorithm under the control of scrambling key K; IV denotes the initialization vector, and ⊕, the bit-wise exclusive-OR operation. Both scrambling key K and the initialization vector (IV) consist of 16 octets or 128 bits.

To scramble an MPEG transport stream packet, the payload is divided into blocks of 16 octets. The last block may consist of less than 16 octets. As shown in Figure I.1, let P1, P2, …, Pm m≥1 denote the plaintext blocks of the payload of an MPEG-2 transport stream packet where P1 is the most significant block and Pm is the least significant block. Correspondingly, let C1, C2, …, Cm denote the cipher text blocks.

- Initialization vector (IV)

Used with the first payload block in each transport packet, the IV shall be a fixed 128-bit value with all bits set to zero.

- Patent/Licensing

The IDSA has been designed with the intent of producing a license-free, patent-free solution. The base algorithm, IV generation, and stream handling methods are well-documented in textbooks and within existing standards.

I.2 ETSI TS 103 197 DVB SimulCrypt

The DVB-SimulCrypt system is related to the concepts of the common scrambling algorithm. It is based on the concept of a shared scrambling and descrambling method.

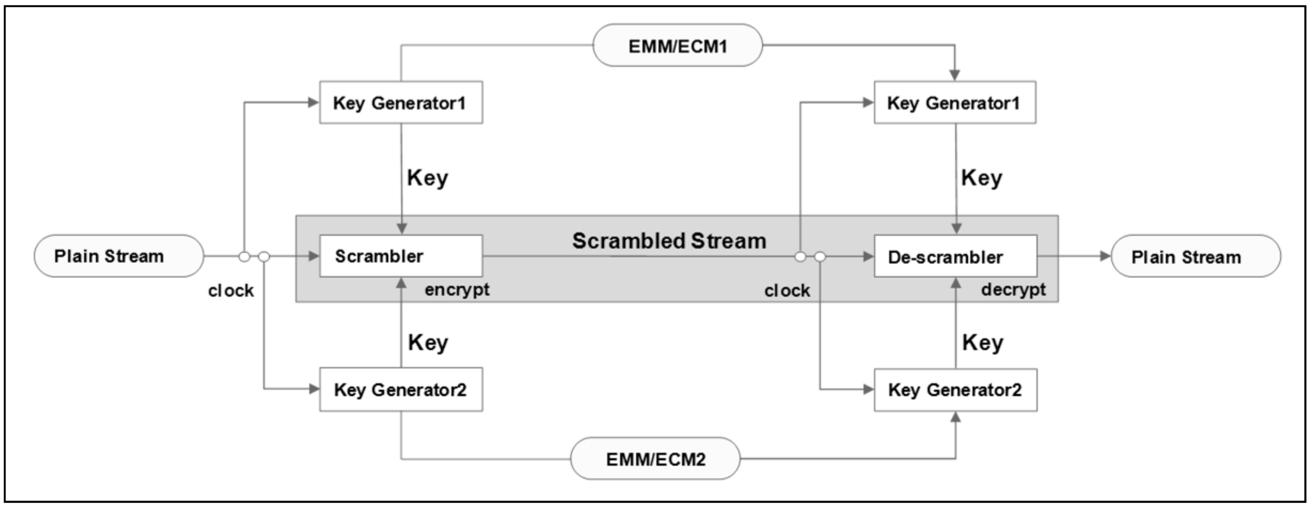


Figure I.2 – Concept of SimulCrypt and CSA

In DVB-SimulCrypt, to use a shared scrambled stream, DVB defines three components: event synchronizing, common scrambler and descrambler, and shared control word.

- Event Synchronizing

The service protection mechanism updates content encryption every 10 seconds or less. Therefore, the terminal device registers the time the key is changed and calculates how to guarantee that this control word is the right one. Therefore, SimulCrypt Synchronizer controls all events of the service protection message.

- Common scrambler and descrambler

To make a common scrambler, SimulCrypt assumes that the scrambling algorithm is shared by all participants. Therefore, the scrambler as one of the server systems shall be able to receive the same control word from SimulCrypt Synchronizer. The common scramble algorithm is not specified in that standard, but all security providers must share the same scrambling algorithm and scrambled contents.

- Shared control word

As for the common scrambler, control words are generated by SimulCrypt components, not a proprietary system. The generated control words are transmitted to SimulCrypt Synchronizer to control the event and scrambling.

The objective of SimulCrypt is to enable multiple security providers to have interoperability between two or more SCP systems at a head-end.

Appendix II

Algorithm selection scheme-related use cases

(This appendix does not form an integral part of this Recommendation.)

SCP descrambling functions are used for contents decryption and time synchronization between encryption keys. In linear TV, the encryption key is updated every 10 seconds; the SCP client is also updated, downloaded, and changed depending on the service selection or service policy.

Appendix II describes three possible scenarios that should require descrambler selection without hardware replacement.

II.1 Definitions of the terms used in the diagram

- IGMP: Internet Group Management Protocol
- $S_{D\text{-}N\text{-}M}$: Contents encryption keys with update period
  - D: indicates user group identification
  - N: indicates content identification
  - M: indicates a sequence of encryption keys within the descrambler

II.2 Scenario 1: SCP update

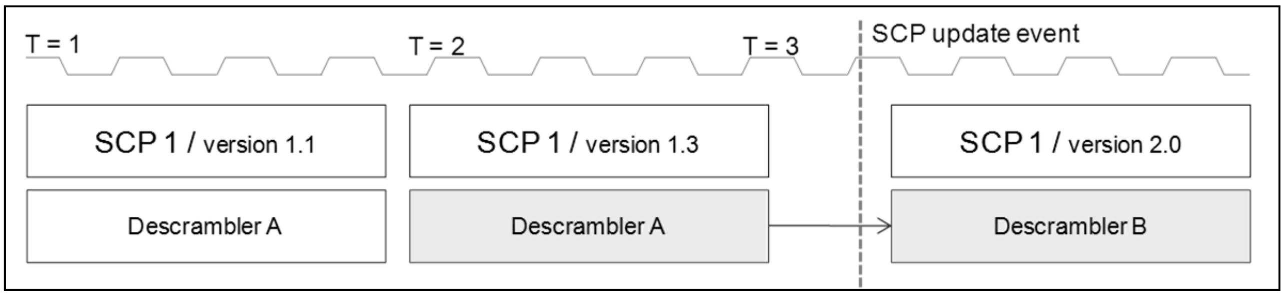


Figure II.1 – SCP update

In linear TV, the working scenario of a descrambler depends on a timeline. Figure II.1 depicts an SCP update. At T3 (T=3), an SCP update is requested from the service provider. Current SCP 1 is version 1.3, and new SCP client is version 2.0. For security reasons, the descrambler is also changed from A to B.

Previously, the change of a descrambler is impossible without any hardware change. If the interfaces supporting descrambler selection are defined, terminal devices support a set of descramblers defined in the global standard. Subsequently, descrambler A can be changed to descrambler B without any modification of the hardware system in the terminal device.

Additionally, the set of descramblers should be flexible depending on the local standard of each country or each service provider. The interface should be open and accessible, however.

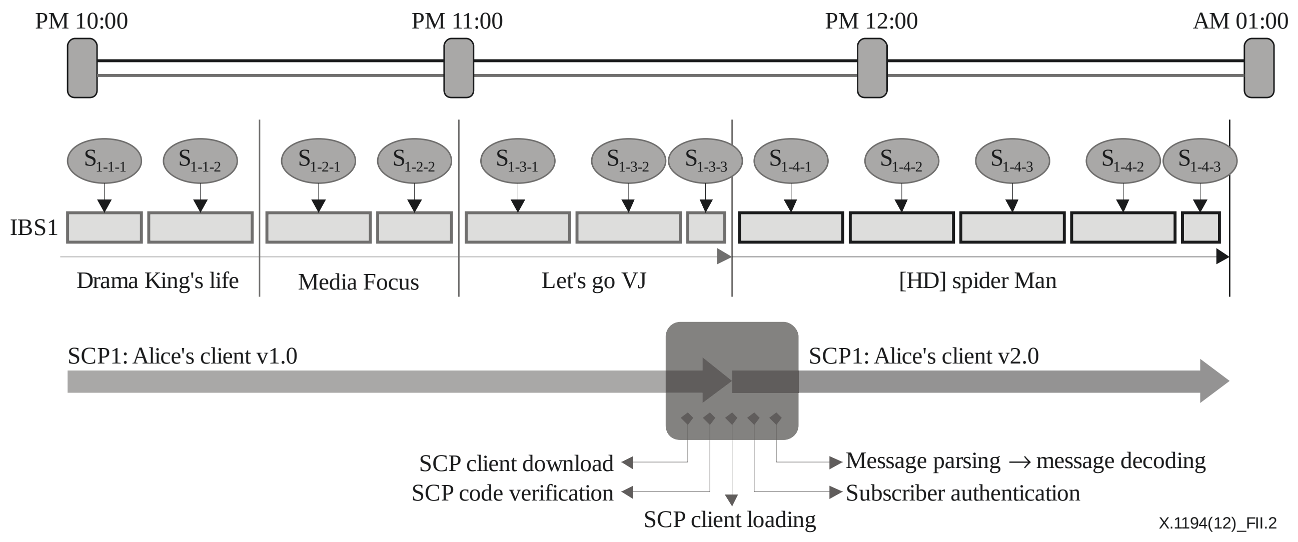


Figure II.2 – Scenario of SCP update

Figure II.2 describes the detailed working scenario of an SCP update. At 11:40 pm, the timing changes the contents from "Let's go VJ" to "[HD] Spider Man". Before 11:40 pm, version 1.0 of SCP1 is working properly. After 11:40 pm, version 2.0 of SCP1 is working properly.

Updating also requires SCP client download, verification, loading, and message parsing. An SCP client update includes minor change for service protection, but the descrambler can be changed if the security provider or service provider so wishes.

II.3 Scenario 2: SCP downloads

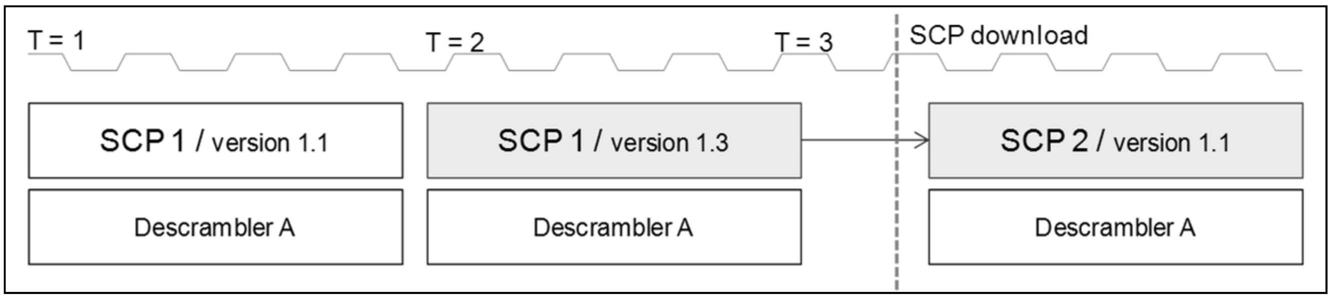


Figure II.3 – SCP download (change)

In Figure II.3 SCP download (change), the SCP client is changed from SCP1 to SCP2 at time T3 (T=3). This means that the applied SCP mechanisms are totally different, and that the security policy is also different. SCP1 and SCP2 share a single descrambler, however.

SCP1 and SCP2 have the same interfaces to access the same descrambler A. By time sequence, the SCP1 client hands over the scrambled contents to the SCP2 client with the same descrambler A.

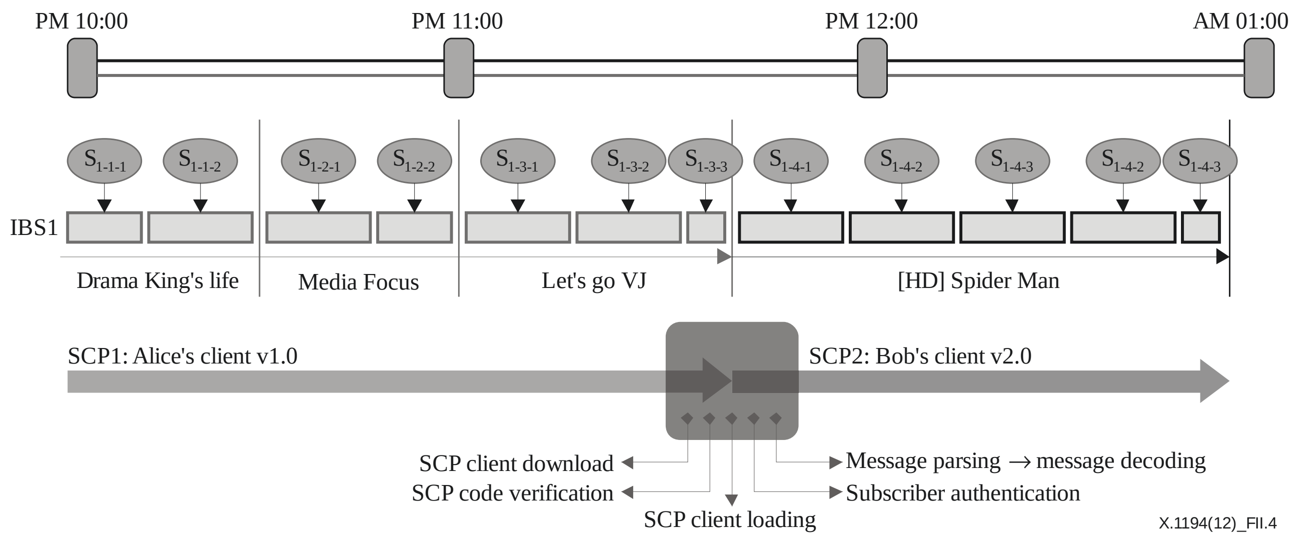


Figure II.4 – Scenario of SCP downloading

Figure II.4 describes the detailed scenario of the SCP download. The working code of the SCP function is changed at 11:40 pm, but the descrambling is the same because SCP1 and SCP2 share the same descrambler. This means that the scrambling algorithm and its server systems are not changed.

II.4 Scenario 3: Multiple SCP

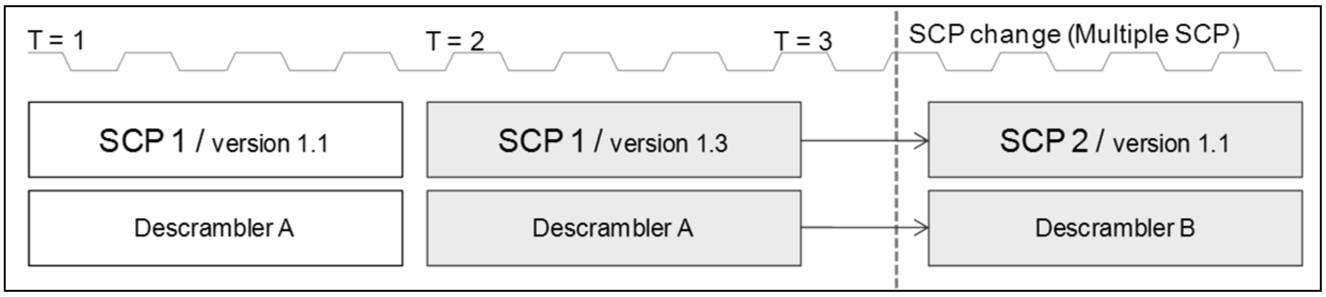


Figure II.5 – Multiple SCP

If two or more content groups deploy different SCP systems, the terminal device should support a multi-descrambler. Sometimes, it can be switched. When the contents provider requires a different level of security or a different security policy, then multiple SCP systems are working simultaneously.

Figure II.5 shows the multiple SCP case. At T3, the old SCP1 stops, and the new SCP2 is subsequently started. Note, however, that multiple SCP systems should be running at the same time, because those two SCP systems are simultaneously working together; mere switching activates an SCP client.

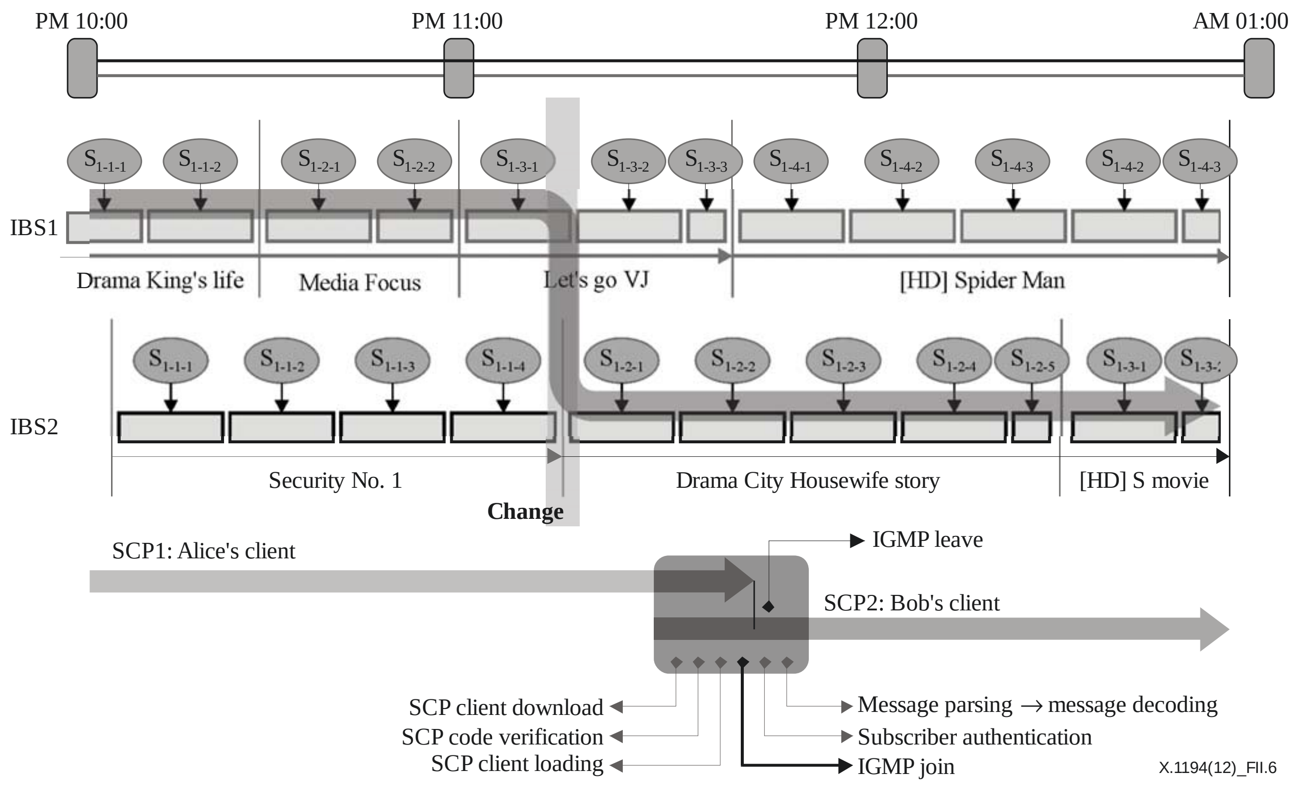


Figure II.6 – Scenario of multiple SCP

Two SCP systems are working together. SCP1 is working on IBS1 contents and SCP2 is working on IBS2 contents. At 11:10 pm the end user requests a change of contents from IBS1 "Let's go VJ" to IBS2 "Drama City housewife story". The active SCP client is also changed from SCP1 to SCP2, with a different descrambler but the same terminal device.

If the SCP2 client is not ready (not downloaded), terminal devices download, verify, and load a new SCP client (SCP2). After loading, the change will be performed, such as IGMP join, IGMP leave, subscriber authentication, and message parsing.

Appendix III

SCP messages in MPEG-2 TS

(This appendix does not form an integral part of this Recommendation.)

Messages for the algorithm selection scheme include a key update message, a synchronization message, and an algorithm select message. In this appendix we provide a case study on the message format. It is informative for easy understanding.

III.1 MPEG-2 TS PSI

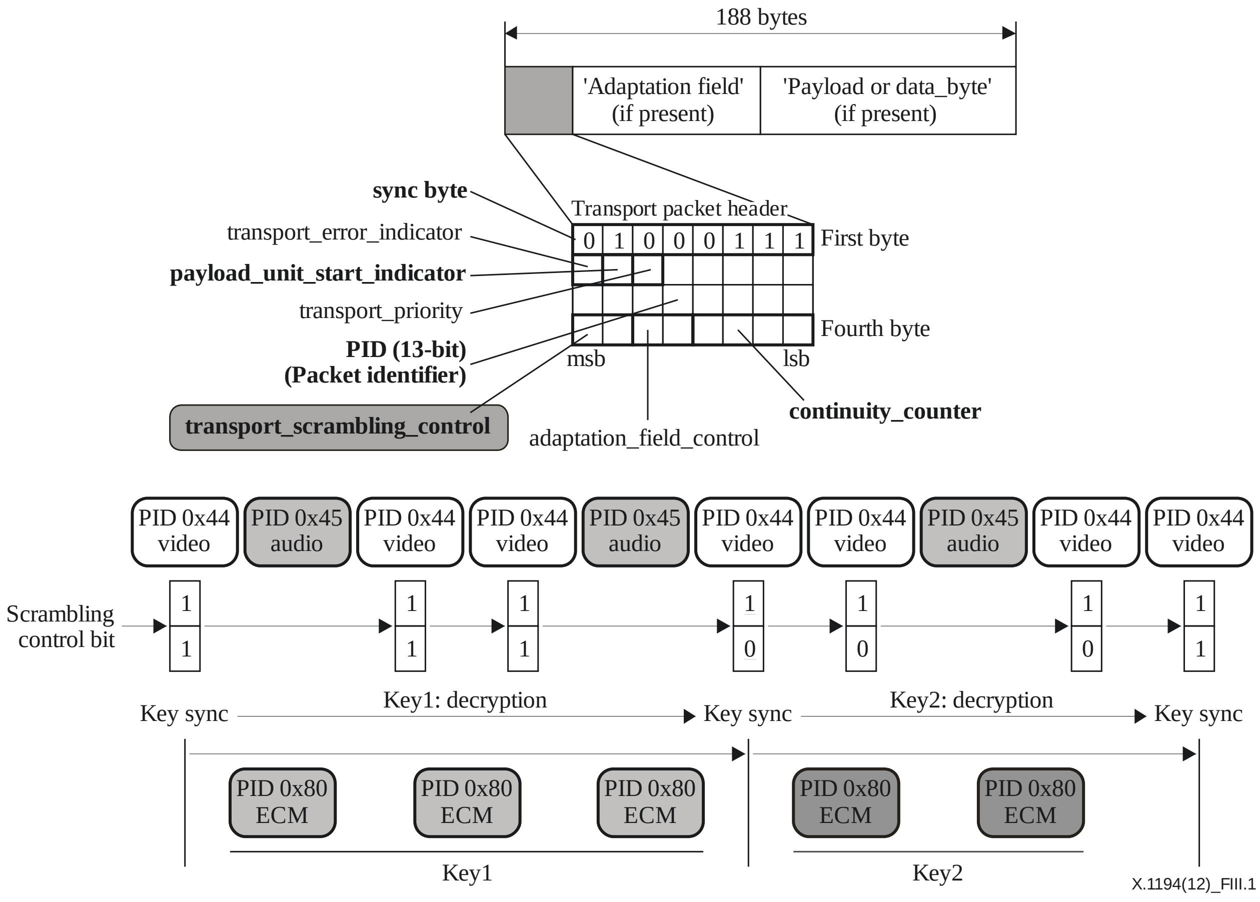


Figure III.1 – Synchronization message in the MPEG-2 TS (traditional CAS)

To synchronize the key update period, we can use the scrambling control bit defined by MPEG-2 TS. MPEG-2 TS is ISO/OSI standard 13818-1, but DVB used those bits for synchronization. "10" is an even key, and "01" is an odd key; the system knows the key change time by checking the *scrambling control bit.*

This is a case study on the key synchronizer, but we also define more suitable messages for IPTV service protection.

Appendix IV

Terminal device provisioning

(This appendix does not form an integral part of this Recommendation.)

ASS uses a certificate of terminal device and it should be shared before ASS starts. If the certificate is fixed, or special hardware is required, then we cannot support a separable or a downloadable SCP function. We assume that there are online key distribution mechanisms and that this is one of the candidate scenarios.

Provisioning is a function to connect a service provider. It includes device authentication, system download, and system configuration. In this appendix, we describe security-related functions within the provisioning functions.

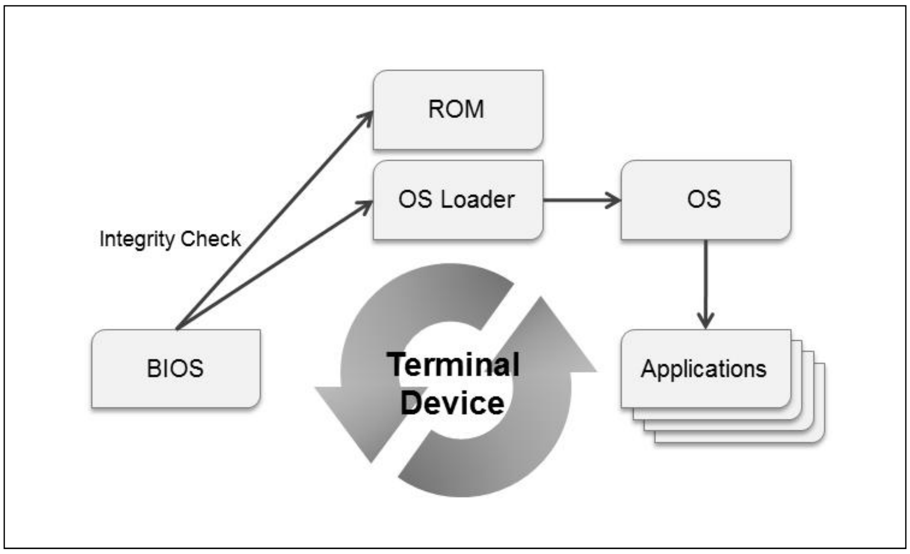


Figure IV.1 – Security-related terminal provisioning

Figure IV.1 shows the booting sequence for the provisioning service. First, BIOS checks the integrity of the system ROM and OS Loader. If the ROM and system are clean, then the operating system is started, and we take a normal provisioning process. Therefore, provisioning functions are developed by the terminal device manufacturer, with the connection procedure designed by a service provider.

Provisioning of a terminal device is divided into two parts – one part is provisioning for the service connection, and the other is integrity check of the system before booting up. In this Recommendation, BIOS, ROM, and OS authentication are just a reference.

– Terminal device authentication: Before booting up, the terminal device checks the integrity of BIOS, ROM, and OS, i.e., compares the digital signature with the original one. If there has been hacking or some other modification undertaken without permission, then the entire system must stop.

– Service connection: To connect a service provider, there are many issues concerning the triple play service, authentication protocol, service registration protocol, etc. Service connections are similar to the network connection. First, there is a need to assign a network address, to authenticate the terminal device, and finally, to download system files and profiles.

In this appendix, the terminal device has three processes for provisioning: network connection, service authentication (key distribution), and program installation. The first and second are similar to network provisioning. Note, however, that the third is program installation, which requires updated system programs or applications after network connection and authentication.

Appendix V

Example of one-time ID generation

(This appendix does not form an integral part of this Recommendation.)

With one-time ID (OTID), authentication procedures are different because some verification processes are added to the traditional authentication procedure. In traditional authentication, two parties are involved; in contrast, this has three parties for third-party verification, as in Figure 16. The authentication logic and authentication server are considered to be one party because they simply relay the required data for verification.

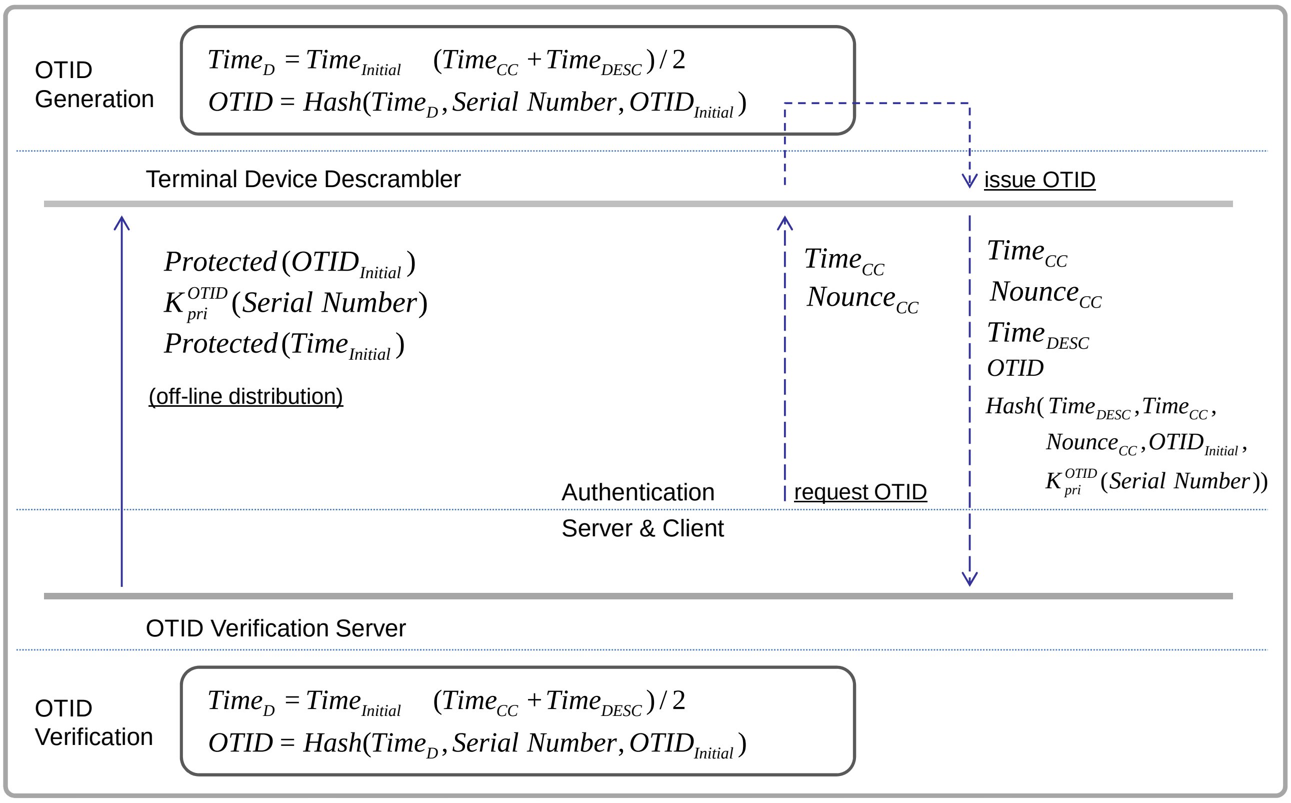

OTID Generation
$Time_D = Time_{Initial} \quad (Time_{CC} + Time_{DESC})/2$
$OTID = Hash(Time_D, Serial\ Number, OTID_{Initial})$
Terminal Device Descrambler
issue OTID
$Protected(OTID_{Initial})$
$K_{pri}^{OTID}(Serial\ Number)$
$Protected(Time_{Initial})$
(off-line distribution)
$Time_{CC}$
$Nounce_{CC}$
$Time_{CC}$
$Nounce_{CC}$
$Time_{DESC}$
$OTID$
$Hash(Time_{DESC}, Time_{CC}, Nounce_{CC}, OTID_{Initial}, K_{pri}^{OTID}(Serial\ Number))$
Authentication Server & Client
request OTID
OTID Verification Server
OTID Verification
$Time_D = Time_{Initial} \quad (Time_{CC} + Time_{DESC})/2$
$OTID = Hash(Time_D, Serial\ Number, OTID_{Initial})$

Figure V.1 – OTID generation and verification

Figure V.1 shows the OTID generation and verification protocol. In this protocol, secure information is $OTID_{Initial}$ and $Time_{Initial}$. The control client can capture all information except $OTID_{Initial}$ and $Time_{Initial}$. The secret information is delivered to the descrambler in the factory; this means that the secret information is delivered via offline distribution.

Every time the identity of the descrambler needs to be verified, a request is made (OTID_Request) to the control client with $Time_{CC}$ and $Nounce_{CC}$. The control client then sends the relevant data to the authentication server, which in turn sends it to the verification server. In this case, if the service provider accepts multiple OTID providers, then the relevant messages (OTID_Response, DESC_AUTH_Request) should include the verification server address.

$$\boxed{OTID = Hash(Time_{Initial} \quad (Time_{CC} + Time_{DESC})/2, Serial\ Number, OTID_{Initial})}$$

OTID has a one-time concept; therefore, we used time difference and $OTID_{Initial}$ for the hash function. The time difference is $TimeD = Time_{Initial} - (Time_{CC} + Time_{DESC}) / 2$ instead of $(TimeD = Time_{DESC} - Time_{CC})$ because the variation is too small.

This illustrates an example of OTID generation and verification procedures because it could be adding an industrial variation. For example, Figure V.1 uses the time difference between the control client and initial time, but we can change it with the serial counter, a general one-time password concept.

Bibliography

b-ITU-T H.264]	Recommendation ITU-T H.264 (2011), *Advanced video coding for generic audiovisual services*.
[b-ITU-T X.1193]	Recommendation ITU-T X.1193 (2011), *Key management framework for secure Internet protocol television (IPTV) services*.
[b-ITU-T X.1195]	Recommendation ITU-T X.1195 (2011), *Service and content protection (SCP) interoperability scheme*.
[b-IETF RFC 2104]	IETF RFC 2104, HMAC: *Keyed-hashing for Message Authentication*.
[b-ETSI TS 101 699]	ETSI TS 101 699 V1.1.1 (1999-11), *Digital Video Broadcasting (DVB); Extensions to the Common Interface Specification*.
[b-ETSI TS 102 825]	ETSI TS 102 825, *Digital Video Broadcasting (DVB); Content Protection and Copy Management (DVB-CPCM)*.
[b-ETSI TS 103 197]	ETSI TS 102 825 V1.5.1 (2008), *Digital Video Broadcasting (DVB); Head-end implementation of DVB SimulCrypt*.
[b-ISO/IEC 13818-1]	ISO/IEC 13818-1, *Information technology – Generic coding of moving pictures and associated audio information: Systems*.
[b-ISO/IEC 18033-3]	ISO/IEC 18033-3:2010, *Information technology – Security techniques – Encryption algorithms – Part 3: Block ciphers*.
[b-ATIS-0800006]	ATIS standard ATIS-0800006 (2007), *IIF Default Scrambling Algorithms (IDSA) IPTV Interoperability Specification*.

SERIES OF ITU-T RECOMMENDATIONS

Series A	Organization of the work of ITU-T
Series D	General tariff principles
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	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	Terminals and subjective and objective assessment methods
Series Q	Switching and signalling
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 and next-generation networks
Series Z	Languages and general software aspects for telecommunication systems

Printed in Switzerland
Geneva, 2012