

International Telecommunication Union

**ITU-T**

TELECOMMUNICATION  
STANDARDIZATION SECTOR  
OF ITU

**X.1198**

(06/2013)

SERIES X: DATA NETWORKS, OPEN SYSTEM  
COMMUNICATIONS AND SECURITY

Secure applications and services – IPTV security

---

**Virtual machine-based security platform for  
renewable IPTV service and content protection**

Recommendation ITU-T X.1198



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        |
| <b>IPTV security</b>               | <b>X.1180–X.1199</b> |
| 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.1198

### Virtual machine-based security platform for renewable IPTV service and content protection

#### Summary

Recommendation ITU-T X.1198 specifies a virtual machine-based security platform for the renewable service and content protection (SCP) system. The virtual machine supports an abstract function of hardware devices. This Recommendation defines a common interface and functional logic in the Internet protocol television (IPTV) terminal device and includes the data structure of SCP client and system components for a terminal device such as an embedded SCP, media client and control client.

#### History

| Edition | Recommendation | Approval   | Study Group |
|---------|----------------|------------|-------------|
| 1.0     | ITU-T X.1198   | 2013-06-13 | 17          |

#### Keywords

Conditional access, downloadable security, exchangeable security, renewable security, service protection, terminal device.

## FOREWORD

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

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

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

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

## NOTE

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

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

## INTELLECTUAL PROPERTY RIGHTS

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

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

© ITU 2013

All rights reserved. No part of this publication may be reproduced, by any means whatsoever, without the prior written permission of ITU.

## Table of Contents

|  | <b>Page</b> |
|--|-------------|
| 1 Scope .....  | 1           |
| 2 References.....  | 1           |
| 3 Definitions .....  | 1           |
| 3.1 Terms defined elsewhere .....                                  | 1           |
| 3.2 Terms defined in this Recommendation.....                      | 2           |
| 4 Abbreviations and acronyms .....                                 | 2           |
| 5 Conventions .....  | 3           |
| 6 Overview of a virtual machine platform.....                      | 4           |
| 6.1 General architecture of a virtual machine .....                | 4           |
| 6.2 Requirements for a virtual machine-based security system ..... | 5           |
| 6.3 Architecture of the SCP virtual machine.....                   | 6           |
| 7 Virtual machine-based security platform for the SCP system.....  | 9           |
| 7.1 System architecture .....                                      | 11          |
| 7.2 SCP virtual machine .....                                      | 12          |
| 7.3 Service and content protection (SCP) client.....               | 13          |
| 7.4 Protection of the SCP client and SVM.....                      | 13          |
| 7.5 Security policy.....   | 14          |
| 7.6 System call.....   | 15          |
| 7.7 Event.....   | 16          |
| Annex A – Data structure of SCP client package .....               | 19          |
| A.1 Header of SCP client .....                                     | 19          |
| A.2 Policy data .....  | 20          |
| A.3 Code data .....  | 21          |
| A.4 Signature.....   | 21          |
| Bibliography.....  | 22          |



# Recommendation ITU-T X.1198

## Virtual machine-based security platform for renewable IPTV service and content protection

### 1 Scope

This Recommendation develops a virtual machine-based security platform for renewable service and content protection (SCP) under Internet protocol television (IPTV) services. This includes the virtual machine architecture and the mechanism for organizing virtual machine-related components such as SCP client, terminal client (embedded SCP) and media client. The following items are addressed:

- a) Behaviour of SCP client: An SCP client has its pre-determined legitimate behaviour of execution. An SCP virtual machine (VM) should monitor the behaviour of an SCP client in run-time. This allows the system to prevent malicious code or unintentional code. Moreover, an SCP VM may support a reporting ability in case of illegal situations.
- b) Protection of SCP client: An SCP client code should not be seen by any unauthorized person or device during transmission. In addition, an SCP client code may be converted to an unreadable form even while loading a process.
- c) Integration with embedded SCP: An embedded SCP should obey the policy of the SCP client, including storing and retransmission.
- d) Integration with media client: A media client should deliver every event that has occurred in the device to the SCP VM. Deleting or modifying an event or its parameter is not allowed.

### 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.1194] Recommendation ITU-T X.1194 (2012), *Algorithm selection scheme for service and content protection descrambling*.

### 3 Definitions

#### 3.1 Terms defined elsewhere

This Recommendation uses the following terms defined elsewhere:

**3.1.1 access control** [b-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 (entity) authentication** [b-ITU-T X.1252]: A process used to achieve sufficient confidence in the binding between the entity and the presented identity.

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

- 3.1.4 confidentiality** [b-ITU-T X.800]: The property that information is not made available or disclosed to unauthorized individuals, entities, or processes.
- 3.1.5 digital signature** [b-ITU-T X.800]: Data appended to, or a cryptographic transformation (see cryptography in [b-ITU-T X.800]) 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.
- 3.1.6 integrity** [b-ITU-T X.800]: The property that data has not been altered or destroyed in an unauthorized manner.
- 3.1.7 key** [b-ITU-T X.800]: A sequence of symbols that controls the operations of encipherment and decipherment.
- 3.1.8 key management** [b-ITU-T X.800]: The generation, storage, distribution, deletion, archiving and application of keys in accordance with a security policy.
- 3.1.9 process** [ITU-T X.1194]: Instance of a computer program that is being executed; contains the program code and its current status.
- 3.1.10 scrambling algorithm** [b-ITU-T X.1191]: Algorithm used in a scrambling or a descrambling process.
- 3.1.11 service and content protection** [b-ITU-T X.1191]: A combination of service protection and content protection or the system or implementation thereof.
- 3.1.12 service protection** [b-ITU-T X.1191]: Ensuring that an end user can only acquire a service and the content hosted therein by extension as what he/she is entitled to receive; service protection includes protecting service from unauthorized access as IPTV contents traverse through the IPTV service connections.
- 3.1.13 virtual machine** [ITU-T X.1194]: Software implementation of a machine that can execute programs just like a physical machine; supports separated operating systems.

## 3.2 Terms defined in this Recommendation

This Recommendation defines the following terms:

- 3.2.1 conditional access:** A function served by a conditional access system; often used as an abbreviation for conditional access system.
- 3.2.2 conditional access system:** 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.
- 3.2.3 trap:** A code or signal function designed to capture errors and reveal where they have occurred.

## 4 Abbreviations and acronyms

This Recommendation uses the following abbreviations and acronyms:

|     |  |
|-----|--|
| AAA | Authentication, Authorization and Accounting |
| API | Application Programming Interface            |
| CPU | Central Processing Unit                      |
| ECM | Entitlement Control Message                  |
| EMM | Entitlement Management Message               |
| EPG | Electronic Programme Guide                   |
| HAL | Hardware Abstraction Layer                   |
| H/W | Hardware                                     |

|      |                                 |
|------|---------------------------------|
| ID   | Identity                        |
| IDE  | Integrated Drive Electronics    |
| I/O  | Input/Output                    |
| IPTV | Internet Protocol Television    |
| OS   | Operating System                |
| OTID | One-Time Identifier             |
| PC   | Program Counter                 |
| SCP  | Service and Content Protection  |
| SCSI | Small Computer System Interface |
| SIM  | Subscriber Identity Module      |
| SSL  | Secure Socket Layer             |
| SVM  | Secure Virtual Machine          |
| S/W  | Software                        |
| TLS  | Transport Layer Security        |
| UI   | User Interface                  |
| USB  | User Serial Bus                 |
| UX   | User experience                 |
| VM   | Virtual Machine                 |
| VoD  | Video on Demand                 |
| XML  | extensible Markup Language      |

## 5 Conventions

In this Recommendation:

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

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

The words "**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 this Recommendation.

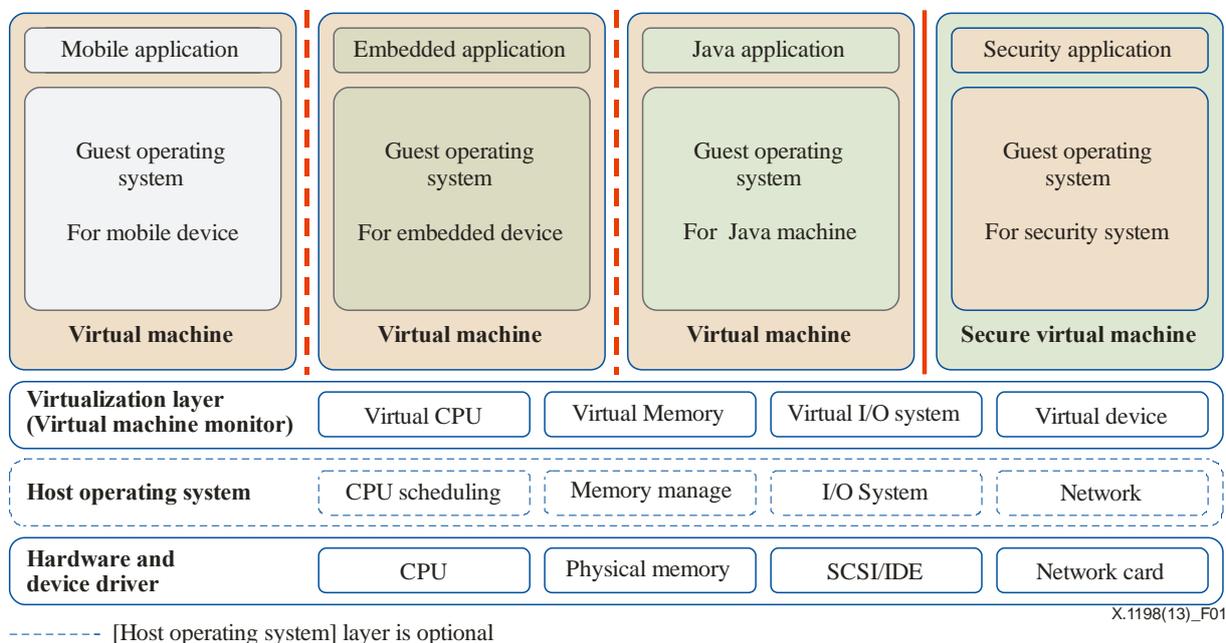
In the body of this Recommendation and its annex, the words *shall*, *shall not*, *should*, and *may* sometimes appear, in which case they are to be interpreted, respectively, as *is required to*, *is prohibited from*, *is recommended* and *can optionally*. The appearance of such phrases or words in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

## 6 Overview of a virtual machine platform

There are two ways to support a renewable client. The first way is a non-standardized method; just a single mechanism is implemented such as a hardware chip (a security co-processor) or a physical devices with an encapsulated or hardwired protection mechanism. The second way is through an open and standard-based approach. This Recommendation develops open and common interfaces to support an interoperable and renewable SCP client with a virtual machine. It is a platform-independent run-time environment for the execution of an SCP client in the terminal device, which provides a downloadable and renewable service protection mechanism. Because this platform is a virtual environment, it enables the writing of a downloadable SCP code which has instruction-level code compatibility. This platform can be implemented in software (S/W) or hardware (H/W) environments.

### 6.1 General architecture of a virtual machine

Generally, a virtual machine provides functions of code mobility and hardware abstraction. As a security feature, a virtual machine provides a safe termination and a controlled execution of a mobile code. Figure 1 shows the concept of a virtual machine.



**Figure 1 – General architecture of a virtual machine**

The system is divided into a host operating system and several virtual machines, including a secure virtual machine, because it has a role of separating a host operating system (OS) and a guest OS. In the secure application, accessing a host operating system is only allowed through the virtual machine, which has the privilege to call the run-time library and system call from the policy monitor.

Components of virtual machine-based systems are illustrated below:

- Application (App): A program that executes a set of instructions and which is commonly defined with single operations for the specified platform such as a mobile device, an embedded device, a Java machine, or a security system.
- Security application (S-App): This application can access all security functions, which are low level disk access and protected instructions such as policy monitor, access token and a content encryption key. The platform should protect security applications from other applications or other illegal processes.

- Guest operating system: This is an operating system with a virtual machine instead of a native driver and a physical device such as a central processing unit (CPU), memory, an input/output (I/O) system and a network card.
- Virtual machine: This is an abstraction platform to support multiple OS environments. It includes a guest operating system and applications. Functionally, this includes application provisioning, maintenance and interoperability functions between different operating systems.
- Secure virtual machine: This is a virtual machine that additionally includes security-enhanced features such as access control, memory protection, process isolation and SCP client-related instructions.
- Virtualization layer: This is a hardware virtualization to support multiple operating systems, which means that multiple instances of a variety of operating systems may share the virtualized hardware resources such as the CPU, memory, I/O system and devices.
- Host operating system: This is a common instruction set to provide a native code. It is not needed in a situation of hardware-level virtualization.

## 6.2 Requirements for a virtual machine-based security system

A virtual machine provides an abstraction of system functions with the common application programming interface (API), but the open API has some vulnerability in the junction of the virtual machines. This clause defines general security requirements for the virtual machine and virtual machine-based renewable systems.

The following security requirements support a renewable system in IPTV service:

- R 6.3.3-04: 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 algorithm.
- R 6.3.3-13: IPTV architecture is required to support a mechanism to securely retrieve the SCP parameters (e.g., configuration, status) from an IPTV terminal device.
- R 6.3.3-14: 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: 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 a universal serial bus (USB) dongle and subscriber identity module (SIM) cards.
- R 6.3.3-19: IPTV architecture is prohibited from precluding support for a mechanism for the selection of an SCP system from the available SCP systems without any hardware replacement except for removable devices.
- R 6.3.3-20: IPTV architecture is prohibited from precluding support for the secure downloading of an SCP system. The specific downloading can optionally depend on specific service protection requirements.
- RR 6.7.3-01: IPTV architecture is recommended to support a mechanism for end users to select IPTV network providers, IPTV service providers and IPTV content providers according to their preferences.

### 6.2.1 General security requirements for a virtual machine

A virtual machine has similar requirements to those of traditional operating systems such as the prevention of illegal access of external codes and system memory protection from unauthorized access. In the virtual machine, the external code, which is imported from outside the system, could

be a malicious one. Hence, the system should prevent the illegal access of those codes that may affect system memory; in many cases, the system is cracked by this kind of illegal access.

To protect the virtual machine, the SCP systems should satisfy the following general security requirements to protect the system itself and applications from various attacks:

- Process isolation: The security process should be protected from other processes including memory access. In addition, a multi-process system should control the intercommunication channel and shared memory.
- Data protection: The security system should protect the data from other processes and host machine access. The control logic is also managed by the security system instead of the host operating system.
- Protection of system resources: The system resources include a physical CPU and memory access rights, and they should be controlled for the protection of the security system.
- Availability: The security system should be available while the system is running and should be responsible in situations of security event emergencies.

### **6.2.2 Security requirements for a virtual machine-based renewable system**

A virtual machine-based renewable system provides a dynamic execution of the SCP client; however, the machine that executes the SCP client could have control and access rights for the SCP client. If the system or virtual machine is malicious, then the SCP system is not secure. Furthermore, the virtual machine could become vulnerable if the SCP client tries to attack the virtual machine with an unauthorized code execution.

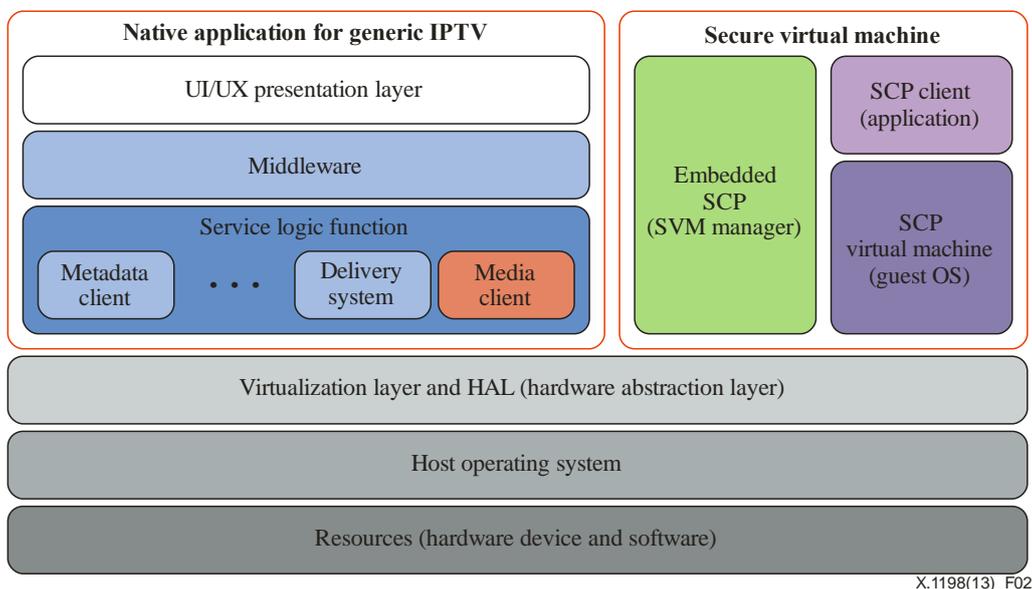
Therefore, the virtual machine should satisfy the following security requirements to protect the content and the service from various attacks:

- SCP client authenticity: The SCP client should contain a valid digital signature of a trusted party.
- Behaviour of SCP client: The SCP client has its predetermined legitimate behaviour of execution. The SCP VM should monitor the behaviour of the SCP client in run-time. This will enable the system to prevent malicious code or unintentional code. Moreover, the SCP VM may support a reporting capability in case of illegal situations.
- Protection of SCP client: The SCP client code should not be seen by any unauthorized person or devices during transmission. In addition, the SCP client code may convert to an unreadable form during a loading process.
- Key protection: Cryptographic keys for the SCP client and embedded SCP should not be exported to an external system or SCP client in plaintext form.
- Security of system calls: System calls which are called by the SCP client should be secured against buffer overflow attacks for pointer addresses.
- Integration with embedded SCP: The embedded SCP should obey the policy of the SCP client including that for storing and retransmission.
- Integration with media player: The media player should deliver every event that has occurred in the device to the SCP VM. Deleting or modifying an event or its parameter is not allowed.
- Access an access token: The system call can read the access token which is managed by the embedded SCP. However, the system call cannot modify the access token in the device.

### **6.3 Architecture of the SCP virtual machine**

The SCP virtual machine is a supporting logic for a renewable SCP client; generally, an SCP virtual machine has the role of downloading and managing the SCP client. However, the IPTV system has three components for renewable SCP clients. The first component is the media client, which

supports loading a media stream, extracting metadata and playing the contents. The second component is the embedded SCP, which is a virtual machine manager to download an SCP client, authenticate user identity and make a secure socket channel, etc. The third component is the SCP virtual machine, which supports machine instructions such as event, system call and security policy. Figure 2 shows the relationship between these components.



**Figure 2 – Architecture of SCP virtual machine**

Figure 2 has six layers as follows:

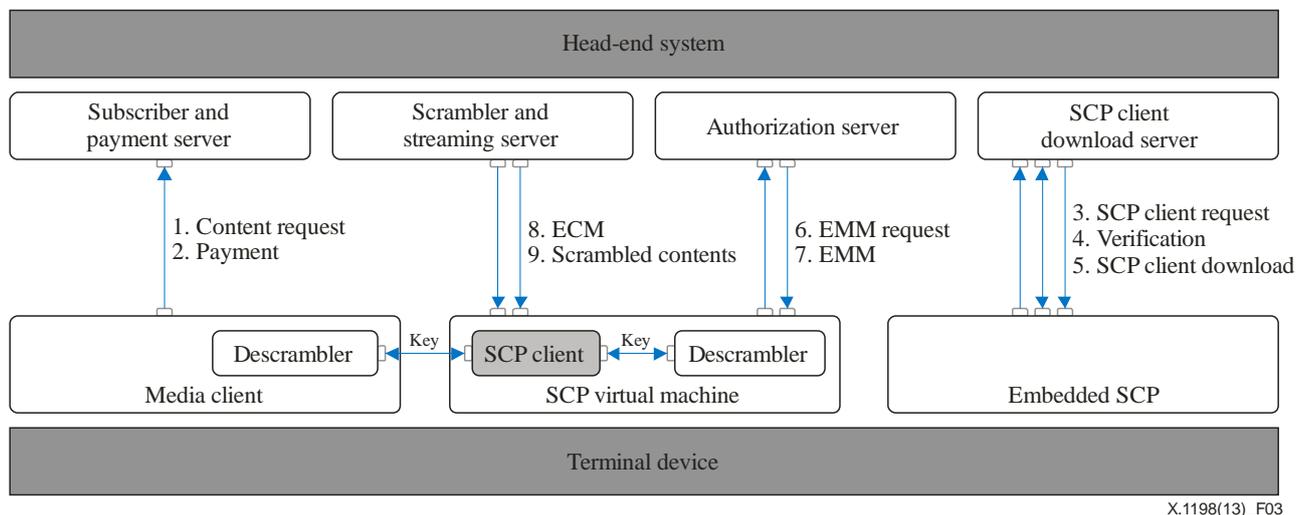
- **Resource layer for hardware and driver software:** It contains the primitive components of the hardware device, which plays IPTV content with a privileged access. For example, the resource layer includes a media processor, a central processor, a memory, a display controller, a demuxer, a descrambler and a remote controller.
- **Host operating system:** The basic operating system for running and controlling the resource layer. It could be removed if the virtualization layer supports these functions.
- **Virtualization layer and hardware abstraction layer (HAL):** A hardware virtualization or hardware abstraction layer to support a common function of the operating systems such as a virtual CPU, a virtual memory and a virtual I/O system.
- **Service logic function layer:** It includes a common library to support IPTV middleware such as event handling, message filtering and message display.
- **Middleware layer:** Decodes IPTV content and displays messages such as an electronic programme guide (EPG), data broadcasting content and SCP messages such as "access denied".
- **UI/UX (user interface or user experience) layer:** It displays a user interface and receives an input from the subscriber to act in the terminal device including the menu button, display windows and service navigation (user interaction).

The SCP system has three parts: embedded SCP, SCP virtual machine and SCP client. The embedded SCP has an interoperability role with the media client. It is an interoperability channel between SCP and IPTV middleware, and a unique communication channel between SCP and IPTV middleware.

- **Embedded SCP:** It is an extended VM manager or secure virtual machine (SVM) manager to support a part of the SCP function, downloading the SCP client, installing the SCP client and verification of the SCP client.

- SCP virtual machine: It is a virtual machine that executes an SCP client. Further details are given in clause 7. The SCP virtual machine just receives messages including entitlement management messages (EMMs), entitlement control messages (ECMs) and descrambling messages, and hands them over to the SCP client.
- SCP client: It decodes SCP messages, analyses the semantics of messages and generates the encryption key. The details are shown in clause 7.3.

Figure 3 represents a detailed scenario of the SCP virtual machine.



**Figure 3 – Virtual machine-based SCP system**

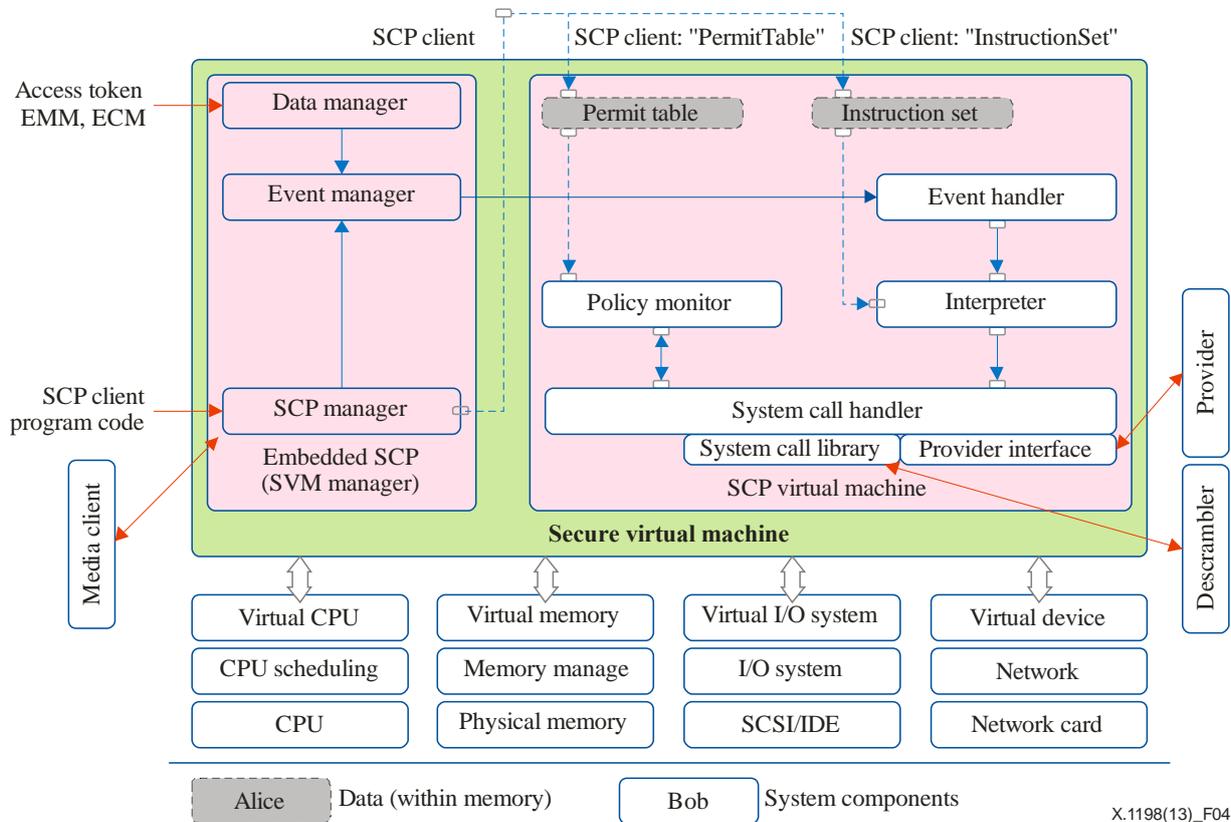
The SCP virtual machine is a platform-independent run-time environment for the execution of an SCP client in the SCP system, which provides a downloadable and a renewable SCP client mechanism. The system works according to the following sequence:

1. Content request: A subscriber selects content and media client request permission from the subscriber server. The subscriber server requests a payment for the selected content.
2. Payment: Covers the physical payment by using a wire transfer, credit card, mobile phone, etc.
3. SCP client request: The embedded SCP requests a proper SCP client.
4. Verification/Authentication: Verification of the SCP client and authentication of the embedded SCP or terminal device with terminal device ID.
5. SCP client download: downloading of the SCP client.
6. EMM request: The SCP virtual machine requests an EMM message.
7. EMM: EMM is sent to the terminal device (SCP virtual machine).
8. ECM: ECM is sent to the terminal device (SCP client).
9. Scrambled contents: The SCP client descrambles the contents.

Figure 3 shows two descramblers in the terminal device. The first one is a hardware descrambler, or an embedded descrambler, located in the media client. Generally, the descrambler is implemented by the hardware logic because of performance issues; in this case, the media client controls the hardware or embedded modules of descrambling and the SCP client just sends an encryption key in a protected channel. The other descrambler is located in the SCP virtual machine; the descrambler is implemented by software logic in the SVM or by a hardware-supported system call. This descrambler is only implemented within the SCP virtual machine.

## 7 Virtual machine-based security platform for the SCP system

The virtual machine-based security platform, which is a secure virtual machine, is also a virtual machine. It has a virtual machine and a virtual machine manager. The virtual machine manager (embedded SCP, SVM manager) controls the status of the virtual machine with the event. Figure 4 depicts how the system works.



**Figure 4 – A virtual machine-based security platform**

There are three supporting components in the SVM:

- **Media client:** This provides an interface for the terminal device middleware. It displays the message of the SCP function and generates system events such as channel change, and system error messages.
- **Provider:** These are external security providers, the private and security enhanced functions are implemented here.
- **Descrambler:** This is an external descrambler, which is a hardware-based descrambler. Therefore, the system call library should include the hardware control and secure communication mechanism.

There are three components in the embedded SCP:

- **Data manager:** Receives an access token such as EMM or ECM, and monitors the IPTV content to check if there is any new data or not.
- **Event manager:** Generates system events in order to control the SVM status; the SVM is working based on the system event. In other words, it generates system events of the data manager and SCP manager.
- **SCP manager:** Downloads and installs the SCP client into the virtual memory of the SCP virtual machine. The SCP client is divided into the permit table and instruction set in the virtual memory.

There are two memory data spaces ("PermitTable" and "InstructionSet") and six components in the SCP virtual machine:

- **Memory:** In the SCP virtual machine, every instance of the SCP client has two memory spaces: the first one is the "Permit Table" which defines the policy of the SCP client in the instruction level. The second one is the "Instruction Set" which stores a set of instructions for the SCP client; each instruction is executed by the interpreter in the virtual machine. The SCP client has four data parts (Header, Policy, Code and Sign) as illustrated in Figure 7: one of them is policy and code. The policy is loaded into the memory (permit table), and the code is also loaded into the memory (instruction set).
- **Event handler:** Takes and analyses the event when an event has occurred. Regarding the event, the event handler changes the status of the SCP client. Detailed events are shown in Figure 6.
- **Interpreter:** Executes an instruction in the SCP client; each instruction is defined by individual functions, which are system calls in the virtual machine. Hence, a set of system calls is implemented in the virtual machine (referred to as 'system call library' in Figure 4).
- **Policy monitor:** Gives a decision to the system call handler; the requested system call is either permitted or not permitted by the permit table in the SCP client. Therefore, the policy is defined in the system call level and not in the instruction level.
- **System call handler:** Requests a permit from the policy monitor and calls a set of corresponding system calls, when the interpreter requests a system call.
- **System call library:** Implements a native function for each system call. This could be coupled with hardware functions such as direct memory access, direct I/O system and hardware descrambler.
- **Provider API:** Provides a private function for a third-party SCP provider; it is not included in the standardization part. However, the interface for the private function could be located in this logic.

In Figure 4, the system works as follows:

- The end user requests a program channel (or change): The media client checks which SCP client is needed for the requested channel. If it is different from the current SCP client, the media client sends an SCP client change message to the SCP manager.
- The media client sends information about the new SCP client to the SCP manager.
- The SCP manager downloads the requested SCP client and stores it in the memory, the instruction set is stored in the InstructionSet memory and the permit table is stored in the PermitTable memory.
- The SCP manager sends a start message to the event manager.
- The event manager generates a start event to the event handler: `EVENT_Initialize`.

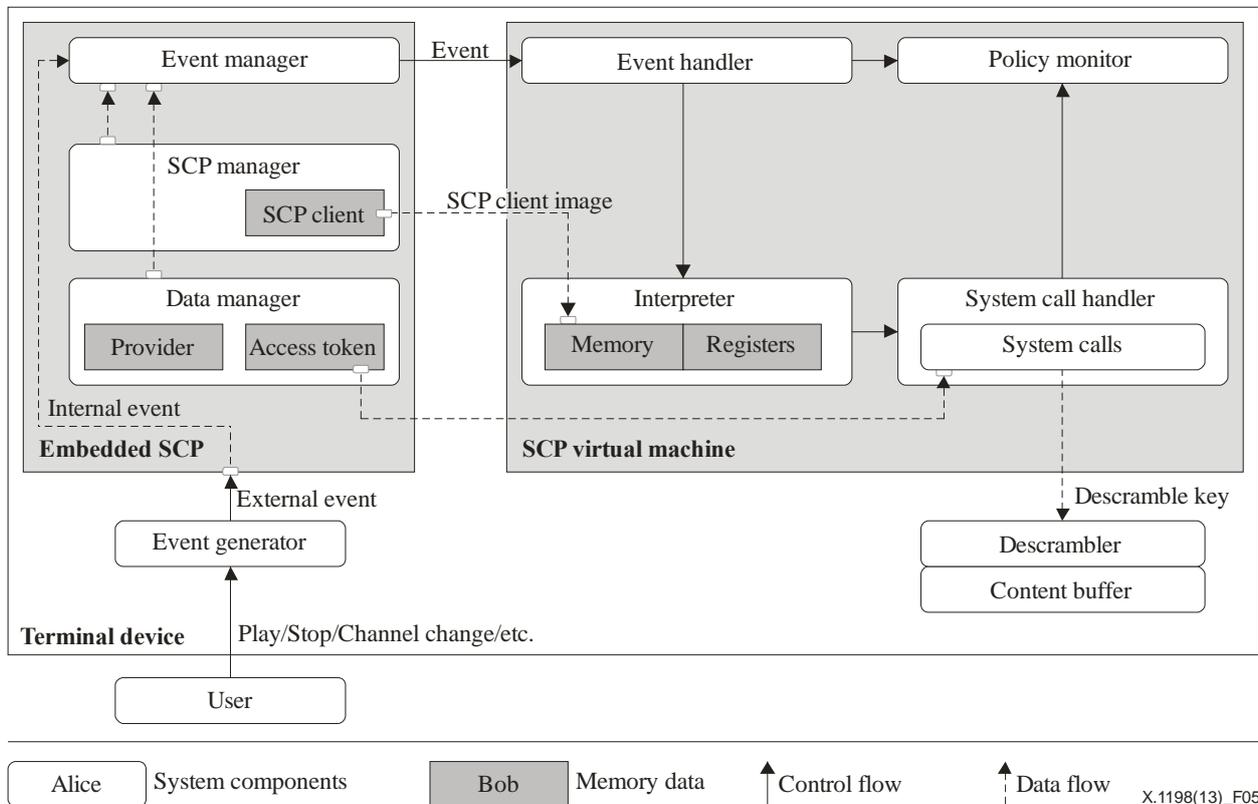
The interpreter fetches the next instruction in the memory (SCP client image) and generates corresponding events such as:

`EVENT_Player`, `EVENT_Timer`, `EVENT_Terminate`, `EVENT_Descramble`

- Before the execution of the instruction, the policy monitor checks the PermitTable so as to allow or not allow the requested instruction.
- If the instruction needs an additional function such as a system library or an external function, then the interpreter passes the function call to the system call handler.
- Access Token (EMM, ECM) eventually occurs and it also generates events such as: `EVENT_ChangeKey`, `EVENT_Entitlement`, `EVENT_Descramble` and `EVENT_SelectContent`.

## 7.1 System architecture

The SCP virtual machine includes the following generic components: interpreter, registers, memory, system calls, event handler, descrambler, policy monitor, content buffer, event generator, SCP client image, access token and provider. The generic system architecture is depicted in Figure 5 as follows:



**Figure 5 – SCP virtual machine architectures**

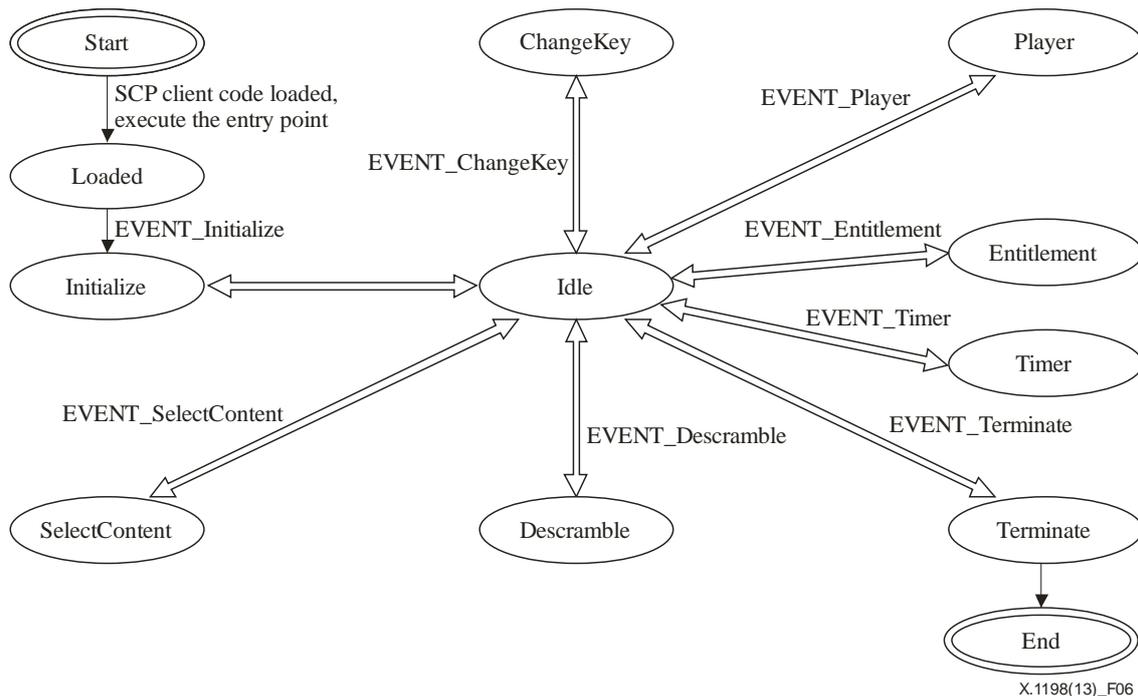
- **Interpreter:** It executes the instruction code iteratively.
- **Registers:** All registers have a 32-bit length. There are 32 registers for general purposes and a program counter (PC).
- **Memory:** It is part of the SCP VM and stores instruction codes and stacks, etc.
- **System calls:** Are functions provided by the SCP VM. A system call is used as an interface to the outside of the SCP VM or to support a complex operation.
- **Event handler:** The media client may generate various events for the SCP VM. The event handler is a part of the SCP client that processes each event.
- **Descrambler:** It is a descrambling algorithm which is defined in the SCP client. The descrambler may be hardware implemented as part of the media client.
- **Policy monitor:** This module continuously monitors that the SCP client does not violate the attached policy.
- **Content buffer:** The descrambler accesses this buffer, and the buffer may be located just before the decoder in the data path.
- **Event generator:** It generates events for the SCP VM to control. Different kinds of parameters are included in the event.
- **SCP client image:** It contains the storage for the SCP client, which belongs to the embedded SCP. The image of the SCP client will be loaded into the SCP VM before initialization.

- Access token: It contains the storage for the access token, which belongs to the embedded SCP. The SCP VM accesses the access token data when the SCP client requests the access token.
- Provider: Holds the storage for the provider, which belongs to the embedded SCP. When the SCP client is loaded to the SCP VM, the VM manager requests a globally identifiable SCP identity (ID) from the provider.

## 7.2 SCP virtual machine

The SCP virtual machine is a simple event-driven system which runs the media client initiated event. Therefore, to save CPU power the virtual machine becomes idle if there is no input event.

Figure 6 represents all machine states from the SCP client loading to termination.



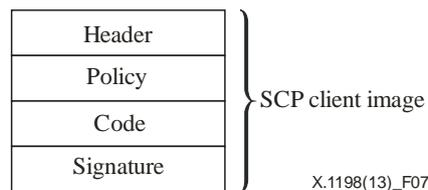
**Figure 6 – SCP virtual machine running states**

- Start: When the SCP manager downloads an SCP client, the system sets the status of the SCP client to "Start". Another SCP client does not load until the SCP client is terminated.
- Loaded: When an SCP client finishes loading program codes, the SCP virtual machine sets the status of SCP client to "Initialize".
- Initialize: When an SCP client finishes the initialization of a program code, the SCP virtual machine sets the status of the SCP client to "Idle".
- Idle: When an SCP client finishes requested events such as EVENT\_ChangeKey, EVENT\_Player, EVENT\_Entitlement, EVENT\_Timer, EVENT\_Terminate, EVENT\_Descramble and EVENT\_SelectContent, the SCP virtual machine returns the status of SCP client to "Idle", which is the waiting status for a new request.
- ChangeKey: When an SCP client receives EVENT\_ChangeKey, the SCP virtual machine launches a ChangeKey function and the state is changed during the execution of the function.
- Player: When an SCP client receives EVENT\_Player, the SCP virtual machine launches a player function and the state is changed during the execution of the player function including contents decoding.

- Entitlement: When an SCP client receives an EVENT\_Entitlement request for a new entitlement, the SCP virtual machine runs the entitlement function of the SCP client.
- Timer: When an SCP client receives a time event EVENT\_Timer, the SCP virtual machine launches the timer function, which is waiting for a requested time period.
- Terminate: When an SCP client receives a terminate event request EVENT\_Terminate, the SCP virtual machine deletes the SCP client and returns memory to the virtual machine.
- End: Final state of the SCP client. The SCP virtual machines are set; the system does not run an SCP client in this state.
- Descramble: When an SCP client receives the descrambling request EVENT\_Descramble, the SCP virtual machine launches a descramble function, which is defined in the SCP client.
- SelectContent: When an SCP client receives the channel change request EVENT\_SelectContent, the SCP virtual machine launches a program change process.

### 7.3 Service and content protection (SCP) client

The SCP client and the policy are distributed in an SCP client package form. The policy defines the SCP client permission to protect a system from illegal attacks and unexpected defects. Before launching the SCP client, the SCP virtual machine scans the SCP client instruction codes. After launching, the SCP virtual machine gives permissions to the SCP client with this policy, which is a policy monitor. The SCP client is composed of the following items (see Figure 7):



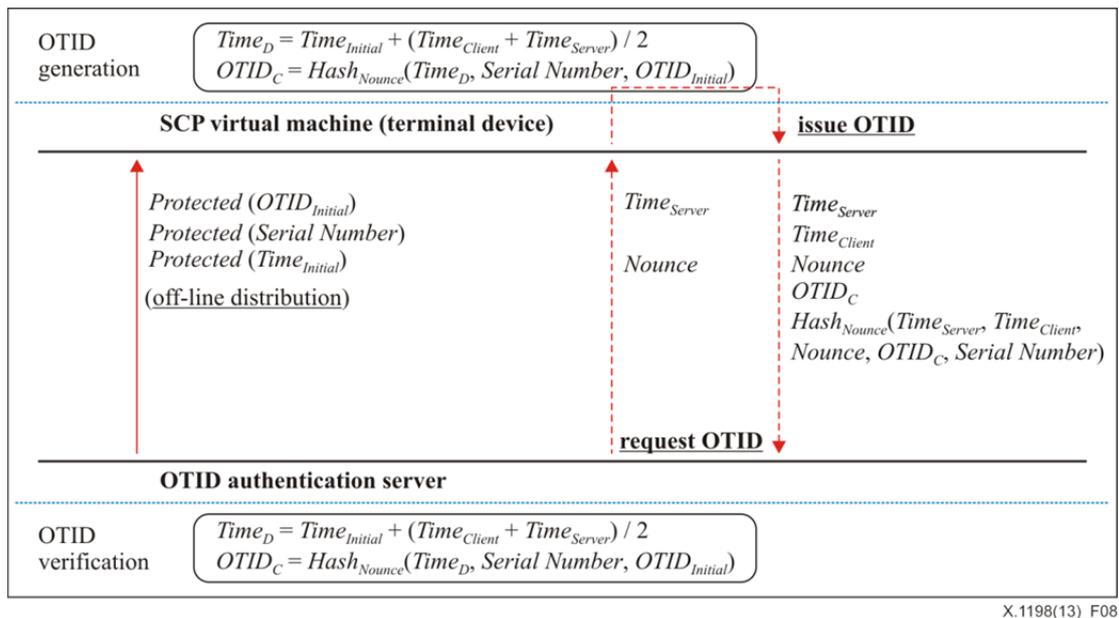
**Figure 7 – SCP client image**

- Header: This contains the metadata which defines the version of the SCP client, the SCP client ID and length of other sections.
- Policy: The policy data can define the security policy for the SCP client. This policy is loaded into the policy monitor.
- Code: The executable instruction code will be loaded into the virtual machine code area. The instruction format is not defined in this Recommendation.
- Signature: The digital signature of all the above items is generated by a certificate authority to prevent forgery or modification.

### 7.4 Protection of the SCP client and SVM

The SCP client is protected with secure socket layer/transport layer security (SSL/TLS) during distribution, but strong authentication mechanisms are required for the protection of the SCP client and SVM which are part of the embedded SCP.

To prevent or detect a duplicated or clone SCP client, the terminal device or SCP virtual machine should provide a one-time identifier (OTID) based authentication mechanism, which is defined in [ITU-T X.1194]. In the virtual machine, the authentication process is required in the SCP client download and entitlement phase. Figure 8 shows the details of device authentication.



**Figure 8 – OTID generation for the detection of a clone client**

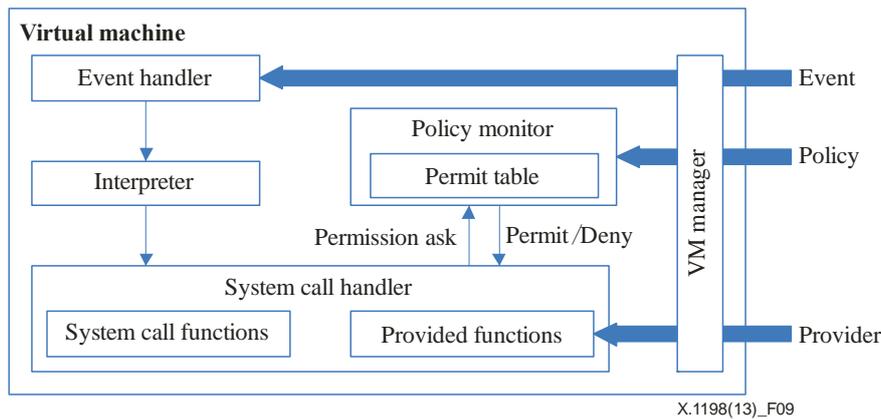
OTID in [ITU-T X.1194] uses a private key encryption instead of a password. This Recommendation uses iterated hash functions to detect a replay attack and an SCP clone client. If an SCP clone client uses the same values for the authentication, the OTID authentication server could deny access.

### 7.5 Security policy

The SCP client on the virtual machine runs as an SCP application, which may make the entire SCP system vulnerable if the SCP client has a security weakness. Because of this, a digital signature is attached to every SCP client to prevent illegal modification. However, if the SCP client contains an unintentional bug, the digital signature is not a sufficiently preventive method against any potential vulnerability.

Therefore, every SCP client has its own policy which defines the lifetime and the allowed operations of the SCP client. The policy is loaded into the policy monitor that observes the behaviour of the SCP client. If an SCP client conducts operations that contradict this policy, the SCP VM should stop the execution and report the event to the media client appropriately, for example, by providing a return error code to the media player or by invoking an error event.

The observation of the SCP client execution may be implemented by the policy monitor. Figure 9 demonstrates the policy monitor implementation.



**Figure 9 – Policy monitor implementation**

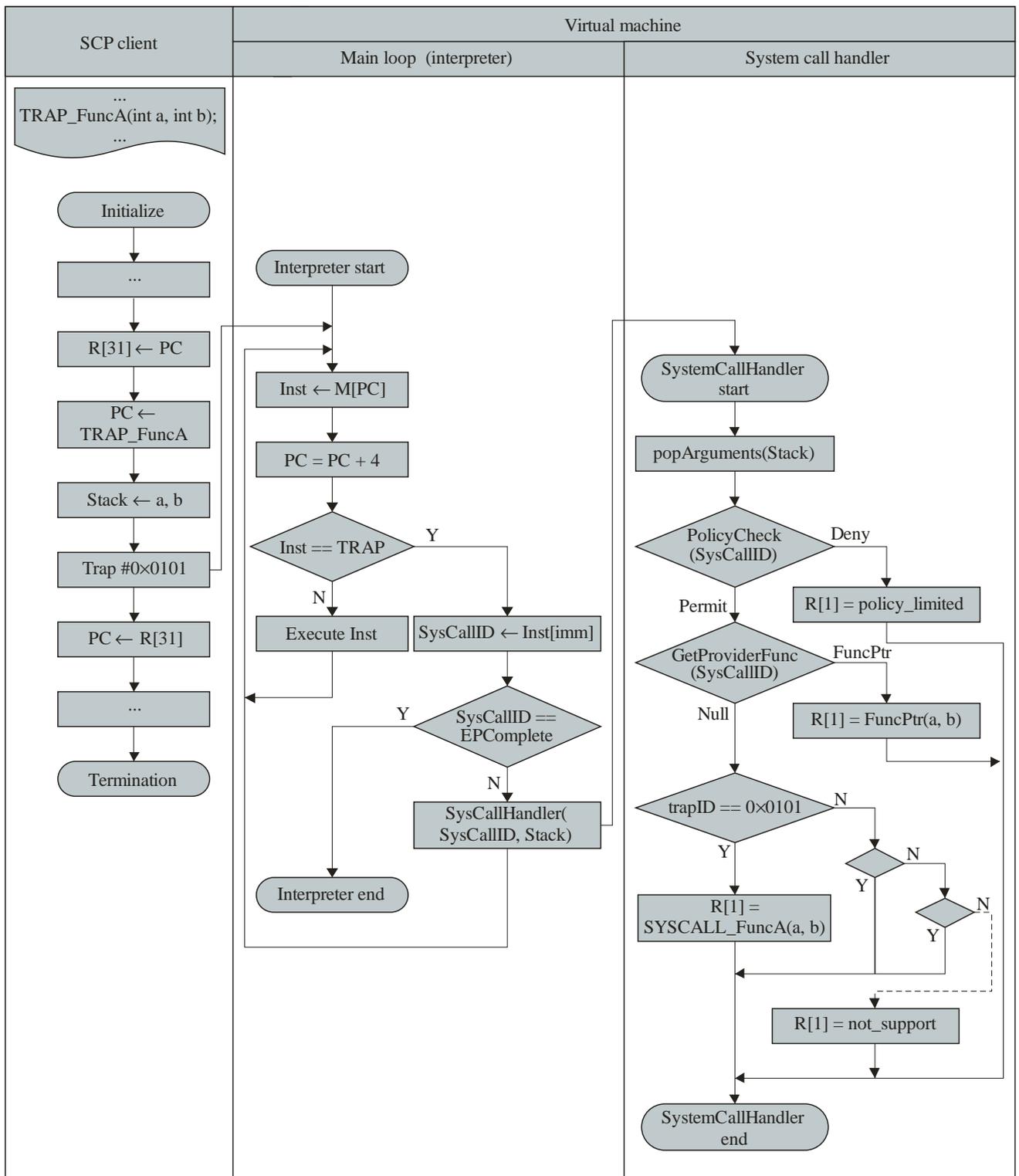
In this example, the policy monitor examines the policy data and retrieves the system call that can be accessible for each event. It may be implemented as a permit table. When an event has occurred, the SCP client is invoked and the virtual machine checks if the system call is allowed through the policy monitor.

## 7.6 System call

The SCP client can invoke a system call by executing a trap instruction after the SCP client pushes parameters into the stack as in the following sequence:

- [Interpreter] Store the current PC to register 31.
- [Interpreter] Push parameters of the "Trap" function into the stack in order.
- [Interpreter] Fetch and decode 'TRAP [system call ID]' instruction.
- [Interpreter] Call the system call handler with a system call ID and a stack address.
- [System Call Handler] Pop parameters of the system call from stack.
- [System Call Handler] Check if the system call is permitted by the policy monitor.
- [System Call Handler] Retrieve the pointer of the provided system call function from the provider. If a certain pointer is returned, perform the system call by the returned pointer, or perform a native system call function.
- [Interpreter] Restore PC with register 31 after the system call has finished.

Figure 10 illustrates the calling sequence and an implementation example of the SVM system call handler.



X.1198(13)\_F10

Figure 10 – Call sequence of system call

### 7.7 Event

An event causes a meaningful action of the media client for the virtual machine. The transition of the execution state is triggered by events, and how the event is handled in each state is defined in the SCP client.

An event of the virtual machine is similar to an external interruption of a generic computer. The SCP client registers the event handler in its loading and initialization phase through `SYSCALL_SetEventHandler`. When the player delivers an event to the virtual machine, it invokes the event handler of the SCP client.

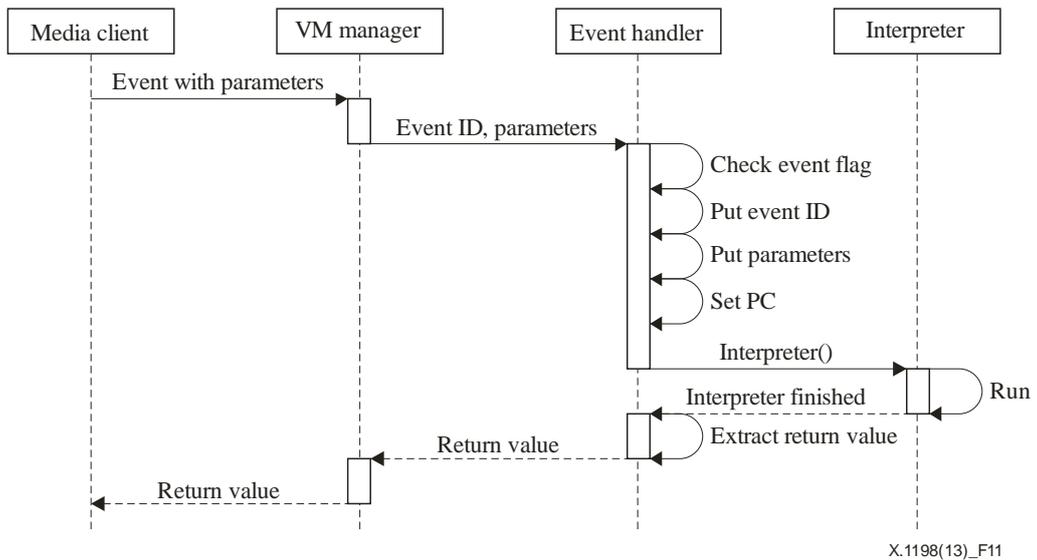
Event parameters are stored in the virtual machine memory which is registered as an event message buffer by `SYSCALL_SetEventParamBuffer`. Event parameters include an event ID for identifying each event, a return value for the result and various input values. When the event handling has finished, the event handler writes back the return value on the event parameter buffer.

The virtual machine cannot process two or more events simultaneously; any new incoming event will not be processed before the end of the current event handling. The event waiting mechanism is first-in-first-served.

Because only one event can be processed at a time, the event handler can use a semaphore or a mutex to prevent processing a new event during event handling.

An event can be handled according to the sequence shown in Figure 11:

1. Media client: The media client recognizes that the event has occurred.
2. Media client: The media client delivers the event notice and the parameters to the virtual machine.
3. Virtual machine manager: When the event signal is received, the virtual machine manager passes the event and its parameters to the event handler.
4. Event handler: Checks that the previous event is still running. If the previous event has not ended yet, it waits for the end of the previous event processing.
5. Event handler: Assigns the event ID of the given event as the 1st element of the event parameter buffer.
6. Event handler: Fills the event parameter buffer with the parameters of the given event. The third and latter elements are reserved for event parameters. The second element of the event parameter buffer is reserved for the return value.
7. Event handler: Sets the program counter to the event handler function which is assigned by `SYSCALL_SetEventHandler` of the SCP client.
8. Event handler: The event handler calls the interpreter.
9. Interpreter (SCP client): The interpreter runs the PC.
10. Event handler: After the interpreter has finished, the event handler extracts the return value of the event from the event parameter buffer, and returns it to the virtual machine manager.
11. Virtual machine manager: Returns the return value to the media client.
12. Media client: Obtains the result of the event process.



X.1198(13)\_F11

**Figure 11 – Event handling sequence**

## Annex A

### Data structure of SCP client package

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

The SCP client includes in this order a package header, a policy, a code and a signature. The header of the SCP client has 48 octets to indicate the type, version, data length (policy, code, signature, etc.) and the entry point of the attached code.

#### A.1 Header of SCP client

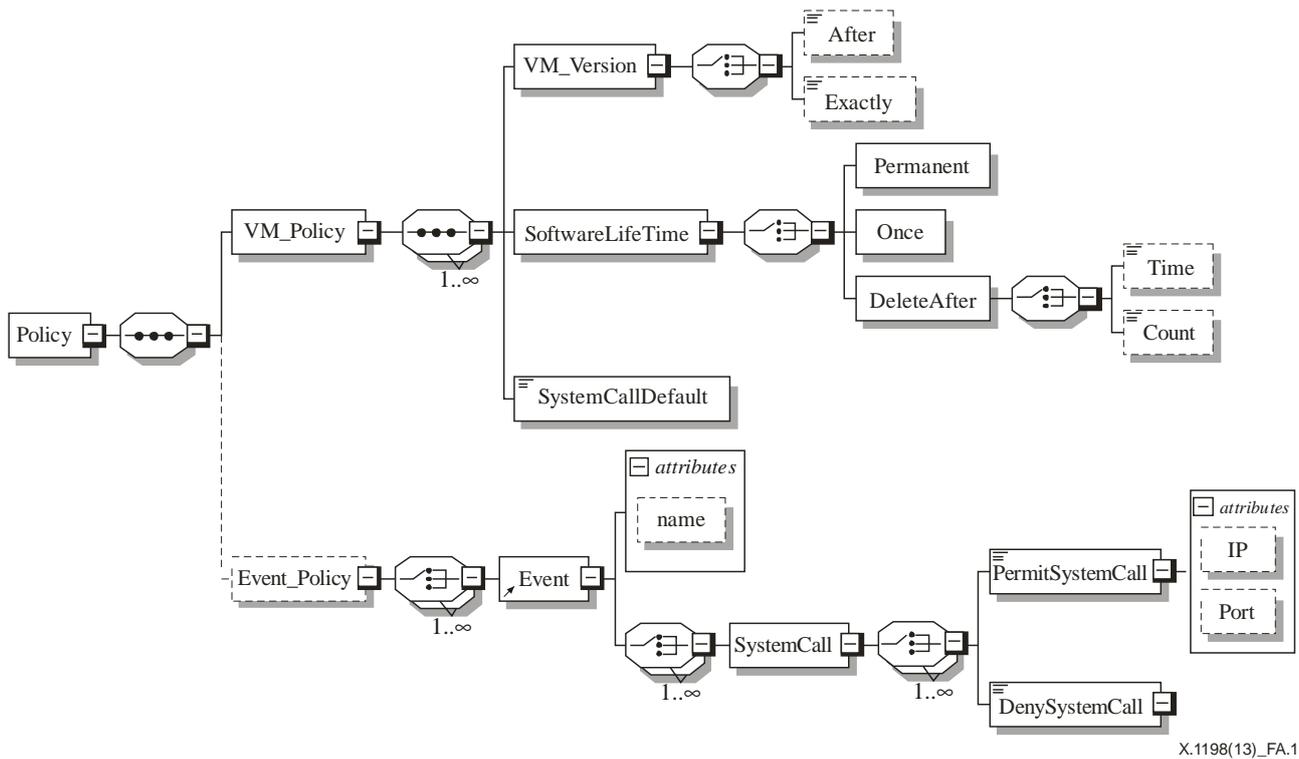
**Table A.1 –Header of SCP client**

|     | (msb)                         |   |   |   |   |   |   | (lsb) |
|-----|-------------------------------|---|---|---|---|---|---|-------|
| bit | 7                             | 6 | 5 | 4 | 3 | 2 | 1 | 0     |
| 0   | Prefix (8 octets)             |   |   |   |   |   |   |       |
| ... |                               |   |   |   |   |   |   |       |
| 7   |                               |   |   |   |   |   |   |       |
| 8   |                               |   |   |   |   |   |   |       |
| 9   | Version (2 octets)            |   |   |   |   |   |   |       |
| 10  | Type (2 octets)               |   |   |   |   |   |   |       |
| 11  |                               |   |   |   |   |   |   |       |
| 12  |                               |   |   |   |   |   |   |       |
| 13  | SCP Client ID (4 octets)      |   |   |   |   |   |   |       |
| 14  |                               |   |   |   |   |   |   |       |
| 15  |                               |   |   |   |   |   |   |       |
| 16  |                               |   |   |   |   |   |   |       |
| 17  | Super SCP ID (4 octets)       |   |   |   |   |   |   |       |
| 18  |                               |   |   |   |   |   |   |       |
| 19  |                               |   |   |   |   |   |   |       |
| 20  |                               |   |   |   |   |   |   |       |
| 21  | Policy Length (4 octets)      |   |   |   |   |   |   |       |
| 22  |                               |   |   |   |   |   |   |       |
| 23  |                               |   |   |   |   |   |   |       |
| 24  | Code Length (4 octets)        |   |   |   |   |   |   |       |
| 25  |                               |   |   |   |   |   |   |       |
| 26  |                               |   |   |   |   |   |   |       |
| 27  |                               |   |   |   |   |   |   |       |
| 28  | Signature Type (1 octet)      |   |   |   |   |   |   |       |
| 29  | Signature Length (3 octets)   |   |   |   |   |   |   |       |
| 30  |                               |   |   |   |   |   |   |       |
| 31  |                               |   |   |   |   |   |   |       |
| 32  | Object Code Offset (4 octets) |   |   |   |   |   |   |       |
| 33  |                               |   |   |   |   |   |   |       |
| 34  |                               |   |   |   |   |   |   |       |
| 35  |                               |   |   |   |   |   |   |       |
| 36  | Load Offset (4 octets)        |   |   |   |   |   |   |       |
| 37  |                               |   |   |   |   |   |   |       |
| 38  |                               |   |   |   |   |   |   |       |
| 39  |                               |   |   |   |   |   |   |       |
| 40  | Entry Point (4 octets)        |   |   |   |   |   |   |       |
| 41  |                               |   |   |   |   |   |   |       |
| 42  |                               |   |   |   |   |   |   |       |
| 43  |                               |   |   |   |   |   |   |       |
| 44  | Reserved (4 octets)           |   |   |   |   |   |   |       |
| 45  |                               |   |   |   |   |   |   |       |
| 46  |                               |   |   |   |   |   |   |       |
| 47  |                               |   |   |   |   |   |   |       |

- Prefix (8 octets): This represents the system type. For example, SCP\_VM\_K type can be represented as '0x5343505F635F4B52', which is 'SCP\_VM\_K' in ASCII text.
- Version (2 octets): The version of the SCP client.
- SCP client ID (4 octets): The identifier of the SCP client which is issued by the SCP client supplier.
- Type (2 octets): The type of the SCP client package:
  - 0x00: The SCP client contains uncompressed data.
  - 0x01: The SCP client contains compressed code and policy.
- Super SCP ID (4 octets): The identifier of the SCP client supplier.
- Policy length (4 octets): The length of the policy data. This is an unsigned integer value, for example, 0x1F4 or 500 octets of policy.
- Code length (4 octets): The length of the code data, unsigned integer value.
- Signature type (1 octet): The type of signature. The following values are defined:
  - 0x00: PKCS#1 RSASSA-PSS signature scheme with 1024-bit key and SHA-1,
  - 0x01: reserved.
- Signature length (3 octets): The length of the signature. This is an unsigned integer value.
- Object code offset (4 octets): Offset of the start point of the code in the SCP client package. This value is the addition of the header length and policy length.
- Load offset (4 octets): Offset of the loading point of the SVM memory. Normally 0.
- Entry point (4 octets): Offset of the initialization code which performs the registration of the event handler and the event parameter buffer.
- Reserved (4 octets): This should be 0x00000000.

## A.2 Policy data

Policy data contains XML data, which defines the security policy of each instruction. The length of this field is defined in the header.



X.1198(13)\_FA.1

**Figure A.1 – XML structure of policy**

### A.3 Code data

Code data contains the SCP client code. This is the series of instructions. The length of this field is defined in the header.

### A.4 Signature

Signature contains the digital signature data. The length of this field and the digital signature algorithm are defined in the header.

## Bibliography

- [b-ITU-T X.800] Recommendation ITU-T X.800 (1991), *Security architecture for Open Systems Interconnection for CCITT applications.*
- [b-ITU-T X.1191] Recommendation ITU-T X.1191 (2009), *Functional requirements and architecture for IPTV security aspects.*
- [b-ITU-T X.1252] Recommendation ITU-T X.1252 (2010), *Baseline identity management terms and definitions.*



## 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   |
| <b>Series X</b> | <b>Data networks, open system communications and security</b>                               |
| Series Y        | Global information infrastructure, Internet protocol aspects and next-generation networks   |
| Series Z        | Languages and general software aspects for telecommunication systems                        |