INTERNATIONAL STANDARD

ISO/IEC 13818-11

First edition
2004-02-01

Information technology — Generic coding of moving pictures and associated audio information —

Part 11: IPMP on MPEG-2 systems

Technologies de l'information — Codage générique des images animées et du son associé —

Partie 11: IPMP sur les systèmes MPEG-2

Reference number
ISO/IEC 13818-11:2004(E)

Contents

Foreword

ISO (the International Organization for Standardization) is a worldwide federation of national standards bodies (ISO member bodies). The work of preparing International Standards is normally carried out through ISO technical committees. Each member body interested in a subject for which a technical committee has been established has the right to be represented on that committee. International organizations, governmental and non-governmental, in liaison with ISO, also take part in the work. ISO collaborates closely with the International Electrotechnical Commission (IEC) on all matters of electrotechnical standardization.

International Standards are drafted in accordance with the rules given in the ISO/IEC Directives, Part 2.

The main task of technical committees is to prepare International Standards. Draft International Standards adopted by the technical committees are circulated to the member bodies for voting. Publication as an International Standard requires approval by at least 75 % of the member bodies casting a vote.

ISO/IEC 13818-11 was prepared by Joint Technical Committee ISO/IEC JTC 1, *Information technology*, Subcommittee SC 29, *Coding of audio, picture, multimedia and hypermedia information*.

ISO/IEC 13818 consists of the following parts, under the general title *Information technology — Generic coding of moving pictures and associated audio information*:

— *Part 1: Systems*

— *Part 2: Video*

— *Part 3: Audio*

— *Part 4: Conformance testing*

— *Part 5: Software simulation*

— *Part 6: Extensions for DSM-CC*

— *Part 7: Advanced Audio Coding (AAC)*

— *Part 9: Extension for real time interface for systems decoders*

— *Part 10: Conformance extensions for Digital Storage Media Command and Control (DSM-CC)*

— *Part 11: IPMP on MPEG-2 systems*

Information technology — Generic coding of moving pictures and associated audio information —

Part 11: IPMP on MPEG-2 systems

1 Scope

This International Standard specifies IPMP (Intellectual Property Management and Protection) on the MPEG-2 system, including:

a) syntax and semantics for IPMP control information which includes tool list, tool container and rights container;

b) syntax and semantics for IPMP descriptors, which facilitates IPMP protection signalling;

c) syntax and semantics of IPMP data extending from the common base class IPMP_Data_BaseClass to support the following functionalities:

- mutual authentication for IPMP tool to IPMP tool as well as IPMP tool to terminal communication,
- the requesting by IPMP tools of the connection/disconnection to requested IPMP tools,
- the notification to IPMP tools of the connection/disconnection of IPMP tools,
- common IPMP processing,
- IPMP tool to/from user interaction;

d) syntax and semantics for IPMP information carriage to and from IPMP tools;

e) syntax and semantics for the request and transfer of content and IPMP tools between terminals;

f) XML syntax and semantics for the description of the environment in which the MPEG-2 Terminal/application is operating;

g) a list of registration authorities required for the support of the specifications found herein.

This document is organized as follows.

Clause 1 provides an introduction to the document. Clause 4 explains the compatibility between the Conditional Access framework and MPEG-2 IPMP framework. Clause 5 provides an overview of the process supported by the IPMP framework, and identifies different normative elements in this process. Clause 6 provides specifications for components identified in Clause 5. Clause 7 provides specifications for the messaging architecture and all supported messages.

2 Normative references

The following referenced documents are indispensable for the application of this document. For dated references, only the edition cited applies. For undated references, the latest edition of the referenced document (including any amendments) applies.

ISO/IEC 10646-1:2000, *Information technology — Universal Multiple-Octet Coded Character Set (UCS) — Part 1: Architecture and Basic Multilingual Plane*

ISO/IEC 13818-1, *Information technology — Generic coding of moving pictures and associated audio information: Systems*

ISO/IEC 14496-1:2001, *Information technology — Coding of audio-visual objects — Part 1: Systems*

ISO/IEC 14496-13[1)], *Information technology — Coding of audio-visual objects — Part 13: Intellectual Property Management and Protection (IPMP) extensions*

XML Schema Part 0: Primer, Part 1: Structures, and Part 2: Datatypes, W3C Recommendation, 2 May 2001, available at <http://www.w3.org/TR/2001/REC-xmlschema-0-20010502>, <http://www.w3.org/TR/2001/REC-xmlschema-1-20010502>, and <http://www.w3.org/TR/2001/REC-xmlschema-2-20010502>, respectively

3 Terms and definitions

For the purposes of this document, the following terms and definitions apply.

3.1
Binary Representation
In the context of an IPMP Tool, this is the format of the implementation of that IPMP Tool. Examples: Platform Dependent Native Code, Java ™ bytecode

3.2
Content
This implies part or whole of an MPEG presentation

3.3
Content Consumption
Any experience of given Content implies consumption of that content. Access, Playback, Denial of Access and Creation of a Copy are all types of content consumption

3.4
Content Stream
This is the incoming content, of MPEG-2 system

3.5
IPMP Device
An implemented application that implements an MPEG-2 Terminal supporting the use of MPEG-2 IPMP

3.6
IPMP Information
Information directed to a given IPMP Tool to enable, assist or facilitate its operation

3.7
IPMP Tool
IPMP tools are modules that perform (one or more) IPMP functions such as authentication, decryption, watermarking, etc. A given IPMP Tool may coordinate other IPMP Tools

1) To be published.

3.8
IPMP Tool Identifier
This refers to the IPMP Tool ID. It identifies a Tool in an unambiguous way, at the presentation level or at a universal level

3.9
IPMP Tool List
The IPMP Tool List identifies, and enables selection of, the IPMP Tools required to process the Content

3.10
IPMP Tool Manager
The IPMP Tool Manager is a conceptual entity within the Terminal that processes IPMP Tool List(s) and retrieves the Tools that are specified therein

3.11
IPMP Tool Message
A message passed between any combination of IPMP Tool or Terminal

3.12
Message Router
A conceptual entity within the Terminal that implements the Terminal-side behaviour of the Terminal-Tool interface

3.13
Mutual Authentication
Protocols carried out to determine the proper and correct identity of a communicating partner and the securing of communication channels between communicating partners

3.14
Parametric Description
Information that enables a Terminal to choose a specific Tool implementation that will support all functionality required by a presentation

3.15
Renewability
Renewability implies that a Terminal may acquire Missing IPMP Tools, or update an existing IPMP Tool with an updated version

3.16
Representation Format
The binary format, platform and communication mechanisms applicable to a given implementation of an IPMP Tool or Terminal

3.17
Scope of Protection
Scope of protection refers to the elementary stream and/or program governed by a given IPMP Tool instance

3.18
Secure Communication
Communication between two entities is said to be secure to a sufficient level when the communication between the two entities can not be attacked by being altered, removed, forged or evesdropped on to the levels required by the two entities

3.19
Terminal
A Terminal is an environment that consumes possibly protected content in compliance with the usage rules

3.20
User
A hardware, software or human entity that is the initiator and/or target of content consumption

3.21
Verification
Establishing, to an acceptable level of certainty, that a certain condition that was claimed to be true is true

4 Compatibility with Conditional Access framework (Informative)

4.1 The existing MPEG-2 CA framework

MPEG-2 systems provides a framework for the protection of content carried in Transport Streams and Program Streams. In the case of Transport Streams, the framework is based on these three distinct elements:

- A 2 bit **transport_scrambling_control** field in the TS packet header that is used to notify the receiver that the contents of the TS packet are scrambled. In practice, these bits may identify which of two "control words" is used as the scrambling key for the packet. It is left to the CA vendor to manage the transport and availability of the control words in the receiver. The same field is present in the PES packet header and is used to signal scrambling at the PES level.

- Conditional Access descriptors are located in Program Map sections to identify the content streams which are protected by the CA system and the streams carrying the ECM private data. These descriptors may optionally carry CA parameters associated with the protected content streams.

- The Conditional Access sections carry further Conditional Access descriptors that identify the Conditional Access system(s) associated with the entire Transport Stream and notify the receiver of the streams containing EMM private data.

For Program Streams, the following framework is defined:

- A 2 bit **PES_scrambling_control** field in the PES packet header that is used to notify the receiver that the contents of the PES packet are scrambled. In practice, these bits may identify which of two "control words" is used as the scrambling key for the packet. It is left to the CA vendor to manage the transport and availability of the control words in the receiver.

- Conditional Access descriptors are located in the Program Stream Map to identify the content streams which are protected by the CA system and the streams carrying the ECM private data. These descriptors may optionally carry CA parameters associated with the protected content streams.

- The Program Stream Map carry further Conditional Access descriptors that identify the Conditional Access system(s) associated with the entire Transport Stream and notify the receiver of the streams containing EMM private data.

These elements are fully described in ISO/IEC 13818-1.

4.2 Backward compatibility – the carriage of IPMP in the CA framework

In order to allow CA systems to recognize the presence of IPMP protection of content in Transport and Program Streams, the CA framework described above must be used to signal the presence of IPMP protection. IPMP enabled receivers may choose to locate IPMP streams using either the CA framework or the traditional IPMP mechanisms described elsewhere in this document. Systems designed to work only with the CA framework will be able to identify the use of a protection mechanism that renders the content inaccessible and deal with it accordingly.

The presence of IPMP protection is identified using the Conditional Access descriptor defined in ISO/IEC 13818-1 and which is carried in the Conditional Access sections and Program Map sections within Transport Streams and in the Program Stream Map in a Program Stream.

The Conditional Access descriptor is duplicated below and the relevant fields amended below the table to show how IPMP presence is signalled in the descriptor.

Syntax	No. of bits	Mnemonic
CA_descriptor() {		
descriptor_tag	**8**	**uimsbf**
descriptor_length	**8**	**uimsbf**
CA_system_ID	**16**	**uimsbf**
Reserved	**3**	**bslbf**
CA_PID	**13**	**uimsbf**
For (i=0; i<N; i++) {		
private_data_byte	**8**	**uimsbf**
}		
}		

CA_system_ID – This is a 16 bit field indicating the type of CA system applicable for either the associated ECM and/or EMM streams. The coding of this is privately defined and is not specified by ITU-T | ISO/IEC.

CA_PID – This is a 13 bit field indicating the PID of the Transport Stream packets which shall contain either ECM or EMM information for the CA systems as specified with the associated CA_system_ID. The contents (ECM or EMM) of the packets indicated by the CA_PID is determined from the context in which the CA_PID is found, i.e. a TS_program_map_section or the CA table in the Transport Stream, or the stream_id field in the Program Stream.

For Transport Streams, the CA_PID value of 0x0003 shall be used to indicate that there is IPMP protection within the system, since this PID is reserved for the IPMP Control Information Table.

4.3 Forward compatibility – the carriage of CA data within MPEG-2 IPMP

If a CA system is required to have data transported in MPEG-2 IPMP, an IPMP Tool ID should be registered. If the CA system is used to descramble the scrambled elementary stream, its tool ID should be included in the IPMP Tool List in IPMP Control Information.

There should be an IPMP Descriptor [6.3.3] created for this CA system. The IPMP Tool ID shall be set to the IPMP Tool ID registered. The control point should be set to 0x01, being the Control Point after Transport Buffer TB_n in STD if the content is scrambled at Transport Stream level and at control point 0x02, being the Control Point between the decode buffer (B_n or ES_n) and the decoder (D_n) in STD in the case of PES layer scrambling for Program or Transport Streams..

The ECMs and EMMs can be carried in either IPMP Descriptors or IPMP_StreamDataUpdate in IPMP Streams [6.4], in the form of IPMP_OpaqueData [7.5.2] extending from IPMP_Data_BaseClass, This information will then be routed by the terminal to the IPMP Tool (CA system) with the specified IPMP Tool ID.

In this way, an IPMP compliant terminal is able to set up the CA system at the indicated Control Point and descrambling can take place as it does in Conditional Access systems that otherwise employ the CA framework.

4.4 Co-existence of MPEG-2 IPMP and Conditional Access

A terminal can be designed in such a way as to allow the co-existence of MPEG-2 IPMP and Conditional Access systems and content as the two systems are able to operate independently or in cooperation with one another.

Content can be protected by both Conditional Access systems and MPEG-2 IPMP. Doing this means having hybrid content with IPMP Descriptors, CA descriptors, ECMs, EMMs, IPMP Stream, IPMP Control Information Table and Conditional Access Table.

The presence of CA_Descriptors in the CAT and/or Program Map sections (or the Program Stream Map in the case of Program Streams) that use a CA_system_id supported by the terminal (and CA_PID not equal to 0x0003) indicates that the stream may be received and decoded by a terminal regardless of any IPMP functions that are applied to the content. If the originator of the stream wishes to enforce the application of IPMP tools, then only CA_PID of 0x0003 should appear in any CA_descriptors applicable to that stream. In this case, conventional CA is achieved by using an IPMP Tool ID as described in 4.3 above.

There are at least two example scenarios envisaged.

- A CA system with a descrambler running at the TS control point, together with an IPMP watermarking tool running after the decoder, governed by an IPMP rights management tool to provide extended functionality and security beyond the CA system.
- Another example would be signalling the method for descrambling content using either a CA system or an IPMP Tool by the use of alternative tools in the tool list, if both of them are operating at the same control point 0x01 (or 0x02 in the case of PES level scrambling).

Handling this hybrid content demands a terminal enabled with both CA system and IPMP. Designing this type of terminal is feasible, but outside the scope of this specification.

5 Overview of MPEG-2 IPMP (Informative)

5.1 IPMP Architecture

This clause describes an IPMP architecture that may operate in cooperation with or independently of the IPMP specifications.

5.1.1 IPMP tool acquisition and protection signaling

The IPMP Tool Manager is a conceptual entity in a given IPMP Terminal that manages tools.

IPMP Control Information [6.2] carries IPMP Tool List Class [6.2.1.3.1] which conveys the list of IPMP tools required to access the content protected by IPMP, and may include a list of alternate IPMP tools or parametric descriptions of tools required to access the content. The conceptual entity Tool Manager parses the tool list, makes sure all tools are available, and retrieves missing tools if any.

IPMP Control Information may also carry IPMP Tool Container Class [6.2.1.3.8] which enables carriage of IPMP tool inside MPEG-2 system. Tool Manager may retrieve the IPMP tool from the content stream, load it, instantiate it and immediately use it in order to play out the content.

A missing IPMP Tool can also be acquired from a neighbouring IPMP device via tool transfer messages defined in Annex A, or it can be acquired by Tool Manager by connecting to a remote server and providing the terminal description as defined in Annex B.

The entire IPMP protection scheme can be described by embedding IPMP Descriptor’s [6.3.3] into Program Map Table (PMT) in Transport stream or Program Stream Map (PSM) in Program Stream. It associates IPMP tool with each individual stream under its protection. It also indicates the control point at which a specific IPMP tool should be running.

The presence of IPMP Descriptor within a program loop but outside the elementary stream loop indicates that all elementary streams of this program is protected by tools given in the IPMP Descriptor.

The presence of IPMP Descriptor within the loop of an elementary stream indicates that the given elementary stream is protected by tools given in the IPMP descriptor.

5.1.2 IPMP Information carriage in MPEG-2 Content

IPMP Information can be carried in MPEG-2 content, which will then be directed to a given IPMP Tool to enable, assist or facilitate its operation.

All IPMP Information should be derived from IPMP_Data_BaseClass [7.1]. Syntax and semantics are defined for carriage of opaque data, tool initialization related data, etc. IPMP Information can be carried in either IPMP Stream [6.4] or IPMP Descriptors [6.3.3].

When IPMP Information is carried in IPMP Descriptor, the destination tool to which the IPMP Information should be directed is specified by the Tool ID specified therein. When IPMP Information is carried in IPMP Stream as a payload of IPMP_StreamDataUpdate [6.4.2], the destination tool is indirectly specified by the IPMP_Descriptor_ID specified therein.

5.1.3 Messaging

To facilitate the cooperation of multiple tools in the protection and governance of content, a message based architecture is provided. The message based architecture has three advantages over functional interface type architectures. The first is that security can more easily be maintained as messages are less difficult to protect in an open framework then parameters in a function parameters list. The second is that the only entities that need be concerned with a given message's definition are those that need to generate or act upon a given message and so additional functionality can be created and supported simply through the addition of required messages. The third is that full interoperability with IPMP tools can be easily achieved by defining a single messaging API by a third-party forum who adopts IPMP.

Physical routing of information is handled by a conceptual entity called the Message Router. The Message Router abstracts all platform-dependent routing and delivery issues, from the IPMP Tools. The interface between the Message Router and the tools is non-normative and is not defined in this specification.

All IPMP Tool interactions take place via the Terminal. IPMP Tools do not communicate directly with each other within the scope of this standard.
The delivery of both bit stream sourced IPMP information as well as IPMP tool and Terminal generated information is supported through the use of three separate messages which are passed via the Message Router to IPMP tools. The three messages are, IPMP_MessageFromBitstream [7.7.2], which is used to deliver IPMP stream data, IPMP_DescriptorFromBitstream [7.7.3], which is used to deliver IPMP Descriptors [6.3.3] and IPMP_MessageFromTool [7.7.4], which is used to deliver messages from either other IPMP tools or the Terminal itself.

5.1.4 Mutual Authentication

The most important aspect of a secure messaging architecture is the use of cryptographic algorithms and protocols that allow one to perform a number of important security functions.

At any point in IPMP Information or content processing, IPMP Tools may be required to communicate with one another or the Terminal. The degree of security required for such communication is determined by a number of variables including information that may be included by the content provider in the Content and conditions of trust established between tool providers a priori and out of band. It is generally the case that a given elementary stream is protected by multiple tools but that certain types of tools are complex (e.g. Rights Management tools) and others are utilities (e.g. Decryption engines). Complex tools may control the instantiation of other tools or make decisions about content use in response to usage queries from the terminal. Mutual authentication may occur between any pair of tools but the level of security required for this communication will in part be dictated by data contained in the bitstream in an opaque manner. The mechanism for making the determination of this security level is non-normative.

Mutual authentication is executed as follows:

1. The Tool that initiates mutual authentication with another tool determines the conditions of trust to be achieved by such authentication, i.e. the initiating tool determines whether it needs integrity protected communication or full secure, authenticated communication. This level may or may not be dictated by IPMP Information in the Content.

2. The communicating tools then engage in a message exchange to determine which authentication protocol will be used. In some cases, this protocol will have been determined by an a priori out of band negotiation between the tool providers in their security audits of one another.

Messages are defined to support the identified authentication functionalities. The first message IPMP_InitMutualAuthentication [7.2.1] may be used and delivered to a given IPMP tool such that the receiving IPMP tool is informed as to its required communication partner as well as security measures that must be in place. Following this message or the absence thereof, IPMP tools required to do so will use the IPMP_MutualAuthentication [7.2.2] message as required to determine or create secure channels of communication as needed based on the application.

As one purpose of Mutual Authentication is the verification of trust relationships existing between two entities these specifications provide for the carriage of trust and security metadata. This metadata may include zero or more certificates, credentials or integrity verification information. The creation or establishment of trust relationships are established by out of band relationships between the different entities involved in protecting and managing the content. However, the trust metadata that results from such relationships needs to be made available to permit static and dynamic verification of trust. IPMP_TrustSecurityMetadata [7.2.3] is defined to provide these additional security related data usable by IPMP tools to determine trust related information.

Once Mutual Authentication is performed, the IPMP_SecureContainer [7.2.5] may be used to pass information securely between IPMP tools and IPMP tools and Terminal.

5.1.5 IPMP Tool connection and disconnection

In the IPMP architecture, IPMP tool may be connected as the result of the presence of IPMP Descriptor in PMT or PSM and in addition may be connected due to requests from already connected IPMP tools.

Instantiation of the Tools to be connected is implementation dependent. IPMP tools may use the IPMP_GetTools [7.3.1] and IPMP_GetToolsResponse [7.3.2] messages to request a list of tools available for connection that exist as well as to respond to the request, respectively.

IPMP tools as well as the Terminal may also query a given IPMP tool as to its capabilities and functionality by using the IPMP_ToolParamCapabilitiesQuery [7.3.3] message with the tool being queried using the IPMP_ToolParamCapabilitiesResponse [7.3.4] message as a reply.

Knowing that a given tool is needed for processing, an IPMP tool may request the connection of another IPMP tool by using the IPMP_ConnectTool [7.3.5] message and may request the disconnection of another IPMP tool by using the IPMP_DisconnectTool [7.3.6] message. A connection may require the actual instantiation of a tool or may be accomplished through physical/electronic means.

The IPMP_ConnectTool message contains control point and sequence information to determine the exact location to connect the requested tool. Tools connected at the request of other tools inherit the same scope of protection as the requesting tool. Note that some control points are not associated with any known point on an Elementary Stream.

Each instantiation of an IPMP Tool shall establish a new logical instance of the tool, for a particular scope of protection. This logical instance can be uniquely identified using the IPMP_Descriptor_ID specified in the IPMP Descriptor which causes the instantiation. The binding between IPMP_Descriptor_ID and the logical instance of a tool ensures unambiguous message addressing.

The instantiation of an IPMP Tool should also resolve in a link between the MR and the Tool instance.

The tool requesting connection shall receive an IPMP_NotifyToolEvent [7.4.3] message indicating the instantiation of the IPMP Tool. The requesting tool and the instantiated tool may perform mutual authentication thereafter.

5.1.6 Notification of IPMP Tool connection and disconnection

During the processing of IPMP protected content, a number of IPMP tools may be involved, for communication and various security purposes, notification messages are supplied to notify IPMP tools when other IPMP tools have either been connected, disconnected or processed watermark information. Additionally IPMP tools may request at any time a list of all the IPMP tools currently connected at various specifiable scopes of protection.

The IPMP_AddToolNotificationListener [7.4.1] message may be used to indicate the sending IPMP tools intent to receive notifications of events and scopes of protection as specified in the IPMP_AddToolNotificationListener message. To remove oneself from being notified in the future for specified events, an IPMP tool may use the IPMP_RemoveToolNotificationListener [7.4.2] to do so.

As events occur for which notifications have been requested, the IPMP_NotifyToolEvent [7.4.3] message is sent to requesting IPMP tools.

5.1.7 Common IPMP processing

Direct support has been provided for the carrying out of common IPMP processing operations. Specific support and the related messages are as follows:

— IPMP_CanProcess [7.5.1] for the notification by an IPMP tool to the Terminal that stream processing may begin. All tools connected and processing data within a given stream must send permission before any tools in the stream receive stream data;

— IPMP_Opaquedata [7.5.2] for the carriage of user defined data;

— IPMP_KeyData [7.5.3] for the carriage of decryption key data as well as timing information to determine the validity period of time varying keys;

— IPMP_RightsData [7.5.4] for the carriage of rights expressions;

— IPMP_SelectiveDecryptionInit [Annex C] for the configuration of a decryption tool;

— IPMP_AudioWatermarkingInit [Annex D] for the configuration of an audio watermarking tool;

— IPMP_SendAudioWatermark [Annex D] for the sending of information to be embedded or that was extracted;

— IPMP_VideoWatermarkingInit [Annex E] for the configuration of a video watermarking tool;

— IPMP_SendVideoWatermark [Annex E] for the sending of information to be embedded or that was extracted.

5.1.8 IPMP tool to/from User interaction

During IPMP processing, direct interaction between IPMP tools and a User may be required. The IPMP_UserQuery [7.6.1] message is used to provide information to be relayed to a User and to request information as well. The IPMP_UserQueryResponse [7.6.2] is used to relay information provided by a User back to the originator of the User query.

6 Specifications (Normative)

6.1 Overview

6.1.1 MPEG-2 IPMP architecture

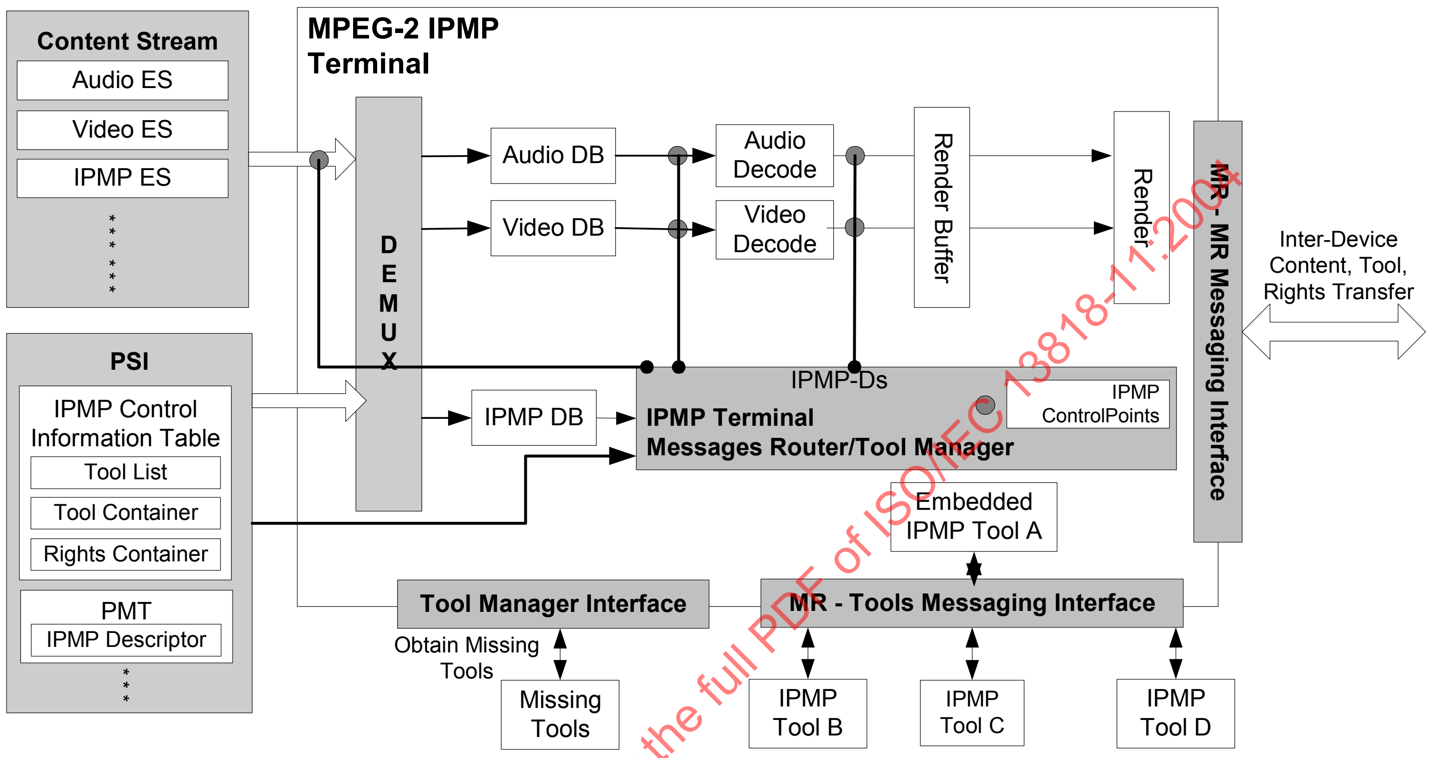

Figure 1 — MPEG-2 IPMP Architecture

The mapping of this diagram to a simplified use domain is shown in the informative Annex F.

6.1.2 Structure of IPMP protected MPEG-2 content

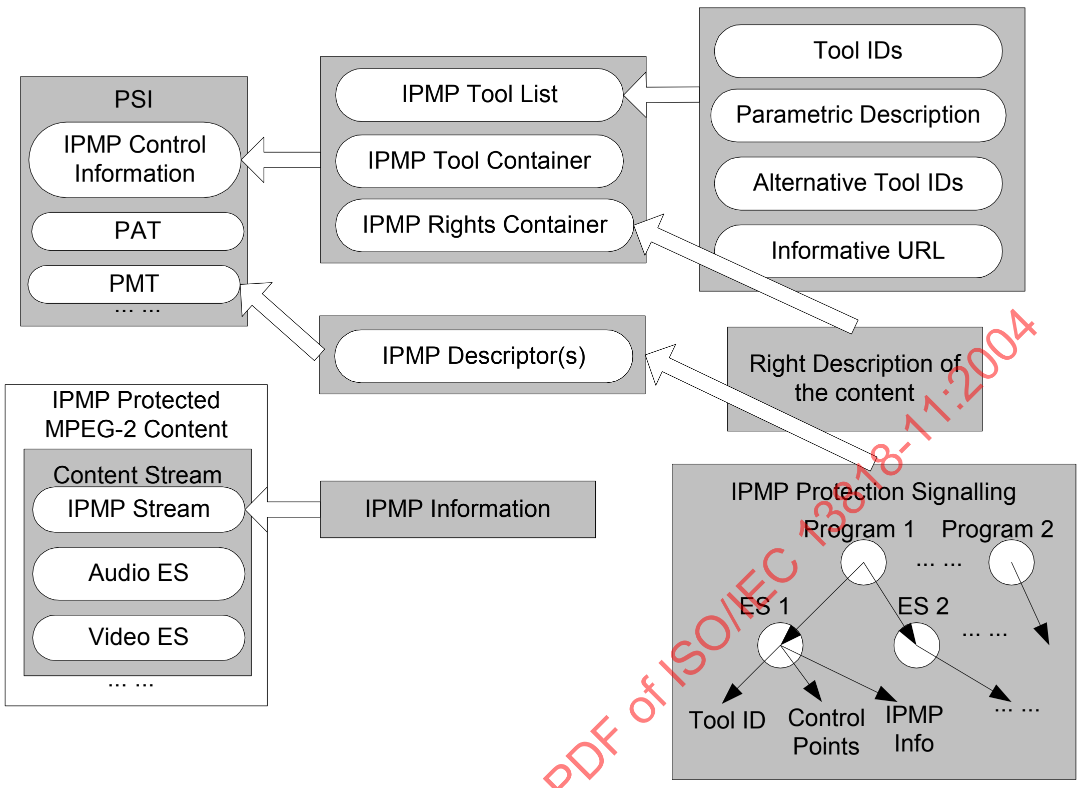

Figure 2 — Structure of IPMP protected MPEG-2 content

6.2 IPMP Control Information

IPMP Control Information contains information including Tool List, IPMP Rights Container and Tool Container. The IPMP Tool List identifies and enables selection of, the IPMP Tools required to process the Content. Tool Container enables the carriage of binary tool in content streams. The IPMP Rights Container contains rights description that describes usage rules associated with the IPMP protected content.

In short, the IPMP Control Information describes the IPMP tools needed to play the content, and the usage rights associated with the content. It could possibly carry the binary tool itself.

The entire IPMP Control Information can be optionally signed to ensure its integrity.

In Transport Stream, it exists in the form of the IPMP Control Information Table. While in Program Steam, it exists as one or more PES packets where the stream_id is extended_stream_id 0xFD and stream_id_extension of 0b000 0000, as defined in ISO/IEC 13818-1.

6.2.1 IPMP Control Information in Transport Stream

IPMP Control Information Table exists in PSI. The PID assigned to IPMP Control Information Table is 0x03 as defined in ISO/IEC 13818-1. The IPMP Control Information table may be segmented into one or more sections, before insertion into Transport Stream packets, with the following syntax.

6.2.1.1 Syntax of IPMP Control Information Section in Transport Stream

Syntax	No. of bits	Mnemonic
IPMP_Control_Info_section() {		
table_id	8	**Uimsbf**
section_syntax_indicator	1	**Bslbf**
'0'	1	**Bslbf**
Reserved	2	**Bslbf**
section_length	12	**uimsbf**
Reserved	2	**bslbf**
IPMP_control_info_version	5	**uimsbf**
current_next_indicator	1	**bslbf**
section_number	8	**uimsbf**
last_section_number	8	**uimsbf**
IPMP_control_info_classes_total_len	16	**uimsbf**
IPMP_control_info_classes_len_in_this_section	16	**uimsbf**
for (i=0; i<N;i++) {		
IPMP_control_info_class()		
}		
If (section_number==last_section_number) {		
IsSigned	1	**bslbf**
Reserved	7	**bslbf**
if (isSigned)		
Signature		**ByteArray**
NumCerts	8	**uimsbf**
for (i=0; i<numCerts;i++) {		
CertType	8	**uimsbf**
Certificate		**ByteArray**
}		
Verifying_Tool_ID	128	**uimsbf**
}		
}		
CRC_32	32	**rpchof**
}		

6.2.1.2 Semantics

table_id -- This is an 8 bit field, which shall be always set to 0x07.

section_syntax_indicator -- The section_syntax_indicator is a 1 bit field which shall be set to '1'.

section_length -- This is a twelve bit field. It specifies the number of bytes of the section, starting immediately following the section_length field, and including the CRC. The value in this field shall not exceed 4093.

IPMP_control_info_version -- This 5 bit field is the version number of the whole IPMP Control Information Table. The version number shall be incremented by 1 modulo 32 when a change in the information carried within the IPMP control info table occurs. When the current_next_indicator is set to '1', then the version_number shall be that of the currently applicable IPMP Control Information Table. When the current_next_indicator is set to '0', then the version_number shall be that of the next applicable IPMP Control Information Table.

current_next_indicator -- A 1 bit indicator, which when set to '1' indicates that the IPMP Control Information Table sent is currently applicable. When the bit is set to '0', it indicates that the IPMP Control Information Table sent is not yet applicable and shall be the next IPMP Control Information Table to become valid.

section_number -- This 8 bit field gives the number of this section. The section_number of the first section in the IPMP Control Information Table shall be 0x00. It shall be incremented by 1 modulo 256 with each additional section in the IPMP Control Information Table.

last_section_number -- This 8 bit field specifies the number of the last section (that is, the section with the highest section_number) of the IPMP Control Information Table.

IPMP_Control_Info_classes_total_len – This 16 bit field specifies the total length of the IPMP Control Information Classes, which may be carried in more than one IPMP Control Information sections.

IPMP_Control_Info_classes_len_in_this_section – This 16 bit field specifies the length of the IPMP Control Information immediately following this field in this section.

IPMP_Control_Info_class – The IPMP Control Information Classes, including Tool List, Tool Container, Rights Container. Since the length of IPMP Control Information class could possibly exceeds 4093, it may split into more than one IPMP Control Information sections. When the IPMP Control Information sections are assembled back to form the complete IPMP Control Information Table, the IPMP Control Information classes can be assembled back as well, with help of IPMP_Control_Info_classes_total_len.

isSigned – This 1 bit field indicates the presence of a signature in the IPMP Control Information table.

Signature – The signature of the entire IPMP Control Information classes including Tool List, IPMP Rights Container and Tool Container. ByteArray is defined in [7.2.2.4].

CertType – The type of certification mechanism being used, value assigned by a Registration Authority.

NumCerts – The number of certificates included.

Certificate -- The array of certificates.

Verifying_Tool_Id – The ID of the Tool that is required to verify the certificate(s). A value of 0 indicates the device.

CRC_32 -- This is a 32 bit field that contains the CRC value that gives a zero output of the registers in the decoder defined in Annex B in ISO/IEC 13818-1 after processing the entire IPMP Control Information section.

6.2.1.3 IPMP Control Information Classes

The IPMP Control Information includes the following IPMP Control Information Classes, with different IPMP Control Information class tags.

IPMP Control Information Class Tag	Class Name
0x00	Forbidden
0x01	IPMP Tool List Class
0x02	IPMP Tool Info Class
0x03	IPMP Tool Container Class
0x04	IPMP Rights Container Class
0x05	IPMP Parametric Description Class
0x06-0xC0	ISO/IEC 13818-1 Reserved
0xC1-0xFE	User private
0xFF	Forbidden

6.2.1.3.1 IPMP Tool List Class

The IPMP Tool List Class includes a list of IPMP tools. It is used to specify all IPMP tools that should be used in order to play back the content.

Syntax	No. of bits	Mnemonic
IPMPToolList () {		
Control_info_class_tag	**8**	**uimsbf**
Length	**16**	**uimsbf**
NumTools	**8**	**uimsbf**
for (i=0; i<numTools; i++) {		
IPMPTool_Info ()		
}		
}		

6.2.1.3.2 Semantics of fields in IPMP Tool List Class

Control_info_class_tag – this 8 bit field should be set to 0x01 to indicate this is an IPMP Tool List Class.

Length – number of bytes of this class, starting immediately following this "Length" field.

NumTools – This 16 bit field specified how many tools are specified in this IPMP Tool List Class.

IPMPTool_Info – This class carries information about a tool including its tool ID, possible alternate tools, etc. It is defined in the following subclause.

6.2.1.3.3 IPMP Tool Info Class

IPMPTool_Info_Class contains information for a logical IPMP Tool required by the Device. The logical tool may be one of the following:

1) A vendor-specific IPMP Tool specified by IPMP_ToolID,

2) One of a list of alternate IPMP Tools,

3) An IPMP Tool specified by a parametric description.

Syntax	No. of bits	Mnemonic
IPMPTool_Info () {		
Control_info_class_tag	**8**	**uimsbf**
Length	**16**	**uimsbf**
IPMP_ToolID	**128**	**uimsbf**
isAltGroup	**1**	**uimsbf**
isParametric	**1**	**uimsbf**
Reserved (0b111111)	**6**	**uimsbf**
if (isAltGroup) {		
NumAlternatives	**8**	**uimsbf**
for (i=0; i< numAlternatives; i++) {		
Specific_Tool_ID	**128**	**uimsbf**
}		
}		
if (isParametric) {		
IPMP_ParametricDescription		
}		
NUMURLS	**8**	**uimsbf**
ToolURL[numURLs]		**ByteArray**
}		

6.2.1.3.4 Semantic definitions of fields in IPMP Tool Info Class

Each IPMPTool_Info_Class identifies one IPMP Tool that is required by the terminal to consume the content. This Tool shall be specified either as a unique implementation, as one of a list of alternatives, or through a parametric description.

A unique implementation is indicated by the isAltGroup and isParametric fields both set to zero. In this case, the IPMP_ToolID shall be from the range reserved for specific implementations of an IPMP Tool and shall directly indicate the required Tool.

In all other cases, the IPMP_ToolID serves as a Content-specific abstraction for an IPMP Tool ID since the actual IPMP Tool ID of the Tool is not known at the time of authoring the Content, and will depend on the Terminal implementation at a given time for a given piece of Content.

A parametric description is indicated by setting the isParametric field to one. In this case, the Terminal shall select an IPMP Tool that meets the criteria specified in the following parametric description. In this case, the IPMP_ToolID shall be from the range reserved for Parametric Tools or Alternative Tools. The actual IPMP Tool ID of the Tool that the terminal implementation selects to fulfill this parametric description is known only to the Terminal. All the Content, and other tools, will refer to this Tool, for this Content, via the IPMP_ToolID specified.

A list of alternative Tools is indicated by setting the isAltGroup flag to ”1”. The subsequent specific ToolIDs indicate the Tools that are equivalent alternatives to each other. If the isParametric field is also set to one, any Tool that is selected under the conditions for parametric tools (as discussed in the paragraph above) shall be considered by the Terminal to be another equivalent alternative to those specified via specific ToolIDs. The Terminal shall choose one from these equivalent alternatives at its discretion. The actual IPMP Tool ID of this Tool is known only to the Terminal.

Control_info_class_tag – this 8 bit field should be set to 0x02 to indicate this is an IPMP Tool Info Class.

Length – number of bytes of this class, starting immediately following this “Length” field.

IPMP_ToolID – the identifier of the IPMP Tool.

isAltGroup – if set to one, this IPMP_Tool contains a list of alternate IPMP Tools.

numAlternates – the number of alternative IPMP Tools specified in IPMP_Tool.

Specific_Tool_ID – an array of the IDs of specific alternative IPMP Tools that can allow consumption of the content.

isParametric – IPMP_Tool contains a parametric description of an IPMP Tool. In this case, IPMP_ToolID is an identifier for the parametrically described IPMP Tool, and the Terminal shall route information specified in the bitstream for IPMP_ToolID to the specific IPMP Tool instantiated by the terminal.

IPMP_ParametricDescription -- IPMP Parametric Description as defined in the following subclause.

ToolURL – An array of numURL informative URLs from which one or more tools specified in this class may be obtained.

6.2.1.3.5 IPMP Parametric Description Class

Using a parametric description, the content provider can now describe what type of IPMP tool is required to playback the content, instead of using fixed tool IDs. For example, the content provider can specify that an AES tool, with block size of 128 bits is required to decrypt video stream. The IPMP terminal, upon receiving such description specifying this tool, can then choose an optimised AES tool from the embedded tools.

This clause contains an illustration of the hierarchy that a parametric description would follow. It does not attempt to define any specific scheme for any specific Tool type. It is anticipated that such definitions will be added over time to the overall scheme as a need is identified and an optimal schema developed. We anticipate that only a basic framework will appear in the current version of the specification, and enhancements to the same will be left for future addendums and/or versions.

- Optional comment
- Version of parametric description syntax
- Class of Tool
- e.g. Decryption, Rights Language Parser
- Sub-class of Tool
  - e.g. for Decryption: AES, DES, NESSIE etc
  - e.g. for Watermarking: “Panos’s watermarking tool” etc
  - e.g. for Rights Language Parser: “Fred’s Rights Parser”
  - e.g. for Protocol Parser: “Mary’s Protocol Parser”
- Sub-class-specific information
  - e.g. for DES: number of bits, stream and/or block decipher capability
  - e.g. for Rights Language Parser : version

The parametric description is defined to allow a generic description of any type of IPMP tool, no matter the type of tool.

Syntax	No. of bits	Mnemonic
IPMP_ParametricDescription () {		
Control_info_class_tag	8	uimsbf
Length	16	uimsbf
descriptionComment		ByteArray
majorVersion	8	uimsbf
minorVersion	8	uimsbf
numOfDescriptions	8	uimsbf
for (i=0; i< numOfDescriptions; i++) {		
class		ByteArray
subClass		ByteArray
typeData		ByteArray
type		ByteArray
addedData		ByteArray
}		
}		

6.2.1.3.6 Semantic definitions of fields in IPMP Tool Info Class

`class` - class of the parametrically described tool, for example, decryption.

`subClass` - sub-class of the parametrically described tool, for example, AES under decryption class.

`typeData` - specific type data to describe a particular type of tool, for example, Block_length, to further specify a AES decryption tool.

`type` - value of the type data above, for example, 128 for the Block_length.

`addedData` - Any additional data which may help to further describe the parametrically defined tool.

6.2.1.3.7 IPMP Tool ID

The IPMP Tool Identifier is 128-bits long, platform independent, and shall contain a unique identification number for the IPMP Tool. A registration authority for IPMP Tools that use a unique ID is required. The registration authority may further maintain an association of the download URLs for various implementations of the given tool for various platforms. These platforms will be described to adequate detail using a structured representation. The IPMP ToolID identifies a specific IPMP Tool, unless in the reserved range for parametrically defined tools or alternative tools. Specific values within this 128-bit space are reserved for indicating parametric tools, the bitstream, the terminal, and other special addresses. These values may not be assigned to registered Tools.

IPMP_ToolID	**Semantics**
0x0000	Forbidden
0x0001	Content
0x0002	Terminal
0x0003 - 0x2000	Reserved for ISO use
0x2001 - 0xFFFF	Carry over from 14496-1 RA
0x10000 - 0x100FF	Parametric Tools or Alternative Tools
0x100FF – 2^128-2	Open for registration
2^128-1	Forbidden

6.2.1.3.8 IPMP Tool Container Class

IPMP Tool Container Class enables carriage of IPMP tool inside MPEG-2 system. The device may retrieve the IPMP tool from the content stream, load it, instantiate it and immediately use it in order to play out the content.

One implementation of a given tool is carried as the payload of one IPMP Tool Container Class, the representation format, packaging information and IPMP Tool ID of which is also specified in the container.

Syntax	**No. of bits**	**Mnemonic**
IPMPToolContainer() {		
Control_info_class_tag	8	**uimsbf**
Length	16	**uimsbf**
IPMP_ToolID	128	**uimsbf**
Tool_Format_ID	32	**uimsbf**
Tool_Package_ID	16	**uimsbf**
SizeofTool	16	**uimsbf**
for (i=0; i<sizeofTool; i++) {		
Toolbody	8	**uimsbf**
}		
}		

Handling and usage of Tools carried in Tool Container Class is implementation dependant.

6.2.1.3.9 Semantic definitions of fields in IPMP Tool Container Class

Control_info_class_tag – this 8 bit field should be set to 0x03 to indicate this is an IPMP Tool Container class.

Length – number of bytes of this class, starting immediately following this “Length” field.

IPMP_ToolID – the ID of the Tool carried in this stream.

Tool_Format_ID - This is defined as 0x0001 for a structurally described tool. Otherwise, the Tool_Format_ID indicates the Binary Representation of the Tool and is assigned by a registration authority.

NOTE A structurally described tool implies a description of the IPMP Tool in terms of a network of primitives that can be combined to provide some or all IPMP functionalities required for content consumption. For example, a DES decryption algorithm could be described as a sequence of opcodes calls receiving the ciphertext as input and providing the plaintext as output.

Tool_Package_ID indicates the details of the package of the Tool – examples are CAB or a Winzip self-install executable. Values are assigned by a registration authority.

6.2.1.3.10 IPMP Rights Container Class

IPMP Rights Container Class conveys the usage rules & states associated with the IPMP protected content.

Syntax	No. of bits	Mnemonic
IPMP_Rights_Container () {		
Control_info_class_tag	8	uimsbf
Length	16	uimsbf
rights_data		ByteArray
}		

6.2.1.3.11 Semantic Definitions of Fields in IPMP Rights Container Class

Control_info_class_tag – this 8 bit field should be set to 0x04 to indicate this is a IPMP Rights Container.

Length – number of bytes of this class, starting immediately following this “Length” field.

rights_data – This contains the details of usage rights information.

6.2.2 IPMP Control Information in Program Stream

In Program Stream, the IPMP Control Information is present as one or more PES packets where the stream_id is extended_stream_id 0xFD and stream_id_extension of 0b000 0000, as defined in ISO/IEC 13818-1.

6.2.2.1 Syntax of IPMP Control Information in Program Stream

Syntax	No. of bits	Mnemonic
IPMP_Control_Info () {		
packet_start_code_prefix	24	bslbf
stream_id	8	Uimsbf
IPMPControlInfo_packet_length	16	Uimsbf
‘100000010000000’	15	bslbf
PES_extension_flag	1	bslbf
PES_header_data_length	8	Uimsbf
‘0000000’	7	bslbf
PES_extension_flag_2	1	bslbf
‘10000001’	8	bslbf
stream_id_extension_flag	1	bslbf
stream_id_extension	7	uimsbf
IPMP_control_info_version	5	Uimsbf
current_next_indicator	1	Bslbf
Reserved	2	Bslbf
IPMPControlInfo_packet_number	8	Uimsbf
last_ packet_number	8	Uimsbf
IPMP_control_info_classes_len_in_this_section	16	Uimsbf
for (i=0; i<N;i++) {		
IPMP_control_info_class()		
}		
If (IPMPControlInfo_packet_number==last_packet_number) {		
IsSigned	1	Bslbf
Reserved	7	Bslbf
if (isSigned) {		
Signature		ByteArray
NumCerts	8	Uimsbf
for (i=0; i<numCerts;i++) {		
CertType	8	Uimsbf
Certificate		ByteArray
}		
Verifying_Tool_ID	128	Uimsbf
}		
}		
CRC_32	32	Rpchof
}		

6.2.2.2 Semantics

packet_start_code_prefix -- The packet_start_code_prefix is a 24-bit code. Together with the stream_id that follows it constitutes a packet start code that identifies the beginning of a packet. The packet_start_code_prefix is the bit string '0000 0000 0000 0000 0000 0001' (0x000001 in hexadecimal)

stream_id -- This is an 8 bit field whose value is always 0xFD (extended_stream_id) in hexadecimal.

PES_extension_flag – This is a 1 bit field whose value should be always set to ‘1’.

PES_header_data_length – This is a 8 bit field whose value should be always set to 0x03, indicating the length of bytes immediately following the field and before the IPMP_control_info_version field.

PES_extension_flag_2 – This is a 1 bit field whose value should be always set to ‘1’.

stream_id_extension_flag – This is a 1 bit field whose value should be always set to ‘1’.

stream_id_extension – This is a 7 bit field whose value should be always set to 0b000 0000 (IPMP Control Information stream).

IPMPControlInfo_packet_length -- This is a 16 bit field indicating the total number of bytes in this IPMPControlInfo PES packet immediately following this field.

IPMP_control_info_version -- This 5 bit field is the version number of the whole IPMP Control Information in this Program Stream. The version number shall be incremented by 1 modulo 32 when a change in the information carried within the IPMP control info occurs. When the current_next_indicator is set to '1', then the version_number shall be that of the currently applicable IPMP Control Information. When the current_next_indicator is set to '0', then the version_number shall be that of the next applicable IPMP Control Information.

current_next_indicator -- A 1 bit indicator, which when set to '1' indicates that the IPMP Control Information sent is currently applicable. When the bit is set to '0', it indicates that the IPMP Control Information sent is not yet applicable and shall be the next IPMP Control Information to become valid.

IPMPControlInfo_packet_number -- This 8 bit field gives the number of this PES packet. The IPMPControlInfo_packet_number of the first PES packet in the entire IPMP Control Information shall be 0x00. It shall be incremented by 1 modulo 256 with each additional PES packet in the IPMP Control Information Table.

last_ packet_number -- This 8 bit field specifies the number of the last IPMP Control Information PES packet (that is, the PES packet with the highest IPMPControlInfo_packet_number) of the IPMP Control Information.

IPMP_Control_Info_classes_len_in_this_section – This 12 bit field specifies the length of the IPMP Control Information immediately following this field in this section.

IPMP_Control_Info_class – The IPMP Control Information Classes, including Tool List, Tool Container, Rights Container.

isSigned – This 1 bit field indicates the presence of a signature in the IPMP Control Information.

Signature – The signature of the entire IPMP Control Information classes including Tool List, IPMP Rights Container and Tool Container.

CertType – The type of certification mechanism being used, value assigned by a Registration Authority.

NumCerts – The number of certificates included.

Certificate -- The array of certificates.

Verifying_Tool_Id – The ID of the Tool that is required to verify the certificate(s). A value of 0 indicates the device.

CRC_32 -- This is a 32 bit field that contains the CRC value that gives a zero output of the registers in the decoder defined in Annex B in [1] after processing the entire IPMP Control Information section.

6.3 IPMP Protection Signalling

The entire IPMP protection scheme can be described by embedding IPMP Descriptor's into Program Map Table (PMT) in Transport stream or Program Stream Map (PSM) in Program Stream. IPMP Descriptor is specified in subclause 6.3.3. It associates IPMP tool with each individual stream under its protection. It also indicates the control point at which a specific IPMP tool should be running.

6.3.1 IPMP Protection Signalling in Transport Stream

The following table is the Transport Stream Program Map Section in MPEG-2 system.

Syntax	No. of bits	Mnemonic
TS_program_map_section() {		
table_id	**8**	**Uimsbf**
section_syntax_indicator	**1**	**Bslbf**
'0'	**1**	**Bslbf**
Reserved	**2**	**Bslbf**
section_length	**12**	**Uimsbf**
program_number	**16**	**uimsbf**
Reserved	**2**	**bslbf**
version_number	**5**	**uimsbf**
current_next_indicator	**1**	**bslbf**
section_number	**8**	**uimsbf**
last_section_number	**8**	**uimsbf**
Reserved	**3**	**bslbf**
PCR_PID	**13**	**uimsbf**
Reserved	**4**	**bslbf**
program_info_length	**12**	**uimsbf**
for (i=0; i<N; i++) {		
descriptor()		
}		
for (i=0;i<N1;i++) {		
stream_type	**8**	**uimsbf**
Reserved	**3**	**bslbf**
elementary_PID	**13**	**uimsnf**
Reserved	**4**	**bslbf**
ES_info_length	**12**	**uimsbf**
for (i=0; i<N2; i++) {		
descriptor()		
}		
}		
CRC_32	**32**	**rpchof**
}		

The presence of IPMP Descriptor in the descriptor() directly under PMT and outside the elementary stream loop indicates that all elementary streams of this program is protected by tools given in the IPMP Descriptor.

The presence of IPMP Descriptor in the descriptor() under the loop of elementary stream, associated with one elementary stream, indicates that the given elementary stream is protected by tools given in the IPMP descriptor.

6.3.2 IPMP Protection Signalling in Program Stream

The following table is program stream map in Program Stream.

Syntax	No. of bits	Mnemonic
program_stream_map() {		
packet_start_code_prefix	**24**	**bslbf**
map_stream_id	**8**	**uimsbf**
program_stream_map_length	**16**	**uimsbf**
current_next_indicator	**1**	**bslbf**
reserved	**2**	**bslbf**
program_stream_map_version	**5**	**uimsbf**
reserved	**7**	**bslbf**
marker_bit	**1**	**bslbf**
program_stream_info_length	**16**	**uimsbf**
for (i=0;i<N;i++) {		
descriptor()		
}		
elementary_stream_map_length	**16**	**uimsbf**
for (i=0;i<N1;i++) {		
stream_type	**8**	**uimsbf**
elementary_stream_id	**8**	**uimsbf**
elementary_stream_info_length	**16**	**uimsbf**
for (i=0;i<N2;i++) {		
descriptor()		
}		
}		
CRC_32	**32**	**rpchof**
}		

The presence of IPMP Descriptor in the descriptor() directly under PSM and outside the elementary stream loop indicates that all elementary streams of this program is protected by tools given in the IPMP Descriptors.

The presence of IPMP Descriptor in the descriptor() under the loop of elementary stream, associated with one elementary stream, indicates that the given elementary stream is protected by tools given in the IPMP descriptors.

6.3.3 IPMP Descriptor

IPMP Descriptor conveys the control point information of the IPMP Tool, including at which control point the tool resides, and its sequence in relation to other tools residing at that control point. The descriptor_id of IPMP Descriptor is 41 as defined in ISO/IEC 13818-1.

The presence of IPMP Descriptor in PMT in Transport stream or PSM in Program Stream indicates the association of IPMP tool and protected elementary stream. It also indicates the sequence and/or control point of the IPMP tool.

6.3.3.1 Syntax of IPMP Descriptor

Syntax	No. of bits	Mnemonic
IPMP_descriptor() {		
descriptor_tag	8	**Uimsbf**
descriptor_length	8	**Uimsbf**
IPMP_Descriptor_ID	32	**Uimsbf**
IPMP_ToolID	128	**Uimsbf**
ControlPoint	8	**uimsbf**
SequenceCode	8	**uimsbf**
IPMP_Data_length	16	**uimsbf**
for (i=0; I< N; i++) {		
IPMP_Data		
}		
isSigned	8	**Uimsbf**
if (isSigned)		
Signature		**ByteArray**
NumCerts	8	**uimsbf**
for (i=0; i<numCerts;i++) {		
CertType	8	**uimsbf**
Certificate		**ByteArray**
}		
Verifying_Tool_ID	128	**uimsbf**
}		
}		

6.3.3.2 Semantic Definitions of Fields in IPMP Descriptor

IPMP_Descriptor_ID – a unique ID of this IPMP descriptor. This could be used to refer to this particular descriptor. 0x00000000 and 0xFFFFFFFF are prohibited.

Since an IPMP Tool's instantiation is signalled by a unique IPMP Descriptor, this IPMP_Descriptor_ID can also used as a unique identification of an IPMP Tool instance for messaging.

IPMP_ToolID – Unique ID of the IPMP Tool that is protecting in this scope.

controlPoint – value specifying the IPMP control point at which the IPMP Tool resides, and is one of the following values:

Terms of TB_n, B_n, EB_n, D_n are defined in STD model of MPEG-2 Systems ISO/IEC 13818-1.

Control Point	**Description**
0x00	No control point.
0x01	Control Point after Transport Buffer TB_n in STD
0x02	Control Point between the decode buffer (B_n or ES_n) and the decoder (D_n) in STD
0x03	Control Point between the decoder (D_n) and the rendering.
0x04 - 0xDD	ISO reserved
0xDE - 0xFE	User private
0xFF	Forbidden.

sequenceCode - value specifying the relation of the IPMP Tool to IPMP Tool(s) residing at the same control point. The value of this field specifies the priority of this IPMP Tool at this specific control point. For example, a value of “20” means this tool has a higher priority than an IPMP tool with sequenceCode “12”. Data will be routed to the IPMP tool with a higher priority first, before the data goes to the next lower priority IPMP tool. Two tools shall not have the same sequence number at the same control point for the same stream.

IPMP_Data **--** The IPMP Data that is extended from IPMP_Data_BaseClass [7.1].

isSigned – This 1 bit field indicates the presence of a signature in the IPMP Descriptor.

Signature – The signature of the entire IPMP Descriptor.

CertType – The type of certification mechanism being used, value assigned by a Registration Authority.

NumCerts – The number of certificates included.

Certificate -- The array of certificates.

Verifying_Tool_Id – The ID of the Tool that is required to verify the certificate(s). A value of 0 indicates the device.

6.4 IPMP Stream

6.4.1 IPMP Stream Specification

The IPMP Stream is an elementary stream that carries IPMP data to be routed to different IPMP Tools. It is similar to ECM in CA system, while IPMP Control Information [6.2] can be treated as EMM in CA system.

IPMP Stream type is 0x1A as defined in MPEG-2 Systems ISO/IEC 13818-1.

Stream_id specifies the type and number of the elementary stream as defined by 13818-1. stream_id of 0b 1111 1101 (extended_stream_id) together with stream_id_extension of 0b 000 0001 identifies an IPMP Stream.

The IPMP stream should be wrapped in PES packets as defined by MPEG-2 Systems ISO/IEC 13818-1. The PES packets shall be of non-zero length. The mapping of the PES packets into MPEG-2 Transport Stream packets is defined in MPEG-2 Systems ISO/IEC 13818-1.

The IPMP stream specification uses the standard PES packet syntax and semantics with the following constraints:

stream_id: this field shall be set to the value of 0b1111 1101 (extended_stream_id).

stream_id_extension: this field shall be set to the value of 0b 000 0001 (IPMP Stream).

When the payload_start_indicator is set to 1, the first byte immediately following the PES header starts with IPMP_StreamDataUpdate.

One PES packet contains an integer number of IPMP_StreamDataUpdate.

6.4.2 IPMP Stream Syntax

The IPMP Stream should be a concatenation of IPMP_StreamDataUpdate's, with the syntax defined below. Keys are possibly carried in IPMP Stream.

Syntax	No. of bits	Mnemonic
IPMP_StreamDataUpdate() {		
IPMP_descriptor_id	32	uimsbf
IPMP_Data_length	16	uimsbf
for (i=0; i<N; i++) {		
IPMP_data		
}		
}		

The IPMP_descriptor_id clearly defines the destination of this IPMP_StreamDataUpdate. It unambiguously identifies a particular IPMP Tool instance since IPMP Tool is instantiated based on the signalling of an IPMP Descriptor. This message should be routed by the message router to the IPMP Tool instance.

`IPMP_data` - The IPMP Data that is extended from IPMP_Data_BaseClass [7.1].

6.4.3 IPMP Stream Decoder Model

6.4.3.1 IPMP Stream Decoder Model in Transport Stream

The Transport Stream IPMP Stream Decoder Model is a conceptual model for decoding IPMP streams. The decoder model is used to specify the delivery of the bits in time. The decoder model does not specify the operation or behavior of a real decoder. Implementation and implementations which do not follow the architecture or timing of this model are not precluded.

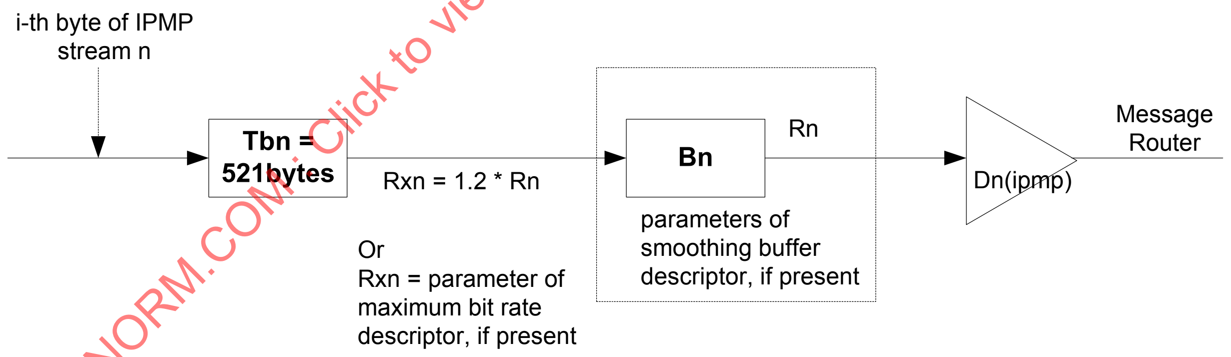
i-th byte of IPMP stream n
Tbn = 521bytes
Rxn = 1.2 * Rn
Or
Rxn = parameter of maximum bit rate descriptor, if present
Bn
Rn
parameters of smoothing buffer descriptor, if present
Dn(ipmp)
Message Router

Figure 3 — IPMP Stream Decoder Model in Transport Stream

The above figure shows the structure of the IPMP Stream decoder model for a single IPMP stream n, which is similar to the Transport System Target Decoder (T-STD) model of ISO/IEC 13818-1. The symbols Tb_n , B_n , Rx_n , and R_n are defined as follows:

Tb_n is the transport buffer for IPMP stream n;

B_n is the main buffer for IPMP stream n;

Rx_n is the rate at which data is removed from Tb_n ; and

R_n is the rate at which data is removed from B_n .

Complete TS packets containing data from the IPMP stream n are inserted in the transport buffer for stream n, Tb_n.

All bytes that enter the buffer Tb_n are removed at rate Rx_n specified below. Bytes which are part of a PES packet, a section, or the contents of these containers are delivered to the main buffer B_n . Other bytes are not, and may be used to control the system. Duplicate TS packets are not delivered to B_n . All bytes that enter the buffer B_n are removed at rate R_n specified below.

The transport buffer Tb_n is 512 bytes.

The transport buffer leak rate Rx_n , the size of the buffer B_n, and the leak rate R_n are specific to a particular service.

The service may indicate the values for Rx_n, B_n and R_n by means of the MPEG-2 maximum_bit_rate_descriptor and the smoothing_buffer_descriptor (see ISO/IEC 13818-1). If used, the descriptors shall be included in the PMT of the content.

The maximum_bit_rate field of the maximum_bit_rate descriptor shall indicate the value that is applied for Rx_n. The sb_size field of the smoothing_buffer_descriptor shall contain the value of B_n . The sb_leak_rate field shall contain the value of R_n.

If the maximum_bit_rate_descriptor is not included in PSI, but the smoothing_buffer_descriptor is included, then $Rx_n = 1{,}2R_n$.

If the smoothing_buffer_descriptor is not included in PSI, but the maximum_bit_rate_de-scriptor is included, then the two buffer model becomes a single buffer model that consists of the Transport buffer Tb_n with a leak rate Rx_n .

If neither of the descriptors are included in PSI, then the buffer model is not applicable. In this case, the delivery of the bits is service specific.

6.4.3.2 IPMP Stream Decoder Model in Program Stream

The Program Stream IPMP Stream Decoder Model is a conceptual model for decoding IPMP streams. The decoder model is used to specify the delivery of the bits in time. The decoder model does not specify the operation or behavior of a real decoder. Implementation and implementations which do not follow the architecture or timing of this model are not precluded.

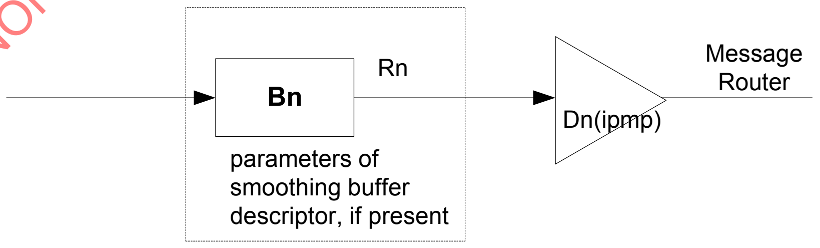

Figure 4 — IPMP Stream Decoder Model in Program Stream

The above figure shows the structure of the IPMP Stream decoder model for a single IPMP stream n, which is similar to the Program System Target Decoder (P-STD) model of ISO/IEC 13818-1. The symbols B_n , Rx_n , and R_n are defined as follows:

B_n is the main buffer for IPMP stream n;

R_n is the rate at which data is removed from B_n .

The PES packet data from IPMP stream n is passed to the input buffer for stream n, B_n. Transfer of byte i from the system target decoder input to B_n is instantaneous, so that byte i enters the buffer for IPMP stream n, of size BS_n, at time t(i).

Bytes present in the pack header, or PES packet headers of the Program Stream such as SCR, DTS, PTS, and packet_length fields, are not delivered to any of the buffers, but may be used to control the system.

All bytes that enter the buffer B_n are removed at rate R_n specified below.

The size of the buffer B_n, and the leak rate R_n are specific to a particular content. The content may indicate the values for B_n and R_n by means of the MPEG-2 smoothing_buffer_descriptor (see ISO/IEC 13818-1). If used, the descriptor shall be included in the PSM (Program Stream Map) of the content.

The sb_size field of the smoothing_buffer_descriptor shall contain the value of B_n . The sb_leak_rate field shall contain the value of R_n.

If no smoothing_buffer_descriptor is included in PSM, then the buffer model is not applicable. In this case, the delivery of the bits is implementation specific.

7 IPMP Data and Messages (Normative)

NOTE The syntax in this clause is described using a language defined by ISO/IEC 14496-1 called Syntactic Description Language (SDL) 0 that allows the description of a bitstream's syntax. SDL assumes an object-oriented underlying framework in which bitstream units consist of "classes." This framework is based on the typing system of the C++ and Java programming languages. SDL extends the typing system by providing facilities for defining bitstream-level quantities, and how they should be parsed.

7.1 IPMP_Data_BaseClass

The IPMP_Data_BaseClass is intended to be extended to provide the carriage of ISO defined as well as user defined IPMP related data.

7.1.1 Syntax

```
abstract aligned(8) expandable(2^28-1) class IPMP_Data_BaseClass:
   bit(8) tag=0 .. 255
{
   bit(8)  Version;
   bit(32) dataID;
   // Fields and data extending this message.
}
```

7.1.2 Semantics

`Version` - indicates the version of syntax used in the IPMP Data and shall be set to "0x01".

`dataID` – used for the purpose of identifying the message. Tools replying directly to a message shall include the same dataID in any response.

`tag` indicates the tag for the extended IPMP data. The exact values for the extension tags are defined below.

IPMP data extending from IPMP_Data_BaseClass can be carried in the following three places:

- IPMP Descriptor [6.3.3]
- IPMP_StreamDataUpdate defined in [6.4.2] which is subsequently carried in IPMP Stream.
- Messages defined in [7.7] specified to carry messages between IPMP tools.

7.1.3 Extension tags for the IPMP_Data_BaseClass message

Table 1 — Tags for IPMP Data extending IPMP_Data_BaseClass

8-bit Tag Value	Symbolic Name
0x00	Forbidden
0x01	IPMP_OpaqueData_tag
0x02	IPMP_AudioWatermarkingInit_tag
0x03	IPMP_VideoWatermarkingInit _tag
0x04	IPMP_SelectiveDecryptionInit_tag
0x05	IPMP_KeyData _tag
0x06	IPMP_SendAudioWatermark_tag
0x07	IPMP_SendVideoWatermark _tag
0x08	IPMP_RightsData _tag
0x09	IPMP_Secure_Container_tag
0x0A	IPMP_AddToolNotificationListener_tag
0x0B	IPMP_RemoveToolNotificationListener_tag
0x0C	IPMP_InitAuthentication_tag
0x0D	IPMP_MutualAuthentication_tag
0x0E	IPMP_UserQuery_tag
0x0F	IPMP_UserQueryResponse_tag
0x10	IPMP_ToolParamCapabilitiesQuery_tag
0x11	IPMP_ToolParamCapabilitiesResponse_tag
0x12	IPMP_GetTools_tag
0x13	IPMP_GetToolsResponse_tag
0x14	IPMP_ConnectTool_tag
0x15	IPMP_DisconnectTool_tag
0x16	IPMP_NotifyToolEvent_tag
0x17	IPMP_CanProcess_tag
0x18	IPMP_TrustSecurityMetadata_tag
0x19– 0x3F	Reserved for Inter-device messages
0x40 – 0xCF	ISO Reserved
0xD0 – 0xFE	User Defined
0xFF	Forbidden

7.2 Mutual Authentication

7.2.1 IPMP_InitAuthentication

7.2.1.1 Syntax

```
class IPMP_InitAuthentication extends IPMP_Data_BaseClass:
bit(8) tag = IPMP_InitAuthentication_tag
{
   bit (32)    IPMP_Descriptor_ID;
   bit (8)     AuthType;
}
```

7.2.1.2 Semantics

`IPMP_Descriptor_ID` – the 32-bit IPMP_Descriptor_ID that uniquely identifies the logical instance of the Tool with which mutual authentication is to be performed. A "0" value indicates the terminal.

`AuthType` has the following values.

Value of AuthType	Semantic Meaning
0x00	Forbidden
0x01	No Authentication Required
0x02	No ID verify, Do secure channel
0x03	Do ID verify, No secure channel
0x04	Do ID verify, Do secure channel
0x05-0xFE	ISO Reserved
0xFF	Forbidden

7.2.1.3 Response

IPMP_Mutual_Authentication between receiving tool and indicated tool.

7.2.2 IPMP_Mutual_Authentication

7.2.2.1 Syntax

```
class IPMP_Mutual_Authentication extends IPMP_Data_BaseClass
: bit(8) tag=IPMP_MutualAuthentication_tag;
{
   bit (1)    requestNegotiation;
   bit (1)    successNegotiation;
   bit (1)    failedNegotiation;
   bit (1)    inclAuthenticationData
   bit (1)    inclAuthCodes;
   const bit (3) reserved = 0b000;
   if (requestNegotiation) {
      bit(8)                nCandidateAlgorithm;
      AlgorithmDesciptor    candidateAlgorithms[nCandidateAlgorithm];
   }
   if (successNegotiation) {
      bit(8)                nAgreedAlgorithm;
      AlgorithmDescriptor   agreedAlgorithms[nAgreedAlgorithm];
   }
   if (inclAuthenticationData) {
      ByteArray             AuthenticationData
   }
   if (inclAuthCodes) {
      bit(8)     type;
      if (type == 0x01) {
         bit(8)          nCertificate;
         bit(8)          certType;
         ByteArray       certificates[nCertificate];
      } elseif (type == 0x02) {
         KeyDescriptor      publicKey;
      } elseif (type == 0xFE) {
         ByteArray          opaque;
      }
      IPMP_Data_BaseClass  trustData;
      ByteArray            authCodes;
   }
}
```

7.2.2.2 Semantics

An instance of `IPMP_Mutual_Authentication` is generated by and exchanged between IPMP tools, and IPMP Tools and a Terminal for the purpose of mutual authentication.

`inclAuthCodes` – if this flag is set, authentication codes for the current message are specified.

`type` – specifying the type of the authentication codes. Semantics of each value for this variable is defined as follows

Value of Type	Semantics
0x00	No information regarding the type is explicitly specified.
0x01	The value specified in the variable authCodes is digital signature. One or more SPKI certificates or X.509 V3 certificates, depending on the value of certType are specified as a method to verify the signature in the variable certificates[].
0x02	The value specified in the variable authCodes is a digital signature. At the same time, a public key accompanied by algorithm specification is specified as a method to verify the signature.
0x03 – 0xDF	ISO Reserved
0xE0 – 0xFD	User Private
0xFE	Application-dependent information regarding the type is specified in the variable opaque.
0xFF	Forbidden

`certType` – specifies type of a certificate, value assigned by a Registration Authority.

`certificates []` – specifies data of one or more certificates. The certificate type is determined by the value of `certType`. This variable is specified, only when the variable type specifies 0x01.

`publicKey` – specifies a cryptographic public key accompanied by algorithm specification. This variable is specified, if the variable `type` specifies 0x02.

`opaque` – specifies an application-dependent method to verify the authentication codes. This variable is specified if the variable `type` specifies 0xFE.

`trustData` – if present, specifies trust and security metadata.

`authCodes` – specifies authentication codes to this message. Note that data to be signed excludes data beginning at the `type` specification of the `authCodes`.

`requestNegotiation` – indicates that the originator is requesting negotiation about an authentication method to be used in the subsequent communication.

`candidateAlgorithms` – specifies a list comprised of one or more algorithm identifiers of candidate authentication methods. The originator of the `IPMP_Mutual_Authentication` message shall specify identifiers of algorithms and/or protocols that it supports. The recipient of this message shall select ones out of the list to fulfil the requirements, which the recipient IPMP tool supports. The recipient sends another `IPMP_Mutual_Authentication` message such that `agreedAlgorithm` specifies the identification of the selected algorithms and/or protocols. If the recipient cannot find algorithms and/or protocols that it supports in the list, it returns an `IPMP_Mutual_Authentication` message specifying a "0b1" for `failedNegotiation`. The syntax of `AlgorithmDescriptor` is provided in [7.2.2.7].

Note: When the field `candidateAlgorithms` contains algorithms of mutual authentication that support the generation of a shared secret, additional algorithm identifiers are listed to support the negotiation of message encryption, and message authentication. The purpose of each algorithm included will be implicit from its functionality. That is, if DES is a candidate algorithm it is assumed it is specified for message encryption/decryption.

Additional Note: For algorithms that support a number of options within one algorithm such as AES supporting a number of block lengths and key lengths, only one configuration will be specified for a given identifier or registration authority listed ID. Additionally padding requirements and solutions will be included in the identifiers and Ids as well.

`inclAuthenticationData` – when set to '1', this implies that the message contains method specific data used during mutual authentication.

`AuthenticationData` – specifies data to be used for mutual authentication and whose value depends on method specific processing. This variable exists only when the flag `inclAuthenticationData` is set.

7.2.2.3 Response

IPMP_Mutual_Authentication, until authentication is successful.

7.2.2.4 ByteArray

The Class `ByteArray` is a generic string or array of bytes container used extensively throughout the IPMP specifications.

7.2.2.4.1 Syntax

```
expandable class ByteArray
{
    bit(8) data[sizeOfInstance()];
}
```

7.2.2.4.2 Semantics

`data` - the string or array of bytes carried.

7.2.2.5 BaseAuthenticationDescriptor

7.2.2.5.1 Syntax

```
abstract aligned(8) expandable(2^28 -1) class BaseAuthenticationDescriptor : bit(8) tag=0
{
// empty. To be filled by classes extending this class.
}
```

0x00	Forbidden
0x01	IPMP_AlgorithmDescr_tag
0x02	IPMP_KeyDescr_tag
0x03-0x7F	ISO Reserved
0x7F-0xFE	User defined
0xFF	Forbidden

7.2.2.6 Key Descriptor

7.2.2.6.1 Syntax

```
class KeyDescriptor extends BaseAuthenticationDescriptor :
bit(8) tag = IPMP_KeyDescr_tag
{
   ByteArray  keyBody;
}
```

7.2.2.6.2 Semantics

This class is for specifying a cryptographic algorithm and a key conforming to the algorithm.

`keyBody` – the body of the key. The value shall be data that conforms to a rule for data structure of the key defined outside of this document.

Example – If an encryption algorithm is RSA, the value of this parameter shall be the BER encoded data which conforms to PKCS#1.

```
RSAPublicKey ::=
      SEQUENCE {
         modulus        INTEGER, -- n
         publicExponent     INTEGER -- e
      }
```

7.2.2.7 AlgorithmDescriptor

7.2.2.7.1 Syntax

```
class AlgorithmDescriptor extends BaseAuthenticationDescriptor:
bit(8) tag = IPMP_AlgorithmDescr_tag {

   bit (1)    isRegistered;
   const bit (7)   reserved = 0b0000.000;
   if (isRegistered) {
      bit (16)       regAlgoID;
   }
   else {
      ByteArray      specAlgoID;
   }
   ByteArray OpaqueData;
}
```

7.2.2.7.2 Semantics

This class is for specifying an identifier of an arbitrary algorithm.

`isRegistered` – a 0b1 indicates the ID included is a normative identifier of the algorithm. A 0b0 indicates the algorithm is identified by an ID assigned by a registration authority

`regAlgoID` – a identifier of the algorithm as assigned by a registration authority.

`specAlgoID` – a normative identifier of the algorithm.

`SpecAlgoID` –

SpecAlgoID	Content	Confidentiality	Integrity
DH-G-H-x-y	Diffie-Hellman Key Agreement Scheme on a base group G. The parameter setting of the both parties is specified in certificates.	No	Yes
EDH-G-H-x-y	Ephemeral Diffie-Hellman Key Agreement Scheme on a base group G. Both parties are assumed to agree on the base group G, and each party is supposed to generate ephemeral parameters (a public key pair) for the purpose of authentication and the parameters are submitted being signed by the party. A certificate of the party is used for the opponent party to verify the signature.	No	Yes
DH-G-E-H-x-y-z	DH-G-H-x-y being used in combination with a symmetric key cipher algorithm E.	Yes	Yes
EDH-G-E-H-x-y-z	EDH-G-H-x-y being used in combination with a symmetric key cipher algorithm E.	Yes	Yes
RSA-OAEP-H-x	Needham-Schroeder Scheme deploying RSA-based OAEP as the public key encryption.	No	Yes
RSA-OAEP-E-H-x-z	RSA-OAEP-H-x being used in combination with a symmetric key cipher algorithm E.	Yes	Yes
SCHNRR- G-x-y	Schnorr Identification Scheme on a base group G.	No	No

In the above table, the significance of the variables is as follows:

Variable	Definition
G	Specifies the type of a base group (e.g. a sub-groups of an elliptic curve)
E	Specifies a symmetric key cipher algorithm to realize confidentiality of messages.
H	Specifies a hash function to realize message integrity. Hash function to be used being SHA-1.
x	Specifies the size of a base field (e.g. the length in bits of the base field of an elliptic curve).
y	Specifies a size of the order of the base group (e.g. the length in bits of the order of a base group).
z	Specifies the length in bits of a symmetric key cipher to be used.

Remark. The actual cryptographic scheme is constructed on a proper sub-group instead of being constructed on an entire elliptic curve or an entire multiplicative group of a finite field. Further, the order of the base sub-group shall be a prime number, and its length is specified to be y bits. This value of y influences the strength of the scheme against the Pohlig-Hellman method, which finds out solutions to DLP on arbitrary groups.

7.2.2.8 Generation of a MAC

In the case where a value is specified for the variable H, if either or both of the parties require the functionality of message integrity, MAC (Message Authentication Code) is attached to messages. Usage of a MAC as part of the message is optional. MAC is generated based on the hash function specified by the variable H and the shared secret being used as a key to generate MAC. MAC is generated in accordance with the following formula, where H denotes Hashing and | denotes concatenation.

MAC = H(H(Shared_Secret)| Message | H(Shared_Secret))

7.2.2.9 Generation of Keys for Message Encryption

Keys of a symmetric key cipher to be used between the parties are generated from the secret shared by the parties through the authentication procedures. For example, in the case that the key length is specified as 56 bits, the write_key_A, which the party A uses to encrypt messages, is generated by the following formula.

```
write_key_A =  TRUNC(56, SHA-1(100_LSBits_Of_Secret | “WRITE” | ID_Of_A | Rest_Of_Secret))
```

ID_Of_A is defined as the byte data with value "0" if Party A is the initiator of the communication, whilst it is the byte data with value 0xFF otherwise.

7.2.3 IPMP_TrustSecurityMetadata

7.2.3.1 Syntax

```
class IPMP_TrustSecurityMetadata : public IPMP_Data_BaseClass
: bit(8) tag = IPMP_ TrustSecurityMetadata_tag
{
   bit(16) numTrustedTools;
   for (int I=0; I<numTrustedTools; I++)
   {
      bit(128)      toolID;
      DateClass  AuditDate;
      bit(16)    numTrustSpecification;
      for (int i=0; i<numTrustSpecification; ++i)
      {
         bit (1)    hasCCBasedTrustMetadata;
         const bit(7) reserved = 0b0000.000;

         if (!hasCCBasedTrustMetadata) {
            DateClass  startDate;
            bit(2)     attackerProfile;
            const bit(6) reserved = 0b0000.00;
            bit(32)    trustedDuration;
         } else {
            ByteArray  CCTrustMetadata;
         }
      }
   }
}
```

7.2.3.2 Semantics

For numTrustedTools referenced by toolID, the trust metadata for each tool consists of the date of audit, AuditDate by the trust auditor.

For each of numTrustSpecification, the tool shall be trusted as of startDate , to protect against the specified AttackerProfile for the TrustedDuration minutes.

Or an alternative, though opaque means of specifying the trust metadata using Common Criteria contained in CCTrustMetadata.

Attacker profile is defined as class I, II, or III where;

0x00	forbidden
0x01	Class I
0x02	Class II
0x03	Class III
0x04 – 0x0F	User defined
0x10 – 0xFE	ISO reserved
0xFF	forbidden

7.2.4 DateClass

7.2.4.1 Syntax

```
class DateClass : {
   bit(40) Date;
}
```

7.2.4.2 Semantics

This descriptor identifies the date on which the audit took place.

`Date` – contains the audit date of IPMP tool in question, in Universal Time, Co-ordinated (UTC) and Modified Julian Date (MJD). This field is coded as 16 bits giving the 16 least significant bits of MJD followed by 24 bits coded as 6 digits in 4-bit Binary Coded Decimal (BCD). If the audit date is undefined all bits of the field are set to 1.

7.2.5 IPMP_SecureContainer

7.2.5.1 Syntax

```
class IPMP_SecureContainer extends IPMP_Data_BaseClass
: bit(8) tag = IPMP_ Secure_Container_tag
{
   bit(1)  hasEncryption;
   bit(1)  hasMAC;
   bit(1)  isMACEncrypted;
   const bit(5) reserved=0b0000.0;

   if(hasEncryption)
      ByteArray  encryptedData;

      if(hasMAC && !isMACEncrypted) {
      ByteArray  MAC;
      }
   }
   else {
      IPMP_Data_BaseClass  protectedMsg;
      if(hasMAC) {
      ByteArray  MAC;
      }
   }
}
```

7.2.5.2 Semantics

This forms a secure container for any message extending the `IPMP_Data_BaseClass`. The use of this message is optional and is defined for transmitting any message in a secure manner when needed by the application. The payload message is optionally encrypted. The MAC allows verification of the integrity of the `protectedMsg`, and is optionally encrypted. `isMACEncrypted` is not explicitly carried in the secure message, but is negotiated as part of an external authentication mechanism, as is the key and algorithm for creating and verifying the MAC, and decrypting the encrypted payload.

`hasEncryption` - Indicates that the encrypted message is contained in `encryptedData`.

`hasMAC` – Indicates that a MAC, Message Authentication Code is contained in the field `MAC`.

`isMACEncrypted` - indicates that the encrypted portion contained in `encryptedData` also contains MAC information.

protectedMsg – Unencrypted message derived from IPMP_Data_BaseClass to which the contained MAC has been applied.

7.3 IPMP Tool connection and disconnection

7.3.1 IPMP_GetTools

7.3.1.1 Syntax

```
class IPMP_GetTools extends IPMP_Data_BaseClass:
   bit(8) tag = IPMP_GetTools_tag
{
}
```

7.3.1.2 Semantics

Message IPMP_GetTools is used by a tool to find all the Tools, instantiated or not, that are available on the terminal.

7.3.1.3 Response

IPMP_GetToolsResponse.

7.3.2 IPMP_GetToolsResponse

7.3.2.1 Syntax

```
class IPMP_GetToolsResponse extends IPMP_Data_BaseClass:
   bit(8) tag = IPMP_GetToolsResponse_tag
{
   IPMPTool_Info  tools[];
}
```

7.3.2.2 Semantics

tools – A list of tools available to the Terminal.

7.3.2.3 Response

None required

7.3.3 IPMP Tool Parametric Capabilities Query

7.3.3.1 Syntax

```
class IPMP_ToolParamCapabilitiesQuery extends IPMP_Data_BaseClass
: bit(8) tag = IPMP_ToolParamCapabilitiesQuery_tag
{
   IPMP_ParamtericDescription_toolParamDesc;
}
```

7.3.3.2 Semantics

This message allows a terminal to query a tool as for support for a specific parametric description. The contents of the message are the actual tool parametric descriptor that would be contained in the tools list.

toolParamDesc: The parametric description of the capability being queried. IPMP_ParamtericDescription is defined in [6.2.1.3.5].

7.3.3.3 Response

`IPMP_ToolParamCapabilitiesResponse.`

7.3.4 IPMP Tool Parametric Capabilities Query Response

7.3.4.1 Syntax

```
class IPMP_ToolParamCapabilitiesResponse extends IPMP_Data_BaseClass
: bit(8) tag = IPMP_ToolParamCapabilitiesResponse_tag
{
   bit(1) capabilitiesSupported;
   const bit(7) reserved = 0b0000.000;
}
```

7.3.4.2 Semantics

This message is the response to the above parametric capabilities query and simply returns a boolean value as to whether or not the parametric description is supported by the tool.

`capabilitiesSupported`: If this bit is set to '1', this implies that the tool does support the parametric description queried in the preceding message. If set to '0' this implies that the tool does not support the parametric description.

7.3.4.3 Response

None required.

7.3.5 IPMP_ConnectTool

7.3.5.1 Syntax

```
class IPMP_ConnectTool extends IPMP_Data_BaseClass:
   bit(8) tag = IPMP_ConnectTool_tag
{
   IPMP_Descriptor toolDescriptor;
}
```

7.3.5.2 Semantics

Message `IPMP_ConnectTool` allows a tool to request the Terminal to create a connection to a tool identified in the `toolDescriptor`.

`toolDescriptor` - contains an IPMP descriptor to be used for determining location of the new tool to be connected. Scope of connected tool shall be the same as the requesting tool.

The Terminal shall link the new Tool Instance to the Message router, before it responds to this message.

7.3.5.3 Response

`IPMP_NotifyToolEvent` with an `eventType` of "`CONNECTED`" or an `eventType` of "`CONNECTIONFAILED`".

7.3.6 IPMP_DisconnectTool

7.3.6.1 Syntax

```
class IPMP_DisconnectTool extends IPMP_Data_BaseClass:
   bit(8) tag = IPMP_DisconnectTool_tag
{
     bit(32)  IPMP_Descriptor_ID;
}
```

7.3.6.2 Semantics

Message `IPMP_DisconnectTool` allows a tool to disconnect a tool it has previously connected at a control point.

`IPMP_Descriptor_ID` is the ID of the IPMP Descriptor that triggered the instantiation of the IPMP Tool Instance which is to be disconnected.

7.3.6.3 Response

`IPMP_NotifyToolEvent` - with an `eventType` of `"DISCONNECTED"` or an `eventType` of `"DISCONNECTIONFAILED"`.

7.4 IPMP Tool notification

7.4.1 IPMP_AddToolNotificationListener

7.4.1.1 Syntax

```
class IPMP_AddToolNotificationListener extends IPMP_Data_BaseClass :
bit(8) tag = IPMP_AddToolNotificationListener_tag
{
   bit(3) scope;
   bit(5) reserved; 0b0000.0
   bit(8) eventTypeCount;
   bit(8) eventType[eventTypeCount];
}
```

7.4.1.2 Semantics

Message `IPMP_AddToolNotificationListener` allows an existing IPMP Tool running on the terminal to request notification any defined event at the terminal.

`scope` - The value of the scope parameter can be:

```
SCOPE_ES = 0x0
SCOPE_PROGRAM = 0x1
SCOPE_LOCAL = 0x2
```

Other values are reserved for future use.

When called with a scope of `SCOPE_ES`, this message will limit notifications for tools protecting the same elementary stream as the tool who sends the message.

When called with a scope of `SCOPE_PROGRAM`, this message will limit notifications for tools protecting the same program as the tool who sends the message is declared in (at the Program level or the Elementary Stream level).

When called with a scope of `SCOPE_LOCAL`, this message will limit notifications to all tools connected in the same name scope as the tool who sends the message is declared in.

`eventType` – Type of event for which the sender shall receive notifications for and defined by the following table.

0x00	CONNECTED
0x01	CONNECTIONFAILED
0x02	DISCONNECTED
0x03	DISCONNECTIONFAILED
0x04	WATERMARKDETECTED
0x05-0xC0	ISO Reserved
0xC1-0xFF	User Defined

7.4.1.3 Response

None required.

7.4.2 IPMP_RemoveToolNotificationListener

7.4.2.1 Syntax

```
class IPMP_RemoveToolNotificationListener extends IPMP_Data_BaseClass :
   bit(8) tag = IPMP_RemoveToolNotificationListener_tag
{
   bit(8) eventTypeCount;
   bit(8) eventType[eventTypeCount];
}
```

7.4.2.2 Semantics

Message IPMP_RemoveToolNotificationListener allows an existing IPMP Tool running on the terminal to terminate the receipt of a previous registered event listener. If an eventTypeCount of “0” is present, all previously registered event types shall be unregistered.

7.4.2.3 Response

None required.

7.4.3 IPMP_NotifyToolEvent

7.4.3.1 Syntax

```
class IPMP_NotifyToolEvent extends IPMP_Data_BaseClass:
   bit(8) tag = IPMP_NotifyToolEvent_tag
{
   bit(8)      eventType;
   bit(32)     IPMP_Descriptor_ID;
}
```

7.4.3.2 Semantics

Message `IPMP_NotifyToolEvent` notifies an IPMP Tool of a tool event for which it had previous registered as a listener.

`IPMP_Descriptor_ID` – This `IPMP_Descriptor_ID` indicates the IPMP Tool Instance that is either connected or disconnected, or the failed state of either, as indicated by the `eventType`.

7.4.3.3 Response

None required.

7.5 IPMP Processing

7.5.1 IPMP_CanProcess

7.5.1.1 Syntax

```
class IPMP_CanProcess extends IPMP_Data_BaseClass:
bit(8) tag = IPMP_CanProcess_tag
{
   bit(1) canProcess;
   bit(7) reserved = 0b0000.000;
}
```

7.5.1.2 Semantics

`canProcess` – used to indicate to the receiver of the message that the sender is able to begin processing data or must discontinue processing.

0x0	DISCONTINUE
0x1	BEGIN

7.5.2 IPMP Opaque data

7.5.2.1 Syntax

```
class IPMP_OpaqueData extends IPMP_Data_BaseClass
: bit(8) tag = IPMP_OpaqueData_tag
{
   ByteArray opaqueData;
}
```

7.5.2.2 Semantics

`opaqueData` – information to be delivered.

7.5.3 IPMP_KeyData

This IPMP data could possibly be carried in IPMP_StreamDataUpdate which is subsequently carried in IPMP Stream. This facilitates the time variant keys carried in IPMP Stream.

7.5.3.1 Syntax

```
class IPMP_KeyData extends IPMP_Data_BaseClass :
   bit(8) tag = IPMP_KeyData_tag
{
   ByteArray keyBody;
   bit(1) hasStartPTS;
   bit(1) hasExpirePTS;
   const bit(6) reserved = 0b000000;
   if (hasStartPTS)
   {
      bit(33) startPTS;
      const bit(7) reserved = 0b000000;
   }
   if (hasExpirePTS)
   {
      bit(33) expirePTS;
      const bit(7) reserved = 0b000000;
   }
   ByteArray OpaqueData;
}
```

7.5.3.2 Semantics

keyBody – the body of the key. The value shall be data that conforms to a rule for data structure of the key defined outside of this document.

hasStartPTS – A value of "1" indicates the presence of the following startPTS field,

StartPTS – PTS of when the contained key is to first be applied to packet processing.

hasExpirePTS – A value of "1" indicates the presence of the following expirePTS field.

ExpirePTS – PTS of when the contained key is still valid for processing but after which validity is revoked packet processing.

OpaqueData – Any other opaque data carried in this IPMP data, which may be used to help the synchronization of the key with the protected media stream.

7.5.4 IPMP_RightsData

7.5.4.1 Syntax

```
class IPMP_RightsData extends IPMP_Data_BaseClass :
  bit(8) tag = IPMP_RightsData_tag
{
  ByteArray rightsInfo;
}
```

7.5.4.2 Semantics

rightsInfo – This contains the details of usage rights information.

7.5.5 IPMP_SelectiveDecryptionInit

Defined in Annex C.

7.5.6 IPMP_AudioWatermarkingInit

Defined in Annex D.

7.5.7 IPMP_SendAudioWatermark

Defined in Annex D.

7.5.8 IPMP_VideoWatermarkingInit

Defined in Annex E.

7.5.9 IPMP_SendVideoWatermark

Defined in Annex E.

7.6 User Interaction Messages

These messages allow information to be exchanged between the User and an entity requiring information from the User. How the requested information is displayed or presented to the user as well as how the User supplies possibly requested information is application and implementation dependent.

7.6.1 IPMP_UserQuery

7.6.1.1 Syntax

```
class IPMP_UserQuery extends IPMP_Data_BaseClass:
bit(8) tag = IPMP_UserQuery_tag
{
    bit(1)         inclDisplayTitle;
    bit(1)         inclDisplayText;
    bit(1)         needReplyText;
    bit(1)         inclOptionSelect;
    bit(1)         inclSMIL;
    const bit(3)   reserved = 0b000;
    bit(24)        numOfAltText;
    for(int i=0; i< numOfAltText; i++)
    {
        bit(24)       languageCode;
        if  (inclDisplayTitle) {
           ByteArray titleText;
        }
        if  (inclDisplayText) {
           bit(8)     nDisplayText;
           DTArray    displayText[nDisplayText];
        }
        if  (needReplyText) {
           bit(8)     nPromptText;
           QTArray    promptText[nPromptText];
        }
        if  (inclOptionSelect) {
           bit(8)        nOption;
           OptionArray   option[nOption];
        }
        if (inclSMIL) {
           bit(1)     isReferenced;
           const bit(7)  reserved = 0b0000.000;
           if (isReferenced){
              ByteArray  SMIL_URL;
           }
           else {
              ByteArray  SMIL;
           }
        }
    }
}
```

7.6.1.2 Semantics

`languageCode` - Contains the ISO 639-2:1998 bibliographic three character language code of the corresponding audio/speech or text object that is being described.

`titleText` - Title of dialog display.

`displayText` - Text to be displayed to the user.

`needReplyText` - Text expected back from the user.

`promptText` - Text to be displayed to the end user to indicate purpose of text input field, i.e. “User ID”, “Password”, “PIN”, etc.

`inclOptionSelect` - An option, true/false, input is needed from the user.

`isExclusive` - If more than one option is associated with a given display text, this indicates mutual exclusivity of options.

`optionText` - Text to be displayed indicating purpose of option selection, i.e. “One month purchase”, “One time play”, “Render at 1024/768”, etc.

`SMIL_URL` - Fully qualified location of SMIL file to be displayed.

`SMIL` - SMIL file to be displayed.

7.6.1.3 DTArray

7.6.1.3.1 Syntax

```
class DTArray
{
   bit(16)    ID;
   ByteArray  displayText;
}
```

7.6.1.4 RTArray

7.6.1.4.1 Syntax

```
class RTArray
{
   bit(16)    ID;
   bit(16)    SubID;
   ByteArray  promptText;
}
```

7.6.1.5 QTArray

7.6.1.5.1 Syntax

```
class QTArray
{
   bit(16)    ID;
   bit(16)    SubID;
   bit(1)        isHidden ;
   bit(7)        0b0000.000 ;
   ByteArray  promptText;

}
```

7.6.1.6 OptionArray

7.6.1.6.1 Syntax

```
class OptionArray
{
   bit(1)     isExclusive;
   const bit(7)  reserved = 0b0000.000;
   bit(16)    ID;
   bit(16)    SubID;
   ByteArray  promptText;
}
```

7.6.1.6.2 Semantics:

`IsExclusive` : When set to ‘1’, this implies a mutually exclusive condition of the option selector.

`ID` : A serial number to be associated with the contained data or to associate contained data with other data.

`SubID` : A serial number to associate items within an `ID` group.

`isHidden`: If set to one, the response field for this text shall not reveal the text input by the user, i.e. password input.

`SizeOfArray` : The size in bytes of the data contained in the field *`Data`*.

`Data` : Characters to be displayed.

7.6.1.7 Response

IPMP_UserQueryResponse

7.6.2 IPMP_UserQueryResponse

7.6.2.1 Syntax

```
class aligned (8)UserIPMP_UserQueryResponse extends IPMP_Data_BaseClass:
bit(8) tag = IPMP_UserQueryResponse_tag
{

     bit(1)         inclReplyText;
     bit(1)         inclOptionSelect;
     const bit(6)   reserved = 0b0000.00;
     bit(24)         languageCode;
     if  (inclReplyText) {
        bit(8)     nReplyText;
        RTArray    ReplyText[nReplyText];
      }
      if  (inclOptionSelect) {
        bit(16)         numOfOptions;
        for (i=0; i<numOfOptions; i++)
        {
           bit(1) optionResult;// 0b1 = TRUE, 0b0 = FALSE;
           const bit(7) reserved = 0b0000.000;
        }
    }
}
```

7.6.2.2 Semantics

`replyText` - Text entered by user. Only fields with entered text need be included. If the original request contained reply text fields but the terminal does not support text input then an `inclReplyText` of “`0b0`“ would indicate so.

`optionResult` - Identical semantics and rules as with `replyText` except that all options must be represented and in the order they were initially specified.

7.6.2.3 Response

None Required.

7.7 IPMP Information Delivery Functions

These messages facilitate delivery of IPMP Information among IPMP Tools and terminal, and from bitstream to IPMP Tools.

7.7.1 IPMP_ToolMessageBase

7.7.1.1 Syntax

```
Aligned(8) abstract expandable(228-1)class IPMP_ToolMessageBase: bit(8) tag = 0 {
   bit(8) Version;
   bit(32) sender;
   bit(32) recipient;
}
```

7.7.1.2 Semantics

`IPMP_ToolMessageBase` is an expandable base class for IPMP Tool Messages. The extension tags are defined in the following table.

`Version` indicates the version of syntax used in the messages and shall be set to "0x01".

`Sender` is the IPMP_Descriptor_ID that indicates the tool instance of the originator of the message. "0" is reserved for the terminal.

`Recipient` the IPMP_Descriptor_ID that indicates the tool instance of the intended recipient of the message. "0" is reserved for the terminal.

Table 2 — Tags for messages extending IPMP_ToolMessageBase

8-bit Tag Value	Symbolic Name
0x00	Forbidden
0x01	IPMP_MessageFromBitstream_tag
0x02	IPMP_DescriptorFromBitstream_tag
0x03	IPMP_MessageFromTool_tag
0x04 – 0xCF	ISO Reserved
0xD0 – 0xFE	User Defined
0xFF	Forbidden

7.7.2 IPMP_MessageFromBitstream

7.7.2.1 Syntax

```
class IPMP_MessageFromBitstream extends IPMP_ToolMessageBase :
   bit(8) tag = IPMP_MessageFromBitstream_tag
{
   IPMP_StreamDataUpdate streamData[];
}
```

7.7.2.2 Semantics

Message `IPMP_MessageFromBitstream` is used to deliver `IPMP_StreamDataUpdate` received in the IPMP Stream in the Content to the IPMP Tool specified by the `IPMP_Descriptor_ID` in the `IPMP_StreamDataUpdate`. If an IPMP PES packet delivered in the IPMP Elementary Stream contains more than one `IPMP_StreamDataUpdate` for a specific IPMP Tool, all `IPMP_StreamDataUpdate` for that tool will be included in a single `IPMP_MessageFromBitstream` message.

`messages[]` - an array of `IPMP_StreamDataUpdate's`, the syntax for which is defined in [6.4.2].

7.7.2.3 Response

None required.

7.7.3 IPMP_DescriptorFromBitstream

7.7.3.1 Syntax

```
class IPMP_DescriptorFromBitstream extends IPMP_ToolMessageBase :
   bit(8) tag = IPMP_DescriptorFromBitstream_tag
{
   IPMP_Descriptor toolDescriptor;
}
```

7.7.3.2 Semantics

Message `IPMP_DescriptorFromBitstream` is used to deliver an `IPMP_Descriptor` received in the bitstream to the IPMP Tool specified in the `IPMP_Descriptor`.

`toolDescriptor` - the `IPMP_Descriptor` received in the bitstream.

7.7.3.3 Response

None required.

7.7.4 IPMP_MessageFromTool

This message allows any data derived from IPMP_Data_BaseClass to be exchanged among IPMP Tools and Terminal.

7.7.4.1 Syntax

```
class IPMP_MessageFromTool extends IPMP_ToolMessageBase:
    bit (8) tag = IPMP_MessageFromTool_tag
{
    IPMP_Data_BaseClass messages[];
}
```

7.7.4.2 Semantics

`message – a container for IPMP_Data_BaseClass derived data.`

7.7.4.3 Response

Received message dependant.

Annex A
(normative)

Tool/Content Transfer Messages among Distributed IPMP Devices

A.1 Introduction

This annex is normative only for devices that implement IPMP inter-device communication for the transfer of content and/or IPMP tools.

A.2 Addressing of distributed devices

To address different IPMP devices in a network domain, every IPMP device shall be assigned with a unique 128 bit deviceID. How the deviceID is assigned and maintained unique is an implementation issue.

A.3 IPMP_DeviceMessageBase

A.3.1 Syntax

```
Aligned(8) abstract expandable(228-1)class IPMP_DeviceMessageBase: bit(8) tag = 0
{
   bit(8) Version;
   bit(128) sender_deviceID;
   bit(128) recipient_deviceID;
}
```

A.3.2 Semantics

`IPMP_DeviceMessageBase` is an expandable base class for IPMP Device to Device Messages. The extension tags are defined in Table A-1.

`Version` indicates the version of syntax used in the messages and shall be set to 0x02.[2]

`Sender_DeviceID` indicates the device ID of the originator of the message.

`Recipient_DeviceID` indicates the device ID of the intended recipient of the message.

Table A.1 — Tags for messages extending IPMP_DeviceMessageBase

8-bit Tag Value	Symbolic Name
0x00	Forbidden
0x01	IPMP_MessageFromDevice_tag
0x02 – 0xCF	ISO Reserved
0xD0 – 0xFE	User Defined
0xFF	Forbidden

2 Version number 0x01 is reverved for IPMP_ToolMessageBase

A.4 Device to Device IPMP Message

A.4.1 IPMP_MessageFromDevice

A.4.1.1 Syntax

```
class IPMP_MessageFromDevice extends IPMP_DeviceMessageBase:
    bit (8) tag = IPMP_MessageFromDevice_tag
{
    IPMP_Data_BaseClass message[1..255];
}
```

A.4.1.2 Semantics

This message allows any data derived from IPMP_Data_BaseClass to be exchanged among IPMP Devices. Possible IPMP data carried in the message include the IPMP data defined in [7.1.3], and the device messages extending IPMP_Data_BaseClass with tag values defined in Table A-2 below.

`message` – a container for IPMP_Data_BaseClass derived data, for example, the content/tool transfer messages, or the mutual authentication data for secure content/tool transfer.

Table A.2 — Tags for device messages extending IPMP_Data_BaseClass

8-bit Tag Value	Symbolic Name
0x19	IPMP_RequestContent _tag
0x1A	IPMP_ResponseToContentRequest _tag
0x1B	IPMP_ContentTransfer_tag
0x1C	IPMP_RequestTool_tag
0x1D	IPMP_ResponseToToolRequest_tag
0x1E	IPMP_DeviceID_Notification_tag
0x1F – 0x2F	User Defined
0x30 – 0x3F	ISO Reserved

A.4.1.3 Response

Received message dependant.

A.5 Content Transfer Messages

A.5.1 IPMP_RequestContent

A.5.1.1 Syntax:

```
class IPMP_RequestContent extends IPMP_Data_BaseClass :
   bit(8) tag = IPMP_RequestContent_tag
{
   bit(128) ContentID;
   if (ContentID == 0)
      ByteArray opaqueData;
   bit(128) IPMP_DomainID;
}
```

A.5.1.2 Semantics:

Content_ID – An identification number that identifies the requested content.

OpaqueData – Data to identify requested content.

IPMP_Domain_ID – Identifies the domain. Every IPMP compatible device within the same domain should obtain the same Domain_ID via some secure means from the operator, possible during the time of registration to the operator.

A.5.1.3 Response:

IPMP_ResponseToContentRequest.

A.5.2 IPMP_ResponseToContentRequest

A.5.2.1 Syntax:

```
class IPMP_ ResponseToContentRequest extends IPMP_Data_BaseClass :
   bit(8) tag = IPMP_ ResponseToContentRequest _tag
{
   bit(2) response;
   const bit(6) reserved=0b0000.00;
}
```

A.5.2.2 Semantics:

Response	Note
00	Reserved
01	No Such content
10	Prohibit in Copy/Move
11	Approved

A.5.3 IPMP_ContentTransfer

A.5.3.1 Syntax:

```
class IPMP_ ContentTransfer extends IPMP_Data_BaseClass :
   bit(8) tag = IPMP_ContentTransfer_tag
{
   bit(128) ContentID;
   bit(8)  Sequence;
   bit(32)    PayloadSize;
   bit(8)  Payload[PayloadSize];
}
```

A.5.3.2 Semantics:

Sequence – an 8 bit field indicates the sequence number of this payload, this enables the recipient device to re-form the content. It accumulates upon each IPMP_ContentTransfer message, and it is reset to 0 when it reaches 256.

To securely transfer the content, this entire message could be set as a payload of the IPMP_SecureContainer as defined in [7.2.5].

A.5.3.3 Response:

Not required

A.6 Tool Transfer Messages

A.6.1 IPMP_RequestTool

A.6.1.1 Syntax:

```
class IPMP_RequestTool extends IPMP_Data_BaseClass :
   bit(8) tag = IPMP_RequestTool_tag
{
   bit(128)   ToolID;
   bit(128)   IPMP_DomainID;
   ByteArray  opaqueData;
}
```

A.6.1.2 Semantics

OpaqueData – Data useful for the identification of an appropriate tool type or an indication of the requesting device's platform capability.

A.6.1.3 Response:

IPMP_ResponseToToolRequest.

A.6.2 IPMP_ResponseToToolRequest

A.6.2.1 Syntax:

```
class IPMP_ResponseToToolRequest extends IPMP_Data_BaseClass :
   bit(8) tag = IPMP_ResponseToToolRequest_tag
{
   bit(128) ToolID;
   bit(2)   response;
   bit(6)   reserved;
   if (response==0b11)
   {
   bit(32)    PayloadSize;
   bit(8)   Payload[PayloadSize];
   bit(16)    ToolDescriptionSize;
   bit(8)   ToolDescription [ToolDescriptionSize];
   }
}
```

A.6.2.2 Semantics:

Response	Note
00	Reserved
01	No Such tool
10	Prohibit for transferring
11	Approved

Payload – carries binary tool

ToolDescription – description of the binary tool

To securely transfer the tool, this entire message could be set as a payload of the IPMP_SecureContainer as defined in [7.2.5].

A.6.2.3 Response:

Not required

A.7 Device ID messages

For a device to know all other devices that are connected to it within the same network, the following message is used to broadcast its existence to all other device connected.

A.7.1 PMP_DeviceID_Notification Message

A.7.1.1 Syntax:

```
class IPMP_ DeviceID_Notification extends IPMP_Data_BaseClass :
   bit(8) tag = IPMP_ DeviceID_Notification_tag
{
   bit(128)   IPMP_DomainID;
   ByteArray    OpaqueData[];
}
```

A.7.1.2 Semantics:

`IPMP_DomainID` – The existing Domain ID or Domain ID to be assigned.

`OpaqueData` – Application dependant data.

Note : The `sender_deviceID` and `recipient_deviceID` contained in the `IPMP_DeviceMessageBase` container of this message and may be defined as needed to indicate newly connected devices. Example, in a ring network, all devices may respond to an `IPMP_ReceiverID` of “0” or conversely only a central domain server shall respond in the case of a star network arrangement.

A.7.1.3 Response:

`IPMP_DeviceID_Notification` as needed.

Annex B
(normative)

Schema for Terminal Platform

The XML schema for terminal platform specification supports the specification of CPU, memory, Operating system, virtual machine, etc. Two possible applications within the IPMP scope are:

- Based on the terminal platform XML schema, an IPMP terminal can describe what it is, what the CPU is, what the operating system is, whether there are any assistant hardwares, etc. The IPMP terminal can send the information to the remote site while trying to retrieve a missing IPMP tool. The remote site, after receiving the information, and XML schema parsing, can then decide which tool (a windows DLL or a Java Byte Code, for example) to send.
- Based on the terminal platform XML schema, an IPMP tool can also describe on what platform it can run, and carry the information in the metadata associated with the IPMP tool, or in the Tool Container. When receiving such an IPMP tool, an IPMP terminal can decide whether this tool can run on it.

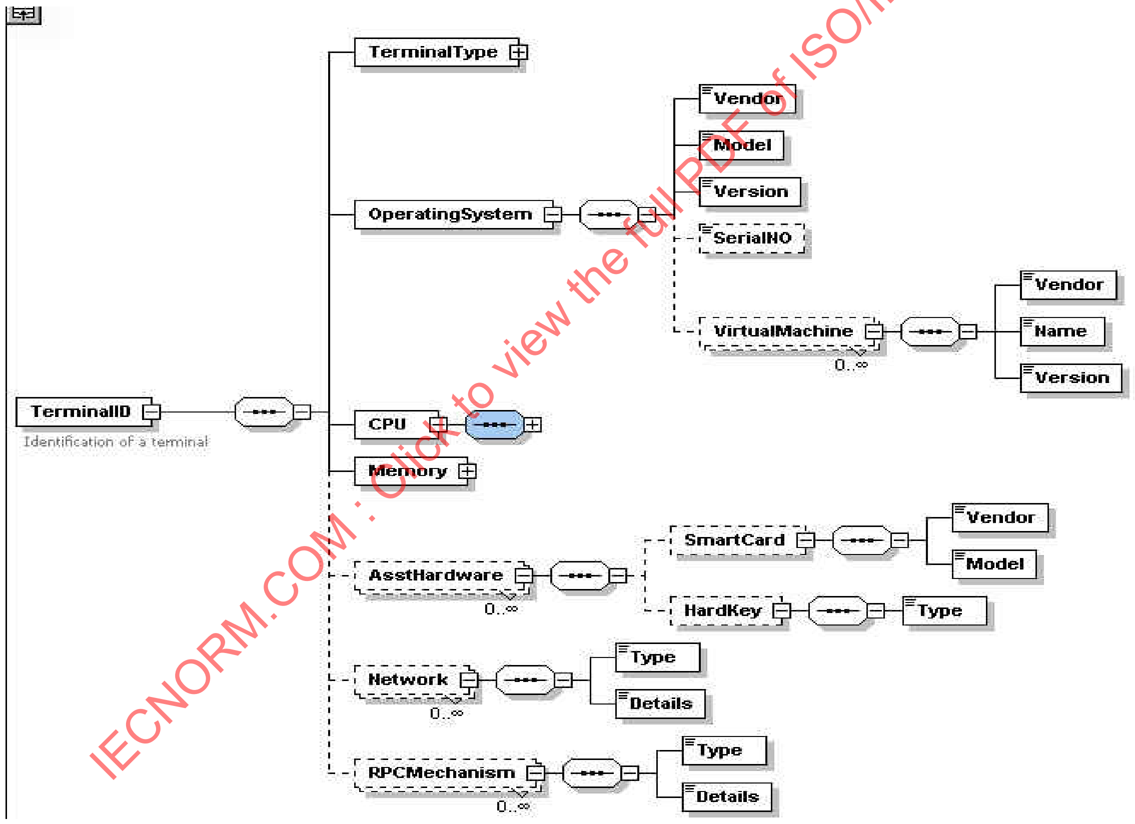

Table B.1 — Schema for Platform Representation

```
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema targetNamespace="urn:mpeg:mpeg2:IPMPSchema:2002"
xmlns:mpeg2ipmp="urn:mpeg:mpeg2:IPMPSchema:2002" xmlns:xsd="http://www.w3.org/2001/XMLSchema"
elementFormDefault="qualified" attributeFormDefault="unqualified">
    <xsd:element name="TerminalID" type="mpeg2ipmp:TerminalIDType">
        <xsd:annotation>
            <xsd:documentation>Identification of a terminal</xsd:documentation>
        </xsd:annotation>
    </xsd:element>
    <xsd:complexType name="TerminalIDType">
        <xsd:sequence>
            <xsd:element name="TerminalType">
                <xsd:complexType>
                    <xsd:sequence>
                        <xsd:element name="Vendor" type="xsd:string"/>
                        <xsd:element name="Model" type="xsd:string"/>
                        <xsd:element name="SerialNO" type="xsd:string" minOccurs="0"/>
                    </xsd:sequence>
                </xsd:complexType>
            </xsd:element>
            <xsd:element name="OperatingSystem" type="mpeg2ipmp:OSType"/>
            <xsd:element name="CPU" type="mpeg2ipmp:CPUType"/>
            <xsd:element name="Memory" type="mpeg2ipmp:MemoryType"/>
            <xsd:element name="AsstHardware" minOccurs="0" maxOccurs="unbounded">
                <xsd:complexType>
                    <xsd:sequence>
                        <xsd:element name="SmartCard" minOccurs="0">
                            <xsd:complexType>
                                <xsd:sequence>
                                    <xsd:element name="Vendor" type="xsd:string"/>
                                    <xsd:element name="Model" type="xsd:string"/>
                                </xsd:sequence>
                            </xsd:complexType>
                        </xsd:element>
                        <xsd:element name="HardKey" minOccurs="0">
                            <xsd:complexType>
                                <xsd:sequence>
                                    <xsd:element name="Type" type="xsd:string"/>
                                </xsd:sequence>
                            </xsd:complexType>
                        </xsd:element>
                    </xsd:sequence>
                </xsd:complexType>
            </xsd:element>
            <xsd:element name="Network" minOccurs="0" maxOccurs="unbounded">
                <xsd:complexType>
                    <xsd:sequence>
                        <xsd:element name="Type" type="xsd:string"/>
                        <xsd:element name="Details" type="xsd:string"/>
                    </xsd:sequence>
                </xsd:complexType>
            </xsd:element>
            <xsd:element name="Downloading" minOccurs="0" maxOccurs="unbounded">
                <xsd:complexType>
                    <xsd:attribute name="Capability" type="xsd:boolean" use="required"/>
                </xsd:complexType>
            </xsd:element>
            <xsd:element name="RPCMechanism" minOccurs="0" maxOccurs="unbounded">
                <xsd:complexType>
                    <xsd:sequence>
                    <xsd:element name="Type" type="xsd:string"/>
                    <xsd:element name="Details" type="xsd:string"/>
                </xsd:sequence>
            </xsd:complexType>
        </xsd:element>
        <xsd:element name="Firmware" minOccurs="0" maxOccurs="unbounded">
            <xsd:complexType>
                <xsd:attribute name="Vendor" type="xsd:string" use="required"/>
                <xsd:attribute name="Name" type="xsd:string" use="required"/>
                <xsd:attribute name="Version" type="xsd:string" use="required"/>
            </xsd:complexType>
        </xsd:element>
    </xsd:sequence>
</xsd:complexType>
<xsd:complexType name="OSType">
    <xsd:sequence>
        <xsd:element name="Vendor" type="xsd:string"/>
        <xsd:element name="Model" type="xsd:string"/>
        <xsd:element name="Version" type="xsd:string"/>
        <xsd:element name="SerialNO" type="xsd:string" minOccurs="0"/>
        <xsd:element name="VirtualMachine" minOccurs="0" maxOccurs="unbounded">
            <xsd:complexType>
                <xsd:sequence>
                    <xsd:element name="Vendor" type="xsd:string"/>
                    <xsd:element name="Name" type="xsd:string"/>
                    <xsd:element name="Version" type="xsd:string"/>
                </xsd:sequence>
            </xsd:complexType>
        </xsd:element>
    </xsd:sequence>
</xsd:complexType>
<xsd:complexType name="CPUType">
    <xsd:sequence>
        <xsd:element name="Vendor" type="xsd:string"/>
        <xsd:element name="Model" type="xsd:string"/>
        <xsd:element name="Speed" type="xsd:integer"/>
    </xsd:sequence>
</xsd:complexType>
<xsd:complexType name="MemoryType">
    <xsd:sequence>
        <xsd:element name="Vendor" type="xsd:string"/>
        <xsd:element name="Model" type="xsd:string"/>
        <xsd:element name="Size" type="xsd:integer"/>
        <xsd:element name="Speed" type="xsd:integer"/>
    </xsd:sequence>
</xsd:complexType>
</xsd:schema>
```

Annex C
(normative)

Selective Decryption Configuration Data

C.1 Introduction

The selective decryption configuration data is used to communicate to the decryptor, how the encryption of the received content bitstream is encrypted, whether all bits are encrypted or only portions of the received bits are encrypted. In the case when only portions of the received bits are encrypted, what portions of the received are encrypted and therefore need to be decrypted.

C.2 IPMP_SelectiveDecryptionInit

C.2.1 Syntax

```
class IPMP_SelectiveDecryptionInit extends IPMP_Data_BaseClass
: bit(8) tag = IPMP_SelectiveDecryptionInit_tag;
{
   bit(8) mediaTypeExtension;
   bit(8) mediaTypeIndication;
   bit(8) profileLevelIndication;
   bit(8) compliance;
   bit(8) numBufs;
   for(i=0, i<numBufs; i++)
      Struct bufInfoStruct {
         bit(128) cipher_Id;
         bit(8) syncBoundary;

         bit(1) isBlock;
         const bit(7) reserved = 0b0000.000;
         if(isBlock)  {
            bit(8) mode;
            bit(16) blockSize;
            bit(16) keySize;
         } else {
            ByteArray Stream_Cipher_Specific_Init_Info;
         }
      }
   }
   bit(1) isContentSpecific;
   const bit(7) reserved = 0b0000.000;
   if(isContentSpecific) {
      bit(8) numFields;
      for(i=0, i< numFields; i++) {
         Struct fieldStruct {
            bit(8) field_Id;
            bit(3) field_Scope;
            const bit(5) reserved = 0b0000.000;
            bit(8) buf;
            bit(1) isMapped;
            const bit(7) reserved = 0b0000.000;
            if(isMapped){
               bit(1) sendMapTable;
               bit(1) isShuffled;
               const bit(6) reserved = 0b0000.000;
               if(sendMapTable){
                  bit(16) sizeMapTable;
                  bit(16) mappingTable[sizeMapTable]
               }
               if(isShuffled){
                  ByteArray shuffleSpecificInfo;
               }
            }
         }
      }
   } else {
      bit(16) nSegments;
      bit(16) RLE_Data[nSegments]
   }
}
```

C.2.2 Semantics

This message allows a terminal to configure a selective decryption tool.

mediaTypeExtension: extends mediaTypeIndication below; The following values are defined.

mediaTypeExtension	Semantics
0x00	ISO/IEC
0x01	ITU
0x02 – 0xCF	ISO Reserved
0xD0 – 0xFE	User Defined
0xFF	Forbidden

mediaTypeIndication: indication of format definition, e.g., MPEG-4 or MPEG-2 etc.; example values could be those of objectTypeIndication in 8.6.6 of ISO/IEC 14496-1.

profileLevelIndication: indication of profile/level of mediaTypeIndication; example values could be those of visualProfileLevelIndication in 8.6.4 of ISO/IEC 14496-1.

compliance: level of compliance to the compression format, i.e., the smallest logical unit in the encrypted bitstream that are correctly recognizable/parsable by the media decoder. For any non-compliant stream, the decryption tool will need to know what marker emulation prevention mechanism was deployed by the encryption tool. The method to emulation prevention is out of scope of this definition. The following values are defined.

Compliance	Semantics
0x00	fully compliant
0x01	compliant to video packet level for video
0x02	compliant to VOP level for MPEG-4 video or Frame level for MPEG-2 video.
0x03	non-compliant for video
0x04	Compliant to GOB level for H.263 baseline
0x05-2F	ISO Reserved for video
0x30	compliant to data frame level for AAC audio
0x31	non-compliant for AAC audio
0x32 – 0x5F	ISO Reserved for audio
0x60 – 0xCF	ISO Reserved
0xD0 – 0xFE	User Defined
0xFF	Forbidden

numBufs: number of buffers needed to hold data to be decrypted separately. The cipher text will be de-multiplexed into different buffers for separate decryption.

bufInfoStruct: a structure holding information needed for each buffer

cipher_Id: type of decryption algorithm used, e.g. , DES, 3DES, AES etc., registered through a Registration Authority.

syncBoundary: this field provides the tool information on what shall be considered a "unit of cipher-text". Bits from different cipher-text unit are decrypted separately. The following values are defined.

syncBoundary	Semantics
0x00	Video packet for video
0x01	VOP (AU) for video
0x02	GOV for video
0x03-2F	ISO Reserved for video
0x30	Data frame for AAC audio
0x31 – 0x5F	ISO Reserved for audio
0x60 – 0xCF	ISO Reserved
0xD0 – 0xFE	User Defined
0xFF	Forbidden

isBlock: block or stream cipher.

mode: mode of block cipher, e.g., CBC, ECB, CFB, OFB, etc., registered through an Registration Authority.

blockSize: block size, in bytes, used for the block cipher

keySize: key size, in bytes, used for the block cipher.

stream_cipher_specific_init_info: normative initialization information if a stream cipher is specified by the cipher_Id.

The above data structures are for representing information pertaining to the buffers used for the decryption process. Below is information pertaining to the fields chosen for decryption.

isContentSpecific: a value of 0b1 indicates that the selective decryption is based on the selection of the fields. Otherwise RLE_Data will specify a collected range of the bitstream selected for decryption, using run length coding of this range information.

numFields: number of fields to be selected for decryption, e.g., MV, DC, DCTsign, Dquant, etc.

fieldStruct: a structure holding information about the fields chosen for decryption

field_Id: index of field based on a predefined list for the given syntax. The following values are defined.

field_Id	Semantics
0x00	Motion vector (MV) for video
0x01	DC coefficients for video
0x02	DCT sign bits for video
0x03	Quantization parameter Dquant for video
0x04	DCT coefficients for video
0x05	All fields ie. All bits in a “unit of cipher text”
0x06-2F	ISO Reserved for video
0x30	Sign bits for AAC audio
0x31	Run-length codewords for AAC audio
0x32	Scale factors for AAC audio
0x32 – 0x5F	ISO Reserved for audio
0x60 – 0xCF	ISO Reserved
0xD0 – 0xFE	User Defined
0xFF	Forbidden

field_scope: represented in three bits, i.e., b2b1b0. A value of 0b1 for b2, b1, and b0 indicates that this field in I, P, and B VOPs, respectively, is selected and put into the buffer that it is associated with.
buf: which buffer this field will be put into.

isMapped: a value of 0b1 indicates that the codewords of a specific field will be mapped, using a mapping table, to an index which is then subject to decryption.

sendMapTable: a value of 0b1 indicates that the mapping table mappingTable will follow. Otherwise, the default mapping of the codeword table defined in the media format definition (e.g. MPEG4 video specification) is used.

sizeMapTable: size, number of entries of the mapping table.

mappingTable: entries of the mapping table. MappingTable[i] indicates that the *ith* codeword in the codeword table defined in the media format definition is mapped to the index of MappingTable[i].

isShuffled: a value of 0b1 indicates the mapped index sequence will be shuffled using a shuffling table specified in Shuffle_Specific_Info.

nSegments: number of disjoint segments that have been generated to signify which segments are selected for decryption

RLE_Data: specifies the number of bits that are to be decrypted or skipped interleavingly, starting from the first segment that is to be decrypted. If the first segment is not to be decrypted, then the value of the first entry shall be zero.

C.3 An example of a selective decryption configuration data (Informative)

One good example of using parametrically configured tools is a configurable selective encryption framework for securing MPEG-2 video content.

It is recognized that for some applications, simplistic, direct application of encryption to content bitstreams poses many problems, most often due to the lack of syntactic structure of the result. The complexity of encrypting the entire content bitstream can also be prohibitive for both very high bit rate content and low power devices. Selective encryption solves both of these problems, the latter due to the reduced complexity associated with only encrypting a portion of the stream and the former by designing the encryption method such that it results in a format compliant, yet encrypted stream. To achieve format compliance, the tool extracts bits from the fields that have been chosen for encryption, concatenates them in an appropriate way, encrypts the concatenation with a chosen cipher appropriate for the application, and then puts the encrypted bits back into their original positions. To maintain compliance, a fixed length index is assigned to each codeword in the VLC code table, and instead of encrypting the code word concatenation, the index concatenation is encrypted, and then the encrypted index concatenation is mapped back to codeword. Any pattern resulting from the encryption of index concatenation can be mapped back to a compliant codeword concatenation. FLC coded fields are treated as special cases of VLC, where the code length does not vary.

In MPEG IPMP, the goal to define a standardized messaging interface between the tools and the terminal provides for an important functionality, namely that of a single configurable tool that could support a wide range of requirements and could be configured to apply only the subset of those that a particular application specified. Using the guidelines for field selection and tools for encrypting VLC in a syntax compliant way, this subclause illustrates an example of a selective decryption configuration message. It shall be noted that in designing these messages, it is assumed that the framework is intended to be applied to MPEG-2 video.

An example of a message is defined here that could be used to configure a format-compliant, selective decryption tool that implemented the method described above. The for-loops are unrolled, but the for-loop syntax is there so it is clear that repetition of various fields was a result of the logic in the data structures. This particular example message tells the decryption tool that it is working with a compliant bit stream. The tool will need two buffers, both of which will use DES as the decryption algorithm in CBC mode with a block size of 64 and a key length of 64. The fields will be grouped on a video packet basis. The fields involved will be MV, DC, DCT sign, and Dquant. The latter three are inserted into a buffer using the values that they take on in the bit stream, while the MVs are mapped to a set of 64 indices (known by the tool) and those are inserted in a separate buffer.

Syntax field name	**Syntactic Value**	**Semantic value**	**Bits**
mediaTypeExtension	0	ISO/IEC MPEG set	8
mediaTypeIndication	97	Visual ISO/IEC 13818-2	8
(visual)profileLevelIndication	8	Main Profile @ Mail Level	8
compliance	0	Bit-level compliant	8
numBufs	2	2 buffers	8
/* For(i=0;i<numBufs;i++) */			
/* numBufs=2 */			
/* buffer 0: motion vectors */			
cipher_Id	0	DES	128
syncBoundary	0	Video packet	8
isBlock	1	Block cipher	1
if(isBlock==1) {			
mode	1	CBC	8
blockSize	64	64	16
keySize	64	64	16
}			
/* i=1 buffer 1: DCT information buffer */			
cipher_Id	0	DES	128
syncBoundary	0	Video packet	8
isBlock	1	Block cipher	1
if(isBlock==1) {			
mode	1	CBC	8
blockSize	64	64	16