INTERNATIONAL STANDARD

ISO/IEC 16500-7

First edition
1999-12-15

Information technology — Generic digital audio-visual systems —

Part 7: Basic security tools

Technologies de l'information — Systèmes audiovisuels numériques génériques —

Partie 7: Outils de sécurité de base

Reference number
ISO/IEC 16500-7:1999(E)

ISO copyright office
Case postale 56 • CH-1211 Geneva 20
Tel. + 41 22 749 01 11
Fax + 41 22 734 10 79
E-mail copyright@iso.ch
Web www.iso.ch

Printed in Switzerland

Contents Page

Foreword

ISO (the International Organization for Standardization) and IEC (the International Electrotechnical Commission) form the specialized system for worldwide standardization. National bodies that are members of ISO or IEC participate in the development of International Standards through technical committees established by the respective organization to deal with particular fields of technical activity. ISO and IEC technical committees collaborate in fields of mutual interest. Other international organizations, governmental and non-governmental, in liaison with ISO and IEC, also take part in the work.

International Standards are drafted in accordance with the rules given in the ISO/IEC Directives, Part 3.

In the field of information technology, ISO and IEC have established a joint technical committee, ISO/IEC JTC 1. Draft International Standards adopted by the joint technical committee are circulated to national bodies for voting. Publication as an International Standard requires approval by at least 75 % of the national bodies casting a vote.

Attention is drawn to the possibility that some of the elements of this part of ISO/IEC 16500 may be the subject of patent rights. ISO and IEC shall not be held responsible for identifying any or all such patent rights.

International Standard ISO/IEC 16500-7 was prepared by DAVIC (Digital Audio-Visual Council) and was adopted, under the PAS procedure, by Joint Technical Committee ISO/IEC JTC 1, *Information technology*, in parallel with its approval by national bodies of ISO and IEC.

ISO/IEC 16500 consists of the following parts, under the general title *Information technology — Generic digital audio-visual systems*:

— *Part 1: System reference models and scenarios*

— *Part 2: System dynamics, scenarios and protocol requirements*

— *Part 3: Contours: Technology domain*

— *Part 4: Lower-layer protocols and physical interfaces*

— *Part 5: High and mid-layer protocols*

— *Part 6: Information representation*

— *Part 7: Basic security tools*

— *Part 8: Management architecture and protocols*

— *Part 9: Usage information protocols*

Introduction

ISO/IEC 16500 defines the minimum tools and dynamic behavior required by digital audio-visual systems for end-to-end interoperability across countries, applications and services. To achieve this interoperability, it defines the technologies and information flows to be used within and between the major components of generic digital audio-visual systems. Interoperability between these components and between individual sub-systems is assured through specification of tools and specification of dynamic systems behavior at defined reference points. A reference point can comprise one or more logical (non-physical) information-transfer interfaces, and one or more physical signal-transfer interfaces. A logical interface is defined by a set of information flows and associated protocol stacks. A physical interface is an external interface and is fully defined by its physical and electrical characteristics. Accessible reference points are used to determine and demonstrate compliance of a digital audio-visual subsystem with this international standard.

A summary of each part follows.

ISO/IEC 16500-1 (DAVIC 1.3.1a Part 2) defines the normative digital audio-visual systems technical framework. It provides a vocabulary and a Systems Reference Model, which identifies specific functional blocks and information flows, interfaces and reference points.

ISO/IEC 16500-2 (DAVIC 1.3.1a Part 12) defines system dynamic behavior and physical scenarios. It details the locations of the control functional entities along with the normative protocols needed to support the systems behavior. It is structured as a set of protocol walk-throughs, or "*Application Notes*", that rehearse both the steady state and dynamic operation of the system at relevant reference points using specified protocols. Detailed dynamics are given for the following scenarios: video on demand, switched video broadcast, interactive broadcast, and internet access.

ISO/IEC 16500-3 (DAVIC 1.3.1a Part 14) provides the normative definition of DAVIC Technology Contours. These are strict sets of Applications, Functionalities and Technologies which allow compliance and conformance criteria to be easily specified and assessed. This part of ISO/IEC 16500 contains the full details of two contours. These are the Enhanced Digital Broadcast (EDB) and Interactive Digital Broadcast (IDB). ISO/IEC 16500-3 specifies required technologies and is a mandatory compliance document for contour implementations.

ISO/IEC 16500-4 (DAVIC 1.3.1a Part 8) defines the toolbox of technologies used for lower layer protocols and physical interfaces. The tools specified are those required to digitize signals and information in the Core Network and in the Access Network. Each tool is applicable at one or more of the reference points specified within the Delivery System. In addition a detailed specification is provided of the physical interfaces between the Network Interface Unit and the Set Top Unit and of the physical interfaces used to connect Set Top Boxes to various peripheral devices (digital video recorder, PC, printer). The physical Delivery System mechanisms included are copper pairs, coaxial cable, fiber, HFC, MMDS, LMDS, satellite and terrestrial broadcasting.

ISO/IEC 16500-5 (DAVIC 1.3.1a Part 7) defines the technologies used for high and mid-layer protocols for ISO/IEC 16500 digital audio-visual systems. In particular, this part defines the specific protocol stacks and requirements on protocols at specific interfaces for the content, control and management information flows.

ISO/IEC 16500-6 (DAVIC 1.3.1a Part 9) defines what the user will eventually see and hear and with what quality. It specifies the way in which monomedia and multimedia information types are coded and exchanged. This includes the definition of a virtual machine and a set of APIs to support interoperable exchange of program code. Interoperability of applications is achieved, without specifying the internal design of a set top unit, by a normative Reference Decoder Model which defines specific memory and behavior constraints for content decoding. Separate profiles are defined for different sets of multimedia components.

ISO/IEC 16500-7 (DAVIC 1.3.1a Part 10) defines the interfaces and the security tools required for an ISO/IEC 16500 system implementing security profiles. These tools include security protocols which operate across one or both of the defined conditional access interfaces CA0 and CA1. The interface CA0 is to all security and conditional access functions, including the high speed descrambling functions. The interface CA1 is to a tamper resistant device used for low speed cryptographic processing. This cryptographic processing function is implemented in a smart card.

ISO/IEC 16500-8 (DAVIC 1.3.1a Part 6) specifies the information model used for managing ISO/IEC 16500 systems. In particular, this part defines the managed object classes and their associated characteristics for managing the access network and service-related data in the Delivery System. Where these definitions are taken from existing standards, full reference to the required standards is provided. Otherwise a full description is integrated in the text of this part. Usage-related information model is defined in ISO/IEC 16500-9.

ISO/IEC 16500-9 (DAVIC 1.3.1a Part 11) specifies the interface requirements and defines the formats for the collection of usage data used for billing, and other business-related operations such as customer profile maintenance. It also specifies the protocols for the transfer of Usage Information into and out of the ISO/IEC 16500 digital audio-visual system. In summary, flows of audio, video and audio-visual works are monitored at defined usage data collection elements (e.g. servers, elements of the Delivery System, set-top boxes). Information concerning these flows is then collected, processed and passed to external systems such as billing or a rights administration society via a standardised usage data transfer interface.

Additional Information

ISO/IEC TR 16501 is an accompanying Technical Report. Further architectural and conformance information is provided in other non-normative parts of DAVIC 1.3.1a (1999). A summary of these documents is included here for information.

ISO/IEC TR 16501 (DAVIC 1.3.1a Part 1) provides a detailed listing of the functionalities required by users and providers of digital audio-visual applications and systems. It introduces the concept of a contour and defines the IDB (Interactive Digital Broadcast) and EDB (Enhanced Digital Broadcast) functionality requirements which are used to define the normative contour technology toolsets provided in ISO/IEC 16500-3.

DAVIC 1.3.1a Parts 3, 4 and 5 are DAVIC technical reports. They provide additional architectural and other information for the server, the delivery-system, and the Service Consumer systems respectively. Part 3 defines how to load an application, once created, onto a server and gives information and guidance on the protocols transmitted from the set-top user to the server, and those used to control the set-up and execution of a selected application. Part 4 provides an overview of Delivery Systems and describes instances of specific DAVIC networked service architectures. These include physical and wireless networks. Non-networked delivery (e.g. local storage physical media like discs, tapes and CD-ROMs) are not specified. Part 5 provides a Service Consumer systems architecture and a description of the DAVIC Set Top reference points defined elsewhere in the normative parts of the specification.

DAVIC 1.3.1a Part 13 is a DAVIC technical report, which provides guidelines on how to validate the systems, technology tools and protocols through conformance and / or interoperability testing.

Information technology — Generic digital audio-visual systems — Part 7: Basic security tools

1. Scope

This part of ISO/IEC 16500 deals with security in DAVIC systems. A number of security tools are specified, including security protocols which operate across DAVIC interfaces, as well as two security related interfaces in the STU. These security tools must be implemented in ISO/IEC 16500 systems conforming to the security profiles.

2. Normative references

The following normative documents contain provisions which, through reference in this text, constitute provisions of this part of ISO/IEC 16500. For dated references, subsequent amendments to, or revisions of, any of these publications do not apply. However, parties to agreements based on this part of ISO/IEC 16500 are encouraged to investigate the possibility of applying the most recent editions of the standards indicated below. For undated references, the latest edition of the normative document referred to applies. Members of ISO and IEC maintain registers of currently valid International Standards. The Telecommunication Standardization Bureau (TSB) maintains a list of currently valid ITU-T Recommendations.

2.1 ISO/IEC and ITU normative references

ISO/IEC 7816-1: 1987, *Identification cards - Integrated circuit(s) cards with contacts - Part 1: Physical characteristics.*

ISO/IEC 7816-2: 1988, *Identification cards - Integrated circuit(s) cards with contacts - Part 2: Dimensions and location of the contacts.*

ISO/IEC 7816-3: 1989, *Identification cards - Integrated circuit(s) cards with contacts - Part 3: Electronic signals and transmission protocols.*

ISO/IEC 7816-4: 1995, *Information technology - Identification cards - Integrated circuit(s) cards with contacts - Part 4: Inter-industry commands for interchange.*

ISO/IEC 7816-5: 1994, *Identification cards - Integrated circuit(s) cards with contacts - Part 5: Numbering system and registration procedure for application identifiers.*

ISO/IEC 7816-6: 1996, *Identification cards - Integrated circuit(s) cards with contacts - Part 6: Inter-industry data elements.*

ISO/IEC 7816-8, *Identification cards - Integrated circuit card(s) with contacts - Part 8: Security related interindustry commands.*

ISO/IEC 8824: 1990, *Information technology - Open Systems Interconnection - Specification of Abstract Syntax Notation One (ASN.1).*

ISO/IEC 8825: 1990, *Information technology - Open Systems Interconnection - Specification of Basic Encoding Rules for Abstract Syntax Notation One (ASN.1).*

ISO/IEC 13818-1: 1996, *Information technology - Generic coding of moving pictures and associated audio information: Systems.*

ISO/IEC 13818-6: 1998, *Information technology - Generic coding of moving pictures and associated audio information - Part 6: Extensions for DSM-CC.*

ITU Recommendation X.509, *Information technology - Open System Interconnection - The Directory: Authentication Framework*, Amendment 2, 1996.

ITU Recommendation X.511, Information technology - Open System Interconnection - The Directory: Abstract Service Definition, Amendment 4, 1996.

2.2 Other normative references

CENELEC EN 50221 : 1997, *Common Interface for Conditional Access and other Digital Video Decoder Applications.*

CENELEC R 206-001 : 1997, *Guidelines for the Implementation and Use of the Common Interface for DVB Decoder Applications.*

ETSI ETR 162: 1995, *Digital Video Broadcasting (DVB): Allocation of Service Information (SI) codes for DVB Systems*.

ETSI ETR 332: 1996-11, *Security requirements capture.*

ETSI ETS 300 468: 1997-01, *Digital Video Broadcasting (DVB) - Specification for Service Information (SI) in DVB systems.*

ETSI ETS 300 608: 1997-08, *Digital cellular telecommunication system (Phase 2) - Specification of the Subscriber Identity Module - Mobile Equipment (SIM-ME) Interface (GSM11.11).*

ETSI prEN 726-3, *Terminal equipment (TE) - Requirements for IC cards and terminals for telecommunication use - Part 3: Application independent card requirements.*

Personal Computer Memory Card International Association, *PC Card Standard - Volume 2: Electrical Specification*, February 1995.

Personal Computer Memory Card International Association, *PC Card Standard - Volume 3: Physical Specification*, February 1995.

Personal Computer Memory Card International Association, *PC Card Standard - Volume 4: Metaformat Specification*, February 1995.

3. Definitions

This clause defines new terms, and the intended meaning of certain common terms, used in this part of ISO/IEC 16500. Annex A of ISO/IEC 16500-1 defines additional terms and, in some cases, alternative interpretations that are appropriate in other contexts. For convenience, the normative definitions below are included in the annex.

3.1. access control: Provides means to access services and protection against unauthorized use of resources, including protection against the use of resources in an unauthorized manner.

3.2. conditional access: A means of allowing system users to access only those services that are authorized to them. In general, conditional access relates to environments in which data is broadcast.

3.3. confidentiality: The protection of information from unauthorized disclosure.

3.4. Control Word (CW): The secret key used for a scrambling algorithm.

3.5. data integrity: The detection or protection of unauthorized modification of data.

3.6. data origin authentication: Corroboration that the identity of the source of data is as claimed.

3.7. Entitlement Control Message (ECM): Conditional access messages carrying an encrypted form of the control words or a means to recover the control words, together with access parameters, ie., an identification of the service and of the conditions required for accessing this service.

3.8. Entitlement Management Message (EMM): Conditional access messages used to convey entitlements or keys to users, or to invalidate or delete entitlements or keys.

3.9. host: A piece of equipment where one or more devices can be connected (eg. an IRD, a VCR, a PC or a STU).

3.10. key management: The generation, storage, distribution, archiving, deletion, revocation, registration, and deregistration of cryptographic keys.

3.11. device: A small piece of hardware, not working by itself, designed to run specialized tasks in association with a host, or to provide resources required by an application but not provided directly by the host. Examples include a conditional access sub system or an electronic program guide application device.

3.12. non-repudiation: The proof of the origin and reception of a message. This means that the sender cannot deny the sending of the message and the receiver cannot deny the reception of the message.

3.13. PC card: Physical packaging standard conforming to PCMCIA Type 1 or Type 2 implementations.

3.14. peer-entity authentication: Corroboration that a communicating entity has the claimed identity.

3.15. resource: A unit of functionality provided by the host for use by a device. A resource defines a set of objects exchanged between device and host by which the device uses the resource.

3.16. scrambling: The process of making a signal unintelligible at the transmission point in order that it can only be received if an appropriate descrambling system is in place at the point of reception. Scrambling can be applied to audio, video or data signals

3.17. service: A set of elementary streams offered to the user as a program. They are related by a common synchronization. They may be made of different data, eg. video, audio, subtitles, other data.

3.18. smart card: Physical packaging standard conforming to ISO/IEC 7816 Parts 1 and 2.

3.19. Transport Stream: MPEG-2 Transport Stream.

4. Acronyms and abbreviations

This clause defines the acronyms and abbreviations used in this part of ISO/IEC 16500. Annex B of ISO/IEC 16500-1 defines acronyms and abbreviations used within ISO/IEC 16500.

AC	Access Conditions
AID	Application Identifier
APDU	Application Protocol Data Unit
API	Applications Programming Interface
ASN.1	Abstract Syntax Notation One
ATM	Asynchronous Transfer Mode
ATR	Answer-to-Reset
BER	Basic Encoding Rules
bslbf	Bit String - Left Bit First
BSS	Business Support System
CA	Conditional Access
CAS	Conditional Access (Sub)System
CBC	Cipher Block Chaining
CD	Committee Draft
CP	Content Provider
CPE	Customer Premises Equipment
CW	Control Word
DE	Data Element
DES	Data Encryption Standard
DF	Dedicated File
DIR	Directory File
DO	Data Object
DSM-CC	Digital Storage Media Command and Control
DVB	Digital Video Broadcast
DVB SI	Digital Video Broadcast Service Information
ECM	Entitlement Control Message
EF	Elementary File
ES	Elementary Stream
ESP	Encapsulated Security Protocol
EMM	Entitlement Management Message
EPG	Electronic Program Guide
ETU	Elementary Time Unit
FP	Filter Program
FS	Filter Set
HFC	Hybrid Fibber Co-ax
HTML	Hypertext Markup Language

IDL	Interface Definition Language
IDO	Inter-industry Data Object
IETF	Internet Engineering Task Force
IP	Internet Protocol
IPPV	Impulse Pay Per View
IPR	Intellectual Property Rights
Ipv4	Internet Protocol Version 4
lsb	Least Significant Bit
lsB	Least Significant Byte
MF	Master File
MJD	Modified Julian Date
MMI	Man Machine Interface
MPEG	Moving Pictures Expert Group
msb	Most Significant Bit
msB	Most Significant Byte
NIU	Network Interface Unit
NP	Network Provider
OMG	Object Management Group
PCR	Program Clock Reference
PDU	Protocol Data Unit
PID	Packet Identifier
PIN	Personal Identification Number
PIX	Proprietary Application Identifier Extension
PMT	Program Map Table
PPV	Pay Per View
PTS	Protocol Type Selection
RFU	Reserved for Future Use
RID	Registered Application Provider Identifier
RP	Record Pointer
RPC	Remote Procedure Call
SP	Service Provider
SPI	Security Parameter Index
STU	Set-Top Unit
SVB	Switched Video Broadcast
TCP	Transmission Control Protocol
TLV	Tag Length Value
TS	Transport Stream
UDP	User Datagram Protocol
URL	Uniform Resource Locator
UTC	Universal Coordinated Time
VoD	Video on Demand

5. Conventions

The style of this specification follows the *Guide for ITU-T and ISO/IEC JTC 1 cooperation. Appendix II: Rules for presentation of ITU-T | ISO/IEC common text (March 1993).*

6. Overview of Security System

The definition of a single, common security system for DAVIC is a very difficult task. Security requirements vary from application to application, and will also depend on the characteristics of the access network, the nature and value of the content, and the business models which govern the system. Furthermore, implementations of security related hardware and software are subject to strict Government regulation, which controls the deployment and movement of such equipment. A final consideration is the very different failure modes of security systems compared to other systems: it is almost impossible to formally verify the strength of individual cryptographic algorithms, and the decreasing cost of computational power makes any given algorithm more vulnerable to attack over time.

For these reasons it is necessary to provide some scope for flexibility in the choice of security mechanisms. In particular, it is desirable to allow the coexistence of multiple cryptographic algorithms, and to provide suitable mechanisms for negotiating their use. This will allow algorithms to be replaced if they are compromised. Note that this will have an implication for interoperability, since a service which makes use of particular security algorithms will not be available unless parties involved in the service delivery have implemented compatible algorithms.

This part of ISO/IEC 16500 defines a number of security tools which must be provided within DAVIC systems implementing the security profiles. These tools include security protocols which operate across DAVIC interfaces. Where possible, these protocols have been designed to be independent of particular cryptographic algorithms. Within certain limits, there exists the flexibility to negotiate the types of algorithms which are used. This is particularly true in the case of algorithms for session confidentiality and integrity.

In the case of conditional access for broadcast services, even more scope for flexibility is required. Due to the nature of the threats in such a system, it is desirable for the design of the system to be kept secret, thereby increasing the cost of an attack on the system. For this reason, it is inevitable that conditional access systems will remain proprietary in nature.

To enhance interoperability, two security related interfaces in the STU are specified. These interfaces allow various components of the security system to be replaced, either in response to attacks, to facilitate customer mobility, or to promote interoperability. By replacing the security devices connected to these interfaces, the STU may be used with different (proprietary) conditional access systems.

The next clause defines a number of security tools. These consist of protocols for

- scrambling the S1 flow;
- scrambling algorithm negotiation and key distribution for the S1 flow;
- authentication and key exchange between the end user and the service provider over the S2 flow;
- confidentiality and integrity of the S2 and S3 flows;
- secure download of data and/or software images;
- parental control.

Clause 9 describes the STU architecture with respect to security, introducing the two conditional access interfaces CA0 and CA1. The CA0 interface is based on the Common Interface (EN 50221), with extensions defined in clause 12 to support interactive services. The CA1 interface is specified in clause 12.

7. Security Tools

This clause defines a number of security tools for DAVIC. These tools are generic in nature, and may be used by a wide variety of applications in different types of systems, to provide interoperable security. Their specification and use does not preclude the implementation of additional, application specific, security services at the application layer.

7.1 S1 Scrambling

A scrambling system is a tool which alters the digital representation of the content (video, audio, coded data) in such a way as to make it unintelligible to unauthorized users of the system, but which may be recovered by authorized users.

The requirement of the use of a scrambling system tool will typically be specified by Content Providers and/or Service Providers (including intermediate service providers), since these parties have a vested interest in the ability to limit the distribution of the content information to selected users on the DAVIC system. The scrambling system tool may be implemented by Content Providers, Service Providers, and/or Network Providers; however, content at any point in the DAVIC system can only be scrambled once.

7.1.1 Scrambled Elements

Scrambling can occur at any location within the system, but it shall be done at the MPEG 2 Transport Stream packet layer. Descrambling shall therefore also be done at the MPEG 2 Transport Stream packet layer.

Scrambling shall be applied only to payload, according to an algorithm determined by a combination of the algorithms available to the scrambling equipment and those available in the STU. The prefix fields must be left in the clear. Since services may be independently supplied to the same MPEG 2 level data multiplex then these services must be independently scrambled within the data multiplex.

The ISO/IEC 13818-1 specification specifically prohibits the encryption of transport packet headers and adaptation fields. This allows transport control and de-multiplexing/re-multiplexing without requiring decryption. This allows the Program Clock Reference (PCR), for example, to be carried and modified by multiplexers without scrambling.

TS packets shall be scrambled independently of each other. This prevents erroneous de-scrambling of information in the event of packet loss.

7.1.2 Control Word Synchronization

Control words (CW) are cryptographic keys necessary to de-scramble the scrambled payload of the MPEG-2 Transport Stream packets. The control words may be distributed or derived in a number of ways, depending on the nature of the security application in use.

For security reasons, the CWs may have a very short life (eg. in the order of seconds). In order to facilitate a real time change of the control words, it may be necessary to store two separate control words. These are known as even and odd control words.

The MPEG-2 Transport Packet header contains the transport scrambling control field to indicate the status and the parity (even or odd) of the CW used to scramble the packet payload. This field shall be used to indicate the use of the scrambling tool. The 00 state is defined in ETR 289 as signifying that the packet is unscrambled. For DAVIC compliant systems, the following additional definitions shall apply:

value	Description
00	Not scrambled
01	Reserved for future use
10	Scrambled with EVEN CW
11	Scrambled with ODD CW

7.1.2.1 Scrambling Algorithm Identification

Considering the potential legal and commercial factors which affect the implementation of scrambling algorithms in various countries, a common scrambling algorithm is not specified for DAVIC systems. As different scrambling algorithms may be used in DAVIC systems, the algorithm to be invoked must be identified.

In the case of retrieval applications, the scrambling algorithm may be negotiated on the S2 flow as described in Subclause 7.5 prior to the delivery of the service. Algorithms are identified using an ASN.1 Algorithm Identifier as defined in ISO/IED 8824 and ISO/IEC 8825. Scrambling algorithms used in DAVIC compliant systems must be registered with an appropriate authority to enable identification.

In the case of distribution applications, the algorithm used to scramble a signal is indicated in a descriptor of the Program Map Table (PMT) during the delivery of the MPEG-2 scrambled service. This allows the STU to determine the correct descrambling algorithm, or to flag appropriate error conditions. Encoding of this descriptor is defined below. The STU may signal the scrambling algorithm and conditional access systems available in the STU as part of the STU profile.

7.1.2.2 The Scrambling Descriptor

The following descriptor may be used in either the global loop, or the local loop of the PMT to indicate the algorithm used to scramble the signal.

Syntax	No. of Bits	Encoding
scrambling_descriptor() {		
descriptor_tag	**8**	**uimsbf**
descriptor_length	**8**	**uimsbf**
scrambling_code	**8**	**uimsbf**
}		

descriptor_tag: This value indicates the type of descriptor. *This value is still to be determined. A request will be made of the DVB to define this descriptor as part of the DVB specification.*

descriptor_length: This is the number of byte immediately following this field to the end of the descriptor. For this descriptor, this value is always 1.

scrambling_code: This field indicates the type of scrambling in use, coded as described below:

Code	Description
0	Reserved
1	DVB Common Scrambling
2-3F	Reserved for DVB
40-7F	Reserved for DAVIC
80-FE	User Private
FF	Reserved

7.1.2.3 Scrambling Algorithm constraints

Because channel change time must be minimized, the decoder must be able to start decryption without significant delay. Therefore it is advisable to originate the scrambler on packet boundaries since this provides for the best access point granularity. That is, such an approach provides the absolute minimum wait time for a stream entry point. Some block cipher scrambling algorithms require the packet payload size to be a multiple of the block length (8 bytes). For these algorithms, some constraints are put on the size of the adaptation field to ensure that the payload is a multiple of a block length. This constraint does not apply to combination block/stream ciphers or to stream ciphers.

7.2 S2/S3 Authentication

A three way authentication mechanism as described in ITU X.509 shall be used to provide mutual authentication between communicating entities in an S2 and S3 flow. Three way authentication eliminates the need for synchronized clocks, and allows for the establishment of session keys and negotiation of session security mechanisms as part of the authentication process. Nevertheless, timestamps appear in the protocol in order to maintain syntactic compatibility with ITU X.511. Their value should be ignored.

7.2.1 Authentication Protocol

The semantics of the authentication exchange are described below:

1. A→B: $A, \{B, T_A, N_A, IntAlgs, ConfAlgs\}^{S_A}, C_A$

   The prover (*A*) initiates the exchange by sending a time stamp T_A and nonce N_A bound to the verifier's identity *B* with the prover's signature. Also included is an optional list of integrity and confidentiality algorithms (*IntAlgs* and *ConfAlgs*) supported by the prover. The prover's identity *A* and certificate C_A are included in the message. The nonce N_A may be either a random number or a sequence number.

2. B→A: $B, \{A, T_B, N_B, N_A, IntAlg, \{K_{AB}^I\}^{P_A}, ConfAlg, \{K_{AB}^C\}^{P_A}\}^{S_B}, C_B$

   The verifier responds with its own time stamp T_B and nonce N_B which is bound to the prover's identity *A*, and the prover's initial challenge N_A with the verifier's signature. Also included is an optional session key and algorithm identifier for encryption and integrity protection of the subsequent session. The algorithms must be chosen from the list provided by the prover in the previous message. These session keys are encrypted with the public key of the prover. This message authenticates the verifier to the prover.

3. A→B: $A, \{B, T_A, N_B\}^{S_A}$

   The final message is a signed response from the prover including the verifier's nonce and identity. This message completes the mutual authentication by verifying the prover's identity to the verifier.

7.2.2 Syntax of Authentication Messages

The following ASN.1 structures are used to transfer authentication messages between the two authenticating parties in an S2 or S3 information flow. Except for **AuthRequestParameter** all structures are taken from ITU X.511. The way in which these data structures are carried in the S2 and S3 information flows is defined below. The **SPIValue** of **AuthRequestParameter** is used to identify the key(s) which may be exchanged during the authentication to the IP security tools defined in Subclause 7.3.

```
AuthRequestParameter ::= SET {
        argument                [0]     BindArgument,
        SPIValue                [1]     BIT STRING OPTIONAL}

BindArgument ::= SET {
        credentials             [0]     Credentials OPTIONAL,
        versions                [1]     Versions DEFAULT {v3} }

Credentials ::= CHOICE {
        simple                  [0]     SimpleCredentials,
        strong                  [1]     StrongCredentials,
        externalProcedure       [2]     EXTERNAL,
        spkm                    [3]     SpkmCredentials }

SimpleCredentials ::= SEQUENCE {
        name                    [0]     DistinguishedName,
        validity                [1]     SET {
        time1                   [0]     UTCTime OPTIONAL,
        time2                   [1]     UTCTime OPTIONAL,
        random1                 [2]     BIT STRING OPTIONAL,
        random2                 [3]     BIT STRING OPTIONAL } OPTIONAL,
        password                [2]     CHOICE {
                unprotected             OCTET STRING,
                protected               SIGNATURE { OCTET STRING} } OPTIONAL }

StrongCredentials       ::= SET {
        certification-path      [0]     CertificationPath OPTIONAL,
        bind-token              [1]     Token,
        name                    [2]     DistinguishedName OPTIONAL }

Token   ::=     SIGNED { SEQUENCE {
        algorithm               [0]     AlgorithmIdentifier
        name                    [1]     DistinguishedName,
        time                    [2]     UTCTime,
        random                  [3]     BIT STRING,
        response                [4]     BIT STRING OPTIONAL,
        bindIntAlgorithm        [5]     SEQUENCE OF AlgorithmIdentifier OPTIONAL,
        bindIntKeyInfo          [6]     BindKeyInfo OPTIONAL,
        bindConfAlgorithm       [7]     SEQUENCE OF AlgorithmIdentifier OPTIONAL,
        bindConfKeyInfo         [8]     BindKeyInfo OPTIONAL,
        dirqop                  [9]     OBJECT IDENTIFIER OPTIONAL,
        attributeCertificate    [10]    AttributeCertificate OPTIONAL,
        clearanceCertPath       [11]    ClearanceCertPath OPTIONAL } }

Versions        ::= BIT STRING { v1(0), v2(1), v3(2) }
```


BindKeyInfo ::= ENCRYPTED { BIT STRING }

The above syntax does not preclude the use of additional private conventions consistent with the syntax which may extend the capabilities of the security services specified herein. In particular, it is to be noted that:

- Use of Distinguished Names may be supplemented, or even supplanted, with use of the Alternative Name attribute in the X.509 certificate. By this means, for example, Internet domain names may be supplied. Also such names are covered by the same certificate as the Distinguished Name. If a Distinguished Name is not supplied, but an Alternative Name is supplied, it is the responsibility of the Directory to establish the mapping between the Alternative Name and the Distinguished Name.
- Clone detection may be based on the use of a monotonically increasing sequence number. Such a sequence number may be incorporated into the Random field provided by the prover. The existence of such a structure is a private arrangement between the prover and the authentication server. Participating servers have the responsibility to implement the detection of clones based on violations of the monotonic increase of the sequence.

7.2.3 Key Use

A single key pair is used for both signing authentication tokens and for encrypting keys in the authentication protocol. This requires only a single certificate chain to be transferred in either direction. Note that long term digital signatures (eg. as used for non-repudiation) may well use a different key pair. This separation of keys for digital signatures on one hand, and authentication and key exchange on the other, is encouraged.

7.2.4 Certificates

Certificates according to ITU-T X.509 Version 3 shall be used.

Certificate revocation is application specific. Certification revocation lists may be obtained by a Directory lookup using a directory access protocol, eg. as described in ITU-T X.500.

7.2.5 Integration with Protocols for S2

Within the S2 flow, the end user will always adopt the role of prover in the authentication exchange. Authentication exchanges are initiated at the discretion of the user, although the server may refuse to interact with the user without such an authentication. This is signaled to the user by issuing a **NO_AUTH** exception.

The authentication messages described above are carried in the **AuthRequest_T** structure in DSM-CC user to user messages. In particular, the first message will be included in the **authInfo** parameter in the **ServiceContextList** of an attempted operation invocation on the server object. The response from the server is carried in the **authData** parameter of a **NO_AUTH** exception. The final response from the client is carried in the **ServiceContextList** of a repeated attempt. Any operation which returns the **NO_AUTH** exception can be used to perform the authentication exchange. For example, the authentication exchange may occur when the client attempts a **directory.open** operation on the server.

The **AuthRequest_T** type is defined as opaque in DSM-CC. DAVIC compliant systems shall make use of the authentication tokens defined in X.511. A BER encoding of the **AuthRequestParameter** ASN.1 structure shall be used for the **authInfo** and **authData** parameters.

7.3 S2/S3 Confidentiality and Integrity

Confidentiality and integrity functions must be performed below DSM-CC because preservation of common bit representations cannot be guaranteed through the RPC operations performed by DSM-CC. Confidentiality shall be provided by encrypting data at the IP level. The IP Encapsulating Security Payload (ESP) [26] shall be used in Transport mode. The IP Authentication Header [25] shall be used to provide integrity and data origin authentication, if required.

7.3.1 Negotiation of Confidentiality and Integrity Algorithms

Confidentiality and integrity algorithms on S2 are negotiated during the authentication exchange described above. This results in the choice of an encryption algorithm and an integrity algorithm, together with session keys for each. In RFC 1825, support of DES CBC mode encryption is recommended. At present, DAVIC does not specify default algorithms.

7.3.2 Definition of the SPI value

The encryption algorithm and integrity algorithm, together with the source and destination addresses, define two uni-directional security associations between the two end-points. These are referenced using a security parameter index (SPI) as described in [24]. The value used for the SPI for a channel is given by **SPIValue** in the authentication exchange. The **SPIValue** shall be defined by the destination party as recommend in [24]. It is a local matter which values are used. The **SPIValue** in the **AuthRequestParameter** of the second message of the authentication exchange shall be used to refer to the security association from the STU to the server. The **SPIValue** in the **AuthRequestParameter** of the final message of the authentication exchange shall be used to refer to the security association from the server to the STU. The **SPIValue** of the **AuthRequestParameter** of the first message may be ignored.

7.3.3 Signaling

The application program shall signal to the IP layer protocol entity if data has to be confidentiality or integrity protected. This signaling is a local matter of the application which is not defined here. The use of confidentiality and/or integrity services is signaled to the receiving party in the IP security protocol headers.

7.3.4 Replay Protection on S2

The integrity mechanisms of IP security protect the PDU which means that an attacker cannot alter the content of TCP layer information since it is protected by an integrity check value. Therefore an already used TCP sequence number will be repeated. This will be detected by the TCP layer.

Note: Two problems are associated with this pragmatic solution:

- *TCP will not report a detection of repeated sequence number as a security error.*
- *TCP sequence numbers may wrap around. An attacker could then insert a packet with a previously used sequence number. This cannot be detected by TCP.*

Proposals for addressing replay protection within IP are being discussed within the IETF. Any solutions developed will be considered by DAVIC in a future revision.

7.4 S2 Digital Signatures

In some applications it is necessary to sign data which is transferred across the S2 flow. This may be for data origin authentication and/or non-repudiation. In general this must be done at the application layer, since the unit of data being signed has significance only in the context of the application protocol being used. Consequently DAVIC does not specify any mechanism to carry out this function.

7.5 DSM-CC Commands for S1 Security Management

Within the retrieval profile, the security services provided on the S1 flow may be negotiated and managed over the S2 flow. This subclause defines DSM-CC User to User commands to configure a security association between the STU and the Server for the S1 flow. This includes the negotiation of scrambling algorithms and the establishment and updating of cryptographic keys. These commands are defined within a separate interface called **S1Security**.

7.5.1 Security Association Configuration

The **assocConfig** operation is used to establish the security association for the S1 flow. This includes negotiation of the scrambling algorithm. Also included is either the transfer of the cryptographic keys needed to descramble

the S1 flow, or an indication of the CA system which will be used to deliver the keys. The **assocConfig** operation forms part of the **S1Security** interface.

7.5.1.1 IDL Syntax

The following syntax shall be used for the **assocConfig** operation:

```
module DAVIC_Security {
        const AccessRole assocConfig_ACR = READER;
        typedef sequence<octet> AlgorithmID;
        typedef sequence<AlgorithmID> AlgorithmIDList;
        typedef sequence<octet> Key;
        typedef sequence<octet> CASystemId;
        exception ALGORITHM_UNAVAILABLE;
        exception UNKNOWN_SYSTEM;
        interface S1Security {
        void assocConfig (
                in AlgorithmIDList supportedAlg,
                in any systemCharacteristics,
                out AlgorithmId s1Alg,
                out Key oddKey,
                out Key evenKey,
                out u_long keyChangeInterval,
                out CASystemId caSystem)
                raises (NO_AUTH, ALGORITHM_UNAVAILABLE, UNKNOWN_SYSTEM);
        };
};
```

7.5.1.2 Semantics

The **S1Security assocConfig()** operation establishes a security association between the STU and the server over the S1 flow. This includes the negotiation of a scrambling algorithm, and the transfer of a cryptographic key from the server to the STU. A lifetime for the key is also defined. In some environments it may be desirable to deliver the cryptographic keys via the CA system. In such a case the server includes the CA system identifier in the return parameters.

The STU lists the scrambling algorithms which it supports via the **supportedAlg** parameter. Algorithms are identified by ISO object identifiers as defined in ITU X.509. The server chooses a single algorithm **s1Alg** which will be used to protect the S1 flow, or raises an **ALGORITHM_UNAVAILABLE** exception. The server returns two keys (**oddKey** and **evenKey**) for use with the scrambling algorithm. The keys are used as the odd and even control words in the scrambling system. The use of two keys facilitates the synchronization of key changes, which are signaled in the MPEG stream.

The lifetime of the keys is returned in the **keyChangeInterval** parameter. If the key change interval is non zero, the STU shall perform a **S1Security getNewKey()** operation within **keyChangeInterval** milliseconds of a key change signaled in the MPEG stream. The key change interval is determined by the server with reference to the **systemCharacteristics** provided by the STU. The **systemCharacteristics** are of the OMG **any** type. The coding of the system characteristics is not defined, but can be used to signal return path delay for example. Unless a pre-determined common encoding exists, the value **null** should be provided by the STU.

Note: While very short key change intervals are supported by this specification, careful consideration must be given to potential network latencies, as well as processing speeds, when determining a suitable value. In general, a minimum key change interval will be of the order of a few seconds, if not minutes or hours. Note that in broadcast conditional access systems the key changes may occur much more frequently.

In some environments it may be necessary to deliver the keys via the CA system. In such a case the server returns a two octet CA system identifier, coded as described in ETR 289. Otherwise a null sequence should be returned.

Note: These messages shall be protected by confidentiality services on the S2 flow. This will be indicated by the ALL SECURE flag in DSM-CC.

7.5.1.3 Privileges Required

The invoking object requires **READER** privileges.

7.5.1.4 Parameters

The following table summarizes the parameters of the **assocConfig** operation:

Type/Variable	Direction	Description
AlgorithmIDList supportedAlg	input	A list of identifiers of scrambling algorithms supported by the STU, ordered by preference (eg. due to speed).
any systemCharacteristics	input	A description of some system characteristics eg. delays in access network or return path. Coding of this parameter is not defined.
AlgorithmID s1Alg	output	The identifier of the scrambling algorithm which will be used to protect the S1 flow.
Key oddKey	output	Odd parity key used for descrambling S1 flow.
Key evenKey	output	Even parity key used for descrambling S1 flow.
u_long keyChangeInterval	output	The minimum time in milliseconds between key changes.
CASystemId caSystem	output	The CA System identifier of the CA system used to deliver cryptographic keys (or null if the keys are to be transferred directly over S2 in the output parameters).

7.5.2 Key Retrieval

If more than one scrambling key is to be delivered over the S2 flow, the **getNewKey** operation shall be used to retrieve subsequent keys.

7.5.2.1 IDL Syntax

The following syntax shall be used for the **getNewKey** operation:

```
module DAVIC_Security {
        const AccessRole assocConfig_ACR = READER;
        typedef sequence<octet> Key;
        interface S1Security {
        void getNewKey (
                out Key key,
                out boolean keyParity,
                out u_long keyChangeInterval)
                raises (NO_AUTH);
        };
};
```

7.5.2.2 Semantics

The **S1Security getNewKey()** operation is used by the STU to retrieve new keys for descrambling the S1 flow. New keys are needed at intervals indicated by the **keyChangeInterval** returned by the previous **getNewKey()** or **assocConfig()** operations. The new key is returned in the **key** parameter, while the parity of this key (odd or even) is indicated by **keyParity**.

Note: These messages must be protected by confidentiality services on the S2 flow.

7.5.2.3 Privileges Required

The invoking object requires **READER** privileges.

7.5.2.4 Parameters

The following table summarizes the parameters of the **getNewKey** operation:

Type/Variable	Direction	Description
Key key	output	New key used for descrambling S1 flow.
boolean keyParity	output	Parity of the above key 0=even; 1=odd.
u_long keyChangeInterval	output	The minimum time in milliseconds until the next key change.

7.6 Secure Download

This subclause defines a mechanism for protecting software and data downloads to the STU. The integrity, source and freshness of the data are protected by this mechanism; however, confidentiality of the data is not provided.

Secure download is implemented by including a security module in the downloaded data image. The **moduleID** of the security module shall be 0xFFFE. No other type of module in a DAVIC compliant system may use the 0xFFFE **moduleID**. The **moduleVersion** for security modules conforming to this specification should be 0x01.

7.6.1 Format of Security DownloadInfoRequest

The STU signals its requirement for a secure download in the **privateDataBytes** in the **DownloadInfoRequest** message. DAVIC systems implementing the security mechanisms defined in this part of ISO/IEC 16500 shall make use of the following format for the **privateDataBytes** (Syntax according to ISO/IEC 13818-6):

Syntax	No. of Bits	Encoding
davicPrivateSecurityData() {		
securityModuleVersion	**8**	**uimsbf**
securityParametersLength	**8**	**uimsbf**
if (securityModuleVersion==1){		
challenge	**32**	**uimsbf**
hashingMechanism	**8**	**uimsbf**
signatureMechanism	**8**	**uimsbf**
certificationPublicKeyHash	**32**	**uimsbf**
}		
for (i=0;i<N;i++) {		
privateDataBytes		
}		
}		

The **securityModuleVersion** indicates which version of the security module definitions the STU supports. A value of 0 indicates it does not support or does not require security. The security module defined in this specification should be indicated by the value 0x01.

The **securityParametersLength** indicates the number of bytes devoted to security parameters in this message. For version 1 defined in this specification the value should be 10 (0x0A).

The **challenge** is a four byte random challenge used to ensure the freshness of the authentication mechanism.

The **hashingMechanism** indicates the hashing mechanism supported by the STU or a security device. Encoding of this field is proprietary, and will be defined by the STU or security device manufacturer.

The **signatureMechanism** indicates the signature mechanism supported by the STU or security device. Encoding of this field is proprietary.

Note: The STU shall send information for identifying the smart card and/or security device manufacturer within the compatibilities or **privateData**. This will allow the server to link the mechanism identifier to a manufacturer and thereby choose the correct mechanism.

The **certificationPublicKeyHash** is the four least significant bytes of the hash of the public key the STU will use to check the signature, or, if a certificate chain is provided, to check the first certificate in the chain. Hashing is according to the hash algorithm indicated by the **signatureMechanism**. It is included as a check that secure download is possible. If not, the download may be aborted, since the signature will fail.

The remainder of the **PrivateData** field is proprietary. The number of proprietary bytes may be computed from the total length of the **PrivateData** field, given in the **DownloadInfoRequest** message.

7.6.2 Format of Security DownloadInfoResponse

The format of DownloadInfoResponse messages in DAVIC is generally unchanged from the definition in DSM-CC; however, where a security module is present, the **ModuleInfo** field corresponding to the security module shall conform to the following format:

Syntax	No. of Bits	Encoding
ModuleInfo() {		
timeStampLength	**8**	**uimsbf**
for(i=0;i<timeStampLength;i++) {		
UTCTimeBytes		
}		
for (i=0;i<N;i++) {		
privateDataBytes		
}		
}		

The **UTCTimeBytes** represent the current time at which the **DownloadInfoResponse** message was generated. It is coded according to the definition of UTC time in ISO/IEC 8824.

The remainder of the **ModuleInfo** field is proprietary. The number of proprietary bytes may be computed from the total length of the **ModuleInfo** field, given in the **DownloadInfoResponse** message.

7.6.3 Format of Security Module

The security module shall conform to the following format:

Syntax	No. of Bits	Encoding
DownloadSecurityModule() {		
signatureLength	**16**	**uimsbf**
for(i=0;i<signatureLength;i++) {		
signatureBytes		
}		
certificateChainLength	**16**	**uimsbf**
for (i=0;i<certificateChainLength;i++) {		
certificateChainBytes		
}		
}		

The **signatureBytes** are obtained by calculating a signature on the concatenation of bit strings used to represent the **DownloadInfoRequest** message, the **DownloadInfoResponse** message and the entire data image excluding the security module. This data image is formed by concatenating the **blockDataBytes** of **DownloadDataBlock** messages, to form individual modules, and ordering these modules as specified in the **DownloadInfoResponse** message. It is recommended that the padding mechanism for the signature generation should be according to ISO 9976-2 (currently CD); however, the actual definition of the padding mechanism will form part of the definition of the signature mechanism used. Note that the headers of **DownloadDataBlock** messages, and the contents of **DownloadDataResponse** messages are not included in the calculation of the signature. In a pure broadcast environment the signature will be calculated over the **DownloadInfoResponse** message and the downloaded data only.

The **certificateChain** is optional. If present it contains a chain of certificates which can be used by the STU to verify the public key of the server. This key would then be used to verify the signature. The certificates are X.509 version 3. The first certificate in the chain must be issued by the certification authority whose key is available at the STU (the hash of which is sent in the **DownloadInfoRequest** message). In this version of the download protocol, it is assumed that there will be at most a single certificate in the chain. If there is no certificate chain, the certification authority public key is used to verify the signature directly.

Note: The certification authority used to verify download images may be different to the general purpose certification authority used for authentication etc, and may take an active role in verifying the correctness/suitability of the code.

7.6.4 Download Reasons in DownloadDataResponse

The **DownloadDataResponse** to the **DownloadDataBlock** of the last block of the secure download, ie. the last block of the security module shall indicate the result of the verification of the signature. The **downloadReason** concerning this are defined in the following table.

downloadReason	Definition
0xF0	Signature OK
0xF1	Signature verification failed
0xF2	Failure due to expiration of certificate
0xF3	Other failure

7.7 Parental Control

7.7.1 Introduction

Parental control enables the user to block certain contents. Parental Control is not a mandatory feature for DAVIC. Supporting parental control may depend on legal issues or the decisions of the provider/manufacturers. However, if parental control is supported the necessary information must be transmitted in an interoperable manner.

DAVIC specifies a rating based system. It is assumed that a specific rating (eg. suitable for children of age 10 years or older) is associated with all content (eg. a movie). However, DAVIC is not responsible for finding the appropriate rating for specific content.

In the context of Parental Control, DAVIC distinguishes between retrieval services, such as VoD, and distribution services such as Pay-per-channel or Switched Video Broadcasting (SVB). In both cases it is necessary that the information about the user on which the access control is based (like this user is allowed to watch all content up to rating xyz) is input to some CPE device (like the STU). This information is called *user access rights* in the following. DAVIC does not specify the user interface for entering the user access rights to the CPE device. DAVIC does not mandate the use of cryptographic methods for enforcing parental control, ie. parental control may or may not be performed by the security subsystem.

7.7.2 Retrieval Services

For retrieval services like VoD the most efficient way for parental control is to block the content at the source, ie. to enforce parental control by the Service Provider. The user access rights must be available to the Service Provider which then can decide if the user is allowed to order a certain content. This information may be stored by the Service Provider in the user profile.

7.7.3 Distribution Services

For distribution services it is assumed that the rating is not necessarily the same for all contents of the channel, ie. the rating shall be content based and not channel based.

For distribution services the network provider usually does not know both pieces of information necessary for parental control, the content rating and the user access rights. Even in the SVB scenario the broadcast control unit and the replication unit in the access network do not know the content rating because the rating is transmitted with the content in S1 and the access network is only a pipe for S1. The broadcast provider usually does not have the ability to block content for certain users.

Therefore, for distribution services (like Pay-per-channel or SVB) DAVIC specifies that if parental control is supported it is supported by the STU. If the content rating is sent it shall be part of the Service Information (SI) data. The content rating shall be based on ETS 300 468, which specifies the transmission of content rating in the SI data of the MPEG transport stream. However, it is recognized that the DVB SI specification does not meet the requirements of content rating schemes of some non-European countries. Therefore, some extensions will be necessary.

The STU reads the rating of the content and compares it with the user access rights. If the user is not allowed to receive the content, the content is blocked. The implementation is left to the STU manufacturer.

In a situation where billing depends on the user behavior (like Pay-per-view or SVB) it must be guaranteed that blocked content is not billed. It is the responsibility of the STU to either not request for a channel with inappropriate rating or to switch (eg. to a default channel) if the program is already received by the STU.

8. Flows and Protocol Stacks

This clause defines the information flows and protocol stacks associated with the security tools specified in clause 7.

8.1 Sample Flow: Scrambled Video on Demand (Informative)

This subclause shows the information flows between different entities associated with the delivery of a scrambled video on demand service. The authentication exchange can take place on the S2 flow at any point during the navigation process. It will be triggered by a no authentication exception raised by the server. The message flows are summarised in Figure 1.

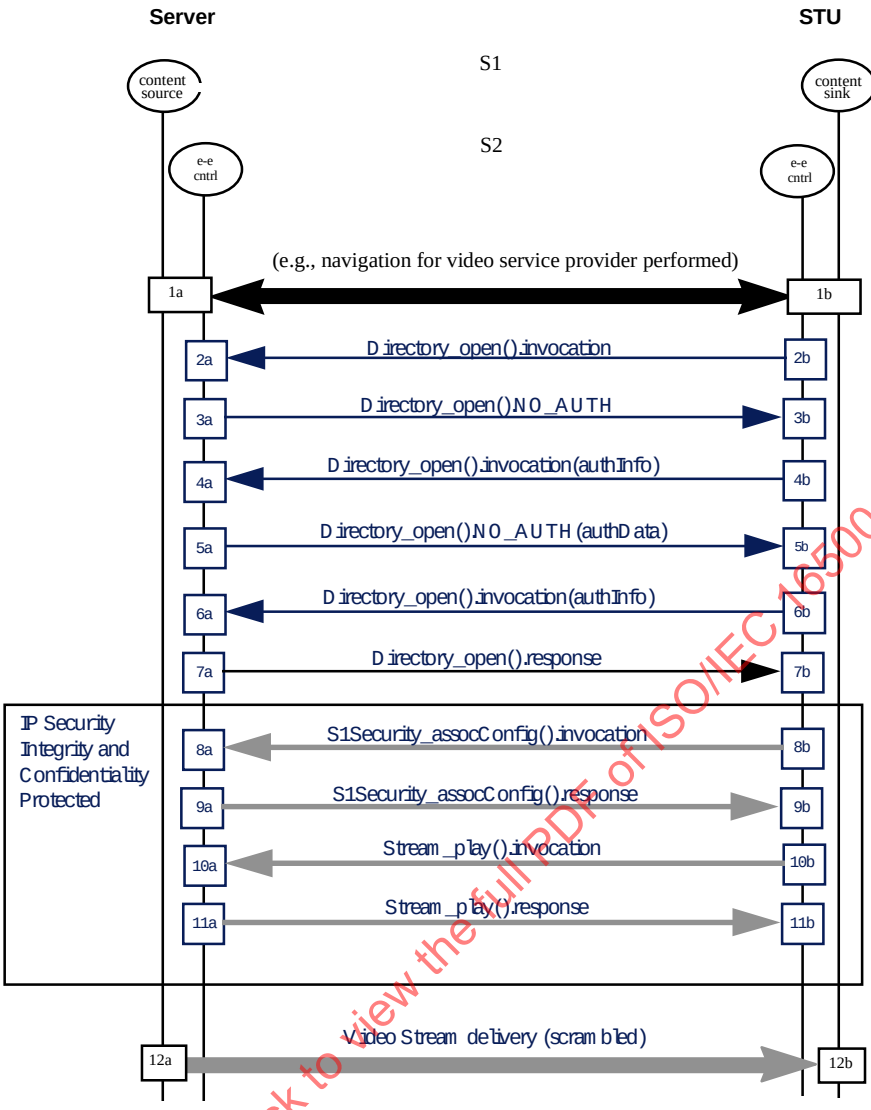

Figure 1. — Information flows for scrambled VoD scenario

<table>
<tr><td>1a-1b</td><td>Navigation through server directory using DSM-CC - no security services have been invoked at this stage.</td></tr>
<tr><td>2a-2b</td><td>Directory open operation invoked by STU, referring to object requiring authentication.</td></tr>
<tr><td>3a-3b</td><td>Server returns NO_AUTH exception with empty authData parameter</td></tr>
<tr><td>4a-4b</td><td>STU repeats the Directory open operation, this time carrying security authenticate information in the service context list of the operation. AuthInfo parameter is BER encoding of following:

AuthRequestParameter ::= SET {
argument [0] BindArgument,
SPIValue [1] BIT STRING }

BindArgument ::= SET {
credentials [0] Credentials,
versions [1] Versions}
Credentials ::= CHOICE {
strong [1] StrongCredentials}

StrongCredentials ::= SET {
certification-path [0] CertificationPath,
bind-token [1] Token }

Token ::= SIGNED { SEQUENCE {
algorithm [0] AlgorithmIdentifier
name [1] DistinguishedName,
time [2] UTCTime,
random [3] BIT STRING,
bindIntAlgorithm [5] SEQUENCE OF AlgorithmIdentifier,
bindConfAlgorithm [7] SEQUENCE OF AlgorithmIdentifier }}</td></tr>
<tr><td>5a-5b</td><td>Server responds with NO_AUTH exception. The authData parameter of this exception contains the following ASN.1 object encoded in BER:

AuthRequestParameter ::= SET {
argument [0] BindArgument,
SPIValue [1] BIT STRING}

BindArgument ::= SET {
credentials [0] Credentials,
versions [1] Versions }

Credentials ::= CHOICE {
strong [1] StrongCredentials }

StrongCredentials ::= SET {
certification-path [0] CertificationPath,
bind-token [1] Token }

Token ::= SIGNED { SEQUENCE {
algorithm [0] AlgorithmIdentifier
name [1] DistinguishedName,
time [2] UTCTime,
random [3] BIT STRING,
response [4] BIT STRING,
bindIntAlgorithm [5] SEQUENCE OF AlgorithmIdentifier,
bindIntKeyInfo [6] BindKeyInfo,
bindConfAlgorithm [7] SEQUENCE OF AlgorithmIdentifier,
bindConfKeyInfo [8] BindKeyInfo } }</td></tr>
</table>

6a-6b	The STU re-issues the Directory open invocation, again including the security authenticate parameter **authInfo** in the service context list. The following ASN.1 type is BER encoded to form **authInfo**: **AuthRequestParameter ::= SET {** **argument [0] BindArgument,** **SPIValue [1] BIT STRING}** **BindArgument ::= SET {** **credentials [0] Credentials,** **versions [1] Versions }** **Credentials ::= CHOICE {** **strong [1] StrongCredentials }** **StrongCredentials ::= SET {** **certification-path [0] CertificationPath,** **bind-token [1] Token }** **Token ::= SIGNED { SEQUENCE {** **algorithm [0] AlgorithmIdentifier,** **name [1] DistinguishedName,** **time [2] UTCTime,** **response [4] BIT STRING } }**
7a-7b	The Directory open operation succeeds. At this stage both ends of the connection have complete information for the S2 security association, and can invoke the security services negotiated for S2. In general this will happen on the first operation to the new object (in this case a stream object providing a movie on demand).
8a-8b	STU invokes **assocConfig()** operation to negotiate the security services on S1. This operation informs the server of the scrambling algorithms supported by the STU.
9a-9b	The server responds indicating the scrambling algorithm it will use to protect the content, and informing the STU of the keys it will use to scramble the content. Note that this message must be protected by S2 confidentiality services.
10a-10b	The STU invokes a **Stream play()** operation to begin playing the movie.
11a-11b	Response from play operation.
12a-12b	The scrambled movie is sent on the S1 flow.

9. Security Interfaces

There are many advantages in providing detachable security device(s) on the STU. Such an approach allows security functionality to be replaced in response to successful attacks, or changing functional requirements. While it is possible to implement changes in the STU functionality which is not related to security by means of software downloads, this approach is not acceptable for security related functions. These will require some form of trusted (tamper-resistant) hardware.

An additional advantage of a detachable device is that it allows the STU to be generic, in that it contains no customer specific data. This allows customer mobility (in that a customer may make use of other STUs - eg. when traveling, or visiting friends), easy replacement of the STU, and reuse of an STU (eg. if a customer moves or ceases to subscribe to a service). This solution permits open competition while maximizing the commonality of the STU.

Two interfaces are defined to different parts of the security system. These are CA0 and CA1 as shown in Figure 2. A DAVIC STB or STU that is compliant with the security tools must offer at least one of these interfaces. The first interface CA0 is to all security and conditional access functions, including the high speed descrambling functions, while CA1 is an interface to a low cost, tamper resistant device used for low speed cryptographic processing. This cryptographic processing function is implemented in a smart card. Where a CA1 interface is implemented, the filtering function described in this specification must also be implemented.

However, if the DAVIC STB utilizes the DAVIC Internal A0 tool, it may support the physical CA1 smart card reader component on the NIU rather than in the STU. In this case all other security elements reside in the STU.

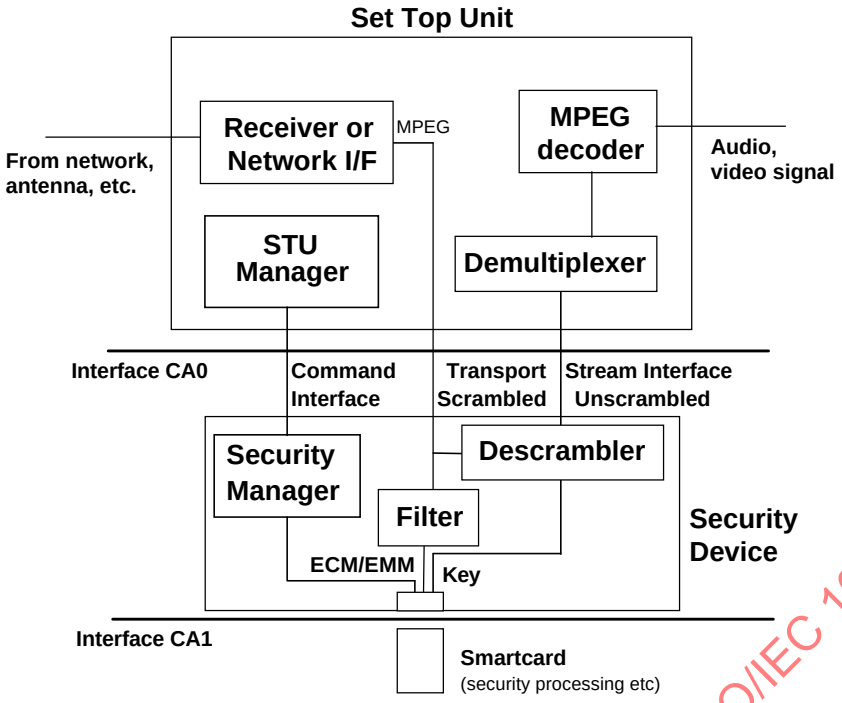

Figure 2. — Security interfaces in the STU

10. Security Interface CA0

10.1 Introduction

The CA0 interface is between a PC card based security device and a host where stream decryption, conditional access, security and more generally defined proprietary functions may be implemented in the PC card. This solution also allows service providers to use security devices containing solutions from different suppliers in the same network thus increasing their choice and anti-piracy options. The interface to be used is based on the Common Interface standard CENELEC EN 50221 and CENELEC R 206-001.

The specification defines a set of functions, partitioned into resources. Not all of the functions required are defined in CENELEC EN 50 221. Necessary additional resources to support DAVIC systems are defined in clause 12. DAVIC has identified these to DVB together with requirements for other additions for incorporation in future revisions of the standard.

Annex B.3 of CENELEC EN 50 221 specifies an optional smart card reader resource class dedicated to electronic commerce applications. The resource is intended to support a smart card reader in a host or on a device used for short term sessions, such as a Bank Card, teleshopping or pay-per-view transactions. This resource is intended to offer users alternative payment options which use smart cards issued by financial services organizations. This Annex of CENELEC EN 50 221 is also adopted, as an optional resource.

10.2 Additional DAVIC Requirements for CA0

10.2.1 Host to Security Device Authentication

This has been identified as needed to support clone detection, and also to associate a PC card with a particular host. These requirements will be refined in a future version of the specification.

10.2.2 Security Services

S2 security is supported by resources defined in clause 12. However applications running on the STU will also require security services, for example to support payment protocols. These services will also be provided by the

PC card, but the exact nature of the Application Programming Interface to be provided will be defined in a future version of the specification.

11. Profiles/Contours

DAVIC 1.3.1 defines two security profiles:

1. a secure distribution services profile which supports
   - the interfaces CA0/CA1,
   - the secure download
   - the STU authentication of the smart card,
2. a secure retrieval services profile which supports
   - the interfaces CA0/CA1,
   - the secure download
   - the S1, S2, S3 security tools,
   - the S1 negotiation.

Both profiles are optional.

12. Security Interface CA1

12.1 Introduction and scope

12.1.1 Introduction

The following text specifies the CA1 interface. The approach taken emphasizes optimization of the implementation cost and user-friendliness. The governing principle in the design is to provide sufficient detail to promote interoperability of components without imposing unnecessary limitations on implementations. The CA1 interface is cost effective and has benefits for both the replacement/change of conditional access systems and for providing the means to allow the STU to be user independent. The CA1 interface is based on proven technology. Using existing standards promotes openness and takes advantage of economies of scale due to previous or concurrent use in other industries.

The specification is compliant to the ISO/IEC 7816 standard for smart cards (parts 1-6). Elements of ISO/IEC 7816 parts 1-6 are used, but the usage of especially parts 3 and 4 are carefully limited to selected, minimum functions so as not to burden the implementation with excessive overhead. The solution is network-independent and is able to co-exist with other security interface definitions. By emphasizing the use of existing standards, the openness of the CA1 interface is enhanced.

12.1.2 CA1 reference model

The architecture of the CA1 interface is shown in Figure 3. In this figure, the incoming transport stream (TSin) is parsed and the appropriate streams are sent to the **descrambling unit** and the **message filter**. The specification assumes that data are descrambled in MPEG-2 Transport Stream [ISO/IEC 13818-1] level. The message filter further parses the input stream to extract eg. CA messages. These are then sent for processing by the smart card. The message filter is initialized by the smart card for filtering. This is shown by the (logical) relation between the smart card and the message filter in Figure 3.

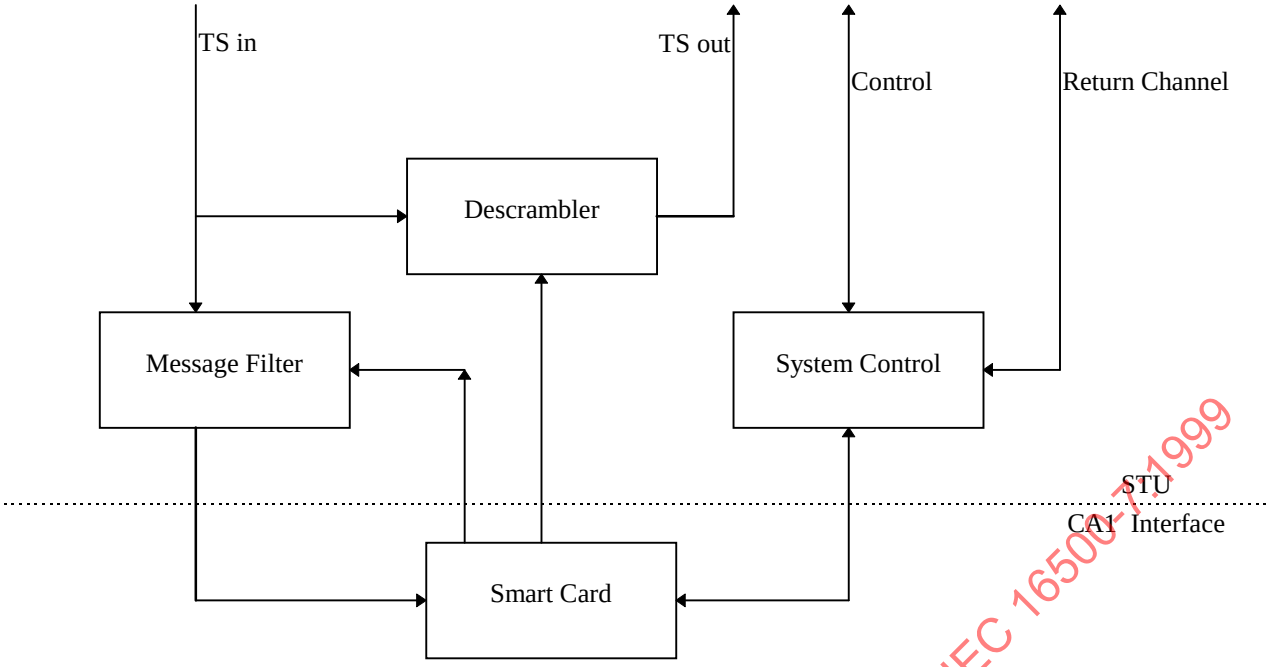

Figure 3. — CA1 Reference Model

The **smart card** processes CA messages that belong to its CA system and uses the results of ECM processing to initialize the **descrambler unit**. This is shown in the figure by the (logical) relation between the smart card and the descrambler. Data exchange may include CA system identification. The **smart card** also supports interaction with external entities by interfacing with the **system control** unit. This interaction can be both with a local user (eg. password entry) and/or with a network or servers (eg. service ordering, authentication) over a **return channel**. This specification is not restricted to a particular implementation of the return channel. For example, the return channel may flow over a different physical path than the forward channel data.

After being initialized by the **smart card**, the **descrambler** uses the CWs to process the incoming MPEG-2 stream into the clear. The stream is then passed as TSout for further processing by the STU or other units.

In this architecture, there is a physical separation of security functions. The **smart card** is distinguished from the **descrambler/message filter** at all layers.

The architecture for the CA1 interface includes a physical layer (based on ISO/IEC 7816-1,2) and also contains electrical, protocol and data structure specifications (based on ISO/IEC 7816-3,4). ISO/IEC 7816-5 and ISO/IEC 7816-6 are also used. Based on these standards this CA1 specification provides specific usage methods/structures and sets limits and constraints that promote cost-effective implementation without significant reduction of functionality.

This specification is based on the following assumptions about DAVIC conditional access implementations:

- Each CA system is identified by its CA_system_id (as defined in ETR 289)
- One STU may function with one or more CA system(s)
- One or more service providers may use the same security device.
- The CA1 interface is delivery network independent

The different elements in the CA1 reference model have the following functionality:

TS in: This is the full transport stream input to the STU.

TS out: This is the full transport stream output to the back-end of the STU. This stream may include descrambled services.

Control: This represents the user interface, and any control the user may apply to the STU, such as channel changes and setting maturity ratings.

System Control: This represents the STU central control microprocessor. This controller can communicate with the smart card for user interfaces, and setting services.

Return Channel: This is the DAVIC return channel which may be used for authentication, or digital interactive services.

CA1 Interface: This is an ISO 7816 compliant interface used to communicate with the smart card.

Descrambler: this is the common descrambler unit used to descramble services using control words from the smart card.

Message Filter: this is a combination of hardware and/or software filters used to reduce the ECM or EMM data rate from the high speed transport stream to a rate suitable for the smart card interface. The filter program (state machine) is read from the smart card and loaded into the message filter.

Smart Card: This is the actual integrated circuit card defined by ISO. This intelligent card contains the conditional access system used to control the services available to the STU. This card also handles DAVIC authentication, digital signatures, and may contain some high-level man-machine interfaces.

12.2 Notation

In all representations of data objects the left most bit represents the most significant bit (msb) and the right most bit represents the least significant bit (lsb). The same holds for bytes: the left most or first byte represents the most significant byte (msB), while the right most or last byte represents the least significant byte (lsB). The Figure 4 illustrates this.

Figure 4. — Representation of data objects.

12.3 Physical characteristics of the CA1 security device

The physical characteristics of the CA1 security device shall satisfy the specifications ISO/IEC 7816-1 and ISO/IEC 7816-2.

12.4 Electronic signals and transmission protocols on the CA1 interface

12.4.1 Additions and restrictions to ISO/IEC 7816-3

The electronic signals and transmission protocols on the CA1 interface shall satisfy ISO/IEC 7816-3, but with the following additions and restrictions:

1. The STU shall be able to support raw data rates of communication of up to 107.563 kilobit per second, between the security device and the STU.
2. The smart card shall respond with the asynchronous answer-to-reset (ATR) as described in ISO/IEC 7816-3.
3. The ATR shall be transmitted in half-duplex mode.
4. The character frame for the ATR shall be as in clause 6.1.2 of ISO/IEC 7816-3. That is, a character shall consist of 12 bits, which are: a start bit, 8 bits of information, a parity bit and 2 stop bits. This is only a repetition of ISO/IEC 7816-3. It does not give any restriction to the standard.
5. The ATR shall specify the T=0 protocol. The STU shall support T=0.
6. There is no requirement for the STU nor the smart card to support Vpp.
7. The STU must support the PTS protocol, but it is not mandatory for the smart card to support it.
8. During ATR the clock frequency shall be in the range defined by ISO.

9. During normal operation, the clock frequency shall be at least 5Mhz.

10. The STU shall execute an external reset before using the card.

11. Support of the retry mode is not mandatory in the STU.

12.4.2 Logical channels

It is mandatory for the STU to support at least two logical channels as described in ISO/IEC 7816-4.

12.5 CA message format and filter specification

12.5.1 CA message mechanism and format

The CA message mechanism and format shall be according to ETR 289. Figure 5 shows the CA message syntax from this specification.

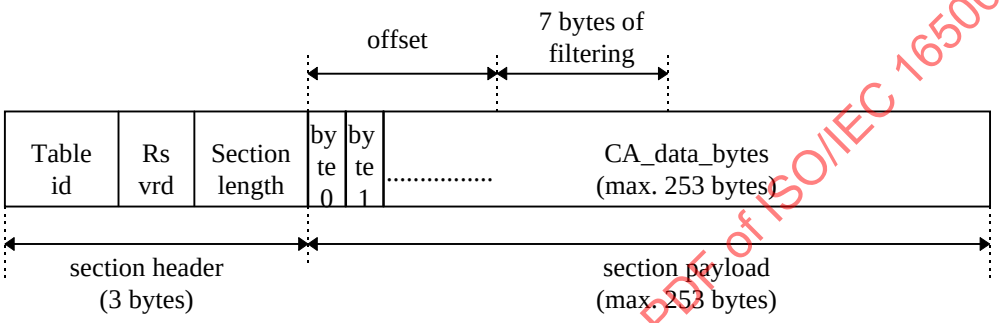

Figure 5. — CA message syntax according to DVB.

Table_id: this is an 8-bit field for message type identification.

Rsvrd: this is a 4-bit field which consist of a **section_syntax_indicator** (1 bit) field, a **DVB_reserved** (1 bit) field, and an **ISO_reserved** (2 bits) field.

Section_length: this is a 12-bit field, that specifies the number of bytes that follow the section_length field up to the end of the section.

CA_data_byte: this is an 8-bit field which carries private CA information.

12.5.2 Filtering of a CA message

Filters are used by a STU to extract required messages for an incoming MPEG-TS stream. In the case of a conditional access system these messages are normally EMMs and ECMs. For the STU to extract these messages it needs to be told the characteristics of these messages. In the case of this specification these characteristics are retrieved from the smart card. The STU uses these characteristics together with the CA_descriptors found in the CAT or PMT to program its filters dedicated to conditional access and assign them to the correct PID. This is not a trivial task as:

- Filters may be time variable.
- Multiple CA messages may exist for one service which requires the set-up of multiple filters.
- Different CA messages for multiple services may be contained in one PID. This requires different filter programs on the same PID to acquire the correct message. The different services should come from the same service provider.
- Support of remultiplexing requires extra association information.

For filtering the table_id byte together with up to 7 filter bytes are compared within the filterable area of a CA message. This area is the first 38 bytes of the payload of the CA message. The offset sets the first location for

comparing the (up to) 7 filter bytes. Use of offset values other than zero may cause some degradation of the filtering data rate. See filter program section for details.

When a message is filtered it shall be passed to the smart card. This filtering scheme is intended for ECM and EMM type messages. For ECMs the smart card normally will return (a) control word(s) required for descrambling.

12.5.3 Filter specification

The filter part for conditional access shall consist of 16 filters in parallel. Each filter:

- must filter on one PID stream
- shall be able to filter on the table_id of a section
- shall be able to filter up to 7 contiguous bytes of section payload
- the 7 contiguous bytes may start within the first 31 bytes of the section payload

Depending on the offset the speed with which the filter is able to process data may be variable. This depends on the physical implementation of the filter. Regardless of the implementation the filter must maintain a throughput of a 1000 sections per second for all offsets.

Any data filtered in the first 7 bytes of the section payload must have a throughput of the full transport data rate. These bytes can be used to decrease the data rate to enable further filtering in the other part of the filterable area such that the input rate for this part does not exceed 1000 sections per second for all filters defined by the filter set.

12.5.4 Filter programming

12.5.4.1 Introduction

Within this specification two main objects used for filtering can be distinguished. These are the filter_set and the filter_program(s). The latter contains the detailed instructions on how to filter a message and how to update the filter. The filter_set groups the various filter programs in different categories, depending on the type of CA message they should operate on. This approach of using a higher level object grouping and referencing the different filter programs has a number of advantages:

- Easy indication of type of filtered message, for example ECM or EMM.
- Easy and fast referencing of filter programs by message type.
- Easy to define global settings for all filter programs or for specific types (for example: command to send message to smart card, indication of preprocessing, etc.).

12.5.4.2 Filter set

The filter set groups the different programs for filtering of one type of message. There must be one filter set to define all of the filtering for all CA messages. A filter set object logically groups filter programs according to the type of message. Typical messages are ECMs and EMMs, each of which could use multiple filter programs to generate the stream of sections to the smart card.

Syntax	No. of bits	Encoding
Filter_set(){		
filter_set_tag	**8**	**uimsbf**
filter_set_length	**8**	**uimsbf**
filter_set_info_length	**8**	**uimsbf**
for(i=0;i<N;i++){		
object()		
}		
for(i=0;i<N1;i++){		
priority	**2**	**uimsbf**
filter_type	**6**	**uimsbf**
number_of_filter_identifiers	**8**	**uimsbf**
for(i=0;i<N2;i++){		
filter_program_number	**8**	**uimsbf**
}		
filter_program_info_length	**8**	**uimsbf**
for(i=0;i<N3;i++){		
object()		
}		
}		
}		

filter_set_info_length: this 8-bit field gives the length in bytes in the loop of objects immediately following.

priority: this 2-bit field allows the STU to prioritize the order that messages are sent to the smart card. A value of zero is the highest priority. If two messages of different priority are in the output buffer to the smart card, the higher priority message is sent first. Messages of equal priority are sent in order of arrival to the output buffer.

filter_type: this 6-bit field identifies which messages are intended for filtering. The filter type is coded in the following way:

Value	Meaning
'00'	reserved for future use
'01'	EMMs
'02'	ECMs
'03'-'1F'	reserved for future use
'20'-'3E'	user private
'3F'	reserved for future use

number_of_filter_identifiers: this 8-bit field identifies the number of bytes in the loop immediately following.

filter_program_number: this 8-bit field gives an index to a filter program. All programs shall have a different filter number. This number can be used to identify a specific filter program.

filter_program_info_length: this 8-bit field gives the length in bytes of the loop of objects immediately following.

object: These objects may be one of those given in Subclause 12.5.5.2 in the FS column. Objects in the global loop are the default if present. Objects in the second loop provide specific overrides for individual filter types.

12.5.4.3 Filter program object

The filter program object describes the programming of a filter.

Syntax	No. of bits	Encoding
Filter_program(){		
filter_program_tag	**8**	**uimsbf**
filter_program_length	**8**	**uimsbf**
filter_program_number	**8**	**uimsbf**
for(i=0;i<N1;i++){		
object()		
}		
}		

filter_program_number: this 8-bit field identifies the number of the filter program. It also serves as the index referencing by the filter set.

filter_program_length: this 8-bit field identifies the number of object bytes.

object: These objects may be one of those given in Subclause 12.5.5.2 in the FP column.

12.5.5 Filter objects

12.5.5.1 Introduction

Filter objects may be included in a filter set object or a filter program object as required. A filter set object logically groups filter programs according to the type of message. Typical messages are ECMs and EMMs, each of which could use multiple filter programs to generate the stream of sections to the smart card.

12.5.5.2 Filtering object identification and location

The following table lists the objects defined for filtering. It also gives the intended placement in the filter set (FS) or filter program (FP).

Object	FS	FP
APDU_object	*	*
CA_descriptor_association_object		*
CA_descriptor_selection_object		*
Pre-processing_object	*	*
CW_association_object		*
filter_state_object		*

12.5.5.3 Filtering object coding

The following semantics apply to all objects in this subsection:

object_tag: the object tag is an 8-bit field which identifies each object.

object_length: the object length is an 8-bit field specifying the total number of bytes of the data portion of the object following the byte defining the value of this field.

12.5.5.3.1 APDU object

The APDU object specifies the instruction to send the filtered message to the smart card. If the object is used in the filter set it holds for the complete set. For a specific filter program the set or the default settings can be overruled by applying this object in the filter program.

Syntax	No. of bits	Encoding
APDU_object(){		
object_tag	**8**	**uimsbf**
object_length	**8**	**uimsbf**
CLA	**8**	**uimsbf**
INS	**8**	**uimsbf**
P1-P2	**16**	**uimsbf**
}		

If this object is not present the default Put_data(send_message) command shall be used.

CLA: this is an 8-bit field specifying the value of the CLA byte in the APDU.

INS: this is an 8-bit field specifying the value of the INS byte in the APDU.

P1-P2: this is a 16-bit value specifying the value of the P1 and P2 bytes in the APDU.

12.5.5.3.2 CA descriptor association object

The CA descriptor association object enables the possibility to make a distinction between ECMs referenced from the global part of the PMT and ECMs referenced from a specific local part of the PMT. The object makes a connection with the component_tag in the stream_identifier_descriptor as defined in DVB-SI ETS 300 468. If the object is not present, the MPEG defined association shall be used. This object enables reliable reference to a PID number.

Syntax	No. of bits	Encoding
CA_descriptor_association_object(){		
object_tag	**8**	**uimsbf**
object_length	**8**	**uimsbf**
component_tag	**8**	**uimsbf**
}		

component_tag: this 8-bit field identifies the CA_descriptor of a component stream for associating a filter program with the correct transport packet PID containing the desired CA messages.

12.5.5.3.3 CA descriptor selection object

The CA descriptor selection object allows selection of the correct CA_descriptor when multiple CA_descriptors occur with the same CA_system_id within one loop of the PMT or CAT. The first CA_descriptor found to match this pattern will be used to determine the PID for the ECM (or EMM) stream.

Syntax	No. of bits	Encoding
CA_descriptor_selection_object(){		
object_tag	**8**	**uimsbf**
object_length	**8**	**uimsbf**
offset	**8**	**uimsbf**
pattern	**8*n**	**uimsbf**
mask	**8*n**	**uimsbf**
}		

offset: this 8-bit field gives the start position of the pattern and mask within the private bytes of the CA_descriptor.

pattern: this 8*n-bit field is a contiguous block of data to compare against the private data bytes in the CA_descriptor.

mask: this 8*n-bit field is a mask used to indicate the individual bits to use when comparing the pattern to the private data. Only bits set (‘1’) in the mask shall be used in the compare. All bits used in the compare must match exactly, or the CA_descriptor shall not be used to determine the PID.

12.5.5.3.4 Pre-processing object

The pre-processing object enables to perform some simple processing on filtered messages before they are sent to the smart card. If preprocessing is done on two or more filtered messages the recommended placement is in the filter set.

Syntax	No. of bits	Encoding
Pre-processing_object(){		
object_tag	**8**	**uimsbf**
object_length	**8**	**uimsbf**
operation	**8**	**uimsbf**
number_of_filter_identifiers	**8**	**uimsbf**
for(i=0;i<N1;i++){		
filter_program_number	**8**	**uimsbf**
}		
for(i=0;i<N2;i++){		
operand_object()		
}		
}		

operation: this 8-bit field gives the action to be performed. It is coded as follows:

Value	Meaning
'00'	reserved for future use
'01'	concatenate operands in given received order
'02'-'7F'	reserved for future use
'80'-'FE'	user private
'FF'	reserved for future use

number_of_filter_identifiers: this 8-bit field identifies the number of bytes in the loop immediately following.

filter_program_number: this 8-bit field references messages filtered by the corresponding filter program. The messages filtered are considered to be operands.

The operand() objects can be used to introduce other operands. They are reserved for future use.

12.5.5.3.5 CW association object

This object enables the association of the result, eg. control word, of processing a CA message with the correct elementary_PID. If the object is not present, the MPEG defined association shall be used.

Syntax	No. of bits	Encoding
CW_association_object(){		
object_tag	**8**	**uimsbf**
object_length	**8**	**uimsbf**
for(i=0;i<N1;i++){		
component_tag	**8**	**uimsbf**
}		
}		

component_tag: this 8-bit field identifies the ES loop in the PMT associated with this control word (CW). This loop then points to the PID of the elementary stream using the CA_descriptor. There must be a stream_identifier_descriptor as specified in DVB-SI in every ES loop of the PMT to use this mechanism. Delivery of an association tag component_tag is a private mechanism in the ECM and allows delivery of multiple control words for one ECM message.

12.5.5.3.6 Filter state object

The filter state object describes a simple programming language to describe a state machine for filtering CA messages. Each filter program may contain a sequence of filter patterns which are executed in order. The filter in the STU shall be initialized with the state having the lowest filter_state_identifier value. After having successfully filtered the first message the programmed filter state shall be updated as soon as possible by the STU given the instructions indicated by next_state and filter_update_condition in the filter_state_object. Multiple of these filter_state_objects are allowed.

Syntax	**No. of bits**	**Encoding**
filter_state_object(){		
object_tag	**8**	**uimsbf**
object_length	**8**	**uimsbf**
reserved	**4**	**bslbf**
filter_state_identifier	**4**	**uimsbf**
filter_update_condition	**4**	**bslbf**
next_state	**4**	**bslbf**
for(i=0;i<N;i++){		
sub_object()		
}		
}		

reserved: this 4-bit field is reserved for future use. It shall be set to 'F'.

filter_state_identifier: this 4-bit field identifies the state name of the filter. This makes it possible to reference different states of the filter for update purposes.

next_state: this 4-bit field identifies the following state in the filtering program.

filter_update_condition: this 4-bit field identifies when a filter set-up should be updated. It is coded as follows:

Value	**Meaning**
'00'	reserved for future use
'01'	update after successful filtering of message to "next_state"
'02'	update after successful filtering of message to the state following the state with matching pattern
'03'	update after permission from smart card
'04'-'0F'	reserved for future use

sub_object: The filter_state_object may contain the following sub_objects:

Table id object

The table_id object describes the filter pattern and mask for the table_id field. Only one object may be present. If none, no changes is made to the previously set pattern or mask. Prior to initialization, the pattern shall be set to zero and the mask shall be set to one.

Syntax	**No. of bits**	**Encoding**
table_id_object(){		
object_tag	**8**	**uimsbf**
object_length	**8**	**uimsbf**
table_id_pattern	**8**	**uimsbf**
table_id_mask	**8**	**uimsbf**
}		

table_id_pattern: this 8-bit field identifies the pattern on which the table_id field shall be filtered.

table_id_mask: this 8-bit field which bits of the table_id_pattern shall be masked. A bit shall be used for filtering if the corresponding bit in the mask has a value of 1.

Filter pattern object

This filter_pattern object describes the filter pattern to filter on.

Syntax	No. of bits	Encoding
filter_pattern_object(){		
object_tag	**8**	**uimsbf**
object_length	**8**	**uimsbf**
offset	**8**	**uimsbf**
for(i=0;i<N;i++){	(N<8)	
CA_data_byte_pattern()		
}		
}		

offset: this 8-bit field gives the offset in bytes from the beginning of the payload data of the CA message.

CA_data_byte_pattern: this 0-7-byte field identifies the bit pattern on which the given CA_data_bytes should be filtered.

Filter mask object

The filter_mask object describes the filter mask to mask the filter pattern. If this object is not present there is no masking.

Syntax	No. of bits	Encoding
filter_mask_object(){		
object_tag	**8**	**uimsbf**
object_length	**8**	**uimsbf**
offset	**8**	**uimsbf**
for(i=0;i<N;i++){	(N<8)	
CA_data_byte_mask()		
}		
}		

offset: this 8-bit field gives the offset in bytes from the beginning of the payload data of the CA message.

CA_data_byte_mask: this 0-7-byte field identifies which bits of the CA_data_byte_pattern shall be masked. A bit in the received message shall be used for filtering if the corresponding bit in the CA_data_byte_mask has the value of 1.

12.5.6 Response objects from smart card

12.5.6.1 Introduction

This subclause defines a number of response objects from the smart card to interact with the filter program.

12.5.6.2 Update filter state

The update_filter_state object indicates if and to what state the filter which filtered the message send to the smart card shall be updated by the STU. This object can only be present when the filter should be updated after indication by the smart card. When in this case the object is absent the filter shall not be updated.

Syntax	No. of bits	Encoding
Update_filter_state_object(){		
object_tag	**8**	**uimsbf**
object_length	**8**	**uimsbf**
reserved	**3**	**uimsbf**
absolute	**1**	**uimsbf**
next_state	**4**	**uimsbf**
}		

absolute: this 1-bit field indicates it the update of the filter state is absolute (value=1) or relative (value=0).

next_state: this 4-bit field gives an absolute goto in the filter program (absolute=1) or a signed relative goto in the filter program (absolute=0). The state of the filter program when this response is returned, depends on the "filter_update_condition" specified by the filter state that issued the message.

For a relative goto the mechanism operates as follows:

- For the update after successful filtering message condition, the filter program will have automatically been updated by the STU by the time this response is returned. Therefore, a value of 'F' is used to replay the last filter state, in the case where there was a transmission error detected. A value of zero (or no object) is normally used to leave the filter state where it is.

- For the update after permission from smart cardî condition, the filter program will have remained untouched. Therefore, a value of zero (or no object) is used to replay the last filter state, in the case where there was a transmission error detected. A value of one is normally used to move the filter to the next state.

12.5.6.3 Update filter conditions

The update_filter_conditions object tells the STU that the filter conditions have changed for a specified filter. It gives the new filter state values for this filter.

Syntax	No. of bits	Encoding
Update_filter_conditions(){		
object_tag	**8**	**uimsbf**
object_length	**8**	**uimsbf**
filter_program_number	**8**	**uimsbf**
filter_state_object()		
}		

filter_program_number: this 8-bit field identifies the number of the filter program to be updated.

filter_state_object: contains the new filter_state object that applies to this filter.

12.5.7 Filter conditions

A CA message must be filtered (and passed to the smart card as indicated by the filter program) if:

{(the filterable area XOR the filter pattern) AND (filter mask)} is equal to zero.

12.5.8 Further filter requirements

The STU shall be able to descramble up to 6 independent (elementary) streams simultaneously, meaning that they may be scrambled with independent control word streams.

12.6 Initialization of the smart card and application

12.6.1 Answer-to-Reset

12.6.1.1 Introduction

The Answer-to-Reset (ATR) is information given by the smart card to the STU at the beginning of a session just after a hardware reset. The ATR contains operational data and consists of the Initial and Format characters, the Interface characters, the Historical characters, and conditionally the Check character TCK, according to ISO/IEC 7816-3. The ATR consists of at most 33 characters. The values and presence of the different characters in this specification are given below. Besides in the ATR, information, for example in the Historical characters, may also be contained in the ATR file or DIR file. These are also described below.

12.6.1.2 The Initial and Format characters

The initial character TS, and the format character T0 are both mandatory. The initial character TS defines the coding convention for all subsequent characters. The format character T0 indicates which interface characters are present and gives the number of historical characters. The coding of both characters is specified in ISO/IEC 7816-3.

12.6.1.3 The interface characters

The table below gives information on the interface characters. It describes the purpose of the different characters and whether they could be sent by the smart card. In addition, it describes whether they will be evaluated by the STU and how the STU shall react to them. The table is comparable to the one given in ETS 300 608.

Interface character	Contents	Sent by the card	a) evaluation by STU b) reaction by STU
TA1 (global)	parameters to calculate the work etu and bit rate adjustment	optional	a) always if present b) if TA1 is not '11' PTS procedure shall be used
TB1 (global)	parameters to calculate the programming voltage and current and indicates if Vpp is connected in the card	optional	a) always if present b) See item 6 subclause 12.4.1
TC1 (global)	parameters to calculate extra guard time between bytes for communication from STU to smart card	optional	a) always if present *b) use the guard time accordingly*
TD1	protocol type; indicator for the presence of interface characters, specifying rules to be used for transmission with the given protocol type	optional	a) always if present b) reject the card if the T=0 protocol is not supported by the card
TA2 (specific)	not used for protocol T=0	optional	a) optional b) ...
TB2 (global)	parameters to calculate the programming voltage	never	See item 6 subclause 12.4.1
TC2 (specific)	parameters to calculate the work waiting time	optional	a) always if present b) use the work time accordingly
TDi (i>1)	protocol type; indicator for the presence of interface characters, specifying rules to be used for transmissions with the given protocol type	optional	a) always if present b) identifying the subsequent characters accordingly
TAi, TBi, TCi (i>2)	characters which contain interface characters for other transmission protocols	*optional*	a) optional b) ...

12.6.1.4 The Historical characters/bytes

The number (0 to 15) of historical characters is indicated by the Format Character T0 as described above. The historical characters give (access to) card identification data. This information may be carried by the historical characters but may also be found in the ATR or DIR file as specified in ISO/IEC 7816-4. The historical characters consist of three fields:

- a mandatory category indicator (1 byte)
- optional COMPACT-TLV data objects as described in ISO/IEC 7816-4, clause 8.3. Also application identification and selection data objects defined in ISO/IEC 7816-5 and/or other IDOs as defined in ISO/IEC 7816-6 may appear in the historical characters. However, if the tag value of these objects is '4X' the coding of the their tag and length fields shall be modified to the COMPACT-TLV format as specified in ISO/IEC 7816-4 clause 8.3.

Remark: If these objects appear in the ATR file, they shall be encoded according to the basic encoding rules of ASN.1 and not in the COMPACT-TLV format!

- a conditional status indicator (3 bytes)

The following table gives the format of the historical bytes which is according to ISO/IEC 7816-4. The possible presence of data objects in the historical bytes, their need for evaluation by the STU and the required reaction of the STU is also indicated in the table.

Object	T/L	Contents	Presence in historical char.	a) Evaluation, b) (re)action by STU
Category indicator	-	"First historical byte"	always	a) always b) Continue in specified way if V(alue) ='00' or '80'
Optional data objects	'XX'	Various (any data allowed by ISO/IEC 7816 to be here)	optional	a) see clause 12.6.4 b) see clause 12.6.4
Status information	-	Category indicator is '00': byte 1: card life status byte 2/3: status bytes SW1-SW2 Category indicator is '80': card life status SW1 and SW2 card life status + SW1, SW2	 - optional - optional - optional - optional - optional	 a) optional b) ... a) always b) ... a) optional b) ... a)always b) ... a) always (SW1 and SW2) b) ...

12.6.1.5 Check character TCK

If only the transmission protocol T=0 is indicated, TCK shall not be sent. In all other cases, TCK shall be sent. In this case the value shall be according to ISO/IEC 7816-3.

12.6.2 Conditional PTS procedure

A PTS procedure according to clause 7 of ISO/IEC 7816-4 is allowed. It shall be supported by the STU. Depending on the desired protocol or some other ATR parameters (F, D) the card shall support PTS.

12.6.3 The ATR and/or DIR file

Besides in the historical bytes card identification and initialization data may also be located in the ATR and/or DIR file. If their presence is indicated in the historical bytes they shall be read as soon as possible, but always after reading possible initial access data.

12.6.4 Card identification and initialization data

The smart card may locate any data allowed by ISO/IEC 7816 in the historical bytes, initial access data, ATR file, and DIR file. The STU shall at least be able to evaluate and react on the data objects given in the following table. This does not mean that it should be able to support all the services which can be indicated in these objects. The (application-independent) card services which at least shall be supported by the STU are described in Subclause 12.6.6.

Object	Tag	Length	Probable location	Description	Reference
Card service data	'31' ('43')	in tag ('01')	Hist. bytes	Denotes methods available in the card for supporting card services	ISO/IEC 7816-4 and 7816-6
Initial access data	'41' '43' ('44')	in tag in tag ('01' to '02')	Hist. bytes	Allows the retrieval of a string of data objects called "initial data string". Use of this object is not recommended	ISO/IEC 7816-4 and 7816-6
Card capabilities	'71' '72' '73' '47'	in tag in tag in tag '01' to '03'	Hist. bytes ATR file	Gives card capabilities for application-independent card services in the first, second, and third software function table.	ISO/IEC 7816-4 and 7816-6
Application identifier	'FY' '4F'	Y='1' to 'F' '01' to '10' (less when in hist. bytes)	Hist. bytes DIR file ATR file	Uniquely identifies the application. The object is mandatory for the smart card to give.	ISO/IEC 7816-5 and 7816-6
Application template	'61'	'03' to '7F'	DIR file ATR file	Container for all application identification and selection objects (not all possible objects in container need to be recognised by STU).	ISO/IEC 7816-5 and 7816-6

12.6.5 Application identification and selection

12.6.5.1 Location of application identification data

In case of a single application smart card the AID or application template shall be located in one of the following locations:

- Historical characters (Not recommended if AID is long).
- ATR file ('2F00') (Accessible with file selection and read commands; presence indicated in historical characters).
- DIR file ('2F01') (Accessible with file selection and read commands; presence indicated in historical characters).

In case of a multi-application smart card the AID or application template shall be located in the DIR file ('2F01') (Accessible with file selection and read commands; presence indicated in historical characters).

12.6.5.2 Application selection

In this specification an application can be selected in at least the following two ways: implicit and direct. It is indicated in the card capabilities object in the historical bytes if the application is selected implicitly. An application can be selected directly with the Select File command, specifying the AID as DF name or with the file identifier. The use of the Select File command is depending on the smart card capabilities as indicated in the first software function table of the card capabilities object. Implicit application selection is not recommended for multi-application cards.

12.6.5.3 Format of application identification data

The AID data object is coded as follows:

An AID number for DAVIC applications will be added to this Subclause as soon as it is available.

Tag	Length	Bytes	Contents
‘4F’ (‘FY’)	‘01’ to ‘10’ Y= ‘1’ to ‘F’ (depending on available length in historical bytes)	00-04 (05-15) 05 06 07-08 09-15	(RID) DAVIC_application_id (PIX) Extended_DAVIC_application_id Application_version System_id RFU

The extended_DAVIC_application_id is coded as follows:

Value	Meaning
‘00’	reserved for future use
‘01’	Broadcast pay-TV functions
‘02’	Interactive security functions
‘03’	Broadcast pay-TV + interactive security functions
‘03’-‘7F’	reserved for future use
‘80’-‘FE’	private use
‘FF’	reserved for future use

The other application objects, besides the application template, specified in ISO/IEC 7816-5 are optional. It is not mandatory for an STU to support these. The application template object however shall be supported by the STU.

The application_version is used to code the version of the application. This specification currently specifies application_version ‘01’ for broadcast pay-TV and interactive security functions. All other values are reserved for future use. The System_id shall be equal to the CA_system_id for extended_DAVIC_application_id values ‘01’ and ‘03’.

12.6.6 Application-independent card services

This specification adopts the application-independent card services as specified in ISO/IEC 7816-4 with the following restrictions:

- Support of application selection by partial DF name as described in clause 9.3.2 of ISO/IEC 7816-4 is not required.
- Support of retrieval of data objects besides (for example indirect retrieval by means of wrapper DE) those in the historical bytes, the initial data string, the ATR file, and the DIR file is not required. See also ISO/IEC 7816-6.

12.7 Smart card security functions

12.7.1 Access conditions

Access to smart card files and data objects by commands is controlled with the access conditions. As long as access conditions are respected the desired command will be performed on the selected file or relevant data object. The different access conditions together with their 4-bit coding levels for the DAVIC application are specified below This specification is an adapted subset of the access conditions specified in ETSI prEN 726-3.

Level	Access Condition
'0'	ALW (always)
'1'	PWRD (password)
'2'	RFU
'3'	PRO (protection)
'4'	AUT (authentication)
'5'	RFU
'6'	PWRD/PRO (password + protection)
'7'	RFU
'8'	PWRD/AUT (password + authentication)
'9'-'E'	RFU
'F'	NEV (never)

The meaning of these access conditions for the application specified is as follows:

ALW: The command shall always be possible without any restriction.

PWRD: The command shall only be possible after a correct PWRD (password) verification.

PRO: The command shall only be possible in a protected way (ie. after checking of data integrity and origin, which for example applies to EMMs and ECMs).

AUT: The command shall only be possible after an internal authentication (ie. an authentication by the STU of the smart card).

NEV: The command shall never be possible.

Although the access conditions have levels they are *not* prioritized.

12.7.2 Commands and access conditions

A file or data object can have different access conditions for different groups of commands. This specification allows commands to be divided into 8 separate groups concerning their access conditions. This grouping of commands is dependent on the data container (file type, data object) the command is supposed to operate on. For each group a separate access condition can be defined.

The following group of commands operating on DF(/MF) files with the same access conditions exist: CA (conditional access). The other 7 groups of commands are RFU or private for DF(/MF) files. Commands of the group CA normally transfer CA-system specific messages from the STU to the smart card, for example ECMs and EMMs.

The following groups of commands operating on EF files with the same access conditions exist: Read, Update, Write, and CA (Conditional Access commands). The other 4 groups of commands are RFU for EF files.

The following groups of commands operating on DOs with the same access conditions exist: Get (Read), Put (Write, Update), and CA. The other 5 groups of commands are RFU or private for DOs.

The following table gives an overview of the different commands in this specification (see also Subclause 12.10). The table indicates if the command operates on a file and/or data object or not. In case the command does not directly operate on a file or data object the corresponding cell is left blank. If a cell has content the command can operate on the corresponding file or DO. The content value of the cell indicates the access condition group the command belongs to. An x means that the command does not have an access condition.

	Current File						
Command	**(MF)**	**DF**	**EF_{trans}**	**EF_{fixed}**	**EF_{var}**	**EF_{cycl}**	**DO**
Select_file	x	x	x	x	x	x	
Read_binary			Read				
Read_record				Read	Read	Read	
Seek				Read	Read	Read	
Verify_pwrd							
Change_pwrd							
Disable_pwrd							
Enable_pwrd							
Internal_authentication							
Write_binary			Write				
Write_record				Write	Write	Write	
Update_binary			Update				
Update_record				Update	Update	Update	
Get_response							
Get_data							Get
Put_data							Put
Put_data(send_message)	CA	CA	CA	CA	CA	CA	CA
Put_data(select)							Get
Change_maturity_rating			Write	Write	Write	Write	
Get_application_status							
Perform_security_operation	CA	CA	CA	CA	CA	CA	CA

12.7.3 Password management

In this specification passwords (PWRDs) are related to DFs. Each DF may have its own password which is indicated in the DF file header. Password handling shall therefore be done by the smart card in the following manner. In all cases the password of the current DF shall be used. If the current DF does not have a password the password of the parent DF shall be used and so on. In the end, if the application does not have a password, the global password (on MF level) of the smart card shall be used. This implicitly requires that the application has access to the file storing the password on MF level. If the application doesn't require a password there is no problem. The access condition PRWD just is not used then.

12.7.4 Authentication

12.7.4.1 Authentication on the S2/S3 Flows

In order to support the authentication mechanisms specified for the S2 flow, the CA1 security device must, in case of the interactive security functions application, perform all private key operations required in the X.509 authentication exchange. These consist of signing a protocol data unit, and decrypting the session keys. In general, the signature will involve hashing followed by the private key operation. Normally, the hashing will be performed on the STU.

The Perform_security_operation command (see 12.10.19) shall be used to perform private key operations on the CA1 security device. The command is defined as a proprietary command for a DAVIC specific application.

The following table summarizes the relevant secure messaging data objects (defined in ISO 7816-4), which are used to support authentication and key establishment:

Tag	Value
'80'	Plain Value (non BER-TLV coded data)
'86'	Padding indicator byte, followed by cryptogram (plain value not coded in BER-TLV)
'9A'	Digital signature input
'9E'	Digital signature
'92'	Certificate (non BER-TLV coded data)
'9C'	Public key

These objects shall be used in the correct context (Perform_security_operation command or Key file(s)) to prevent misinterpretation. The user's certificate is indicated by the tag '92'.

12.7.4.2 Digital Signatures for DSM-CC Downloads

A Certification Authority public key for download is stored on the smart card. A tag of '9C' is used to indicate a public key.

12.8 Data structures in the smart card

12.8.1 Introduction

This subclause describes the method for accessing data stored in the different files within the DAVIC pay-TV smart-card. This specification complies with ISO/IEC 7816-4.

12.8.2 File organization

12.8.2.1 Logical model

The logical organization of data in the smart card is according to ISO/IEC 7816-4, eg. a structure with one Master File (MF), several Dedicated Files (DFs) and Elementary Files (EFs).

12.8.2.2 Elementary file structures

As in ISO/IEC 7816-4, this specification defines the following elementary file (EF) structures: Transparent EF, EF with linear fixed structure, EF with linear variable structure, and Cyclic EF. Their specification is equal to that described in ISO/IEC 7816-4.

12.8.2.3 Allocation of file identifiers

Within ISO/IEC 7816 the following file identifiers are already allocated: '2F00' (DIR), '2F01' (ATR), '3F00' (MF), '3FFF' and 'FFFF'. All other file identifiers may be freely used within the application. However, double file identifiers under the same parent are not allowed as described in ISO/IEC 7816-4.

12.8.2.4 File structure

Figure 6 shows a typical file structure of the smart card in the DAVIC application for broadcast pay-TV. The actual structure including access to data objects is implementation specific. However, for the application data object referencing (Get_data, Put_data) is preferred. In the figure, applications are located in their own DF below the MF. When only one application is present on the smart card application might also be located on MF level. Figure 7 shows the typical file structure for this situation. In the figures, the shown DAVIC broadcast pay-TV application file contains the following files with the following characteristics:

- system application file(s). This file(s) contain general-purpose data objects (eg. application_id, system_id) and common data objects that can be used by all service providers (eg. password, filter_set, filter_program, keys, parental_rating). Data objects defined can be selected by the Get_data command if indicated in the file header. The detailed structure below the system level file(s) is implementation specific.

- service provider file(s). Each service provider is allocated a file (DF or EF) which contains all data objects used for managing a service (eg. service_provider_id, keys, entitlements, specific filter conditions). Data objects defined can be selected by the Get_data command if indicated in the file header. The detailed structure below the service provider file(s) is implementation specific.

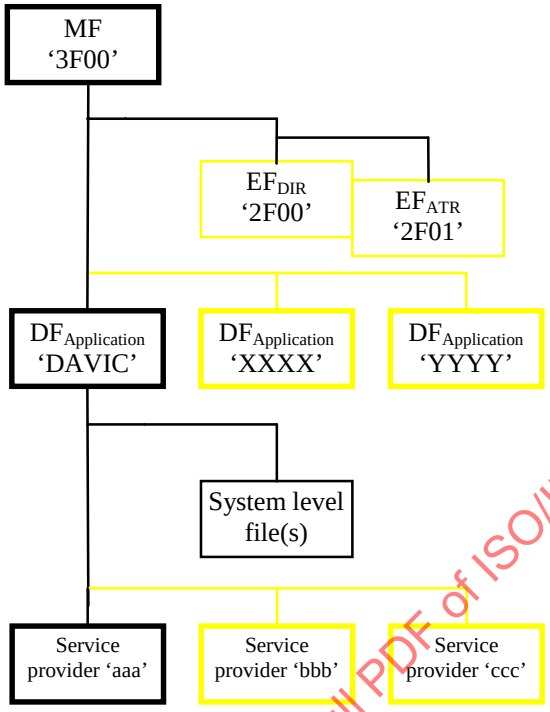

Figure 6. — Typical smart card file structure for the DAVIC CA1 pay-TV application.

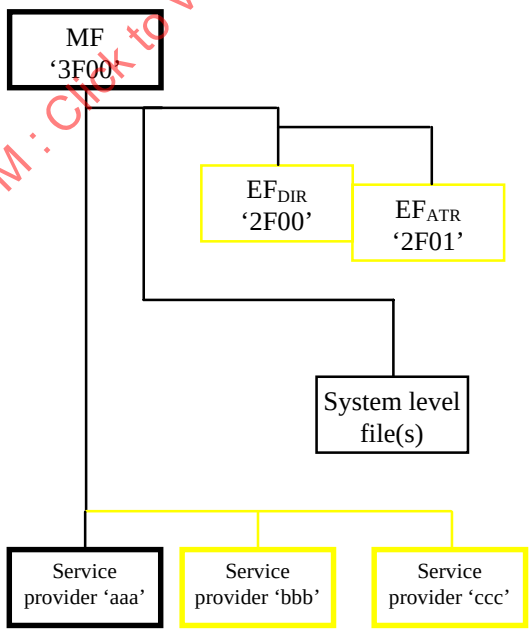

Figure 7. — Typical smart card file structure for the DAVIC CA1 pay-TV application with only one application on the card.

Figure 8 shows a typical file structure of the smart card in the case of the DAVIC application for interactive security services. The key file(s) contain(s) the public key and the certificate. The data objects can be accessed as indicated by the file header.

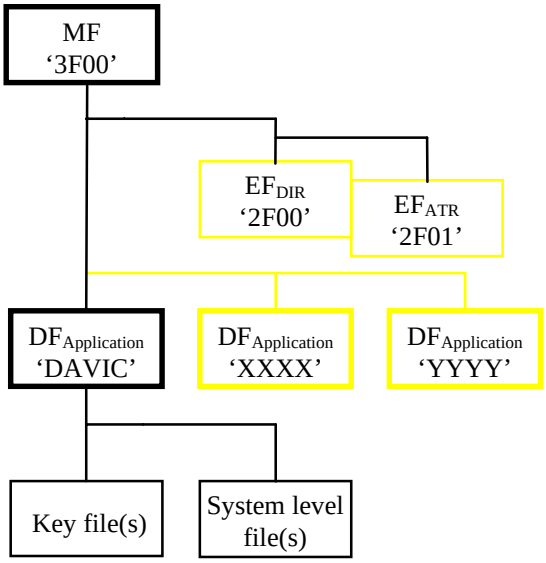

Figure 8. — Typical smart card file structure for the DAVIC interactive application.

Identical files or objects which are lower in the file structure overrule the same files and objects which are higher in the structure if selected.

12.8.3 Data structure headers

12.8.3.1 Introduction

Data structure headers give information on the data structures like MF, DF, and EF. As an example, this information may include the identifier of the structure and access conditions (AC) to the structure. The following paragraphs describe the exact specification of the different structure headers.

12.8.3.2 Header information

All data structure header information is contained in the FCI (file control information) template (Tag=‘6F’). This FCI template may contain all objects defined in clause 5.1.5 of ISO/IEC 7816-4 and other objects allowed by ISO 7816. However, it is not mandatory for the STU to support all these objects. The following table gives the ISO 7816 specified objects the STU shall support

Tag	Length	Value	Support by STU
‘79’	var	|‘4F’|L|AID|: DAVIC tag allocation scheme	Mandatory
‘82’	1 to 3/4	File descriptor	Mandatory
‘83’	2	File identifier	Mandatory
‘84’	1 to 16	DF name	Mandatory
‘85’	var	Application proprietary information	Mandatory
‘XX’	Y	Other ISO objects for file header	Optional

Paragraph 12.8.3.3 specifies the contents of the application proprietary information (tag=‘85’).

The application root (MF or DF) shall at least contain the (DAVIC) tag allocation scheme (tag=‘79’), the file descriptor (tag=‘82’), the file identifier (tag=‘83’) or DF name (tag=‘84’), and the application proprietary information (tag=‘85’).

All the other headers in the (DAVIC) application shall at least contain the application proprietary information (tag='85').

12.8.3.3 Application proprietary information

12.8.3.3.1 General

This subclause describes the contents of the application proprietary information (tag='85') within the FCI template. The application proprietary information consists of two parts: a common part and a specific part which depends on the structure type.

The common part of the application proprietary information is specified as follows:

Byte	Description	Length
1	Tag = '85'	1 byte
2	Length	1 byte
3 - 4	File_identifier	2 bytes
5	Type of file (1)	1 byte

(1) The coding of this byte is equal to the contents of the file descriptor byte as defined in clause 5.1.5 of ISO/IEC 7816-4.

The following paragraphs specify the file specific part for the different file types.

12.8.3.3.2 (MF/)DF

In the case of a DF in the application or as the root of the application, which can also be the MF, the specific part of the application proprietary information is coded as follows:

Byte	Description	Length
6 - 7	RFU	2 bytes
8 - 11	Access conditions (3)	4 bytes
12	File status (4)	1 byte
13	Number of DFs under the selected directory	1 byte
14	Number of EFs under the selected directory	1 byte
15 - 16	Directory file identifier ('FFFF' means no directory file)	2 bytes
17	DF access (5)	1 byte
18	PWRD characteristics (6)	1 byte
19	PWRD status (7)	1 byte

(3) Access conditions (byte 8 - 11) (see also 12.7.1):

Byte 8: b8-b5: RFU
Byte 8: b4-b1: RFU
Byte 9: b8-b5: RFU
Byte 9: b4-b1: RFU
Byte 10: b8-b5: RFU
Byte 10: b4-b1: RFU
Byte 11: b8-b5: CA
Byte 11: b4-b1: RFU

(4) File status (byte 12):

b8-b3: RFU
b2: '0': Readable when invalidated
'1': Not readable when invalidated
b1: '0': Invalidated
'1': Not invalidated

(5) DF access (byte 17):

b8-b3:	RFU	
b2:	'0':	No Get_data/Put_data
	'1':	Get_data/Put_data
b1:	'0':	No Read/Write
	'1':	Read/Write

(6) PWRD characteristics (byte 18):

b8-b7:	'00':	No PWRD
	'01':	PWRD
	'10':	RFU
	'11':	RFU
b6-b1:	PWRD length	

(7) PWRD status (byte 19)

b8:	'0'	Not activated
	'1'	Activated
b7-b1:	Number of allowed PWRD retrials	
	'0000000' = blocked	
	'1111111' = no limit	

The header can be obtained by a Get_response command direct after a Select_file command.

12.8.3.3.3 EF

In the case of an EF (data structure type '03') the specific part of the application proprietary information is coded as follows:

Byte	Description	Length
6 - 7	File size (2)	2 bytes
8 - 11	Access conditions (3)	4 bytes
12	File status (4)	1 byte
13	Record length (if fixed structure)	1 byte

(2) File size (byte 6 - 7):

- Transparent EF: Number of bytes in body part.
- Variable EF: Number of bytes in body part.
- Fixed/cyclic EF: Number of records in EF.

(3) Access conditions (byte 8 - 11) (see also 12.7.1):

Byte 8: b8-b5:	Read
Byte 8: b4-b1:	Update
Byte 9: b8-b5:	Write
Byte 9: b4-b1:	RFU
Byte 10: b8-b5:	RFU
Byte 10: b4-b1:	RFU
Byte 11: b8-b5:	CA
Byte 11: b4-b1:	RFU

(4) File status (byte 12):

b8-b3:	RFU	
b2:	'0':	Readable when invalidated
	'1':	Not readable when invalidated
b0:	'0':	Invalidated
	'1':	Not invalidated

The header can be obtained by a Get_response command direct after a Select_file command.

12.8.4 EF file content

12.8.4.1 MF level EFs

According to ISO 7816 parts 4 to 6, the following files may be present for application identification in the card.

12.8.4.1.1 EF_{DIR}

The directory EF (EF_{DIR}) contains information on the application(s) on the smart card and may contain other (private) information. It is defined within ISO/IEC 7816-4 and ISO 7816-6 and has the following characteristics:

Parameter	**Value**		
File ID	'2F00'		
Structure	-		
Presence	Optional: Mandatory if AID information not in ATR or ATR file		
Access conditions (recommended)			
Length	Private		
	Object	M/O	#DO
TLV objects contained	AID	M	n

12.8.4.1.2 EF_{ATR}

The answer to reset EF (EF_{ATR}) contains information on the application(s) in the smart card and may contain other (private) information. It is defined in ISO/IEC 7816-3 and ISO 7816-4 and has the following characteristics:

Parameter	**Value**		
File ID	'2F01'		
Structure	-		
Presence	Optional: Mandatory if AID information not in ATR of DIR file		
Access conditions (recommended)			
Length	Private		
	Object	M/O	#DO
TLV objects contained	AID	M	n

12.8.4.2 $EF_{directory}$

When present below a DF, this EF may be used to reference data objects available under the DF. This optional file enables easy look-up of the different objects without fixing the file structure. If this file is not present files can be selected using the Select_file command with the selection options select first or only occurrence and next occurrence as specified in Subclause 12.10.3.

For (indirect) referencing data objects the wrapper DO (tag='63') is used as defined in ISO/IEC 7816-6 clause 5.6. Only the tags '5C' (taglist), '51' (path to an EF), and '52' (command to perform) are mandatory for the STU to support. The following figure is an example of a wrapper referencing a number of objects with tags Tag1, Tag2, and Tag3 on a file location indicated by Path:

63	L	5C	L	Tag1, Tag2, Tag3, ...	51	L	Path

Figure 9. — Wrapper object example.

With the Path the desired file can be selected with the Select_file command. The $EF_{dirrectory}$ has the following characteristics:

Parameter	Value		
File ID	Referenced from DF file header		
Structure	-		
Presence	Optional		
Access conditions (recommended)			
Length	Private		
	Object	M/O	#DO
TLV objects contained	Wrapper for Application_related_data	O	1
	Wrapper for System_related_data	O	1
	Wrapper for Parental_rating	O	1
	Wrapper(s) for Service_provider_related_data	O	n
	Wrapper for Filter_set	O	1
	Wrapper(s) Filter_prog	O	n
	Wrapper for Entitlement_template	O	n
	Wrapper for Interactive_security_related_data	O	n
	Wrapper for other data objects	O	n

If a template is referenced with a wrapper, for example the service_provider_related_data template, automatically all data (objects) which may be contained in that template are referenced. This prevents long tag sequences within the wrapper. However, it could be advantageous to indicate each tag contained in the template separately to directly indicate which data are present on the card and if overruling may occur.

It shall not be directly assumed that only one object with an indicated tag is present in the referenced EF. Multiple objects with the same tag could occur in the same EF. The wrappers may contain tags of templates and tags of data objects.

12.8.5 Overview of DAVIC data objects

This subclause gives the list of data objects (for the pay-TV application) that can be found in a DAVIC application DF.

System related data is considered to be on a higher level than service provider related data. Service provider related data shall overrule the same data on system level if content from that specific service provider is selected by the user. If overruling can occur service provider related data shall be allocated within (separate) child DFs in the file structure of the smart card to express the hierarchy compared to system data. Other data objects referenced from the data directory EF are considered to be on system level unless they are within an EF of DF for service provider data. Then they are said to be on service provider level. This can be determined by examining the path contained in the wrapper or by analyzing the response to the Get_data command.

12.8.5.1 System related data

The following table gives the list of all data objects that can be found at the system level:

Object	M/O	#DO
Application_template	O	1
Application_related_data_template	O	1
Application_id (AID)	O	1
Application_label	O	1
Application_expiration_date	O	1
Application_effective_date	O	1
System_related_data_template	O	n
System_id (=CA_system_id for broadcast pay-TV)	M	1
CA_descriptor_selection	O	n
System_label	O	1
System_provider_label	O	1
Preferred_service_provider	O	1
Filter_set	M	1
Filter_program	O	n
Parental_rating	O	1
Entitlement_template	O	n

12.8.5.2 Service provider related data

The following table gives the list of all data objects that can be found at the service provider level:

Object	M/O	#DO
Service_provider_related_data_template	O	n
Service_provider_id	M	1
CA_descriptor_selection	O	n
Service_provider_label	O	1
Base_channel	O	1
Filter_set	O	1
Filter_program	O	n
Entitlement_template	O	n

12.9 Basic data objects

12.9.1 Introduction

This subclause describes the various data objects which may be stored in the smart card and response objects. To present them in a logical way they are grouped according to their context. This subclause only describes the data objects for basic operation. Data objects for special functions are described in separate sections together with the description of the function.

12.9.2 Application_related_data

Application_template: The application_template may be used as a container for other application objects or as a reference to application objects.

Tag	Length	Byte	Contents
see 12.12	var	n	Application objects

Application_related_data_template: The application_related_data_template may be used as a container for other application related objects or as a reference to application related objects.

Tag	Length	Byte	Contents
see 12.12	var	n	Application related objects

Application_id: The application_id identifies the type of DAVIC application and the version of that application. It is equal to byte 5 and 6 of the PIX in the AID (see Subclause 12.6.5.3).

Tag	Length	Byte	Contents
see 12.12	02	01	Application_id
		02	Application_version

Application_label: The application_label gives the name of the CA1 interface application.

Tag	Length	Byte	Contents
see 12.12	var	n	text according to ETS 300 468

Application_expiration_date: The application_expiration_date gives the date the application will stop operating.

Tag	Length	Byte	Contents
see 12.12	02	01-02	Date coded as Modified Julian Date (MJD) according to ETS 300 468

Application_effective_date: The application_effective_date gives the date the application will start operating.

Tag	Length	Byte	Contents
see 12.12	02	01-02	Date coded as Modified Julian Date (MJD) according to ETS 300 468

12.9.3 System_related_data

System_related_data_template: The system_related_data_template may be used as a container for other system objects or as a reference to system related objects.

Tag	Length	Byte	Contents
see 12.12	var	n	System related objects

System_id: The system_id identifies the system which uses the application. For broadcast pay-TV this number is equal to the CA_system_id defined in ISO/IEC 13818-1.

Tag	Length	Byte	Contents
see 12.12	n	01-02	id number according to ETR162 (=CA_system_id)
		03-0n	private data

CA_descriptor_selection: The CA_descriptor_selection enables more detailed selection of the correct CA_descriptor then the CA_system_id alone. The CA_descriptor_selection object is described in more detail in Subclause 12.5.5.3.3.

Tag	Length	Byte	Contents
see 12.12	2n+1	01	offset
		02-n+1	pattern
		n+2-2n+1	mask

System_label: The system_label gives the name of the system.

Tag	Length	Byte	Contents
see 12.12	var	n	text according to ETS 300 468

System_provider_label: The system_provider_label gives the name of the (CA) system provider.

Tag	Length	Byte	Contents
see 12.12	var	n	text according to ETS 300 468

Preferred_service_provider: The preferred_service_provider may be used to indicate the service provider to tune to when using the CA system.

Tag	Length	Byte	Contents
see 12.12	01 or 02	01(-02)	service_provider_id

12.9.4 Service_provider_related_data

service_provider_related_data_template: The service_provider_data_template may be used as a container for other service provider objects or as a reference to service provider objects.

Tag	Length	Byte	Contents
see 12.12	var	n	Service provider objects

service_provider_id: The service_provider_id identifies the service provider operating the system.

Tag	Length	Byte	Contents
see 12.12	01 or 02	01(-02)	service_provider_id

service_provider_label: The service_provider_label gives the name of the service provider operating the system.

Tag	Length	Byte	Contents
see 12.12	var	-	text according to ETS 300 468

base_channel: The base_channel identifies the channel(s) the service provider transmits its programs.

Tag	Length	Byte	Contents
see 12.12	06	01-02	network_id
		03-04	original_network_id
		05-06	transport_stream_id

CA_descriptor_selection: The CA_descriptor_selection enables more detailed selection of the correct CA_descriptor then the CA_system_id alone. It is described in more detail in Subclause 12.5.5.3.3.

Tag	Length	Byte	Contents
see 12.12	2n+1	01	offset
		02-n+1	pattern
		n+2-2n+1	mask

12.9.5 Filtering

filter_set: The filter_set groups filter programs for a certain type of messages.

Tag	Length	Byte	Contents
see 12.12	var	n	filter set as defined in Subclause 12.5.4.2

filter_program: The filter_program describes the programming of a filter.

Tag	Length	Byte	Contents
see 12.12	var	n	filter program as defined in Subclause 12.5.4.3

12.9.6 Parental_rating

parental_rating: The parental_rating object contains the parental rating value. It is encoded according to the contents of the DVB-SI parental rating descriptor (ETS 300 468).

Tag	Length	Byte	Contents
see 12.12	4	01-03	country_code
		04	rating

12.9.7 Entitlement data

entitlement_template: The entitlement_template shall be used to package all entitlement data. Besides proprietary objects it may contain the other objects specified in this paragraph.

Tag	Length	Byte	Contents
see 12.12	var	n	entitlement objects

entitlement_type: The entitlement_type contains information on the type of entitlement.

Tag	Length	Byte	Contents
see 12.12	01	01	entitlement_type

The field entitlement_type is coded as follows:

Value	Meaning
'00'	reserved for future use
'01'	subscription
'02'	PPV
'03'	IPPV
'04'-'7F'	reserved for future use
'80'-'FE'	private use
'FF'	reserved for future use

service_provider_id: The service_provider_id identifies the service provider operating the system.

Tag	Length	Byte	Contents
see 12.12	01 or 02	01(-02)	servide_provider_id

entitlement_label: The entitlement_label gives a description of the entitlement.

Tag	Length	Byte	Contents
see 12.12	var	n	text according to ETS 300 468

entitlement_dates: The entitlement_dates object identifies the beginning and finishing dates of the entitlement. These dates are MJD coded according to ETS 300 468.

Tag	Length	Byte	Contents
see 12.12	04	01-02	Begin-date (MJD coded)
		03-04	End-date (MJD coded)

entitlement_date/duration: The entitlement_date/duration object identifies the start time and duration of an entitlement.

Tag	Length	Byte	Contents
see 12.12	08	01-02 03-05 06-08	Begin-date (MJD coded) Start time Duration

To enable close co-operation with SI information in the stream the following object can also be used.

SI-link: The SI-link object enables to create a backward link from the entitlement data in the smart card to the SI data in the MPEG stream. It contains the following data:

Tag	Length	Bytes	Contents
see 12.12	08 or 10	01-02	network_id
		03-04	original_network_id
		05-06	transport_stream_id
		07-08	service_id
		09-10	event_id (optional)

12.9.8 Response messages

This subclause describes response messages from the smart card which may occur after a challenge.

Even_control_word: This object gives an even control word for descrambling.

Tag	Length	Byte	Contents
see 12.12	var	n	Even control word

Odd_control_word: This object gives an odd control word for descrambling.

Tag	Length	Byte	Contents
see 12.12	var	n	Odd control word

Even/odd_control_word: This object gives an even and an odd control word for descrambling.

Tag	Length	Byte	Contents
see 12.12	2n	01-n	Even control word
		n+1-2n	Odd control word

Status: The status object gives status information from the smart card to the STU. It allows passing of multiple status words at the same time due to its variable length. When the status object contains multiple status words the first word has the highest priority, the last word the lowest.

Tag	Length	Byte	Contents
see 12.12	2n	2n-1	status
		2n	additional information

The status and additional information are coded as follows.

Status		Additional information	
Value	**Meaning**	**Value**	**Meaning**
'00'	reserved	-	-
'01'	soft reset (of smart card due to information change)	'00'	no additional information
		'01'-'7F'	RFU
		'80'-'FE'	user private
		'FF'	RFU
'02'	control word descrambled	'00'	no additional information
		'01'	entitlement available
		'02'	free preview
		'03'-'7F'	RFU
		'80'-'FE'	user private
		'FF'	RFU
'03'	control word not descrambled	'00'	no additional information
		'01'	no entitlement
		'02'	geographical black-out
		'03'	maturity rating
		'04'-'7F'	RFU
		'80'-'FE'	user private
		'FF'	RFU
'04'	reserved for purchase	'00'	no additional information
		'01'-'7F'	RFU
		'80'-'FE'	user private
		'FF'	RFU
'05'-'7F'	reserved		
'80'-'FE'	user private		
'FF'	reserved		

Status_template: In the case additional parameters together with status words should be passed to the STU the status template can be used to serve as a container for the status words and parameters.

Tag	Length	Byte	Contents
see 12.12	var	n	status words and additional parameters

Update_filter_state: See Subclause 12.5.6.2

Update_filter_conditions: See Subclause 12.5.6.3.

Access_conditions_object: With the access_conditions_object the access conditions to data can be returned to the STU as response to a command.

Tag	Length	Byte	Contents
see 12.12	01 or 02	01	high nibble: access conditions granted (as described in paragraph 12.7.1) low nibble: access conditions (as described in paragraph 12.7.1)
		02	If the access condition is PWRD or PWRD/AUT, then the remaining PWRD trials are in this byte. The msb is RFU. The encoding of bit 7 to 1 is equal to that of byte 19 described in 12.8.3.3.2.

12.10 Smart card commands

12.10.1 Introduction

This subclause specifies the functions and their command coding that the smart card may use for this specification. Application specific commands are only valid within the application, eg. the application selected. Most commands are defined as a subset of ISO/IEC 7816-4 and ETSI prEN 726-3. Some commands are proprietary to this specification and are mainly related to the interactive digital TV environment.

12.10.2 Coding of the commands

The following commands are specified in this specification:

Command	CLA	INS	P1	P2
Select_file	see text	'A4'	control	type
Read_binary	see text	'B0'	offset	offset
Read_record	see text	'B2'	rec. no	mode
Seek	see text	'A2'	'00'/offset	type/mode
Verify_password	see text	'20'	'00'	'00'
Change_password	see text	'24'	'00'	'00'
Disable_password	see text	'26'	'00'	'00'
Enable_password	see text	'28'	'00'	'00'
Internal authentication	see text	'88'	'00'	'00'
Write_binary	see text	'D0'	offset	offset
Write_record	see text	'D2'	rec. no	mode
Update_binary	see text	'D6'	offset	offset
Update_record	see text	'DC'	rec. no	mode
Get_response	see text	'C0'	'00'	'00'
Get_data	see text	'CA'	tag ref./ app. data	tag ref./ app. data
Put_data	see text	'DA'	tag ref./ app. data	tag ref./ app. data
Put_data (send_message)	see text	'DA'	'01'	'01'
Put_data (select_object)	see text	'DA'	'01'	'02'
Change_maturity_rating	see text	'F4'	'00'	'00'
Get_application_status	see text	'F8'	'00'	'00'
Perform_security_operation	see text	'8A'	tag ref.	tag ref.

For the application the CLA-bytes '8X' shall be used, where the last nibble is encoded according to Table 9 of ISO/IEC 7816-4 and bit 4 and 3 are equal to 0. For application independent commands according to ISO CLA-bytes '0X' may also be used, where the last nibble is encoded according to Table 9 of ISO/IEC 7816-4 and bit 4 and 3 are equal to 0.

12.10.3 Select_file

12.10.3.1 Functional description

The Select_file function enables the selection of a file (MF, DF or EF) according the methods as specified in ISO/IEC 7816-4. After a selection, the selected file becomes the Current File. Subsequent functions on files will be implicitly on the Current File.

12.10.3.2 Access conditions

No access conditions are defined for this function.

12.10.3.3 Command message

The command message for the Select_file function is as follows:

CLA	See 12.10.2
INS	'A4'
P1	Selection control: '00': Selection by file identifier '01': Select DF under current directory '02': Select EF under current directory '03': Select parent DF '04': Direct selection by DF name
P2	Selection options: '00': First or only occurrence (no identical file IDs allowed in same DF/MF in file structure) '02': Next occurrence
Lc field	Length of the subsequent data field
Data field	If present, according to P1: - file identifier - DF name/ Application identifier
Le field	Empty

12.10.3.4 Response message

The response to the Select_file command is as follows:

Data field	Empty
SW1-SW2	Status bytes

The smart card shall give the file header of the file selected if the Get_response command is given just after the Select_file command.

Status conditions are described in Subclause 12.10.20.

12.10.4 Read_binary

12.10.4.1 Functional description

The Read_binary function enables the retrieval of (part of) the content of a transparent EF.

12.10.4.2 Access conditions

The function will be performed on the Current (EF) File. The command will only be processed if the AC for Read functions of the Current File is satisfied.

12.10.4.3 Command message

The command message for the Read_binary function is as follows:

CLA	See 12.10.2
INS	‘B0’
P1	Selection control: b8=0
P2	P1‖P2 is the offset of the first byte to be read from the beginning of the file.
Lc field	Empty
Data field	Empty
Le field	Number of bytes to be read

12.10.4.4 Response message

The response to the Read_binary command is as follows:

Data field	Data read (Le bytes)
SW1-SW2	Status bytes

Status conditions are described in Subclause 12.10.20.

12.10.5 Read_record

12.10.5.1 Functional description

The Read_record function enables reading records from record type EFs. The mode of the command and the value of the record pointer (RP) determine the record read from the Current (EF) File. Concerning the different modes and record pointer management this specification is (almost) equal to ETSI prEN 726-3.

12.10.5.2 Access conditions

The function will be performed on the Current (EF) File. The command will only be processed if the AC for the Read functions of the Current File is satisfied. AC=PRO is not applicable for this function.

12.10.5.3 Command message

The command message for the Read_record function is as follows:

CLA	See 12.10.2
INS	‘B2’
P1	Record. no. (P1 only has significance if P2=‘04’ or ‘05’. In all other cases it shall be set to ‘00’ by the STU.)
P2	Mode: - ‘00’ = first record; RP set to first record before reading - ‘01’ = last record; RP set to last record before reading - ‘02’ = next record; RP increased before reading - ‘03’ = previous record; RP decreased before reading - ‘04’ = absolute mode/current mode, the record number is given in P1 with P1 = ‘00’ denoting the current record. The RP is not affected. - ‘05’ = read all records from P1 up to the last
Lc field	Empty
Data field	Empty
Le field	Number of bytes to read. If Le =‘00’ the whole record shall be read (P2=‘00’ ‘01’ ‘02’ ‘03’ ‘04’) or all records from P1 until the end of the file shall be read (P2=‘05’).

In the ‘next’ (P2=‘02’) and ‘previous’ (P2=‘03’) mode the RP shall not be increased or decreased if it is set on the last or first record, respectively, and the Read_record command shall not be performed. This shall however not be true in the case of a cyclic file where virtually no first and last records exist and the pointer just shall be increased or decreased to the ‘next’ record before it is read.

12.10.5.4 Response message

The response to the read_record command is as follows:

Data field	Lr (may be equal to Le) bytes
SW1-SW2	Status bytes

Status conditions are described in Subclause 12.10.20.

12.10.6 Seek

12.10.6.1 Functional description

The Seek function locates a record with a certain characteristic within the Current File (record type EF). The command can only be used for EFs containing records.

12.10.6.2 Access conditions

The function will be performed on the Current File. The command will only be processed if the AC for Read functions is satisfied.

12.10.6.3 Command message

The command message for the Seek function is as follows:

CLA	See 12.10.2
INS	‘A2’
P1	Offset (in bytes) within the record. ‘00’ means no offset.
P2	Type/mode: xxxx0000 seek from beginning forward xxxx0001 seek from end backward xxxx0010 seek from next location forward xxxx0011 seek from previous location backward 0000xxxx type 1, no response data 0001xxxx type 2, record number as response data
Lc field	Empty
Data field	Pattern to compare with
Le field	Empty or 1 byte in case of type 2

12.10.6.4 Response message

The response to the Seek command is as follows:

Data field	Empty for type 1 seek 1 byte for type 2 seek
SW1-SW2	Status bytes

Status conditions are described in Subclause 12.10.20.

12.10.7 Verify_password

12.10.7.1 Functional description

The verify_password function, used to verify a password stored in the smart card, may be used with any DF or EF within the application selected. The command is equal to the verify command defined in ISO/IEC 7816-4 with P2=0x00. The smart card should unambiguously know where the find to password stored. In this way it is possible to have a global card password or an application specific password. The nearest (parent) DF header gives the length of the desired password.

12.10.7.2 Access conditions

For this function no access conditions are defined. However, other access conditions may change as a result of the comparison of the password. The number of acceptable unsuccessful comparisons may be limited.

12.10.7.3 Command message

The command message for the Verify_password function is as follows:

CLA	See 12.10.2
INS	'20'
P1	'00'
P2	'00'
Lc field	Length of the subsequent data field
Data field	Password (length of the password is given in the nearest (parent) DF header)
Le field	Empty

12.10.7.4 Response message (nominal case)

The response to the Verify_password command is as follows:

Data field	Empty
SW1-SW2	Status bytes

Status conditions are described in Subclause 12.10.20.

12.10.8 Change_password

12.10.8.1 Functional description

The Change_password function enables the change of the password in the smart card. It may be used with any DF or EF (within the application) selected. The smart card should unambiguously know where to find the password stored. In this way it is possible to have a global card password or an application specific password.

12.10.8.2 Access conditions

For this function no access conditions are defined.

12.10.8.3 Command message

The command message for the Change_password function is as follows:

CLA	See 12.10.2
INS	'24'
P1	'00'
P2	'00'
Lc field	Length of the subsequent data field
Data field	old_password new_password (length of the password is given in the nearest (parent) DF header)
Le field	Empty

12.10.8.4 Response message

The response to the Change_password command is as follows:

Data field	Empty
SW1-SW2	Status bytes

Status conditions are described in Subclause 12.10.20.

12.10.9 Disable_password

12.10.9.1 Functional description

The Disable_password function, disables the necessity for password verification. Effectively, this results in the access condition Password no longer being present. The smart card should unambiguously know where the find to password stored. In this way it is possible to have a global card password or an application specific password.

12.10.9.2 Access conditions

No access conditions are defined for this function. The command is only possible if Disable and Enable are allowed for the EF which stores the password. The disabling of the password remains effective until it is enabled again by the enable command.

12.10.9.3 Command message

The command message for the Disable_password function is as follows:

CLA	See 12.10.2
INS	'28'
P1	'00'
P2	'00'
Lc field	Length of the subsequent data field
Data field	password (length of the password is given in the nearest (parent) DF header)
Le field	Empty

12.10.9.4 Response message

The response to the Disable_password command is as follows:

Data field	Empty
SW1-SW2	Status bytes

Status conditions are described in Subclause 12.10.20.

12.10.10 Enable_password

12.10.10.1 Functional description

The Enable_password function, enables the necessity for password verification. Effectively, this results in the access condition Password being present. The smart card should unambiguously know where the find to password stored. In this way it is possible to have a global card password or an application specific password.

12.10.10.2 Access conditions

No access conditions are defined for this function. The command is only possible if Disable and Enable are allowed for the EF which stores the password. The command shall only be performed if password verification is disabled and the password is not blocked.

12.10.10.3 Command message

The command message for the Enable_password function is as follows:

CLA	See 12.10.2
INS	'28'
P1	'00'
P2	'00'
Lc field	Length of the subsequent data field
Data field	password
Le field	Empty

12.10.10.4 Response message (nominal case)

The response to the Enable_password command is as follows:

Data field	Empty
SW1-SW2	Status bytes

Status conditions are described in Subclause 12.10.20.

12.10.11 Write_binary

12.10.11.1 Functional description

The Write_binary function enables the writing of content in to a transparent EF.

12.10.11.2 Access conditions

The function will be performed on the Current (EF) File. The command will only be processed if the AC for Write functions of the Current File are satisfied.

12.10.11.3 Command message

The command message for the Write_binary function is as follows:

CLA	See 12.10.2
INS	‘D0’
P1	Selection control: b8=0
P2	P1||P2 is the offset of the first byte to be read from the beginning of the file.
Lc field	Length of subsequent data field
Data field	Data to be written
Le field	Empty

12.10.11.4 Response message

The response to the Write_binary command is as follows:

Data field	Empty
SW1-SW2	Status bytes

Status conditions are described in Subclause 12.10.20.

12.10.12 Write_record

12.10.12.1 Functional description

The Write_record command allows writing of records in to record type EFs.

12.10.12.2 Access conditions

The function will be performed on the Current (EF) File. The command will only be processed if the AC for Write functions of the Current File are satisfied.

12.10.12.3 Command message

The command message for the Write_record function is as follows:

CLA	See 12.10.2
INS	‘D2’
P1	Record no. (P1 only has significance if P2=‘04’. In all other cases it shall be set to ‘00’ by the STU.)
P2	Mode: - ‘00’ = first record; RP set to first record before reading - ‘01’ = last record; RP set to last record before reading - ‘02’ = next record; RP increased before reading - ‘03’ = previous record; RP decreased before reading - ‘04’ = absolute mode/current mode, the record number is given in P1 with P1 = ‘00’ denoting the current record. The RP is not affected.
Lc field	Length of the subsequent data field
Data field	Record to be written
Le field	Empty

In the ‘next’ (P2=‘02’) and ‘previous’ (P2=‘03’) mode the RP shall not be increased or decreased if it is set on the last or first record, respectively, and the Write_record command shall not be performed. This shall however not be true in the case of a cyclic file where virtually no first and last records exist and the pointer just shall be increased or decreased to the ‘next’ record before it is written.

12.10.12.4 Response message

The response to the Write_record command is as follows:

Data field	Empty
SW1-SW2	Status bytes

Status conditions are described in Subclause 12.10.20.

12.10.13 Update_binary

12.10.13.1 Functional description

The Update_binary function enables the updating (overwriting) of binary data in a transparent EF.

12.10.13.2 Access conditions

The function will be performed on the Current (EF) File. The command will only be processed if the AC for Write functions of the Current File are satisfied.

12.10.13.3 Command message

The command message for the Update_binary function is as follows:

CLA	See 12.10.2
INS	'D6'
P1	Selection control: b8=0
P2	P1‖P2 is the offset of the first byte to be read from the beginning of the file.
Lc field	Length of subsequent data field
Data field	Data to be written
Le field	Empty

12.10.13.4 Response message

The response to the Update_binary command is as follows:

Data field	Empty
SW1-SW2	Status bytes

Status conditions are described in Subclause 12.10.20.

12.10.14 Update_record

12.10.14.1 Functional description

The Update_record function allows updating (overwriting) of records in record type EFs.

12.10.14.2 Access conditions

The function will be performed on the Current (EF) File. The command will only be processed if the AC for Write functions of the Current File are satisfied. AC=PRO is not applicable for this function.

12.10.14.3 Command message

The command message for the Update_record function is as follows:

CLA	See 12.10.2
INS	'DC'
P1	Record. no. (P1 only has significance if P2='04'. In all other cases it shall be set to '00' by the STU.)
P2	Mode: - '00' = first record; RP set to first record before reading - '01' = last record; RP set to last record before reading - '02' = next record; RP increased before reading - '03' = previous record; RP decreased before reading - '04' = absolute mode/current mode, the record number is given in P1 with P1 = '00' denoting the current record. The RP is not affected.
Lc field	Length of subsequent data field
Data field	Record to be updated
Le field	Empty

In the 'next' (P2='02') and 'previous' (P2='03') mode the RP shall not be increased or decreased if it is set on the last or first record, respectively, and the Update_record command shall not be performed. This shall however not be true in the case of a cyclic file where virtually no first and last records exist and the pointer just shall be increased or decreased to the 'next' record before it is updated.

12.10.14.4 Response message

The response to the Update_record command is as follows:

Data field	Empty
SW1-SW2	Status bytes

Status conditions are described in Subclause 12.10.20.

12.10.15 Get_response

12.10.15.1 Functional description

The Get_response function is used to get APDU(s) which otherwise could not be obtained.

12.10.15.2 Access conditions

No access conditions are defined for this function.

12.10.15.3 Command message

The command message for the Get_response function is as follows:

CLA	See 12.10.2
INS	'C0'
P1	'00'
P2	'00'
Lc field	Empty
Data field	Empty
Le field	Maximum length of data expected in response

12.10.15.4 Response message

The normal response to the Get_response command is as follows:

Data field	(Part of) APDU with length Le
SW1-SW2	Status bytes

Where the Get_response command is issued immediately after a Select_file the response will be the file header.

Where the Get_response command is issued immediately after a Put_data(send_message) command the response can be very diverse and will vary from control words to MMI dialogue objects.

Where the Get_response command is issued immediately after a Get_status command the response will reflect the status of the smart card.

Status conditions are described in Subclause 12.10.20.

12.10.16 Get_data

12.10.16.1 Functional description

The Get_data function is used for the retrieval of one primitive data object or the retrieved of one or more data objects contained in a constructed data object. In case multiple objects with the same tag exist in the same environment the next object with the same tag will be read during the following get_data referencing the same tag. Using the Put_data command the first object can be read again.

12.10.16.2 Access conditions

This functions operates on an object. The function will only be processed if the AC for Get functions is satisfied.

12.10.16.3 Command message

The command message for the Get_data function is as follows:

CLA	See 12.10.2
INS	'CA'
P1-P2	'0000' to '003F': RFU '0040' to '00FF': BER-TLV tag (1 byte) in P2 '0100' to '017F': RFU for application data '0180' to '01FF': Private for application data '0200' to '02FF': Simple-TLV tag in P2 '0300' to '3FFF': RFU '4000' to 'FFFF': BER-TLV tag (2 bytes) in P1-P2
Lc field	Empty
Data field	Empty
Le field	Number of bytes expected in response

12.10.16.4 Response message

The response to the Get_data command is as follows:

Data field	Data read (may be equal to Le bytes)
SW1-SW2	Status bytes

Status conditions are described in Subclause 12.10.20.

12.10.17 Put_data

12.10.17.1 Functional description

The Put_data function is used for storing one primitive data object, or one or more data objects contained in a constructed data object. The exact storing function (writing once and/or updating and/or appending) depends on the context.

12.10.17.2 Access conditions

This functions operates on an object. The function will only be processed if the AC for Put, CA, or Get functions is satisfied.

12.10.17.3 Command message

The command message is as follows:

CLA	See 12.10.2
INS	'DA'
P1-P2	'0000' to '003F': RFU '0040' to '00FF': BER-TLV tag (1 byte) in P2 '0100' - '017F': RFU for application data '0180' - '01FF': Private for application data '0200' - '02FF': Simple-TLV tag in P2 '0300' - '3FFF': RFU '4000' - 'FFFF': BER-TLV tag (2 bytes) in P1-P2
Lc field	Length of the subsequent data field
Data field	Parameter + data to be written
Le field	Empty

The following special Put_data commands are defined:

Transfer CA Message

To transfer a message from the STU to the smart card.

P1-P2	'0101': Send_message
Data field	Origin (TLV coded) Filtered message (Transparent)

The origin object indicates the origin of the accompanying message.

Tag	Length	Byte	Contents
see 12.12	02	01	source
		02	source detailed

The source and source detailed field are coded as follows:

Source		Source detailed	
Value	**Meaning**	**Value**	**Meaning**
'00'	reserved	-	-
'01'	section filter	'xx'	filter_program_number
'02'	MMI	'00'	no additional information
'03'	interaction channel	'00'	no additional information
'04'-'7F'	reserved		
'80'-'FE'	user private		
'FF'	reserved		

Select Objects

To select objects.

P1-P2	'0102': Select_object
Data field	Target object (TLV coded) Pattern (TLV coded) Mask (TLV coded) (optional)

The target_object object indicates the object to work on.

Tag	Length	Byte	Contents
see 12.12	01 or 02	01(-02)	tag value

If the tag value is primitive the indicated pattern shall be looked for. If the tag value is constructed an object within the indicated constructed object which satisfies the pattern (first part pattern is tag and length of intended object) shall be looked for.

The pattern object gives the pattern to compare with.

Tag	Length	Byte	Contents
see 12.12	var	n	pattern

The mask object gives the mask to mask the pattern with.

Tag	Length	Byte	Contents
see 12.12	var	n	mask

12.10.17.4 Response message

The response the Put_data command is as follows:

Data field	Empty
SW1-SW2	Status bytes

Status conditions are described in Subclause 12.10.20.

12.10.18 Get_application_status

12.10.18.1 Functional description

The Get_application_status function enables the STU to regularly poll the status of the smart card. Regularly polling of the status is mandatory.

12.10.18.2 Command message

The command message for the Get_application_status function is as follows:

CLA	See 12.10.2
INS	'F8'
P1	'00'
P2	'00'
Lc field	Length of the subsequent data field
Data field	Date + time
Le field	Empty

12.10.18.3 Response message (nominal case)

The response to the Get_application_status command is as follows:

Data field	Empty
SW1-SW2	Status bytes

Together with the Get_response command smart card output data can be obtained.

Status conditions are described in Subclause 12.10.20.

12.10.19 Perform_security_operation

12.10.19.1 Functional description

The Perform_security_operation function enables security services in the case of interactive applications.

12.10.19.2 Command message

The command message for the Perform_security_operation function is as follows:

CLA	See 12.10.2
INS	'8A'
P1	Tag of the data object the value field of which is transmitted in the response data field (as defined in ISO/IEC 7816-4).
P2	Tag of the data object the value field of which is transmitted in the command data field (as defined in ETS 300 468).
L_c field	Length of subsequent data field
Data field	Value of the data object specified in P2
L_e field	Maximum number of bytes expected in response

12.10.19.3 Response message (nominal case)

The response to the Perform_security_operation command is as follows:

Data field	Data or empty
SW1-SW2	Status bytes

Status conditions are described in Subclause 12.10.20.

Note: The format of this command mirrors functionality currently being defined in ISO DIS 7816-8. Changes in the ISO definition may result in revision to this DAVIC specification.

12.10.20 Status Conditions returned by the smart card

After each command the smart card shall return the status bytes SW1 and SW2 according to ISO/IEC 7816-3. The status conditions defined are in line with ISO/IEC 7816-4. The STU shall support the following minimum set of status bytes. It is allowed for the smart card to also output other status words if desired.

12.10.20.1 Normal ending

SW1	SW2	Description
'90'	'00'	Normal ending of the command
'61'	'XX'	Normal ending of the command. SW2 indicates the length of response bytes still available

12.10.20.2 General errors

SW1	SW2	Description
'67'	'00'	Wrong length
'6B'	'00'	Wrong parameter(s) P1 or P2
'6C'	'XX'	Wrong length Le: SW2 indicates the exact length
'6D'	'00'	Instruction code not supported or invalid
'6E'	'00'	Class not supported
'6F'	'00'	Error: no precise diagnosis

12.10.20.3 Command not allowed

SW1	SW2	Description
'69'	'81'	Command incompatible with file structure
'69'	'82'	Security status not satisfied
'69'	'83'	Authentication method blocked
'69'	'84'	Referenced data invalidated
'69'	'85'	Conditions of use not satisfied
'69'	'86'	Command not allowed (no current EF)

12.10.20.4 Wrong parameter(s) P1-P2

SW1	SW2	Description
'6A'	'80'	Incorrect parameters in the data field
'6A'	'81'	Function not supported
'6A'	'82'	File not found
'6A'	'83'	Record not found
'6A'	'84'	Not enough memory space in file
'6A'	'85'	Lc inconsistent with file structure
'6A'	'86'	Incorrect parameters P1-P2
'6A'	'87'	Lc inconsistent with P1-P2
'6A'	'88'	Referenced data (data objects) not found

12.10.20.5 Additional errors

SW1	SW2	Description
'62'	'83'	Selected file invalidated
'63'	'00'	No information given
'63'	'CX'	Counter privided by 'X'
'65'	'81'	Memory failure

12.10.20.6 Summary of Status Conditions for Commands

The following tables show an overview of the possible status conditions for each command.

	OK		General errors						Additional errors							
Commands	90 00	61 xx	67 00	6B 00	6C xx	6D 00	6E 00	6F 00	62 83	63 00	63 Cx	65 81				
Select_file	*		*	*	*	*	*	*	*							
Read_binary	*		*	*	*	*	*	*								
Read_record	*		*	*	*	*	*	*								
Seek	*		*	*	*	*	*	*								
Verify_password	*		*	*	*	*	*	*		*	*					
Change_password	*		*	*	*	*	*	*								
Disable_password	*		*	*	*	*	*	*								
Enable_password	*		*	*	*	*	*	*								
Unblock_password	*		*	*	*	*	*	*								
Internal_authentication	*		*	*	*	*	*	*								
Write_binary	*		*	*	*	*	*	*				*				
Write_record	*		*	*	*	*	*	*				*				
Update_binary	*		*	*	*	*	*	*				*				
Update_record	*		*	*	*	*	*	*				*				
Get_response	*	*	*	*	*	*	*	*								
Get_data			*	*	*	*	*	*								
Put_data	*		*	*	*	*	*	*				*				
	*															
Put_data(send_message)	*		*	*	*	*	*	*								
Put_data(select)	*		*	*	*	*	*	*								
Change_maturity_rating	*		*	*	*	*	*	*								
Get_application_status	*		*	*	*	*	*	*								
Perform_security_operation	*		*	*	*	*	*	*								

	Command not allowed							Wrong parameter(s) P1-P2								
Commands	69 81	69 82	69 83	69 84	69 85	69 86	-	6A 80	6A 81	6A 82	6A 83	6A 84	6A 85	6A 86	6A 87	6A 88
Select_file									*	*				*	*	
Read_binary	*	*				*			*	*						
Read_record	*	*				*			*	*	*					
Seek																
Verify_password			*	*										*		*
Change_password			*	*										*		*
Disable_password			*	*										*		*
Enable_password			*	*										*		*
Unblock_password																
Internal authentication				*	*									*		*
Write_binary	*	*				*			*	*						
Write_record	*	*				*			*	*	*	*	*			
Update_binary	*	*				*			*	*						
Update_record	*	*				*			*	*	*	*	*			
Get_response														*		
Get_data		*			*				*							*
Put_data		*			*			*	*			*	*			
Put_data(send_message)		*			*			*	*			*	*			
Put_data(select)	*	*			*			*	*		*		*			
Change_maturity_rating		*	*					*	*							
Get_application_status								*	*							
Perform_security_operation					*			*	*							

12.11 Man machine interface

12.11.1 Introduction

The man machine interface will be as much as possible the same as the high level man machine interface of the CA0 interface specification. In practice this means that the syntax and semantics of the high-level MMI objects defined in CENELEC EN 50 221 will be copied into this specification, except for tag allocation which will be such that it fits into this specification.

In addition to the already specified man-machine interactions the high-level interface of the CA0 interface specification (CENELEC EN 50 221) may also be used. Therefore this specification adopts all of the CA0 specified high-level MMI objects. These include:

text()

enq()

answ()

menu()

menu_answ()

list()

However, for inclusion into this specification these objects are modified as follows:

- The object tag fields (text_tag, enq_tag, ...) are replaced by the unified object_tag field. The length of this object_tag field is 16 bits uimsbf.
- The 'length_field()' is replaced by the object_length field. The length of the object_length field is 8 uimsbf.

The following text is copied from CENELEC EN 50 221:

The MMI objects define the transactions required but allow the host to determine the format and type of display. The contents of any text will be interpreted according to the current character set.

12.11.2 MMI objects

12.11.2.1 Text

The text object is used to display a block of text on the screen.

Syntax	No. of bits	Encoding
text(){		
object_tag	**8**	**uimsbf**
object_length	**8**	**uimsbf**
for(i=0;i<length;i++){		
text_char	**8**	**uimsbf**
}		
}		

A text object with length equal to 0 will be interpreted as a null object: nothing is displayed.

text_char: Text information is coded using the character sets and methods described in CENELEC EN 50221.

The text sent by the application may include control characters as are defined by CENELEC EN 50221 to provide indication of how the display is to be presented. The interpretation of these control characters is at the discretion of the host.

12.11.2.2 Enq(uiry)

Enq and Answ allow the application to request user input. There is a single statement, which may be a request for information such as user's PIN code. The response to the Enq is returned by the host in the Answ object. The Enq object allows the application to specify whether the user's input should be echoed back to the user by the host. For example, when entering a PIN code, the numbers entered by the user should not be displayed.

Syntax	No. of bits	Encoding
enq(){		
object_tag	**8**	**uimsbf**
object_length	**8**	**uimsbf**
reserved	**7**	**bslbf**
blind_answer	**1**	**bslbf**
answer_text_length	**8**	**uimsbf**
for(i=0;i<object_length-2;i++){		
text_char	**8**	**uimsbf**
}		
}		

blind_answer: set to 1 means that the user input has not to be displayed when entered. The host has the choice of the replacement character used (star, ...).

answer_text_length: expected length of the answer. Set to hex 'FF' if unknown by the Application.

text_char: Text information coded using the character sets and methods described in CENELEC EN 50221.

12.11.2.3 Answ(er)

This is object is used together with the Enq object to manage user inputs.

Syntax	No. of bits	Encoding
answ(){		
object_tag	**8**	**uimsbf**
object_length	**8**	**uimsbf**
answ_id	**7**	**bslbf**
if(answ_id == answer){	8	uimsbf
for(i=0;i<length;i++){		
text_char	**8**	**uimsbf**
}		
}		
}		

answ_id: This field is used to indicate the type of response received from the user. answ_id values are coded as follows:

answ_id	value
cancel	00
answer	01
reserved	other values

The value answer means that the object contains the user input (which may be of zero length). The value cancel means that the user wishes to abort the dialogue.

12.11.2.4 Menu

This object is used in conjunction with the Menu Answ object to manage menus in the high level MMI mode. A menu is made of one Title, one sub-title, several choices and one bottom line. Text objects with text_length = 0 can be used (.g. if no sub-title or no bottom text are used).

Syntax	No. of bits	Encoding
menu(){		
object_tag	**8**	**uimsbf**
object_length	**8**	**uimsbf**
choice_nb	**8**	**uimsbf**
TEXT() /* title text */		
TEXT() /* sub-title text */		
TEXT() /* bottom text */		
for (i=0; i<choice_nb;i++) /* when choice_nb !='FF' */		
TEXT()		
}		

Choice_nb: = 'FF' means that this field does not carry the number of choices information.

The way the host has to display the text, the menu box and select the choice is manufacturer dependent. For example, the host is free to display the choices on several pages and to manage itself the page-down and page-up functions. The manufacturer is also free to define how the user selects the choice (numerical keypad, arrows, colored keys, voice etc).

12.11.2.5 Menu answ(er)

This object is used in conjunction with the menu object to return the user choice, and the with list object to indicate that the user has finished with that object.

Syntax	No. of bits	Encoding
menu(){		
object_tag	**8**	**uimsbf**
object_length = 1	**8**	**uimsbf**
choice_ref	**8**	**uimsbf**
}		

choice_ref : the number of the choice selected by the user. If the object was preceded by a menu object, then choice_ref = 01 corresponds to the first choice that had been presented by the application in that object (first choice text after the bottom text in the menu object) and choice_ref = 02 corresponds to the second choice text presented by the application.

choice_ref = 00 indicates that the user has cancelled the preceding menu or list object without making a choice.

12.11.2.6 List

This object is used to send a list of items to be displayed (eg. during a consultation of the entitlements). It has exactly the same syntax as the Menu object, and is used in conjunction with the Menu_answ object. A list is made of one Title, one sub-title, several items and one bottom line. Text objects with text_length = 0 can be used (eg. if no sub-title or no bottom text are used).

Syntax	No. of bits	Encoding
menu(){		
object_tag	**8**	**uimsbf**
object_length	**8**	**uimsbf**
item_nb	**8**	**uimsbf**
TEXT() /* title text */		
TEXT() /* sub-title text */		
TEXT() /* bottom text */		
for (i=0; i<item_nb;i++) /* when choice_nb !='FF' */		
TEXT()		
}		

Item_nb: = 'FF' means that this field does not carry the number of items information.

The way the host has to display the title, sub-title, bottom text and items is manufacturer dependant. For example, the host is free to display the items on several pages and to manage itself the page-down and page-up functions.

12.12 Tag allocation

12.12.1 General

In general tag allocation is according to the following table:

Class	Tag range	Primitive or constructed
Universal	'00' - '1E'; '1F01' - '1F7F' '20' - '3E'; '3F01' - '3F7F'	primitive constructed
Application	'40' - '5E'; '5F01' - '5F7F' '60' - '7E'; '7F01' - '7F7F'	primitive constructed
Context-specific	'80' - '9E'; '9F01' - '9F7F' 'A0' - 'BE'; 'BF01' - 'BF7F'	primitive constructed
Private	'C0' - 'DE'; 'DF01' - 'DF7F' 'E0' - 'FE' 'FF01' - 'FF7F'	primitive constructed

This specification reserves the following private tag range for DAVIC use:

‘C0’ - ‘CF’ private, primitive
‘DF0F’ - ‘DF3F’ private, primitive
‘E0’ - ‘EF’ private, constructed
‘FF0F’ - ‘FF3F’ private, constructed

The following range of tags can be used privately:

‘D0’ - ‘DE’ private, primitive
‘DF40’ - ‘DF7F’ private, primitive
‘F0’ - ‘FE’ private, constructed
‘FF40’ - ‘FF7F’ private, constructed

12.12.2 Tag allocation for DAVIC application

DAVIC pay-TV application data objects (including universal class and application class objects):

Tag	Name of Data Element	Description & Reference	Lgth	May be located within template
‘4F’ app, prim	Application identifier (AID)	A DE which uniquely identifies an application in a smart card. (see ISO/IEC 7816-5)	var	‘6E’ ‘61’
‘50’ app, prim	Application label	A DE to give the application a name for the man machine interface (see ISO/IEC 7816-5)	var	‘6E’ ‘61’
‘51’ app, prim	Path	Gives the location of a DE (see ISO/IEC 7816-4)	var	-
‘52’ app, prim	Command to perform	Gives a command to perform a desired action. (see ISO/IEC 7816-5 and ISO/IEC 7816-6)	var	-
‘5C’ app, prim	Taglist	A concatenation of tags without delimiters	var	-
‘5F24’ app, prim	Application expiration date	Gives the date after which the application expires. Coding is according to ETS 300 468 (MJD)	5	‘6E’
‘5F25’ app, prim	Application effective date	Gives the date from after which the application can be used. Coding is according to ETS 300 468 (MJD)	5	‘6E’
‘61’ app, constr	Application template	The application template contains one or more objects related to the application	var	-
‘63’ app, constr	Wrapper	The wrapper enables indirect referencing and retrieval of DOs (see ISO/IEC 7816-5)	var	-
‘6E’ app, constr	Application related data template	As defined in ISO/IEC 7816-6	var	-
‘6F’ app, constr	FCI template	As defined in ISO/IEC 7816-4 (see 4.4.2)	var	-
‘79’ app, constr	Coexistent Tag Allocation Authority template	As defined in ISO/IEC 7816-6 (see 4.4.4)	var	-
‘C0’ priv, prim	System_id	A DE which identifies the system which uses the application. For broadcasting this is equal to the CA_system_id	var	‘E0’

‘C1’ priv, prim	System_label	A DE used at the man machine interface to describe the system	var	‘E0’
‘C2’ priv, prim	System_provider_label	A DE used at the man machine interface to describe the provider of the system	var	‘E0’
‘C3’ priv, prim	CA_des_selector	A DE describing the characteristics of the MPEG CA descriptor meant for the system	var	‘E0’ ‘E1’
‘C4’ priv, prim	Service_provider_id	A DE which identifies an service provider of the system	1 or 2	‘E1’
‘C5’ priv, prim	Service_provider_label	A DE used at the man machine interface to describe an service provider of the system	var	‘E1’
‘C6’ priv, prim	Base_channel	A DE identifying the channel on which an service provider transmits his content	6	‘E1’
‘C7’ priv, prim	Parental_rating	A DE containing the programmed maturity rating value	var	-
‘C8’ priv, prim	Password	Enables TLV coding for the password	var	-
‘C9’ priv, prim	Preferred_service_provider	Indicates the preferred service_provider when using the CA system	01/02	‘E0’
‘CA’ priv, prim	Even/odd_control_word	A DE to transfer an even and odd control word to the STU for descrambling	var	-
‘CB’ priv, prim	Even_control_word	A DE to transfer an even control word to the STU for descrambling	var	-
‘CC’ priv, prim	Odd_control_word	A DE to transfer an odd control word to the STU for descrambling	var	-
‘CD’ priv, prim	Status	A DE which represents the status of the smart card	var	-
‘CE’ priv, prim	Access-conditions	A DE which contains the access conditions to a DO	var	-
‘CF’ priv, prim	Origin	This DE indicates the origin of a filtered message	02	-
‘DF01’priv, prim	Filter_set	This DE lists the different filter programs	var	-
‘DF02’ priv, prim	Filter_program	This DE describes the states of one filter program	var	-
‘DF03’ priv, prim	Update_filter_state	A DE which contains a new filter state for the filter program ìrunningî in the STU	1	-
‘DF04’ priv, prim	Update_filter_conditions	A DE which contains new filter conditions for the filter program ìrunningî in the STU	var	-

‘DF06’ priv, prim	Target	This DE gives the tag value of the target object to look in	01/02	-
‘DF07’ priv, prim	Pattern	This DE gives the pattern to compare with in the target object	var	-
‘DF08’ priv, prim	Mask	This DE gives an optional mask to mask the pattern to compare with in the target object	var	-
‘DF10’ priv, prim	text()	This DE is used to pass a text to display on the screen	var	-
‘DF11’ priv, prim	enq()	This DE allows the application to request user input	var	-
‘DF12’ priv, prim	answ()	This DE transfers the user input on an enquiry from the STB to the smart card	var	-
‘DF13’ priv, prim	menu()	This DE represents a menu to be shown on the screen. It is used in conjunction with menu_ans()	var	-
‘DF14’ priv, prim	menu_ans()	This DE transfers the answer on a presented menu. It is used in conjunction with menu()	01	-
‘DF15’ priv, prim	menu_list()	This DE is used to send a list of items to be displayed to the STB	var	-
‘E0’ priv, const	System_related_ data template	This DE contains system related data. It can also be used to reference system related data by the wrapper	var	-
‘E1’ priv, const	Service_provider_ related_data template	This DE contains service provider related data. It can also be used to reference service provider related data by the wrapper	var	-
‘E2’ priv, const	Entitlement_data_ template	Template which contains context specific entitlement data	var	-
‘E3’ priv, const	Interactive _security _related_data	Template which contains related data for interactive security functions	var	-

The application_related_data template (tag=‘6E’) may contain the following objects:

Tag	**Name of Data Element**	**Description & Reference**	**Lgth**	**May be located within template**
‘4F’ app, prim	Application identifier	A DE which uniquely identifies an application in a smart card. (see ISO/IEC 7816-5)	var	‘6E’
‘50’ app, prim	Application label	A DE to give the application a name for the man machine interface (see ISO/IEC 7816-5)	var	‘6E’
‘5F24’ app, prim	Application expiration date	Gives the date after which the application expires. Coding is according to ETS 300 468 (MJD)	5	‘6E’
‘5F25’ app, prim	Application effective date	Gives the date from after which the application can be used. Coding is according to ETS 300 468 (MJD)	5	‘6E’

The system_data template (tag='E0' priv, const) may contain the following objects:

Tag	Name of Data Element	Description & Reference	Lgth	May be located within template
'C0' priv, prim	System_id	A DE which identifies the system which uses the application. For broadcasting this is equal to the CA_system_id	var	'E0'
'C1' priv, prim	System_label	A DE used at the man machine interface to describe the system	var	'E0'
'C2' priv, prim	System_provider _label	A DE used at the man machine interface to describe the provider of the system	var	'E0'
'C3' priv, prim	CA_descriptor _selector	A DE describing the characteristics of the MPEG CA descriptor meant for the system	var	'E0'
'C9' priv, prim	Preferred_service _provider	A DE used to indicate which is the preferred service provider on the card	1 or 2	'E0'

The service_provider_data template (tag='E1' priv, const) may contain the following objects:

Tag	Name of Data Element	Description & Reference	Lgth	May be located within template
'C6' priv, prim	Service_provider_id	A DE which identifies an service provider of the system	1 or 2	'E1'
'C7' priv, prim	Service_provider_label	A DE used at the man machine interface to describe an service provider of the system	var	'E1'
'C8' priv, prim	Base_channel	A DE identifying the channel on which an service provider transmits his content	var	'E1'
'C5' priv, prim	CA_descriptor _selector	A DE describing the characteristics of the MPEG CA descriptor meant for the system	var	'E1'

The following context specific data objects only may occur in the FCI template (tag='6F' app, const).

Tag	Name of Data Element	Description & Reference	Lgth	Shall be located within template
'82' contxt, prim	File descriptor	A DE describing the corresponding file. As defined in ISO/IEC 7816-4 (see 5.1.5)	1, 3/4	'6F'
'83' contxt, prim	File identifier	Gives the file identifier in the file header. As defined in ISO/IEC 7816-4 (see 5.1.5)	2	'6F'
'84' contxt, prim	DF name	Gives the DF name in the file header. As defined in ISO/IEC 7816-4 (see 5.1.5)	1 to 16	'6F'
'85' contxt, prim	Application proprietary infO	Gives application proprietary information in the file header	var	'6F'
'XX'	Other DOs	Other DOs as defined in ISO/IEC 7816-4 may be contained in the file header (see 5.1.5 for other context specific DOs)	-	-

The following context specific data objects only may occur in the Filter_set object or Filter_program object.

Tag	Name of Data Element	Description & Reference	Lgth	Shall only be located within object
'91' contxt, prim	APDU	A DE containing the APDU which should be used to send a message from the STU to the smart card	4	'DF01' 'DF02'
'92' contxt, prim	CA_descriptor_ association	A DE containing a component_tag (see DVB-SI) to enable correct reference to a CA-descriptor and PID.	1	'DF01' 'DF02'
'93' contxt, prim	CW_association	A DE containing a component_tag (see DVB-SI) to associate a control word with the correct elementary_PID	1	'DF01' 'DF02'
'94' contxt, prim	Filter_state	A DE describing the different states of a filter_program	var	'DF01' 'DF02'
'95' contxt,prim	Table_id	A DE containing the table_id of the section to filter on	2	'94' in 'DF01' 'DF02'
'96' contxt, prim	Filter_pattern	A DE containing the characteristics to filter on for a section	1 to 9	'94' in 'DF01' 'DF02'
'97' contxt, prim	Filter_mask	A DE containing a mask value for the section to filter on	1 to 9	'94' in 'DF01' 'DF02'

The following context specific data objects only may occur in the Entitlement_template (tag='E2' priv, const).

Tag	Name of Data Element	Description & Reference	Lgth	Shall only be located within template
'C6' priv, prim	Service_provider_id	A DE which contains the service provider identification number	1 or 2	-
'91' contxt, prim	Entitlement_type	A DE which represents the type of an entitlement present	1	'E2'
'94' contxt, prim	Entitlement_label	A DE containing a description of the entitlement	var	'E2'
'95' contxt, prim	Entitlement_dates	A DE containing the begin and end date the entitlement is valid	4	'E2'
'96' contxt, prim	Entitlement_date/duration	A DE containing the start time and duration of an entitlement	7	'E2'
'97' contxt, prim	SI_link	A DE containing a backward link to service and or event information in the stream	10	'E2'

The following table gives the basic response messages of the smart card.

Tag	Name of Data Element	Description & Reference	Lgth	May be located within template
'CA' priv, prim	Even/odd_ control_word	A DE to transfer an even and odd control word to the STU for descrambling	var	-
'CB' priv, prim	Even_control_ word	A DE to transfer an even control word to the STU for descrambling	var	-
'CC' priv, prim	Odd_control _word	A DE to transfer an odd control word to the STU for descrambling	var	-
'CD' priv, prim	Status	A DE which represents the status of the smart card	var	-
'CE' priv, prim	Access-conditions	A DE describing the access conditions to a file or data object	4	-
'DF03' priv, prim	Update_filter_ state	A DE which contains a new filter state for the filter program "running" in the STU	1	-
'DF04' priv, prim	Update_filter_ conditions	A DE which contains new filter conditions for the filter program "running" in the STU	var	-

The following data objects may be used for man machine interaction:

Tag	Name of Data Element	Description & Reference	Lgth	Maybe located within template
'DF10' priv, prim	text()	This DE is used to pass a text to display on the screen	var	-
'DF11' priv, prim	enq()	This DE allows the application to request user input	var	-
'DF12' priv, prim	answ()	This DE transfers the user input on an enquiry from the STB to the smart card	var	-
'DF13' priv, prim	menu()	This DE represents a menu to be shown on the screen. It is used in conjunction with menu_ans()	var	-
'DF14' priv, prim	menu_ans()	This DE transfers the answer on a presented menu. It is used in conjunction with menu()	1	-
'DF15' priv, prim	menu_list()	This DE is used to send a list of items to be displayed to the STB	var	-

13. Additional Resources for the DAVIC CA0 Interface

The baseline document CENELEC EN 50 221 for the CA0 specification defines a set of resources, some of which are used in the DAVIC environment. Other resources not in that specification but mandatory in the DAVIC environment are defined here.

13.1 Host-provided Resources

13.1.1 TCP/IP Socket Resource

This resource defines an interface very similar to a subset of the BSD Sockets interface for the CA0 security device to create TCP and UDP connections on an IP communications stack already established by the STU. It gives the PC card general access to TCP/IP services, but in particular is intended to allow it to use directory and name services to find relevant security information.

13.1.1.1 BSD Sockets

Sockets were developed as part of the BSD UNIX implementation as an inter-process communication abstraction. In UNIX both pipes and Unix-domain sockets, as well as TCP/IP are supported by the socket abstraction but in this proposal only TCP/IP is supported. In UNIX the socket interface consists of a set of system service function calls - socket(), bind(), connect(), accept(), listen(), send(), sendto(), recv() and recvfrom(). The functions getsockopt() and setsockopt() are also provided to perform management operations on sockets. The file system functions read(), write(), close() and select() are also supported on sockets, as is the I/O management function ioctl() with a limited set of operations.

13.1.1.2 Resource Concept

Operation of the resource protocol is by exchange of objects, rather than as function calls. However the function call mechanism allows synchronous operation, where the calling process can be blocked until the conditions for function completion exist. In the standard BSD sockets implementation several functions - connect(), accept(), select(), amongst others - naturally make use of this blocking behavior.

In contrast the protocol used by all resources is naturally asynchronous. An object is sent but no blocking occurs at that point. Any response arriving later as a result is processed when it is read by the application. How this is done