ETSI GS ECI 001-3 V1.1.1 (2017-07)

Embedded Common Interface (ECI) for exchangeable CA/DRM solutions; Part 3: CA/DRM Container, Loader, Interfaces, Revocation

Disclaimer

The present document has been produced and approved by the Embedded Common Interface (ECI) for exchangeable CA/DRM solutions ETSI Industry Specification Group (ISG) and represents the views of those members who participated in this ISG.
It does not necessarily represent the views of the entire ETSI membership.

Reference
DGS/ECI-001-3

Keywords
CA, DRM, swapping

ETSI

650 Route des Lucioles
F-06921 Sophia Antipolis Cedex - FRANCE

Tel.: +33 4 92 94 42 00 Fax: +33 4 93 65 47 16

Siret N° 348 623 562 00017 - NAF 742 C
Association à but non lucratif enregistrée à la
Sous-Préfecture de Grasse (06) N° 7803/88

Important notice

The present document can be downloaded from:
http://www.etsi.org/standards-search

The present document may be made available in electronic versions and/or in print. The content of any electronic and/or print versions of the present document shall not be modified without the prior written authorization of ETSI. In case of any existing or perceived difference in contents between such versions and/or in print, the only prevailing document is the print of the Portable Document Format (PDF) version kept on a specific network drive within ETSI Secretariat.

Users of the present document should be aware that the document may be subject to revision or change of status.
Information on the current status of this and other ETSI documents is available at
https://portal.etsi.org/TB/ETSIDeliverableStatus.aspx

If you find errors in the present document, please send your comment to one of the following services:
https://portal.etsi.org/People/CommiteeSupportStaff.aspx

Contents

List of Figures

List of Tables

Intellectual Property Rights

Essential patents

IPRs essential or potentially essential to the present document may have been declared to ETSI. The information pertaining to these essential IPRs, if any, is publicly available for **ETSI members and non-members**, and can be found in ETSI SR 000 314: *"Intellectual Property Rights (IPRs); Essential, or potentially Essential, IPRs notified to ETSI in respect of ETSI standards"*, which is available from the ETSI Secretariat. Latest updates are available on the ETSI Web server (https://ipr.etsi.org/).

Pursuant to the ETSI IPR Policy, no investigation, including IPR searches, has been carried out by ETSI. No guarantee can be given as to the existence of other IPRs not referenced in ETSI SR 000 314 (or the updates on the ETSI Web server) which are, or may be, or may become, essential to the present document.

Trademarks

The present document may include trademarks and/or tradenames which are asserted and/or registered by their owners. ETSI claims no ownership of these except for any which are indicated as being the property of ETSI, and conveys no right to use or reproduce any trademark and/or tradename. Mention of those trademarks in the present document does not constitute an endorsement by ETSI of products, services or organizations associated with those trademarks.

Foreword

This Group Specification (GS) has been produced by ETSI Industry Specification Group (ISG) Embedded Common Interface (ECI) for exchangeable CA/DRM solutions.

The present document is part 3 of a multi-part deliverable covering the CA/DRM Container, Loader, Interfaces, Revocation for the Embedded Common Interface for exchangeable CA/DRM solutions specification, as identified below:

Part 1: "Architecture, Definitions and Overview";

Part 2: "Use cases and requirements";

Part 3: "CA/DRM Container, Loader, Interfaces, Revocation";

Part 4: "The Virtual Machine";

Part 5: "The Advanced Security System ";

Part 6: "Trust Environment".

The use of terms in bold and starting with capital characters in the present document shows that those terms are defined with an **ECI** specific meaning, which may deviate from the common use of those terms.

Modal verbs terminology

In the present document "**shall**", "**shall not**", "**should**", "**should not**", "**may**", "**need not**", "**will**", "**will not**", "**can**" and "**cannot**" are to be interpreted as described in clause 3.2 of the ETSI Drafting Rules (Verbal forms for the expression of provisions).

"**must**" and "**must not**" are **NOT** allowed in ETSI deliverables except when used in direct citation.

Introduction

Service and content protection realized by Conditional Access (CA) and Digital Rights Management (DRM) are essential in the rapidly developing area of digital Broadcast and Broadband services. This includes the distribution of HD and UHD content to various types of customer premises equipment (**CPE**) in order to protect business models of content owners and service providers, including Broadcasters and PayTV operators. While CA systems primarily focus on the protection of content distributed via unidirectional networks as usually used in broadcast environment, DRM systems originate from bidirectional network environments and permit access to content on certified devices for authenticated users, with typically rich content rights expressions. In practice, a clear distinction between CA and DRM functionalities is not feasible in all cases and therefore within the present document the term CA/DRM systems is used.

Currently implemented CA/DRM solutions, whether embedded or as detachable hardware, often result in usage restrictions for service/platform providers on one side and consumers on the other. The consequences for consumers are dependencies with regard to the applicable network, service and content providers and the applied **CPE** suited for classical digital broadcasting, IPTV or OTT (over-the-top) services. While **CPEs** with embedded platform-proprietary CA or DRM functionality bind a customer to a specific platform operator, detachable hardware modules allow using retail **CPE** as e.g. Set-Top-Boxes (STB) and integrated TV sets (iDTV). Due to their form factor and cost, detachable hardware modules do not fulfil future demands, especially those with regard to consumption of protected content on tablets and mobile devices and for cost-critical deployments.

Existing technologies thus bind the freedom of many players in digital multimedia content markets. Due to technological progress, innovative, software-based CA/DRM solutions become feasible. Maximizing interoperability while maintaining a high level of security, these solutions promise to meet upcoming demands in the market, allow for new businesses, and broaden consumer choice with respect to content consumption via broadcast and broadband connections.

It is in consumers' interest that bought and owned **CPEs** are available for further use after a move or a change of the network provider and that those devices can be utilized for services of different commercial video portals. This can be achieved by the implementation of interoperable CA and DRM mechanisms inside **CPEs** based on appropriate security architecture. Further fragmentation of the market for **CPEs** can only be prevented and competition encouraged by ensuring solutions for consumer-friendly and flexible exchangeability of CA and DRM systems, associated with a state-of-the-art security environment.

It is in the Platform Operator's interest that security technology can be deployed flexibly and managed easily across various networks and on all kinds of devices. The advantage of updating existing devices with the latest security systems in a seamless way provides unparalleled business opportunity.

An **ECI Ecosystem** as specified in the present document according to the **ECI** multi-part deliverable addresses important attributes, as flexibility and scalability due to software-based implementation, exchangeability fostering a future-proof solution and enabling innovation. Further aspects are applicability to content distributed via different types of networks, including classical digital broadcasting, IPTV and OTT services. The **ECI** system specification of an open eco-system, fostering market development, provides the basis for exchangeability of CA and DRM systems in **CPEs**, at lowest possible costs for the consumers and with minimal restrictions for CA or DRM vendors to develop their target products for the PayTV market.

Beside Part 4 for the Virtual Machine and Part 5 for the Advanced Security, including two sub-parts, the present document, Part 3 of a multi-part deliverable, specifies all necessary elements, which are essential for the download and exchange of CA/DRM clients (**ECI Clients**) and their execution environment (**ECI Host**) under a trusted environment, including communication with necessary functional entities via APIs, which are specified in detail.

1 Scope

The architecture of the **ECI system** is defined in the **ECI** specification ETSI GS ECI 001-1 [1]. The **ECI** system is based on requirements as defined in the **ECI** specification ETSI GS ECI 001-2 [2]. The present document specifies the core functionality of an **ECI Ecosystem**, including CA/DRM Container, Loader, Interfaces and Revocation details. A major advantage and innovation of the **ECI Ecosystem**, compared with currently deployed systems, is a complete software-based architecture for the loading and exchange of CA/DRM systems, avoiding any detachable hardware modules. Software containers provide a secure ("Sandbox") environment for either CA or DRM kernels, hereafter named as **ECI Clients**, together with their individual **Virtual Machine** instances. Necessary and relevant Application Programming Interfaces (API) between **ECI Clients** and **ECI Host** ensure that multiple **ECI Clients** can be operated in a secure operation environment and completely isolated from the rest of the **CPE** firmware and are specified in full detail. The installation and exchange of an **ECI Host** as well as multiple **ECI Client**s is the task of the **ECI** Loader, which initially is loaded by a chip loader. **ECI Host** and **ECI Clients** are downloaded via the DVB data carousel for broadcast services and/or via IP-based mechanisms from a server in case of broadband access. This process is embedded in a secure and trusted environment, providing a trust hierarchy for installation and exchange of **ECI Host** and **ECI Clients** and thus enabling an efficient protection against integrity- and substitution attacks. For this reason, the **ECI Ecosystem** integrates an advanced security mechanism, which relies on an efficient and advanced processing of control words, specified as **Key Ladder block** and integrated in a System-on-chip (SoC) hardware in order to provide the utmost security necessary for **ECI** compliance. **ECI**-specific advanced security functions play also a key role in a re-encryption process in case of stored protected content and/or associated with export of protected content to an **ECI**-compliant or non-compliant external device. An advanced Micro DRM system provides the necessary functionality and forms an integral part of such a concept. Advanced security functionality is relevant also in case of revocation of a **CPE** or a specific **ECI Client**. Related APIs are specified within the present document, while advanced security is covered in detail by ETSI GS ECI 001-5-1 [4] and ETSI GS ECI 001-5-2 [8].

A number of APIs characterize the **ECI Ecosystem**, guaranteeing communication with relevant entities associated e.g. with **ECI** Loaders, import and export of protected content, advanced security, decryption and encryption, local storage facilities and watermarking. Additional APIs are available for **ECI Client** Man-Machine-Interface (MMI) or for an optional **Smart Card** reader.

Exchange of **ECI Client**s is initiated by the user or may be requested by a platform operator in case of necessary updates. A minimum of two **ECI Client**s are supported, with two additional **ECI Client**s as far as local storage on a Personal Video Recorder (PVR) is available or for export reasons.

The present document covers specification details in the following clauses:

The **ECI** certificate system is specified in clause 5, covering **Certificates** for various purposes as for **ECI Host Loader**, **ECI Client Loader** and **ECI Operator Certificates**, including definition of these **Certificates** and associated **Revocation List**, their composition into chains and the **Root Certificate** structure.

The **ECI Host Loader** is subject of clause 6, where the **ECI Host** loading process addresses storage of an image, verification of the image authenticity of the image by the **CPE** using **ECI TA** provided authentication data and the subsequent activation of the image, including specification of file format, transport protocol and **Operator** specific revocation of **ECI Host Images**.

Clause 7 covers all specification details with regard to the **ECI Client Loader**, based on the fact, that the **ECI Host** can download, store and activate **ECI Client Images** and accompanying data. The **ECI Client** loading process can be split up into several steps ranging from discovery process to download and initialization of **ECI Client**s, allowing the download process to be performed using data from the broadcast stream or from the internet,

Clause 8 deals with Revocation specification details including functionality to selectively exclude delivery of services to **CPEs** based on the **ECI TA** status of the **CPE** hardware, the **ECI Host**, other **Platform Operations** and **ECI Clients** loaded.

Detailed specifications of **ECI Client** interfaces can be found in clause 9, covering among very comprehensive specification details, necessary for the **ECI** eco-system, APIs for general **ECI Host** resources, **ECI**-specific **ECI Host** resources, **ECI Host** decryption resources, **ECI Host** re-encryption resources, content protection-related resources and **ECI Client**-to-**ECI Client** - related resources.

Finally clause 10 deals with mandatory and optional **ECI Host** functionalities.

This **ECI** core specification only applies to reception and further processing of content, which is controlled by a Conditional Access and/or Digital Rights Management system and has been encrypted by the service provider.

Content that is not controlled by a Conditional Access and/or DRM system is not covered by the present document.

The present document is intended to be used in combination with a contractual framework (license agreement), compliance and robustness rules and appropriate certification process agreements under control of a trust authority, which are not subject to technical specifications as represented by **ECI** Group Specifications. Some of these basic aspects can be found in an informative annex to ETSI GS ECI 001-6 [i.11], Trust Environment, which specifies the technical mechanisms and relations concerning a trusted environment.

2 References

2.1 Normative references

References are either specific (identified by date of publication and/or edition number or version number) or non-specific. For specific references, only the cited version applies. For non-specific references, the latest version of the referenced document (including any amendments) applies.

Referenced documents which are not found to be publicly available in the expected location might be found at https://docbox.etsi.org/Reference/.

NOTE: While any hyperlinks included in this clause were valid at the time of publication, ETSI cannot guarantee their long term validity.

The following referenced documents are necessary for the application of the present document.

[1] ETSI GS ECI 001-1: "Embedded Common Interface (ECI) for exchangeable CA/DRM solutions; Part 1: Architecture, Definitions and Overview".

[2] ETSI GS ECI 001-2: "Embedded Common Interface (ECI) for exchangeable CA/DRM solutions; Part 2: Use cases and requirements".

[3] ETSI GS ECI 001-4: "Embedded Common Interface (ECI) for exchangeable CA/DRM solutions; Part 4: The Virtual Machine".

[4] ETSI GS ECI 001-5-1: "Embedded Common Interface (ECI) for exchangeable CA/DRM solutions Part 5: The Advanced Security System Sub-part 1: ECI specific functionalities".

[5] ETSI GS ECI 001-5-2: "Embedded Common Interface (ECI) for exchangeable CA/DRM solutions; Part 5: The Advanced Security System; Sub-part 2: Key Ladder Block".

[6] ISO/IEC 23001-7:2015: "Information technology - MPEG systems technologies - Part 7: Common encryption in ISO base media file format files".

[7] ISO/IEC 23009-1:2014: "Information technology - Dynamic adaptive streaming over HTTP (DASH) - Part 1: Media presentation description and segment formats".

[8] ISO/IEC 13818-1-1:2007: "Information technology - Generic coding of moving pictures and associated audio information - Part 1: Systems".

[9] National Institute of Standards and Technology: "Recommendation for Block 2001 Edition Cipher Modes of Operation", 2001 edition.

[10] NIST U.S. FIPS PUB 197 (FIPS 197) (2001): "Advanced Encryption Standard (AES)".

[11] ISO/IEC 21320: "Information technology - Document Container File - Part 1: Core".

[12] IETF RFC 4122 (July 2015): "A Universally Unique IDentifier (UUID) URN Namespace".

[13] CEN EN 50221 (1997): "Common Interface Specification for Conditional Access and other Digital Video Broadcasting Decoder Applications".

[14] ETSI TS 102 006: "Digital Video Broadcasting (DVB); Specification for System Software Update in DVB Systems".

[15] ETSI EN 301 192: "Digital Video Broadcasting (DVB); DVB specification for data broadcasting".

[16] ETSI TR 101 202: "Digital Video Broadcasting (DVB); Implementation guidelines for Data Broadcasting".

[17] ISO/IEC 13818-6: "Information technology - Generic coding of moving pictures and associated audio information - Part 6: Extensions for DSM-CC".

[18] IETF RFC 2616: "Hypertext Transfer Protocol - HTTP/1.1".

[19] ETSI EN 300 468: "Digital Video Broadcasting (DVB); Specification for Service Information (SI) in DVB systems".

[20] ETSI TS 101 162: "Digital Video Broadcasting (DVB); Allocation of identifiers and codes for Digital Video Broadcasting (DVB) systems".

[21] ETSI TS 101 211: "Digital Video Broadcasting (DVB); Guidelines on implementation and usage of Service Information (SI)".

[22] IETF RFC 768: "User Datagram Protocol (UDP)".

[23] IETF RFC 791: "Internet Protocol (IP)".

[24] IETF RFC 793: "Transmission Control Protocol (TCP)".

[25] IETF RFC 1034: "Domain names - Concepts and Facilities".

[26] IETF RFC 1035: "Domain names - Implementation and Specification".

[27] IETF RFC 2460: "Internet Protocol, Version 6 (IPv6) Specification".

[28] IETF RFC 1123: "Requirements for Internet Hosts -- Application and Support".

[29] IETF RFC 952: "DOD Internet Host Table Specification".

[30] IANA: "Media Types".

NOTE: Available at http://www.iana.org/assignments/media-types/media-types.xhtml.

[31] ISO/IEC 7816-1: "Identification cards - Integrated circuit cards - Part 1: Cards with contacts - Physical Characteristics".

[32] ISO/IEC 7816-2: "Identification cards - Integrated circuit cards - Part 2: Cards with contacts - Dimensions and location of the contacts".

[33] ISO/IEC 7816-3: "Identification cards - Integrated circuit cards - Part 3: Cards with contacts - Electrical Interface and transmission protocols".

[34] ETSI TS 103 205 (V1.2.1) (11-2015): "Digital Video Broadcasting (DVB); Extensions to the CI Plus™ Specification".

[35] ISO/IEC 7816-5: "Identification cards - Integrated circuit cards - Part 3: Cards with contacts - Registration of application providers".

[36] ISO/IEC 7810: "Identification cards - Physical characteristics".

[37] ISO/IEC 23001-9:2014: "Information Systems - MPEG system technologies - Part 9: Common Encryption of MPEG2 transport streams".

[38] ETSI TS 103 285 (2015): "Digital Video Broadcasting (DVB); MPEG-DASH Profile for Transport of ISO BMFF Based DVB Services over IP Based Networks".

[39] ISO/IEC 14496-12:2015: "Information technology - Coding of audio-visual objects - Part 12: ISO base media format".

[40] ETSI ETR 289 (1996): "Digital Video Broadcasting (DVB); Support for use of scrambling and Conditional Access (CA) within digital broadcasting systems".

[41] ETSI TS 103 127: "Digital Video Broadcasting (DVB); Content Scrambling Algorithms for DVB-IPTV Services using MPEG2 Transport Streams".

[42] ETSI TS 100 289: "Digital Video Broadcasting (DVB); Support for use of the DVB Scrambling Algorithm version 3 within digital broadcasting systems".

[43] IETF RFC 7230 (June 2014): "Hypertext Transfer Protocol (HTTP/1.1): Message Syntax and Routing".

[44] IETF RFC 7231 (June 2014): "Hypertext Transfer Protocol (HTTP/1.1): Semantics and Content".

[45] IETF RFC 5246 (August 2008): "The Transport Layer Security (TLS) Protocol Version 1.2".

[46] IETF RFC 5288 (August 2008): "AES Galois Counter Mode (GCM) Cipher Suites for TLS".

[47] IETF RFC 6066 (January 2011): "Transport Layer Security (TLS) Extensions: Extension Definitions".

[48] IETF RFC 5280 (May 2008): "Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile".

[49] IETF RFC 6818 (January 2013): "Updates to the Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile".

[50] W3C Recommendation: "Portable Network Graphics (PNG) Specification (Second Edition)".

NOTE: Available at http://www.w3.org/TR/PNG.

[51] IETF RFC 6151 (March 2011): "Updated Security Considerations for the MD5 Message-Digest and the HMAC-MD5 Algorithms".

[52] IETF RFC 6125 (March 2011): "Representation and Verification of Domain-Based Application Service Identity within Internet Public Key Infrastructure Using X.509 (PKIX) Certificates in the Context of Transport Layer Security (TLS)".

[53] ISO/IEC 8859-1:1998: "Information technology - 8-bit single-byte coded graphic character sets, Part 1: Latin alphabet No. 1".

[54] ISO 3166-1:2006: "Codes for the representation of names of countries and their subdivisions - Part 1: Country codes".

[55] ISO 639-2:1998: "Codes for the representation of names of languages - Part 2: Alpha-3 code".

[56] OIPF: "Release 2 Specification Volume 5a - Web Standards TV Profile".

NOTE: Available at http://www.oipf.tv/docs/OIPF-T1-R2-Specification-Volume5a-Web-Standards-TV-Profile-v2_3-2014-01-24.pdf.

[57] "JPEG File Interchange Format, Version 1.02", Eric Hamilton, C-Cube Microsystems, September 1, 1992.

NOTE: Available at http://www.w3.org/Graphics/JPEG/jfif3.pdf.

[58] "Graphics Interchange Format version 89a", (c)1987,1988,1989,1990, CompuServe Incorporated, Columbus, Ohio.

NOTE: Available at http://www.w3.org/Graphics/GIF/spec-gif89a.txt.

2.2 Informative references

References are either specific (identified by date of publication and/or edition number or version number) or non-specific. For specific references, only the cited version applies. For non-specific references, the latest version of the referenced document (including any amendments) applies.

NOTE: While any hyperlinks included in this clause were valid at the time of publication, ETSI cannot guarantee their long term validity.

The following referenced documents are not necessary for the application of the present document but they assist the user with regard to a particular subject area.

[i.1] ETSI TS 102 034 (V1.4.1): "Digital Video Broadcasting (DVB); Transport of MPEG-2 TS Based DVB Services over IP Based Networks".

[i.2] Richardson, S. Ruby: "RESTfull Web services", L. o'Reilly, 2007.

[i.3] Dash Industry Forum (2015): "Guidelines for Implementation: Dash-IF Interoperability Points version 3.0".

[i.4] Dash Industry Forum: "Identifiers for protection".

NOTE: Available at http://dashif.org/identifiers/protection/.

[i.5] CA Browser Forum: "Baseline Requirements: Certificate Policy for the Issuance and Management of Publicly-Trusted Certificates".

NOTE: Available at https://cabforum.org/.

[i.6] NIST SP 800-52 rev1 (April 2014): "Guidelines for the Selection, Configuration, and Use of Transport Layer Security (TLS) Implementations".

[i.7] CI Plus Specification V1.3.1 (2011-09).

NOTE: Available at http://www.ci-plus.com.

[i.8] DLNA Networked Device Interoperability Guidelines, Digital Living Network Alliance.

NOTE: Available at http://www.dlna.org/guidelines.

[i.9] Hybrid Broadcast Broadband Television (HbbTV®) Operator Applications.

[i.10] ETSI GR ECI 004: "Embedded Common Interface (ECI) for exchangeable CA/DRM solutions; Guidelines for the implementation of ECI".

[i.11] ETSI GS ECI 001-6: "Embedded Common Interface (ECI) for exchangeable CA/DRM solutions; Part 6: Trust Environment".

3 Definitions and abbreviations

3.1 Definitions

For the purposes of the present document, the following terms and definitions apply:

Advanced Security System (AS System): function of an **ECI** compliant **CPE**, which provides enhanced security functions (hardware and software) for an **ECI Client**

NOTE: The details are specified in ETSI GS ECI 001-5-1 [4].

application menu: main settings menu of an **ECI Client**, **an ECI Client** presents on request of the **ECI Host** to the **User**

AS slot: resources of the Advances Security block provided exclusively to an **ECI Client** by the **ECI Host**

AS slot session: resources and computing in an AS slot related to the de-cryption or re-encryption of a content element

broadcast-CPE-ID: special (compact) identification number issued to all **CPEs** that are capable of installing on broadcast-only networks, used to identify the **CPE** to a broadcast security system using manual means

NOTE: This ID does not serve any security purpose.

brother: other **Child** of the same **Father**

NOTE: **Father, Children, Brother** are referring to entities that manage **certificates**.

certificate: data with a complementary secure **Digital Signature** that identifies an **Entity**

NOTE: The holder of the secret key of the signature attests to the correctness of the data - authenticates it - by signing it with its secret key. Its public key can be used to verify the data.

certificate chain: list of **Certificates** that authenticate each other up to and including a Root Revocation List

Certificate Processing Subsystem (CPS): subsystem of the **ECI Host** that provides **certificate** verification processing and providing additional robustness against tampering

child, children: entity (entities) referred to by a **Certificate** signed by a (common) **Father**

NOTE: **Father, Children, Brother** are referring to entities that manage **Certificates:** initialization data and software that is used to start the SoC of a **CPE**.

Content Protection system: systems that employs cryptographic techniques to manage access to content and services

NOTE: The term may be interchanged frequently with the alternate Service Protection system. Typical systems of this sort are either Conditional Access Systems (CAS), or Digital Rights Management systems (DRM).

Customer Premises Equipment (CPE): media receiver which has implemented **ECI**, allowing the **User** to access digital media services

CPE manufacturer: company that manufactures **ECI** compliant **CPEs**

digital signature: data (byte sequence) that decrypted with the public key of the signatory of another piece of data can be used to verify the integrity of that other piece of data by making a digest (hash) of the other piece of data and comparing it to the decrypted data

ECI (Embedded CI): architecture and the system specified in the ETSI ISG "Embedded CI", which allows the development and implementation of software-based swappable **ECI Client**s in customer premises equipment (**CPE**) and thus provides interoperability of **CPE** devices with respect to **ECI**

ECI application: HTML based application hosted on an **ECI Client,** and running in a dedicated browser session for the purpose of interacting with the user and providing user input to the **ECI Client**

ECI Chip Manufacturer: company providing Systems on a Chip that implement **ECI** specified chipset functionality

ECI Client (Embedded CI Client): implementation of a CA/DRM client which is compliant with the Embedded CI specifications

NOTE: It is the software module in a **CPE** which provides all means to receive, in a protected manner, and to control execution of a consumer's entitlements and rights concerning the content that is distributed by a content distributor or **Operator**. It also receives the conditions under which a right or an entitlement can be used by the consumer, and the keys to decrypt the various messages and content.

ECI Client Image: file with software as VM code, and initialization data required by the **ECI Client Loader**

ECI Client Loader: software module part of the **ECI Host** which allows downloading, verifying and installing new

ECI Container: single VM instance with complementary support libraries and **ECI** API that permits a single instance of an **ECI Client** to run on a **CPE**

ECI Ecosystem: commercial operation consisting of a **TA** and several platforms and **ECI** compliant **CPEs** in the field

ECI Host: hardware and software system of a **CPE**, which covers **ECI** related functionalities and has interfaces to an **ECI Client**

NOTE: The **ECI Host** is one part of the **CPE** firmware.

ECI Host Image: file(s) with software and initialization data for an **ECI** environment

NOTE 1: An **ECI Host** image may consist of a number of **ECI Host Image** files.

NOTE 2: It may also contain other software that does not cause interference with or permit undesirable observation of the **ECI Host**.

ECI Host Loader: software module, which allows downloading, verifying and installing **ECI Host** software into a **CPE**

NOTE: In a multi-stage loading configuration this term is used to refer to all security critical loading functions involved in loading the **ECI Host**.

ECI Root Certificate: Certificate which issues to verify items approved by an **ECI TA**

ECI Trust Authority (TA): organization governing all rules and regulations that apply to implementations of **ECI** and manages the interoperability and coexistence of CA and DRM systems within the **ECI Ecosystem**

NOTE: The Trust Authority has to be a legal Entity to be able to achieve legal claims. The Trust Authority needs to be impartial to all players in the downloadable CA/DRM ecosystem.

entity: organization (e.g. manufacturer, **Operator** or **Security Vendor**) or real world item (e.g. **ECI Host**, **Platform Operation** or **ECI Client**) identified by an ID in a **Certificate**

export chain: chain of **Certificates** used for authorization of export to one or a group of **Micro DRM Systems**

export connection: authenticated relation between an **ECI Client** that can decrypt content and a **Micro Server** that can re-encrypt content

export group: Group of Micro DRM-Systems, to which export is permitted

father: signatory of the **Certificate** of the **Child Entity**

NOTE: **Father, Children, Brother** are referring to entities that manage **Certificates**.

image series: series of images for an **ECI Host** or an **ECI Client** that are different depending on the **CPE**_id of the **CPE**, nevertheless represent (nearly) identical functionality

import chain: chain from the POPK of an **ECI Client** to an Entity that represents an export system or an export group

NOTE: An Export Chain and a matching Import Chain can be used to authenticate a micro server session importing content to an exporting **ECI Client**.

import connection: approved connection from an **ECI Client** to a **Micro Server** that permits it to import decrypted content for subsequent re-encryption

manufacturer: entity which develops and sells CPEs, which accommodate an implementation of the ECI system and allow ECI Hosts and ECI Clients to be installed per software download

media handle: reference to a single program decryption or re-encryption processing setup between an **ECI Client** and an **ECI Host**

micro client: ECI Client or non-**ECI** client that can decrypt content which was re-encrypted by a Micro Server

micro DRM operator: party provisioning and maintaining a **Micro DRM System**

micro server: ECI Client that can import decrypted content, re-encrypt this content and authenticate a specific **ECI Client** or group of **ECI Clients** as the target for subsequent decryption

micro DRM system: Content Protection System that re-encrypts content on a **CPE** with a Micro Server and that permits decoding of that re-encrypted content by authenticated Micro Clients, Micro Server and Micro Clients being provisioned by a Micro DRM Operator

operator: organization that provides **Platform Operation**s that is enlisted with the **ECI TA** for signing the **ECI eco system**

NOTE: An **Operator** may operate multiple **Platform Operation**s.

Platform Operation (PO): specific instance of a technical service delivery operation having a single **ECI** identity with respect to security

PO client chain: Certificate Chain, starting with the operator root revocation-list and ending with the platform operation client co-signatures

re-encryption Session: process controlled by a **Micro Server** of importing content from an **Import Connection**, re-encrypting it and producing the decryption information necessary by the authenticated **Target** to subsequently decrypt it

request: message from a sender to a receiver asking for certain information or to perform certain operation, which is specified in the data fields of that request

NOTE: More details are given in clause 9.2.3.

response: message answering a **Request**

NOTE: More details are given in clause 9.2.3.

Revocation List (RL): list of **Certificates** that have been revoked and therefore should no longer be used

root: public key or **Certificate** containing a public key that serves as the basis for authenticating a chain of **Certificates**

root certificate: trusted **Certificate** that is the single origin of a chain of **Certificates**

screen conflict: situation where the **ECI Host** cannot provide access to the screen on behalf of a user interface session as defined in clause 9.4.3 of the present document

Secure Authenticated Channel (SAC): communication path (channel) that has been established between two entities where the entities have securely identified themselves to each other (authenticated) and agreed on an encryption of data transferred between them (secure)

security vendor: company providing **ECI** security systems including **ECI Clients** for **Operators** of **ECI Platform Operations**

service: content that is provided by a **Platform Operation**

NOTE: In the context of **ECI** only protected content is considered.

Sender Public Key (SPK): public key of the sender of the encrypted content used to verify the origin of the signature of the first key of a key chain used to decrypt the content, the sender being part of a platform operation

Smart Card: detachable hardware security device used by several CA or DRM providers to enhance the level of security of their products

target: Micro Client or a group of **Micro Client**s for which content is re-encrypted by a Micro Server

Trust Authority (TA): organization governing all rules and regulations that apply to a certain implementation of **ECI** and targeting at a certain market

NOTE: The Trust Authority has to be a legal entity to be able to achieve legal claims. The Trust Authority needs to be impartial to all players in the **ECI Ecosystem** it is governing.

Trusted Third Party (TTP): security services provider, which issues **certificate**s and keys to compliant **Manufacturer**s of the relevant components of an **ECI-System**

NOTE: It is under control of the **ECI Trust Authority (TA)**.

user: person who operates an **ECI** compliant device

VM Instance: instantiation of VM established by an **ECI Host** that appears to an **ECI Client** as an execution environment to run in

3.2 Abbreviations

For the purposes of the present document, the following abbreviations apply:

3DES	Triple-DES
4CC	Four Character Code (also FourCC)
AEAD	Authenticated Encryption with Associated Data
AES	Advanced Encryption Standard
AES-GCM	AES Galois Counter Mode
AID	Application IDentifier
AK	Authentification Key
APDU	Application Protocol Data Unit
API	Application Programming Interface
AS	Advanced Security
ASCII	American Standard Code for Information Interchange
ATR	Answer to Reset
BAT	Bouquet Association Table
BMFF	Base Media File Format
BSD	Berkeley Software Distribution
CA	Conditional Access
CA/DRM	Conditional Access/Digital Rights Management
CAT	Conditional Access Table
CBC	Cipher Block Chaining
CENC	Common Encryption
CI	Common Interface
CP	Content Property
CPE	Customer Premises Equipment
CPS	Certificate Processing Subsystem
CPU	Central Processing Unit
CRC	Cyclic Redundancy Check
CRL	Certificate Revocation List
CSA	Common Scrambling Algorithm
CSA1	Common Scrambling Algorithm, first version
CSA3	Common Scrambling Algorithm, third version
CSS	W3C Cascading Style Sheets
CSS3	CSS version 3
CTR	Counter Mode
CW	Control Word
Dash	Dynamic Adaptive Streaming over HTTP
DDB	Download Data Block
DDOS	Distributed Denial of Service
DES	Data Encryption Standard
DHE	Ephemeral Diffie-Hellman
DII	Download Info Indication

NOTE: As defined in ETSI TS 102 006 [14].

DLNA	Digital Living Network Alliance
DNS	Domain Name System
DRM	Digital Rights Management
DSI	Download Server Initiate

NOTE: DSI is defined in ETSI TS 102 006 [14].

DSMCC	Digital Storage Media Command and Control
DVB	Digital Video Broadcasting
EAC	Export Authorization Certificate
EAOC	Export Authorization Operator Certificate
ECM	Entitlement Control Message
EGC	Export Group Certificate
EIT	Event Information Table
EMM	Entitlement Management Message

ES	Elementary Stream
ESC	Export System Certificate
GCM	Galois/Counter Mode
GMT	Greenwich Mean Time
HD	High Definition
HDCP	High-bandwidth Digital Content Protection
HTML	Hyper Text Mark-up Language
HTTP	Hypertext Transfer Protocol
HTTP(S)	Hypertext Transfer Protocol Secure
iDTV	integrated Digital TV receiver
IFSC	Information Field Size of Card
IFSD	Information Field Size of Device
IP	Internet Protocol
IPTV	TV using the Internet Protocol (IP)
IPv4	Internet Protocol version 4
IPv6	Internet Protocol version 6
ISO	International Organization for Standardisation
ISOBMFF	ISO Base Media File Format
LAN	Local Area Network
LSB	Least Significant Bit
MIME	Multipurpose Internet Mail Extensions
MMI	Man Machine Interface
MP4	Digital Multimedia Container Format (asl called MPEG-4 part 14)
MPD	Media Presentation Description
MPEG	Motion Picture Experts Group
MSB	Most Significant Bit

NOTE: In **ECI** the MSB is always transmitted first.

n.a.	not applicable
NV memory	Non-Volatile memory
NV	Non-Volatile
OS	Operating System
OTT	Over The Top (over the open Internet)
OUI	Organizationnally Unique Identifier
PAT	Program Association Table
PayTV	Pay Television
PES	Packet Elementary Stream
PID	MPEG Packet Identifier
PIN	Personal Identification Number
PKIX	Public-Key Infrastructure X.509
PMT	Program Map Table
PO	Platform Operation
POC	Platform Operation Certificate
POPK	Platform Operation Public Key
PPS	Protocol and Parameter Selection
PSI	Program Specific Information
PSSH	Protection System Specific Header
PVR	Personal Video Recorder
RAM	Random Access Memory
RFU	Reserved for Future Use

NOTE: Bits currently not defined have to be set to '0'.

RL	Revocation List
SAC	Secure Authenticated Channel
SDT	Service Description Table
SHA	Secure Hash Algorithm
SI	Service Information
SIM	Subscriber Identity Module
SoC	System on Chip
SPK	Signature Public Key (also known as Signature Verification Key)
SSK	Signature Secret Key (also known as Signature Private Key)

SSL	Secure Sockets Layer
SSU	System Software Update
STB	Set Top Box
TA	Trust Authority
TCK	The ChecK byte
TCP	Transmission Control Protocol
TLS	Transport Layer Security
TPC	Transmission Control Protocol
TPDU	Transport Protocol Data Unit
TPEGC	Third Party Export Group Certificate
TS	Transport Stream

NOTE: Transport Stream is defined in ISO/IEC 13818-1-1 [8].

TTP	Trusted Third Party
TV	TeleVision
UDP	User Datagram Protocol
UHD	Ultra High Definition
UI	User Interface
uimsbf	unsigned integer, most significant bit first
UNT	Update Notification Table
URI	Usage Rights Information
URL	Uniform Resource Locator
USB	Universal Serial Bus
UTF	UCS (Universal Character Set) Transformation Format
UUID	Universally Unique Identifier
VM	Virtual Machine
WAN	Wide Area Network
WEB	World Wide Web

4 Conceptual principles

Reserved for future use.

5 ECI Certificate System

5.1 Introduction

5.1.1 Scope

ECI uses **Certificates** for various purposes, like **ECI Host Loader**, **ECI Client Loader** and **ECI Operator Certificates**. The definition of these **Certificates** and associated **Revocation List**, their composition into chains and the **Root Certificate** structure is defined in this clause. The definition uses a compact binary format which is specified in the present document amenable for hardware implementation and suitable cryptography, and a simple signalling system for future versions and extensions.

5.1.2 Notation and conventions of fields

The data structure definitions below map directly onto a sequence of bytes. Any cryptography function is defined to operate on the byte sequence representation.

The data definition follows a natural alignment for 16 and 32 byte fields to simplify data processing on 32-bit CPU core. Padding is used as a generic field to indicate the required stuffing fields for this purpose. This uses the function padding(n_bytes) with n_bytes being the alignment boundary in number of bytes from the start of the defined data structure. Padding fields shall be skipped on interpretation of data structures. The value of padding field shall be set to 0.

Any field defined by another data structure through a type definition has no mnemonic. In general no field length definition is given for such a field.

5.1.3 Extension Field

Many of the more substantial data structures defined have an extension field that permits future (backward compatible) extensions to be added. The definition is in Table 5.1.3-1.

Table 5.1.3-1: Extension field definition

Syntax	No. of bits	Mnemonic
`Extension_Field {`		
`  padding(4)`		
`  length`	32	uimsbf
`  for (i=0; i<length; i++) {`		
`    extension_byte`	8	uimsbf
`  }`		
`}`		

Semantics:

length: integer	Number of bytes in the loop following. The value should be a multiple of 4, and may be 0.
extension_byte: byte	Data field with information that may be ignored by implementations based on versions of the present document that did not define the content of this field.

5.2 ECI Certificates

The **ECI Certificate** has a straightforward structure. The ID of the **Certificate** is simply a binary number intended only for machine interpretations unlike X.509 certificates used on the internet.

The generic layout of a **Certificate** is shown in Figure 5.2-1

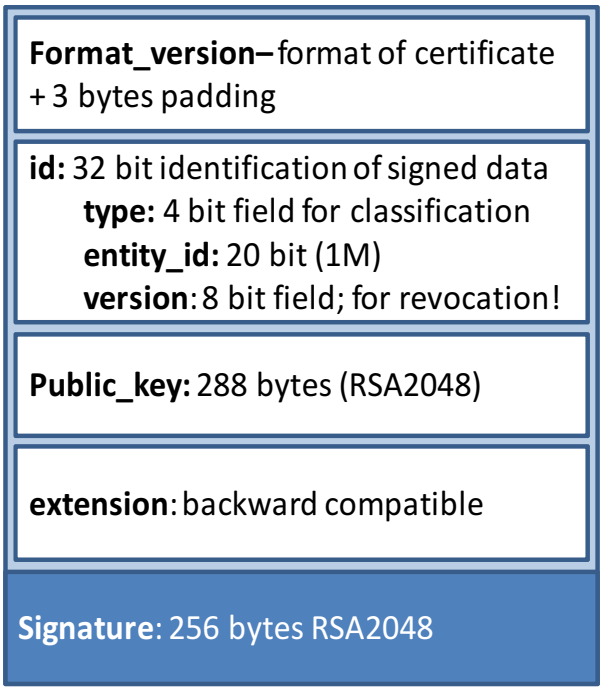


Figure 5.2-1: ECI Certificate format version 1

The **ECI Certificate** format is defined in Table 5.2-1.

Any signed item shall use a distinct 8-byte start field, the first byte being the version format of the item signed, then (for version 1 items) 3 bytes of padding, followed by the second 4 bytes representing a unique ID in the context of the signing Entity's secret key.

Table 5.2-1: ECI Certificate definition

Syntax	No. of bits	Mnemonic
ECI_Certificate_Id {		
padding(4)		
Type	4	Uimsbf
entity_id	20	Uimsbf
Version	8	Uimsbf
}		
ECI_Public_Key_v1 {		
byte **modulus**[256]	2 048	
}		
ECI_Certificate_Data_v1 {		
ECI_Certificate_Id **id**	32	Uimsbf
Public_Key_v1 **public_key**	2 304	
Extension_Field **extension**		
}		
ECI_Signature_v1 {		
byte **signature**[256]	2 048	Uimsbf
}		
ECI_Certificate {		
format_version	8	Uimsbf
if (version == 0x01) {		
ECI_Certificate_Data_v1 **data**		
ECI_Signature_v1 **signature**		
}		
}		

Semantics:

format_version: integer	Values 0x00, 0x02..0xFF: reserved. Value 0x01: **ECI Certificate** format version 1. Implementations that do not recognize a **Certificate** type shall not process it and return fail on validation requests.
id: integer	Certificate Identification in the form of a 32 bit number which is unique in the context of the **Certificate Father** (signatory of the **Certificate**). Values 0x00000 and 0xF0000-0xFFFFF are reserved.
type: integer	**type** defines the type of the Entity, like **Manufacturer**, **ECI Host**, **Operator**, etc. in the context of the signatory (**Father**). **Certificates** with a type value 0x0 .. 0x7 shall require a **Revocation List** for verification of children. Type values of 0x8 and above shall not require a **Revocation List** for verification of children (see Table 5.2-2).
entity_id: integer	Defines the number of the **Entity**. **entity_id** carries various sub formats as defined per **Certificate** type. Unless otherwise defined, entity_id's are unique in the context of the **Father** \(signatory of the **Certificate** or **Revocation List**).
version	Version number of the entities **Certificate**, assigned in ascending order (typically incrementing by 1).
extension: Extension_Field	Data in this field shall be ignored by processing functions not defined to interpret this. This field may be used for specific data in specific application of the generic **Certificate** definition. Its interpretation is context dependent. This field shall not be used for non **ECI** applications unless it is explicitly specified to be permitted.
public_key: ECI_Public_Key_v1	Public key (assigned by **Father**) of the Entity of this **Certificate**.
data: ECI_Certificate_Data	This is the data section of the **Certificate**.
signature: byte[256]	The signature field contains the byte sequence representation of the signature of the **Father** of the **Certificate**, using the cryptographical functions as defined in annex A.

Any verification of an **ECI Certificate** shall include the verification of the total length of the **Certificate** in terms of the accumulation of the field definitions.

Generic type values are used for most **Certificates** and **Revocation Lists**, so as to assure that all assigned values are unique. Table 5.2-2 presents the overview of all **ECI TA** signed data.

Table 5.2-2: ID assignment and Fathers for signed items

Father	Type	ID field	Description
Root	0x0	0xFFFFF	Root
Root	0x1	Manufacturer id,<> 0xFxxxx	Manufacturer **Certificate**
Root	0x1	Manufacturer RL id, == 0xFxxxx	Manufacturer **Revocation List**
Manufacturer	0x0	Host_id, <> 0xFxxxx	**ECI Host Certificate**
Manufacturer	0x0	Host RL, == 0xFxxxx	**ECI Host Revocation List**
Host	0x8	Host Image id	**ECI Host Image**
Host	0x9	Host Image Series id	**ECI Host Image Series Certificate**
Host Image Series	0x9	Image Target Id	**ECI Host** series image
Root	0x2	Vendor id, <> 0xFxxxx	**Security Vendor Certificate**
Root	0x2	Vendor RL id, == 0xFxxxx	**Security Vendor Revocation List**
Vendor	0x0	Client id, <> 0xFxxxx	**ECI Client Certificate**
Vendor	0x0	Client RL, == 0xFxxxx	**ECI Client** and **ECI Client** series **Revocation List**
Client	0x0	Client id	**ECI Client Image**
Client	0x1	Client series id	Client series **Certificate**
Client series	0x8	Image Target Id	Client series image
Root	0x3	Operator id, <> 0xFxxxx	Operator **Certificate**
Root	0x3	Operator RL id, == 0xFxxxx	Operator **Revocation List**

Father	Type	ID field	Description
Operator	0x0	Platform Operation id, <> 0xFxxxx	**Platform Operation Certificate**
Operator	0x0	Platform Operation RL id, == 0xFxxxx	**Platform Operation** RL
Platform Operation	0x0	Platform Operation Client Image Cosignature id <> 0xFxxxx	**Platform Operation** Client image cosignature
Platform Operation	0x0	Platform Operation Client Image RL id == 0xFxxxx	**Platform Operation** Client **Image Revocation List**
Platform Operation or Target Group	0x0	Target Group id, <> 0xFxxxx	Target Group, defined in ETSI GS ECI 001-5-1 [4]
Platform Operation or Target Group	0x0	Target RL id, == 0xFxxxx	Target Revocation list, defined in ETSI GS ECI 001-5-1 [4]
Platform Operation or Target Group	0x8	Micro Client id, <> 0xFxxxx	Micro Client, defined in ETSI GS ECI 001-5-1 [4]
Platform Operation, **Export Group**, Third Party Export Group	0x4	Export Group id, <> 0xFxxxx	**Export Group**
Platform Operation, **Export Group**, Third Party Export Group	0x4	Export Group RL id, ==0xFxxxx	**Export Group** Revocation List
Export Group	0x5	Third Party Export Group id, <> 0xFxxxx	Third Party **Export Group**
Export Group	0x8	Export Group RL id, == 0xFxxxx	**Export Group** Revocation List
Export Group, Third Party Export Group	0xE	Export System id, <> 0xFxxxx	Export System
Root	0x4	Export Authorization Operator id, <> 0xFxxxx	Export Authorization Operator
Root	0x4	Export Authorization Operator id, == 0xFxxxx	Export Authorization Operator Revocation List
Export Authorization Operator, Export Authorization	0x0	Export Authorization id, <> 0xFxxxx	Export Authorization (with children)
Export Authorization Operator, Export Authorization	0x0	Export Authorization id, == 0xFxxxx	Export Authorization Revocation List
Others	Others		Reserved
NOTE: ECI functions may transport and process **data** field and **signature** sections of a **Certificate** or another signed data item separately.			

5.3 ECI Revocation List

A **Revocation List** shall be signed by the same **Entity** that originally signed the **Certificate** being revoked. The **Revocation List** is a list of identifiers of entities defining the minimum acceptable version for their **Certificates**. In case an entry in a revocation list is a **Certificate** that has an associated revocation list(s) there is a minimum version number for the **Revocation List** to be applied with that **Certificate**. The layout of an **ECI Revocation List** is defined in Figure 5.3-1.

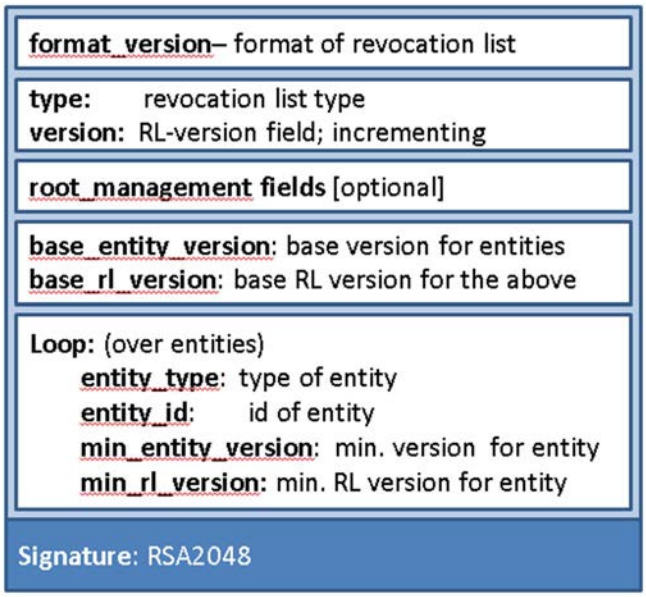


Figure 5.3-1: Revocation List structure

ECI Host implementations shall store the latest (is defined in **rl_version**) received **Revocation List** they receive for an **Entity** that they manage, regardless of the source of the data.

The **Revocation List** (ECI_RL) is defined in Table 5.3-1.

Table 5.3-1: Revocation List definition

Syntax	No. of bits	Mnemonic
ECI_RL_Id {		
padding(4)		
Type	4	Uimsbf
indicator = 0xF	4	Uimsbf
version	24	Uimsbf
}		
ECI_Revocation_List_v1 {		
base_entity_version	8	Uimsbf
base_rl_version	24	Uimsbf
number_of_entities	24	Uimsbf
for (i=0; i<number_of_entities; i++){		
entity_type	4	Uimsbf
entity_id	20	Uimsbf
min_entity_version	8	Uimsbf
min_rl_version	24	Uimsbf
}		
}		
ECI_RL {		
format_version	8	Uimsbf
if (format_version == 0x01){		
ECI_RL_Id **rl_id**	32+24	Uimsbf
root_version_indicator	1	Uimsbf
padding(1)	7	Uimsbf
root_version	8	Uimsbf
min_root_version	8	Uimsbf
padding(4)		
ECI_Revocation_List_v1 **rev_list**		
Extension_Field **extension**		
ECI_Signature_v1 **rl_signature**	2 048 (see note)	Uimsbf
}		
}		
NOTE: .. = in version 1 **Certificate** associated CRLs.		

Semantics:

format_version: integer	Values 0x00, 0x02..0xFF: reserved. Value 0x01: ECI **Revocation List** format version 1. Implementations that do not recognize a **Certificate** type shall not process it and return fail on validation requests.
type: integer	Type field is defined in ECI_Certificate_Id, see Table 5.3-1.
indicator: integer	Indication of **Revocation List**; value shall be equal to 0xF.
version: integer	The version of this RL. Starts at 1 (which is typically empty with a new **Certificate**) and incremented on every update.
base_entity_version: integer	All entities with a **id.version** smaller than **base_id_version** are revoked.
base_rl|_version	All revocation lists for an entity with version equal **base_entity_version** that are smaller than **base_rl_version** are no longer valid.
number_of_entities: integer	Number of entities in the revocation list. See Table 5.3-1. Below for maximum values.
entity_type: integer	Type of entity of which older versions are revoked.
entity_id: integer	Entity_id of the entity of which older versions are revoked.
min_entity_version: integer	Minimum version number of the entity (certificate id) matching **entity_type** and **entity_id**. Lower versions are revoked.
min_rl_version	Minimum version of the revocation list to be applied in conjunction with entity matching **entity_type**, **entity_id** and **entity_min_version**. Lower revocation list versions are no longer valid.
root_version_indicator: bit	If value equal 0 the root_version and min_root_version field shall have no significance. If value equal 1 and the **Father** are a **Root Certificate** the **root_version** and **min_root_version** fields shall have the interpretation as below.
root_version	Version of the **Root Certificate** that is the signatory of this **Revocation List**.
min_root_version: integer	If the **Father**'s (i.e. Root) version is larger or equal than this field all **Root Certificate** versions less than **min_root_version** shall be revoked for verification of **Certificates** of the type is defined in **revocation_id_lead**.
extension: Extension_Field	Additional data: shall be ignored (except for signature calculation) by implementations not designed to interpret this field, except for computation of the signature.
rl_signature: ECI_Signature_v1	The signature of the **ECI Entity** to which the **Revocation List** is associated. The signature is computed over all preceding data.

NOTE: Hardware implementations can process **Revocation Lists** in chunks, looking for an ID of a next **Certificate** that should be validated while accumulating the signature hash and at reaching the end of the list verifying the signature.

As a general rule **ECI Hosts** shall store the **TA Revocation Lists** of all **Certificates** required to verify the entities that are loaded by the **ECI Host**. **ECI Hosts** shall replace a stored **Revocation List** for a **Certificate** or item by a newly received **Revocation List** with a later version number.

The maximum length of the **Revocation Lists** shall be in accordance with clause B.2.

5.4 Certificate Chains and Revocation List Trees

5.4.1 Data structure definitions

A **Certificate Chain** is a sequence of **Certificates** with associated **Revocation Lists** where a **Certificate** has been signed by the entity managing the preceding **Certificate**. It starts with the **Revocation List** of the **Father Certificate** (typically a Root). The minimum (valid) version number of a **Certificate** and the minimum (valid) **Revocation List** version for a **Child** are defined by the **Revocation List** of its **Father**. Chains are used as credentials to verify an item to be loaded, and thus a **Certificate** will typically not appear on the **Revocation List** of its predecessor. Nevertheless **Revocation List** processing is mandatory so as to verify the integrity of the chain. Table 5.4.1-1 presents the structure of a typical **Certificate Chain**.

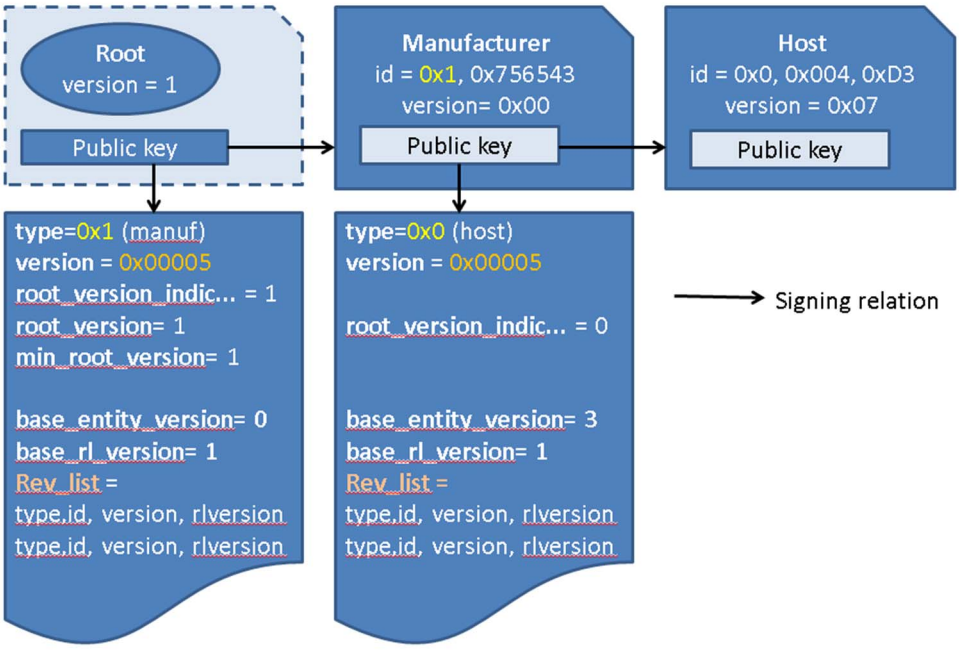


Figure 5.4.1-1: Host Certificate Chain example

Chains can be transported or stored and be composed from different sections.

Revocation list trees are sequences of linked revocation lists that use a **Certificate** in a previous chain as **Father**, thus spanning a large space of certified items. These can be used by **Platform Operation**s to deprecate (indicate the revocation of) other (**TA**-revoked) entities. The definition of **Certificate Chain** and **Revocation List** tree shall be in accordance with Table 5.4.1-1.

Table 5.4.1-1: Certificate Chain and Revocation List Tree definitions

Syntax	No. of bits	Mnemonic
ECI_Certificate_Chain {		
chain_length	8	uimsbf
padding(4)		
for (i=0; i<**chain_length**; i++){		
ECI_RL **rl**		
ECI_Certificate **certificate**		
}		
}		
ECI_RL_Tree {		
ECI_RL **father_revocation_list**		
three_breadth	32	uimsbf
for (i=0; i<**three_breadth**; i++){		
father_node_depth	8	uimsbf
chain_length	8	uimsbf
padding(4)	16	uimsbf
for (i=0; i<**chain_length-1**; i++){		
ECI_Certificate **certificate**		
ECI_RL **rl**		
}		
}		
}		

Semantics:

chain_length: integer	Length of the chain.
rl: ECI_RL	The **Revocation List** for the preceding **Certificate** or **Father** of the chain in case of the first iteration of a chain. The version numbers of the identifier field of the **Revocation Lists** in a chain shall be equal.
certificate: ECI_Certificate	**Father** of next **Certificate** in current sequence.
father_revocation_list: ECI_RL	**Revocation List** for the **Father** of the chain.
three_breadth: integer	Number of sub-chains in the tree.
father_node_depth: integer	Level of the **Father Certificate** in the preceding **Certificate Chain** (including the **Father** of the tree). The inherited **Father** list is the **Father** of this chain, preceded by its **Father**, etc. up to the **Father** of the tree itself.

The ordering rules on **Certificates** in **Revocation List** trees are:

- Trees shall not contain duplicate **Certificates**.
- The tree shall be ordered in a depth first manner: i.e. all **Brothers** of the last leaf **Certificate** shall be listed as chain_length=0 sub-trees immediately after the last **Certificate**, then followed by the sub trees of the **Brother** of the **Father**, etc.
- Brother **Certificates** shall appear in id-order in the tree (lowest first).

5.4.2 Processing rules for Certificate Chains

The **ECI Host** performs the verification of the **Certificate Chains** and provides an appropriate response for revoked items using the **Advanced Security System**. The security critical steps of **Certificate** and **Revocation List** verification are performed by the secure **Advanced Security System**. The **Advanced Security System** also provides **ECI Clients** with the ability to subsequently verify the validity of the applied revocation version numbers of the chains.

The **ECI Host** can process a **Certificate Chain** in an iterative process. This starts with the **ECI TA** Root **Revocation List** and ends with the end item in a chain. The **Certificate Chain** processing fails on any intermediate check failure. In case the **ECI Host** fails on a condition it shall ensure that the present Certificate and **Revocation List** and all preceding **Revocation Lists** and **Certificates** are validated by their signature before triggering the **ECI Host** policy measures on revoked entities or invalid credentials. The **Advanced Security system, as defined in** ETSI GS ECI 001-5-1 **[4]** and ETSI GS ECI 001-5-2 [5] shall ensure appropriate robustness is maintained for **Certificate Chain** processing.

Any order of processing is permitted as long as it yields the same outcome regarding the acceptance of chains.

1) The **ECI Host** shall perform the following verification steps on the **Revocation Lists**:

   a) The **ECI Host** shall verify the **Revocation List format_version** field matches a version that it can interpret and the **rl_id.type** and **rl_id.rl_indicator** field to match the expected values.

   b) The **ECI Host** shall verify whether the length of the **Revocation List** corresponds to its field values.

   c) If **root_version_indicator**=1 the **ECI Host** shall check if a Root is expected as **Father** at this point in the chain processing, check if the **root_version** is present for verification and check if the **min_root_version** does not exceed any root version used so far in chain processing.

   d) The **ECI Host** shall verify if this **Revocation List** has not been invalidated by the minimum version number for this **Revocation List** from the preceding **Revocation List** in the chain or in case of a root revocation list by the min_root_revocation_list number used so far in chain processing.

   e) The **ECI Host** shall verify the signature of the **Revocation List** with the public key of the **Father Certificate**.

   f) The **ECI Host** shall process any extension bytes in the **Revocation List** if capable of doing so.

   g) The **ECI Host** shall verify if the ***next*** <entity type, entity id, version> in the chain is not revoked according to the **Revocation List** and establish the minimum Revocation List version to apply to that **Certificate**.

2) The **ECI Host** shall perform the following pre-verification steps on the next **Certificate**:

   a) The **ECI Host** shall verify the version of the **Certificate**. In case the version is not matching its processing capabilities it shall fail loading the chain.

   b) The **ECI Host** shall verify the type field of the certificate ID and fail if this does not match an expected values.

   c) The **ECI Host** shall verify that the length of the **Certificate** matches its format definition.

   d) The **ECI Host** shall verify the signature of the **Certificate** with the public key of the **Father Certificate**.

   e) The **ECI Host** shall process any additional field and/or extension bytes in the **Certificate** if capable of doing so.

A **Revocation List** chain, as extracted from a **Revocation List**tree can be used to verify the revocation of a specific item that needs to be loaded by the **Advanced Security System**. Such an item can be identified by the sequence of ids of **Certificates** used to verify it on loading it in the **Advanced Security System**. The default processing rules for a **Revocation List** chain shall be identical to that of a **Certificate Chain**.

3) The **CPS** shall load the current **Revocation List** and the <entity type, entity id, version> of the next **Certificate** in the CPS. The CPS shall perform the following verification:

   a) The CPS shall check the **Revocation List format_version** field to match a version that it can interpret and the **rl_id.type** and **rl_id.rl_indicator** field to match the expected values.

   b) In case the **Father** is a **Root Certificate** (**root_version_indicator**=1) the CPS shall select the **Root Certificate** with **root_version** to be the **Father**, otherwise the preloaded or preceding **Certificate** is used.

   c) The CPS shall verify the signature of the **Revocation List** with the public key of the **Father Certificate**.

   d) The CPS shall verify whether the length of the **Revocation List** corresponds to its field values.

   e) The CPS shall verify if the version number of the **Revocation List** has not been invalidated.

   f) The CPS shall verify if the ***next*** <entity type, entity id, and version> in the chain is not revoked according to the **Revocation List** and shall establish the minimum Revocation List version to accompany that **Certificate**.

4) Then the **ECI Host** shall load the **Certificate** in the appropriate CPS processing location, which will perform the following verifications:

   a) The CPS shall check the **Revocation List format_version** field to match a version that it can interpret and the **id.type** and **id.entity_id** field to match the expected values.

   b) The CPS shall verify whether the length of the **Certificate** corresponds to its field values.

   c) The CPS shall perform signature verification against the Public Key of the **Father Certificate**.

5.5 Revocation tree sets and revocation data files

Revocation data for verifying a specific **Entity** should select revocation data containing the RL of the **Father** of the targeted **Entity**.

When distributing revocation data, the chains to revoke multiple targeted entities can be combined into a tree, thus avoiding duplicate Root and **Child Certificates** and their associated **Revocation Lists** and permitting more ordered searching in **CPEs**.

For ease of assembly and disassembly of revocation data revocation trees can also simply be combined into a set of trees. However, sets of trees shall be non-overlapping except for the common **Father** revocation list. Tree sets can contain multiple Root RLs (during an ongoing **TA** Root change rollout).

The definition of **Certificate Chain** and **Revocation List** tree shall be in accordance with Table 5.5-1.

Table 5.5-1: Revocation List Tree set definition

Syntax	No. of bits	Mnemonic
ECI_RL_Tree_Set {		
tree_number	32	uimsbf
for (i=0; i<tree_number; i++) {		
ECI_RL_Tree **tree**		
}	8	uimsbf
}		

Semantics:

tree_number: integer	Number of trees in the set
tree: ECI_RL_Tree	Tree (including **Root Certificate**) of **Certificates** ad their **Revocation Lists**.

NOTE: Online servers can distribute single **Entity** targeted trees (effectively chains) to minimize data traffic. On broadcast networks trees can be split and merged easily to match the number of buckets (see clause 7.7.2) used in the transmission carousel.

Revocation trees or set of trees do not need to be complete in terms of containing all entities of a class. It is up to the **Platform Operation** to compose the set of revocation trees as he sees fit, ensuring minimal risk in deployed **CPEs** in the network of the **Platform Operation**. On broadcast networks **Revocation Lists** can also be alternated in time to expand the revocation coverage.

ECI requires **CPEs** to permanently store **ECI TA** chains for all items to be potentially loaded in order to ensure once revoked entities remain revoked. This is specified in the relevant clauses.

In order for convenience of transport **ECI** revocation tree sets are carried in the format given in Table 5.5-2.

Table 5.5-2: Revocation data file

Syntax	No. of bits	Mnemonic
ECI_revocation_data_file {		
magic = 'ERD'	24	uimsbf
version	8	uimsbf
father_type	4	uimsbf
sub_type	4	uimsbf
ECI_RL_Tree_Set **revocation_data**		
}		

Semantics:

magic: byte[3]	Magic number used for verification of the format of the following data. It has the value of the three 8-bit ASCII representations of the characters 'ERD'. The **ECI Host** shall check the value of this field to verify if an **ECI** file has the expected format for additional data integrity.
version: byte	Format version of the image header. Value 0x01 is the presently defined version; all other values are reserved. The **ECI Host** shall ignore any image with a version number that is not recognized.
father_type: integer	Type of the common **Father** of the **Revocation List** data. 0x0 indicated the **ECI Root Certificate**. Values 0x1-0x7 are reserved. Values 0x8-0xF may be applied for private applications.
sub_type: integer	For father_type field equal 0x0 this defines the type of the common **Revocation List** in accordance with Table 5.2-2 of the **ECI Root Certificate** of the data contained in the revocation data. This value is undefined for other values of father_type.
revocation_data: ECI_RL_Tree_Set	Revocation List Tree set of revocation lists for revoked items.

5.6 Large data item signatures

ECI computes signatures over large data items (e.g. software images) using efficient hash function for bulk data hashing in conjunction with a regular signature operation. In Table 5.6-1 the signature of large data elements is defined.

Table 5.6-1: Definition of the signature of large data elements

Syntax	No. of bits	Mnemonic
ECI_Data_Signature {		
sign_version	8	uimsbf
padding(4)	24	uimsbf
if (sign_version == 0x01){		
for (i=0; i<256; i++){		
signature_byte	8	uimsbf
}		
}		
}		

Semantics:

sign_version: integer	Version of signature. Value 0x01 is the present version; all other version values are reserved. **CPEs** that have not implemented a version shall ignore this field (and any following data).
signature_byte: byte	Sequence of bytes representing the large item signature.

The signing algorithm is defined in annex A.

5.7 Root Certificates

5.7.1 Definition of a Root Certificate

ECI uses a sequence of **Root Certificate *versions***. The **ECI TA** can start using a new **Root Certificate** version for instance when any of the previous **Revocation Lists** for any of the **Children** is too large or if the secret key associated with the **Certificate**'s public key is no longer considered sufficiently secret.

A **Root Certificate** uses the identifier field of **ECI Certificates** with the field definition as given in Table 5.7-1. The type and identifier fields are never used; only the version field is applied.

Table 5.7-1: Definition of the ECI Root_ID field

Syntax	No. of bits	Mnemonic
ECI_Root_Id {		
type /* see Table 5.2-1***/**	4	uimsbf
id /* see Table 5.2-2 ***/**	20	uimsbf
version	8	uimsbf
}		

Semantics:

version: integer	The version number of the **Certificate**; numbering starts at 1, and is incremented by one on every new issue of a **Root Certificate**. Value 0x00 is reserved.

5.7.2 ECI Host Root Certificate Management

The **ECI TA** can start using a new **Root Certificate** with a higher version number. It can at some point in time thereafter issue a **Revocation List** for the new **Root Certificate** that revokes preceding **Root Certificates**. This invalidates all **Certificates** signed by such a Root.

Alternatively the **ECI TA** can decide that a **Revocation List** for specific type of entities (e.g. **Manufacturers**) is too large, decide to reissue new versions of all previously issued **Certificates** by using a higher **min_id_version** field in the **Revocation List** for that type of **Entity**. This effectively invalidates all previously issued **Certificates** for the type of **Entity** up to **min_entity_version**-1. Typically this requires issuing significant new **Certificates** with a higher version number for entities still using a lower **Certificate** version to replace now revoked **Certificates**.

The resources an **ECI Host** shall provide for storage of **Root Certificates** are defined in ETSI GR ECI 004 [i.10].

6 ECI Host Loader

6.1 Introduction

The **ECI Host** loading process distinguishes the following aspects:

1) Storage of an image, verification of the image authenticity of the image by the **CPE** using **ECI TA** provided authentication data and the subsequent activation of the image.

2) The file format of the file(s) containing the image and all other information needed to load the image into the **CPE**.

3) Transport protocol for delivering the **ECI Host Image** to the **CPE**. This includes any discovery by the **CPE** of the location of the required images. It includes any storage of the transported images and complementary **ECI** validation chain and signature data.

4) Any **Operator** specific revocation of **ECI Host Images**; the data format for such information is defined in clause 6; the application is defined in clause 8.

The logic of the verification and image authentication shall be applied on newly downloaded **ECI Host Images** and authentication data, on every reboot of a **CPE** and where so provisioned during the regular functioning of a **CPE**.

6.2 Storage, verification and activation

6.2.1 Principles of Operation

The **ECI Host** ensures that **ECI Clients** can run in a private and tamper free environment in accordance with **ECI** robustness requirements for the implementation of such clients. The **ECI Host** also prevents the interference of one **ECI Client** with another **ECI Client**. For this purpose the **ECI TA** can certify software for **CPEs** and the **CPE** loader shall verify the authenticity of the software images it loads.

Many **CPEs** use multi-stage loaders. **ECI** assumes that the core **CPE** chip loads a number of chip specific initialization images before commencing loading any regular software images. Such images may be implicitly certified under the **ECI TA** chip vendor license agreement. Alternatively they may be made part of the **Manufacturer** certification process defined in this clause.

In case the software of one of the **ECI** managed images later demonstrates to have a security bug the **ECI TA** and **CPE Manufacturer** can revoke it and have it replaced by a version with a bug fix.

In Figure 6.2.1-1 it is assumed that **Img1** is a chip-specific image needed to bring the chip into a state where it can start loading more regular application images. It is protected by a chip specific signature **CS1**, which is verified by the **Chip Loader** using a chip vendor proprietary key.

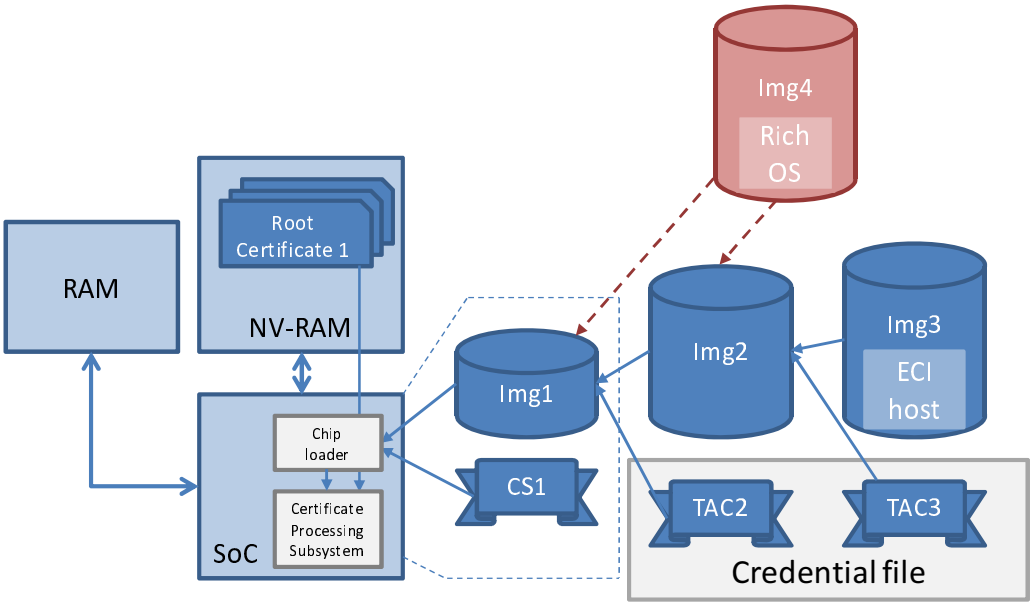


Figure 6.2.1-1: Example of ECI Host Loading Process

Once **Img1** is running the chip proceeds to load other images. It loads **Img2**, which can be authenticated by a **Certificate Chain** and image signature **TAC2**. The verification of the image is performed using the **TA Root Certificate**, the **Certificate Processing Subsystem** and **TAC2**. **Img2** proceeds to load **Img3** which contains the **ECI Host** software. **Img3** is verified by the **Certificate Processing Subsystem**, the **Root Certificates** and Trust Authority **Certificate Chain** and image signature **TAC3**. Additional images like **Img4** containing for instance a rich OS may be loaded that are not certified by the **ECI TA** if the loading environment can guarantee this does not create a security hazard for the **ECI Host**.

The Trust Authority Credentials for the images are carried in a special credential file.

The **ECI TA** certifies the security integrity of the **ECI Host**: its ability to provide client privacy, tamper resistance from threats outside the **ECI Host** and to ensure clients create undesirable interference interfere with each other. **CPE Manufacturer**s may wish to use complementary security measures for loading images using their proprietary image encryption and authentication.

Platform operations can verify the freshness of the **ECI Host Images** and decide not to decrypt services. For this purpose the CPS extracts the minimum **Revocation List** version number used to verify any loaded item, thereby permitting **Platform Operation**s to verify the application of a recent **Revocation List**. These **Platform Operation** specific acceptance procedures for an **ECI Host** are defined in clause 8.

The **ECI Host Loader** shall store the latest **ECI Host Images** and the latest credentials thereof in NV-RAM. The **ECI Host Loader** will re-verify every image it loads on reboot of the **ECI Host**. This procedure re-establishes **ECI Host** authenticity on every reboot.

6.2.2 Credential definition

6.2.2.1 ECI Host Image related Certificates

ECI provides for two types of **ECI CPEs** with respect to **ECI Host Image** diversity:

1) Generic **CPEs** that will load the same set of **ECI Host Images** on every instance of the same **CPE** type and version.

2) Individualized **CPEs** that will load a (partially) different set of images on each **CPE** of the same **CPE** type and version. Such a series of images of the same "type" but individualized per **CPE** is denoted as an **Image Series**.

The **ECI Host Certificate Chain** consists of the following **Certificates** (each certified by its predecessor):

1) Root Certificate:

   - This is the representation of the central **ECI TA** Root **Entity**. The public key of this **Certificate** shall be used to verify.

2) Manufacturer Certificate:

   - This is a representation of the **ECI TA Entity** for a specific **Manufacturer**. The public key of this **Certificate** shall be used to verify.

3) Host Certificate:

   - This is a representation of an **ECI TA** certified **CPE** hardware and **ECI Host** software release. For generic **ECI Hosts** the public key of this **Certificate** shall be used to authenticate all **ECI Host Images**. For "individualized" **ECI Host Images** the public key of this **Certificate** shall be used to verify.

4) Host Image Series Certificate:

   - This **Entity** provides a generic approval for a series of images that are specific for a particular **CPE** configuration, but are otherwise identical from an **ECI TA** perspective. For individualized **ECI Hosts** the public key of this **Certificate** shall be used to authenticate the **ECI Host Image** intended for the specific **CPE** with a **CPE** ID matching the identifier in the **Certificate**.

NOTE: Each Entity identifier has to be interpreted in the context of the authorizing Entity; i.e. IDs are relative.

The **ECI Host Image** and accompanying certification structure is sketched in Figure 6.2.2.1-1 and Table 6.2.2.1-1 gives an overview of the related parameters.

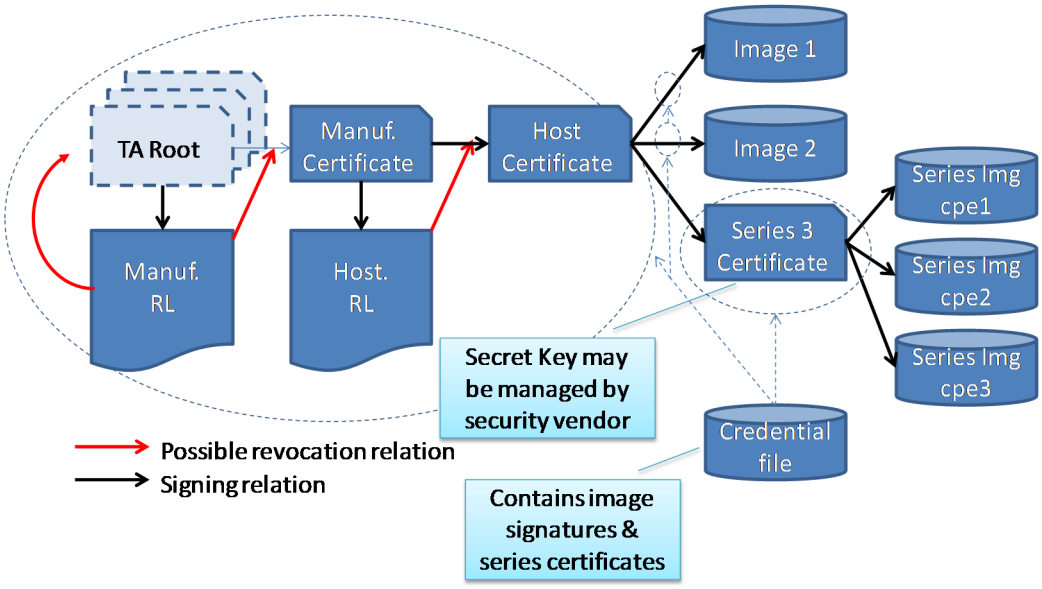


Figure 6.2.2.1-1: ECI Host Image certification structure

Table 6.2.2.1-1: ECI Host related Certificate parameter overview

Type	Entity	Certificate ID field value	Specific Processing by ECI Host
0x0	**Manufacturer**	manufacturer_id, version	Manutacturer_id shall be checked against **CPE**'s Manufacturer ID in the AS-block.
0x0	Host	cpe_type, cpe_model, host_version	cpe_type and cope_model shall be checked against **CPE** type and **CPE** model by the AS-block.
0x8	CPE **Image Series**	target_id	The target_id shall be checked against the **CPE**'s identity.
0x8	CPE image	n.a.	
0x8	**ECI Host Image**	ECI_Host_Image_Id	This is the type for the actual image signature.

The **Certificate** definitions for the **ECI Host** related **Certificates** shall be in accordance with the general **ECI_Certificate** definition is defined in clause 5.2. The definition of the identifier fields of the **Certificates** for **ECI Host** management is given in Table 6.2.2.1-2.

Table 6.2.2.1-2: ID field definition of Host related Certificates

Syntax	No. of bits	Mnemonic
ECI_Manufacturer_Id {		
padding(4)		
type /* see table 5.2-2 */	4	uimsbf
manufuacturer_id	20	uimsbf
Version	8	uimsbf
}		
ECI_CPE_Type_ID {		
cpe_type	12	uimsbf
cpe_model	8	uimsbf
}		
ECI_Host_Id {		
padding(4)		
type /* see table 5.2-2 */	4	uimsbf
ECI_CPE_Type_Id **cpe_type_id**	20	uimsbf
host_version	8	uimsbf
}		
ECI_Host_Image_Series_Id {		
padding(4)		
type /* see table 5.2-2 */	4	uimsbf
image_series_model	8	uimsbf
image_series_model_extension	4	uimsbf
image_series_version	16	uimsbf
}		

Semantics:

type	Value in accordance with Table 5.2-2.
manufacturer_id: integer	Id assigned to **Manufacturer** by the **ECI TA**.
cpe_type: integer	Id assigned to **CPE** model by the **ECI TA**.Values 0x000 and 0x3F0..0x3FF are reserved. **CPEs** of the same model shall have large commonalities and use the same **ECI** security technology.
cpe_model: integer	Id assigned to a version of a specific model that is identical in many respects but has a number of non-trivial differences. Value is assigned by the **ECI TA**. Values 0x00 and 0xF0..0xFF are reserved.
cpe_type_id: ECI_CPE_Type_id	ID of **CPE** hardware type (version + model); unique in the context of **manufacturer_id**.
cpe_host_version	ID assigned to a set of images making up a **CPE ECI Host** configuration for the **CPE**.
image_series_model: integer	ID of images of the same type for **CPEs** supporting **Image Series**, the distinction being made by the **cpe_id**. Values 0x000 and 0xF00..0xFFF are reserved.
image_series_version: integer	Id assigned incrementally to the version of the **Image Series** model by the **ECI TA**. Values 0x0000 and 0xF000..0xFFFF are reserved.

6.2.2.2 ECI Host Image Signatures

The **ECI Host Image** ID shall be equal to the Host Image Series id, and is defined in Table 6.2.2.2-1.

Table 6.2.2.2-1: Host Image ID and Host Series image ID definitions

Syntax	No. of bits	Mnemonic
ECI_Host_Image_Id {		
padding(4)		
type /* see table 5.2-2 */	4	uimsbf
image_model	8	uimsbf
image model_extension	4	uimsbf
image_version	16	uimsbf
}		
ECI_CPE_Id {		
cpe_serial_number	28	uimsbf
cpe_type	12	uimsbf
manufacturer_id	20	uimsbf
}		
ECI_Image_Target_Id {		
padding(4)		
target type	4	uimsbf
if (target type == 0x1){		
ECI_CPE_Id **cpe_id**	60	uimsbf
}		
}		

Semantics:

type	Value in accordance with Table 5.2-2.
image_model: integer	Id assigned to an **ECI Host Image** or series of images replacing each other. Values 0x00 and 0xF0.0xFF are reserved.
image_model_extension: integer	Extension of above field. In regular applications this field should be set to 0x0.
image_version: integer	Version of an image of the same type assigned incrementally. Values 0x00 and 0xF0..0xFF are reserved.
cpe_serial_number: integer	Serial number of the **CPE** for which the image is intended. The cpe_serial_number shall be unique within the context of <manufacturer_id, cpe_type_id>
cpe_type: integer	cpe_type field as defined in the ECI_CPE_Type_Id structure
manufacturer_id: integer	manufacturer_id field as defined in the ECI_Manufacturer_Id structure
target type: integer	Type of target identification for the series image. Value 0x1 defines this structure definition and indicates a cpe_id is used as target, other values are reserved.
cpe-id: ECI_CPE_Id	ID of the **CPE** that is the target of a series (**ECI Host** or **ECI Client**) image.

ECI Host Image signatures and **ECI Host Image Series** signatures that are used to sign the actual **ECI Host Images** shall use the Large Data signature structure as defined in clause 5.5.

6.2.2.3 ECI Host Credentials

Table 6.2.2.3-1 defines the **ECI Host** Credential structure which verifies an **ECI Host Image** set.

Table 6.2.2.3-1: ECI Host credential structure definition

Syntax	No. of bits	Mnemonic
ECI_Host_Credentials{		
image_credential_version	8	uimsbf
if (image_credential_version == 0x01) {		
padding(4)	24	uimsbf
ECI_Certificate_Chain **image_chain**		
nr_images	8	uimsbf
padding(4)	24	uimsbf
for (i=0; i<images; i++){		
ECI_Host_Image_Id **image_id**	32	uimsbf
if (image_id.type == 0x8) {		
ECI_Certificate **series_cert**		
} else if (image_id.type == 0x9){		
ECI_Data_signature **image_signature**		
}		
}		
Extension_Field **extension**		
}		
}		

Semantics:

image_credential_version: byte	Format version of the credentials. Value 0x01 is the presently defined version; all other values are reserved. **ECI Host Loader**s shall ignore any credentials with a value other than the values they recognize.
image_chain: ECI_Certificate_Chain	2-level deep **Certificate Chain** starting with Manufacturer Root RL, up to **ECI Host Certificate**. The last **Certificate** shall be used to verify the image signature of any **Image Series Certificate**.
nr_images: integer	The number of images for which the signatures are included.
image_id	ID of the image for which the signature follows in the loop. The image_ids listed in the loop shall have different **image_id.image_model** field values.
series_cert: ECI_Certificate	Certificate used for verification of an **Image Series**
image_signature: ECI_Data_Signature	Signature of the image (including Host Image Id).
extension: Extension-Field	Backward compatible extension field.

When verifying the **image_chain** the **CPE** shall obey the generic processing rules for chains is defined in clause 5.4.

6.2.3 Loading process of ECI Host Image file

The **CPE** shall store, verify and activate the execution of the set of **ECI Host Image** file required to start the **ECI Hos**t. The actual activation of the **ECI Host Image** typically happens on **CPE** boot.

The **CPE** shall use a robust processing function called the **ECI Host Loader** to download, verify and activate the chosen **ECI Host Image**. If, for example, the **CPE** boot image containing the **ECI Host Loader** starts the execution of a second image, and the second image loads and starts the execution of a third image, the functionality of the second image to actually properly load the third performing verification of the image signature shall be considered **ECI Host Loader** functionality for that **CPE**. Only the **ECI Host Loader** function can verify and start an **ECI Host Image**. The **ECI Host Loader** shall use the **Certificate Processing System** (CPS) to verify the image credentials.

The **CPE** shall store the latest set of **ECI Host Image** files and its credentials it has downloaded in **NV memory**. On **CPE** boot the **ECI Host Loader** shall be able to locate these and start loading the images in a manner that is suitable for the specific **CPE**-type.

The **ECI Host Loader**, using the CPS, shall use the regular chain processing rules in clause 5.4 to verify every image that is loaded. Generic images and **Image Series Certificates** shall be verified using the Host **Certificate** public key.

The **Image Series Certificate** public key shall be used to verify **Image Series** images, and the **CPE** shall verify the cpe_id in the image against the cpe_id of the **CPE**.

In case of a compromised image (signature check failure by the CPS) the **ECI Host Loader** shall reject an image, which means the **CPE** will be unable to instantiate an **ECI Host** on the **CPE**. The **CPE** shall be able to recover from this situation: it shall have a recovery procedure to re-initialize the latest **ECI Host Image** and their credentials, e.g. through reloading the latest set of **ECI Host Image** files from the broadcast channel, from its online **ECI Host Image** server or through some other means.

The **ECI Host** shall store the latest versions of the **ECI Host** chain **Certificates** it acquires regardless of the channel through which it is acquired. Effectively this "locks" the latest available Host **Certificate** as the basis for future image verifications.

The sequence of loading **ECI Host Images** is not verified directly by the signature verification process: this shall be performed by the bootloader for the first **ECI Host image** and for subsequent activations by the preceding **ECI Host images** themselves.

6.3 ECI Host related file formats

The present document does not define any file naming or other Meta attributes for **ECI Host Image** files. It manages **ECI Host Image** data in the form of a set of data containers (**ECI** wise nameless files) identified by their Host Image id, and the **ECI** credentials (**Certificate Chain**s and signatures) needed to authenticate these.

An **ECI Host Image** file shall be a sequence of an ECI_Host_Image_Header and the image content. It shall follow the definition is defined in Table 6.3-1.

Table 6.3-1: ECI Host Image file definition

Syntax	No. of bits	Mnemonic
ECI_Host_Image_File {		
magic = 'EHI'	24	
image_header_version	8	uimsbf
if (image_header_version == 0x01) {		
ECI_Host_Image_Id **host_image_id**	32	uimsbf
ECI_Manufacturer_Id **manufacturer_id**	32	uimsbf
Extension_Field **extensions**		
for (i=0; i<n; i++) {		
host_image_byte	8	uimsbf
}		
}		
}		

Semantics:

host_image_byte: byte	The actual **ECI Host Image**; format proprietary to **CPE**.
magic: byte[3]	The magic number is used for verification of the format of the following data. It has the value of the three 8-bit ASCII representations of the characters `'EHI'`. The **CPE** firmware shall check the value of this field to verify if an **ECI** file has the expected format for additional data integrity.
image_header_version: byte	Format version of the image header. Value 0x01 is the presently defined version; all other values are reserved.
host_image_id: ECI_Host_Image_Id	**ECI Host Image** ID of the image. **CPEs** shall check this field before loading a (new) **ECI Host Image**.
manufacturer_id: ECI_Manufacturer_Id	**ECI** Manufacturer ID of the **Manufacturer** of the **CPE** of the **ECI Host Image**. **CPEs** shall check this field before loading a (new) **ECI Host Image**. See note.
extensions: Extension_Field	See clause 5.1 of the present document: backward compatible extensions.
host_image_byte: byte	The actual **ECI Host Image**.
NOTE: This should also correspond with the Manufacturer OUI used in broadcast carousels to carry the associated file.	

Image Series files have a unique signature which is carries in the image file itself. Therefore a specific file format shall follow the definition as given in Table 6.3-2.

Table 6.3-2: ECI Host Image Series file definition

Syntax	No. of bits	Mnemonic
`ECI_Host_Image_Series_File {`		
`  magic = 'EHS'`		
`  image_header_version`	8	uimsbf
`  if (image_header_version == 0x01) {`		
`    ECI_Data_Signature image_signature`		
`    ECI_Image_Target_Id target_id`	64	
`    Extension_Field extensions`		
`    for (i=0; i<n; i++) {`		
`      host_image_byte`	8	uimsbf
`    }`		
`  }`		
`}`		

Semantics:

host_image_byte: byte	The actual **ECI Host Image**; format proprietary to **CPE**.
magic: byte[10]	Magic number used for verification of the format of the following data. It has the value of the three 8 bit ASCII representations of the characters `'EHS'`. The **CPE** firmware shall check the value of this field to verify if an ECI file has the expected format for additional data integrity.
image_header_version: byte	Format version of the image header. Value 0x01 is the presently defined version; all other values are reserved.
image_signature: ECI_Data_Signature	Signature over all data following in the image file.
target_id: ECI_Series_Image_Target_Id	Target ID for the image. Value for target_id.target_type value is 0x01, all other values are reserved.
extensions: Extension_Field	See clause 5.1: backward compatible extensions.
host_image_byte: byte	Sequence of bytes forming the Host **Image**.

The **ECI Host Image** credentials follow the definition in Table 6.3-3, which in essence is the **Certificate Chain** with the set of the image signatures or **Image Series Certificates**.

Table 6.3-3: ECI Host Image credential file definition

Syntax	No. of bits	Mnemonic
ECI_Host_Image_Credential_File{		
magic = 'EHC'	24	uimsbf
version	8	uimsbf
if (version == 0x01) {		
ECI_Host_Credentials **credentials**		
}		
}		

Semantics:

magic	Magic number used for verification of the format of the following data. It has the value of the three 8 bit ASCII representations of the characters 'EHC'. The **CPE** firmware shall check the value of this field to verify if an **ECI** file has the expected format for additional data integrity.
version	Format version of the file. Value 0x01 is the presently defined version; all other values are reserved.
credentials: ECI_Host_Credentials	The credentials for one or a group of **ECI Host Images**.

The host_image_id is used for the identification of **ECI TA** signatures for a set of **ECI Host Image** files comprising a full download in the **ECI** credential structure.

ECI compliant **CPEs** are allowed to download other proprietary **CPE** software modules using the same transport protocol as the one used for **ECI Host Image** files. There is no specific format required for such images.

On broadcast media it is convenient to distribute the revocation data of many **ECI Hosts** as one large file. **ECI Hosts** receiving such data can use this to check their own **ECI Host Certificate**.

The **ECI Host** revocation data file uses the ECI_Revocation_Data_File format defined in Table 5.5-2. The **ECI Host** revocation data file uses father_type equal 0x0 (**Root Certificate**) and sub_type equal to the **Manufacturer** revocation list type. The revocation_data conforms to the constraint that the leaf revocation list in the trees are **ECI Host** revocation lists.

6.4 ECI Host Image transport protocols

6.4.1 Introduction

The present document distinguishes three types of Host **Image** delivery:

1) **Broadcast: ECI** defines protocols to permit **Platform Operators** to signal and deliver new **ECI Host Image** files from the **CPE Manufacturer** to **CPEs** in the field using DVB-SSU.

2) **On-line: ECI** permits internet connected **CPEs** to download **ECI Host Image** files using any proprietary protocol, suggesting the use of HTTP 1.1 as well as using an **ECI** defined interface to an operator provided web server.

3) **Other: CPE Manufacturers** and/or **Operators** can also use other means to deliver **ECI Host Image** files including off-line methods like delivery via USB stick. This means of transport of images is out of scope of the present document. Nevertheless images loaded with such a protocol shall be in compliance with the file format and image verification in clauses 6.3 and 6.2.

CPEs designed to acquire **service**s from digital broadcast networks shall implement the **ECI Host Image** broadcast transport protocol is defined in clause 6.4.2.

CPEs with an IP connection shall implement an online **ECI Host Image** internet transport protocol is defined in clause 6.4.3 as well as the protocol defined in clause 7.7.3.3.

A **CPE** may implement any complementary **ECI Host Image** transport protocol, including the **ECI Host** broadcast and off-line transport protocols (e.g. USB stick). In all cases the **CPE Manufacturer** shall ensure practical means by which an **ECI Host** can be updated in the field through a combination of the above transport protocols, taking into account practical use cases where some of the network connections are not connected.

6.4.2 ECI Host Broadcast Transport Protocol

6.4.2.1 General and Profiling

The **ECI Host** broadcast transport protocol permits new **ECI Host Image** files and associated data to be transported from **CPE Manufacturer** via the **Operator** broadcast headend infrastructure to the **CPE**. The protocol also permits transport of non **ECI Host Image** files (for non-security critical functions). The **Operator** may play an active role in managing the software version on the **CPE**. This protocol facilitates cooperation by setting standards for the technical interoperability points between the **CPE Manufacturer** and the **Operator**:

- Voluntary standard handover of download data from the **CPE Manufacturer** to the **Operator**;

NOTE: The technical details of such a handover are out of scope of the **ECI** specifications.

- Standard broadcast transport protocol (enabling a single playout provision at the **Operator** broadcast headend); and
- Standard discovery, transport protocol implementation and operational transport protocol parameter choices in receivers.

The **ECI Host** broadcast Transport Stream and **CPE** implementations shall be compliant with DVB SSU [14], and as a consequence comply with the relevant section of the DVB data carousel definition [15] and implementation guidelines [16] and the MPEG data carousel definition [17].

Operators and **CPEs** shall both support DVB-SSU simple profile; and optionally support DVB-SSU UNT profile.

Operators may support multiple simultaneous carousels.

CPEs shall scan all carousels signalled appropriately in the SI, UNT (if applicable) and PMT for relevant download items.

The overall broadcast scheme for downloading images in sketched in Figure 6.4.2.1-1.

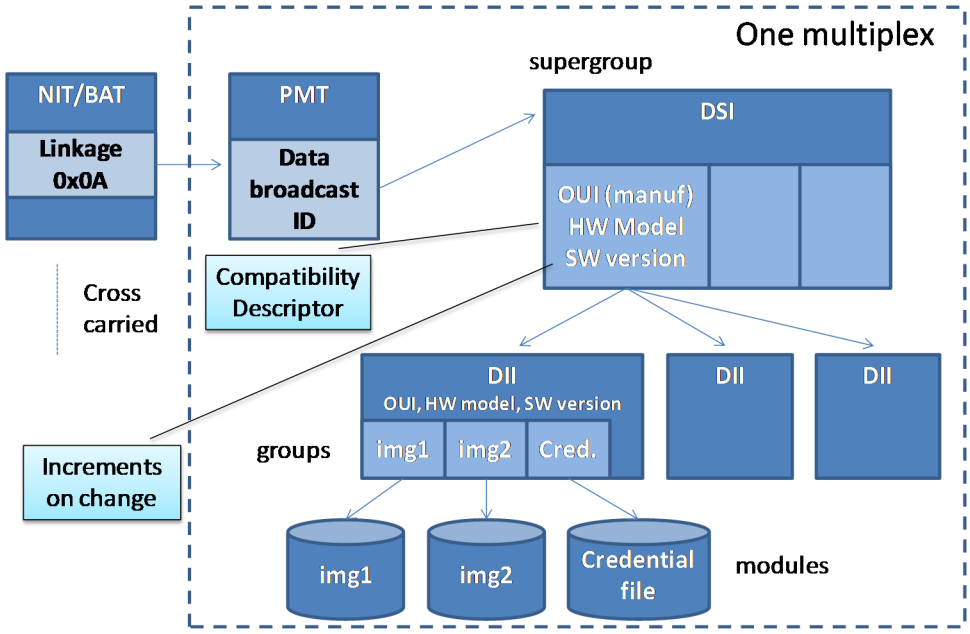


Figure 6.4.2.1-1: Host Image signalling and carousel structure overview (no UNT variant)

6.4.2.2 CPE Manufacturer to Operator handover

Any future **ECI** based Eco-System will have to define a guideline for **Operators** and **CPE Manufacturer**s to provide uniform way of exchanging image file information (both **ECI Host** and non **ECI Host Images**), **ECI** image credentials and meta information regarding the download from (many) **CPE Manufacturers** to (many) **Operators**.

6.4.2.3 DVB SI Signalling

6.4.2.3.1 Download location signalling

Operators shall support DVB-SSU linkage descriptor (linkage type 0x09) with minimally the generic DVB OUI (i.e. non **Manufacturer** specific linkage to all carousels) in all NIT (terrestrial or cable) or in BAT tables (satellite).

Simple profile **CPEs** shall support the DVB-SSU linkage descriptor (linkage type 0x09).

Operators supporting DVB-SSU UNT profile shall support the SSU scan linkage descriptor (linkage type 0xA) in all NIT (terrestrial or cable) or in BAT tables (satellite).

UNT profile **CPEs** shall support the DVB-SSU scan linkage descriptor (linkage type 0x09).

6.4.2.3.2 Emergency Updates

In order to indicate the need to urgently replace an **ECI Host image**, one or more ECI_host_emergency_download descriptors can be placed in NIT, BAT or in one of the SDT entries for a service for which the flagged **ECI Host** can provide access. The **ECI Host** shall be able to retrieve this descriptor from any of the tables in which it appears in any of the currently tuned multiplexes and perform the associated processing, and use any spare tuner to access relevant multiplexes to acquire this descriptor with a worst case 30 minutes interval period, during power-on state. More frequent checking of non-tunes multiplexes (3 minute interval) is recommended.

The ECI_host_emergency_download_descriptor permits targeting specific operation platforms and specific **Platform Operations** and client images in order to minimize the number of **users** experiencing any disturbance potentially caused by emergency updates.

When the **ECI Host** finds a new ECI_host_emergency_download descriptor it shall match its **ECI Host** and **ECI Client** configuration against the targeting information in the descriptor. If a target match is found and the version of the currently installed host image requires an update, the **ECI Host** shall perform this update in accordance with the emergency_indicator. This will cause a disruption for ongoing **user** activities in the **CPE**.

The **ECI** operation descriptor is a DVB private descriptor and shall always be preceded in the table it appears in by the DVB private_data_specifier_descriptor (see ETSI EN 300 468 [19] and ETSI TS 101 211 [21]) using the **ECI** private_data_specifier_field. The syntax of the descriptor is defined in Table 6.4.2.3.2-1.

Table 6.4.2.3.2-1: ECI_host_emergency_download_descriptor

Syntax	No. of bits	Mnemonic
ECI_host_emergency_download_descriptor{		
descriptor_tag	8	uimsbf
descriptor_length	8	uimsbf
/* main loop */		
main_loop_nr	8	uimsbf
for (i=0; i<main_loop_nr; i++){		
/* client loop */		
client_nr		
for (j=0; j<client_nr; j++){		
platform_operation_tag	8	uimsbf
Reserved	3	
client_flag	1	
client_tag	4	uimsbf
}		
/* host image loop */		
host_nr	8	uimsbf
for (j=0; j<host_nr; j++){		
Reserved	4	
emergency_indicator	4	uimsbf
manufacturer_id	20	uimsbf
cpe_type_id	20	uimsbf
min_host_version	8	uimsbf
}		
}		
/* private data till end of descriptor*/		
for (i=0; i<n; i++){		
private_data_byte	8	
}		
}		

Semantics:

descriptor_tag	**ECI** private tag value for descriptor_tag: see ETSI GR ECI 004 [i.10].
descriptor_length	See ETSI EN 300 468 [19].
main_loop_nr	Number of entries in the main loop. The separate main loop entries shall be evaluated separately by the **ECI Host**, i.e. have OR semantics. The various elements in one loop entry shall have AND semantics.
client_nr	Number of entries in the client target loop; value 0x00 shall mean any client will match. The separate loop entries shall have OR semantics and all clients that match shall be considered for an emergency update. The fields of one loop entry shall have AND semantics.
platform_operation_tag	Tag value for the **ECI Platform Operation** as listed in the ECI_platform_operation_descriptor in the NIT/BAT. The **ECI Host** shall consider an emergency update if the platform_operation matches to the platform_operation of one of the installed Clients.
client_flag	Signals whether the client_tag field is relevant for matching. Value=0b0 means not relevant (i.e. any client_id will match), value=0b1 means the client_tag is relevant.
host_tag	Tag value identifying the **ECI Host** as listed in the ECI_platform_operation_descriptor in the NIT/BAT that matches the platform_operation_tag field in the same client loop entry. The **ECI Host** shall consider an emergency update if the referred vendor_id and client_id match with one of the installed clients in the **ECI Host** for the **Platform Operation**.
host_nr	Number of entries in the host loop. The minimum value shall be 1.The loop entries shall have OR semantics; i.e. if any host specification matches the target condition the main loop has a matching state.
emergency_indicator	The **ECI Host** shall use the value of this field to select the appropriate behaviour for starting the download and the subsequent update of the host as defined in Table 6.4.2.3.2-2.
manufacturer_id	Manufacturer_id of the host targeted by an emergency update. The **ECI Host** shall consider an emergency update if the value of this field matches the manufacturer_id of the **ECI Host**.
cpe_type_id	Value as defined by ECI_CPE_Type_ID in Table 6.2.2.1-2. The **ECI Host** shall consider an emergency update if the cpe_type_id of the Host matches to the value of this field. cpe_type_id.cpe_type equal 0x000 shall mean any **ECI Host** cpe_types is a match (and cpe_model and host-version shall be ignored). cpe_type_id.cpe_model equal 0x00 shall mean any **ECI Host** cpe_model is a match (and host version shall be ignored).
min_host_version	The **ECI Host** shall consider an emergency update if and only if its host version is less or equal to the value in this field. NOTE: field value equal 0xFF implies all host versions match.
private_data_byte	Private data: the content may be defined by the operator managing the broadcast of this descriptor.

Table 6.4.2.3.2-1 defines a number of conditions in the main loop (having AND semantics) that shall be met in order for the **ECI Host** to consider performing an emergency update. If all these conditions are met the **ECI Host** shall perform an emergency download and installation of a new host image in accordance with the emergency_indicator field for that **ECI Host**. The indicator field values are defined in Table 6.4.2.3.2-2.

Table 6.4.2.3.2-2: ECI_host_emergency_download_descriptor emergency_indicator field values

Name	Value	Description
System emergency	0x01	The **ECI Host** shall download the new host image install it as quickly as possible interrupting ongoing **user** instigated activities if so required. See note.
Regular urgency	0x03	The **ECI Host** shall download the new host image and install it on the first occasion this does not cause any disruption to any **user** instigated activity or latest during the next power-up event. NOTE: **Platform Operators** can use this for instance if the present **ECI Host** has serious deficiencies for decrypting services but can perform reasonably for regular use cases.
RFU	other	Reserved for future use.
NOTE: **Platform Operators** can use this for instance if the present **ECI Host** has significant performance issues in combination with the targeted platform/client combinations.		

6.4.2.4 PSI signalling

Operators shall support the data_broadcast_id_descriptor in the PMT [19] for every carousel transmitted, but are not required to support any OUI signalling in the selector bytes of this descriptor.

SSU Simple profile **CPEs** shall use the data_broadcast_id_descriptor to locate the PID of the stream carrying a DVB-SSU carousel.

6.4.2.5 UNT option

This clause only applies to **CPEs** and **Operators** supporting the UNT profile.

In the PMT the data_broadcast_id_descriptor shall be used containing the system_software_update_info structure with update_type 0x2 and OUI field set to DVB OUI 0x00015A.

Operators shall carry an SSU table entry in one of the SSU tables for each of **CPE** types they support.

ECI Hosts shall be able to interpret the following UNT descriptors (see ETSI TS 102 006 [14]):

- SSU_location_descriptor (if a carousel for the **CPE**-type is being broadcast).
- Scheduling_descriptor (if a carousel for the **CPE**-type is planned for the foreseeable future).
- Message_descriptor.

CPEs shall be able to consistently perform a successful download of a practically error-free received carousel that is mounted and dismounted on the nominally published times and that makes two full cycles (repetition of all messages in the carousel) under the provision that there is no **user** initiated activity interfering with the download.

6.4.2.6 Carousel structure

ECI DVB SSU carousels (for details see ETSI TS 102 006 [14]) shall use two layer data carousels.

The **ECI** DVB SSU carousel shall use the DSI message with the following constraints:

- There shall be a complete list of all available groups for download.
- Each group shall correspond to one **cpe_type + cpe_model** of one **Manufacturer**, and contain all resources for the **ECI Host** of the **CPE** type. This implies a maximum of 255 modules (image files) can be available (plus one file for the credentials).

NOTE 1: Due to limitations in the values for **ECI_host_id.model_id** the limit is 239.

- The CompatibilityDescriptor in the GroupCompatibility field of the GroupInfoIndication structure (for details see ETSI TS 102 006 [14]) shall use the following convention:
  - The loop shall contain a system hardware descriptor:
    - The OUI shall correspond to the **Manufacturer** of the **CPE**.
    - The model and version fields associated with the system hardware descriptor shall correspond to **CPE**'s **cpe_type** and **cpe_model**, and be equal to the **id.cpe_type** and **id.cpe_model** fields of the **ECI Host Certificate** in credential file of the group.
  - The loop shall contain a system software descriptor; the model field shall be set to 0, the version field shall reflect the version of the total **ECI Host** software in the group (i.e. both **ECI Host** and non **ECI Host Images**).

CPEs shall use the model and version field in the compatibilityDescriptor to match to its own **CPE** model and **CPE** version and shall use the software version field to check if the group contains an update and in case of a new version proceed with downloading new images.

The **ECI** DVB SSU carousel shall use the DII message fields with the following constraints:

- The blockSize shall be set to a value of 2 kbyte (2 048 byte) at minimum.
- The "tDownloadScenario" field shall be given a meaningful value reflecting a download of all modules with at minimum 4x the slowest message repeat time (carousel turnaround time).
- moduleId bits 7..0 shall be equal to the **id.image_model** of the image file.

- moduleVersion be equal to the **ECI id.image_version** of the image file.

CPEs may use the "tDownloadScenario" field to terminate downloads that fail to succeed (e.g. due to high packet error rates) and report this problem to the **user**.

The group of a **CPE** type shall contain the following modules:

- Image files for a **CPE** type (may be a partial image set).
- The **ECI Host Image** credential file containing the (latest) credentials for all images of a **ECI Host**:
  - this module shall have DII moduleId bits 7..0 set to 0xFF; and
  - moduleVersion shall increment on every change.

NOTE 2: **Operators** are allowed to share common files between downloads for various **CPE** types by sharing DownloadDataBlocks between DIIs. However, this implies the need to manage **ECI Host Image** Ids coherently between **CPE** types.

6.4.2.7 ECI Host downloading operation

The **ECI Host Image** loader shall attempt to check all possible carousels every 30 minutes in power-on state if network access resources are available and at least every 6 hours in standby state without disturbing the **user**, e.g. after switching the **CPE** to standby and during non-peak viewing hours.

If a network provider makes UNTs available carrying potential downloads for a **CPE**-type the corresponding **CPE** shall regularly check the UNT for the schedule of a potential new update. The **CPE** shall attempt checks this with the same frequency-conditions as for the **ECI Host** image carousels.

It is recommended that the **user** receives a warning if a broadcast-only mode **CPE** is prevented from performing the above checks for more than 2 weeks.

Once the availability of a new download is detected the **CPE** and the **user** has provided approval, the **CPE** shall attempt to perform the download and install the new image (possibly overwriting a previous version). Any persistent failure to successfully perform a download shall be reported appropriately to the **user**. **ECI Hosts** shall always be able to recover from a failed host image download and recover to a functional state, e.g. by restoring the previous host image or by attempting to reload the new host image.

It should be noted that persistent failure to download new **ECI Host Images** or credentials can lead to denial of service by an **Operator**.

6.4.2.8 Operator Carousel schedules

Operators should provide sufficient bandwidth for **CPE** image data carousels to perform the download in a reasonable time.

6.4.2.9 User Interface Aspects

A **CPE**, able to perform **ECI Host Image** downloads over the broadcast network, shall:

- have a download scan mode of operation that will automate the checks for the availability of new images or credentials on a regular basis; e.g. as part of the standby state, and this is recommended to be the default **Manufacturer** setting for download checking; and
- have a setting in the **CPE** menu that will automate any **user** approval for accepting new **ECI Host Image** files or credentials, and it is recommended to be the default **Manufacturer** setting for download approval.

CPEs shall provide for at least one alternative means for downloading new **ECI Host Image** files in order to prevent **CPEs** operating in broadcast networks that do not provide new **ECI Host Image** files for their **CPE** type experiencing denial of service.

6.4.3 ECI Host Internet Transport Protocol

6.4.3.1 IP Protocol

ECI does not define a specific protocol for the **CPE** to check for new **ECI Host Image** files from a service provided by the **Manufacturer**. It is however recommended to use HTTP1.1 [18] as the file transfer protocol, and the protocol as defined in clause 7.7.3.3 may be used which defines a standardized download service for **ECI Host Image** files from a **Platform Operation** server.

Typically the **ECI Host Image** download server is provided for by the **CPE Manufacturer**. With specific arrangements in place between **CPE Manufacturer** and an **Operator** (or 3rd parties acting on their behalf) these may also be provided by the **Operator** or a third party.

6.4.3.2 Online Loader Operation

The **ECI** online **ECI Host Image** loader shall attempt to check its online server every 30 minutes without disturbing the **user**. It is recommended that the **user** receives a warning if an online-only mode **CPE** is prevented from performing the above checks for a longer period.

Once the availability of a new download is detected, the **CPE** shall attempt to perform the download and install the new image (possibly overwriting previous image versions). Any persistent failure of such download shall be reported appropriately to the **user**.

It should be noted that failure to download new **ECI Host Images** or credentials may lead to denial of service by an **Operator**.

The **CPE** online loader shall deliver a set of (new) images and image credentials as defined in clause 6.3 for verification, storage and activation.

The **ECI** online Host Image loader shall provide emergency download features with the same effect as defined in clause 6.4.2.3.2 for broadcast.

6.4.4 Alternative transport protocols

An **ECI Host** is allowed to use any alternative (proprietary) delivery protocols.

The **CPE** loader shall process a set of (new) images and image credentials as defined in clause 6.3 for verification, storage and activation.

7 ECI Client Loader

7.1 Introduction

The **ECI Host** can download, store and activate **ECI Client Images** and accompanying data. The **ECI Client** loading process can be split up in the following steps:

1) Discovery of **ECI**-based protection of a service/package of services and/or other ways to identify the need for an **ECI Client**. This is part of the regular navigation application of the **CPE**.

2) Determining the network location (broadcast or online) of the resources needed to install an **ECI Client** on the **ECI Host**.

3) Downloading and storing (in NV memory) the **Platform Operation** information required to install the **ECI Client** and verifying the credentials.

4) Registration of the **ECI Host** with the security system of the **Platform Operation**, and receiving (if required) **CPE** specific initialization data for the decryption of the **ECI Client**.

5) Downloading and storing (in NV memory) of the **ECI Client Image** and associated **ECI Client** credentials from the network and verifying the credentials and the image, storing in NV-memory for future use.

6) Initialization of the **ECI Client** using the **ECI Client Image**, the **Platform Operation Certificate**, allocation of an **ECI** Container and required AS resources and starting the execution of the **ECI Client**.

All processes can be performed using data from the broadcast stream or from the internet, with exception of the registration of the **CPE** with the **Operator**, which requires manual assistance in case only a broadcast connection is available.

Operators can renew the **ECI Client** resources at any time by publishing the information on broadcast or online networks. The **ECI Host** regularly checks for such updates.

ECI requires support data for various functions of an **CPE**, e.g. revocation data or updated **Certificate Chain**s required by the **ECI Client** and/or the **ECI Host** to be able to support the **ECI Client**. On broadcast networks the transport protocol allows selective download of the data needed by a **CPE**, based on an "index" (hash) of the identification of the data. The grouping of data by the hash of the index is called "bucketizing". On online networks selective downloading is based on passing the identification of the required data as a parameter to a web services API.

The following data items can be downloaded by the **ECI Host**:

- **ECI Client Images** (in bucketized format on broadcast networks).
- **ECI Client** revocation data (in bucketized format on broadcast networks).
- Platform operation client chain.
- **Platform operation** revocation data (in bucketized format on broadcast networks).
- **ECI Host Image** revocation data (in bucketized format on broadcast networks). **ECI AS** setup client initialization data for decryption of encrypted client images (in bucketized format on broadcast networks).

7.2 Discovery of ECI Clients

7.2.1 Introduction

Typically an **ECI** compliant **CPE** (e.g. an iDTV) will have no **ECI Client** installed when it leaves the factory, because this device may be sold in any market worldwide. The following clause defines the available mechanisms allowing an **ECI** compliant **CPE** to find **ECI Clients** which might be required to descramble services delivered by a network it is connected to.

For the discovery process two types of networks are distinguished:

1) Transport stream based networks (broadcast and typical IPTV networks).
2) IP protocol based networks.

ECI supports two modes of provider and client discovery for transport stream based networks:

1) Manual installation - including basic (broadcast) network setup parameters.
2) Self-discovery (with **user** choice) - this assumes the **CPE** can auto-install for the network autonomously.

Both manual installation and self-discovery protocols on transport stream based networks use common signalling.

For IP protocol based networks **ECI** supports:

1) Manual base-URL entry.

7.2.2 Transport stream based networks

7.2.2.1 Common signalling

In order to reduce manual entry of parameters for the **user**, **ECI** provides for online signalling of key **ECI** parameters to install a client:

- One or more ECI_ platform_operation_descriptors in the NIT carrying the available clients (by ID) per **Platform_Operation**. The descriptor includes the platform provider name and a short-id (to permit compact representation in the manual installation string).
- A platform provider may specify a base URL for the web API in the ECI_base_URL_descriptor.

7.2.2.2 ECI_ platform_operation _descriptor

The ECI_ platform_operation_descriptor provides key information about a **Platform Operation** that offers access services for a transport stream based network.

For each **Platform_Operation** the NIT_{actual} (and/or BAT on satellite networks) shall carry the ECI_ platform_operation_descriptor at minimum on the central multiplex & table identified in the installation string for networks that offer only manual installation, and on all multiplexes, except for satellite networks, for networks that offer self-discovery. Satellite networks are permitted to carry the ECI_ platform_operation_descriptor only on the multiplexes on which the provider carries services: either as part of the NIT or a BAT.

The ECI_ platform_operation _descriptor is a DVB private descriptor, using **ECI**'s private data specifier in the DVB private_data_specifier_descriptor [20]. It is defined in Table 7.2.2.2-1.

Table 7.2.2.2-1: ECI_ platform_operation _descriptor

Syntax	No. of bits	Mnemonic
ECI_ platform_operation_descriptor(){		
descriptor_tag	8	uimsbf
descriptor_length	8	uimsbf
platform_tag	8	uimsbf
operator_id	20	uimsbf
platform_operation_id	20	uimsbf
platform_name_length	8	uimsbf
/* platform name loop */		
for (i=0; i<N; i++){		
platform_name_char	8	uimsbf
}		
for (i=0; i<N; i++){		
extension_byte	8	uimsbf
}		
}		

Semantics:

descriptor_tag	**ECI** private tag value for descriptor_tag see ETSI GR ECI 004 [i.10].
platform _tag	This 8-bit field specifies the tag of the **Platform_Operation** for the purpose of manual installation. On each NIT and BAT of a network supporting each **Platform_Operation** shall have a unique platform_tag value. Each platform_tag shall appear only once in each NIT or BAT. The platform_tag shall not be used for ordering providers and shall not be presented in the **CPE user** interface for **Platform_Operation** selection.
operator_id	Operator ID as defined in clause 7.5.2 of the present document. This is the identifier of the operator of the **Platform_Operation**.
platform_operation_id	**Platform_Operation** ID as defined in clause 7.5.3 of the present document.
platform_name_length	Length of the octet sequence of the platform name loop. If the length is 0 the provider does not support self-discovery, and shall not be listed in any provider selection menu in the **CPE**'s client installation menu. The maximum value of this field shall be 40.
platform_name_char	sequence of UTF8 characters representing the name of the platform operation.
extension_byte	Additional bytes; reserved for future use by the present document.

7.2.2.3 ECI_base_url_descriptor

The ECI_base_url_descriptor allows the **Platform_Operation** to signal the base URL: of his web-API (see clause 7.7.3), which can be used to provide client installation related services in case of online access.

For each **Platform_Operation** the NIT_{actual} (and/or BAT on satellite networks) may carry the ECI_base_url_descriptor in the same table carrying the ECI_platform_operation_descriptor.

The ECI_base_url_descriptor is a DVB private descriptor, using **ECI**'s private data specifier in the DVB private_data_specifier_descriptor [19]. It is defined in Table 7.2.2.3-1.

Table 7.2.2.3-1: ECI_base_url_descriptor

Syntax	No. of bits	Mnemonic
`ECI_base_url_descriptor(){`		
`  descriptor_tag`	8	uimsbf
`  descriptor_length`	8	uimsbf
`  platform_tag`	4	uimsbf
`  reserved`	4	
`  base_url_length`	8	uimsbf
`  /* base url loop */`		
`  for (i=0; i<N; i++){`		
`    base_url_char`	8	uimsbf
`  }`		
`}`		

Semantics:

descriptor_tag	**ECI** private tag value for descriptor_tag see ETSI GR ECI 004 [i.10].
platform_tag	This 4-bit field specifies the tag of the provider for the purpose of manual installation. On each NIT and BAT of a network supporting each **Platform_Operation** shall have a unique platform_tag value. Each platform_tag shall appear only once in each NIT or BAT. The platform_tag shall not be used for ordering **Platform_Operations** and shall not be presented in the **CPE user** interface for **Platform_Operation** selection.
base_url_length	This field shall indicate the number of octets in the base URL loop.
base_url_char	The sequence of UTF8 characters forming the base URL for a platform operation.

7.2.2.4 Manual installation

The **Platform_Operation** can provide an installation string to the **user** which he can enter into a suitable installation menu item of the **CPE user** interface in order to install an **ECI Client**. The installation string shall be defined in accordance with this clause. The installation string is a digit representation of a binary number of variable lengths. The binary number in a most significant bit first representation can be constructed by concatenating the 3-bit binary values of the digits in a most significant bit first representation.

The number is presented to the **user** in chunks of 4 digits, and the entry on the **CPE** UI shall equally represent 4 digit chunks.

The installation string identifies the parameters are defined in Table 7.2.2.4-1

Table 7.2.2.4-1: Installation string parameters (in number of bits)

Parameter	DVB-T/DVB-T2	DVB-C/DVB-C2	DVB-S/DVB-S2	IPTV	Mnemonic
Network type	3	3	3	3	uimsbf
Network ID	16	17	17	16	uimsbf
Platform tag	8	8	8	8	uimsbf
Client tag	4	4	4	4	uimsbf
Padding	0	0	0	0	uimsbf
Checksum	5	5	5	5	uimsbf
Number of bits	36	36	36	36	uimsbf
Number of digits	12	12	12	12	uimsbf
Number of chunks	3	3	3	3	uimsbf

Semantics:

Network type	3-bit field. The values for network type are presented in Table 7.2.2.4-2.
Network ID	The DVB SI Table-id containing the ECI_service_provider_descriptor (see clause 7.2.2.2) that provides detailed information required to access services as defined in Table 7.2.2.4-3.
Platform tag	4-bit field representing the provider tag of the required service provider in the ECI_service_provider_descriptor in the NIT or BAT.
Client tag	4-bit field representing the provider tag of the required client in the ECI_service_provider_descriptor selected by the provider tag in the NIT or BAT.
Padding	0..2 bit field with value 0 that pads the previous string to a multiple of 3 bits.
Checksum	5 bit field formed by adding successive 5-bit chunks of the previous string. The last part of the string is padded with additional leading zeroes to a length of 5-bits. E.g. the checksum of string 0b01011010 is 0b01011 + 0x00010 = 0b01101. The checksum shall be used by the **user** interface of the **CPE** to reject any erroneous entries by the **user**.

Table 7.2.2.4-2: Network type value representation

Network type	Value
DVB-T/T2	0
DVB-C/C2	1
DVB-S/S2	2
IPTV	3
Reserved	4..7

Table 7.2.2.4-3: Network ID representation

Network type	Network ID value	Number of bits
DVB-C	0b0 followed by Network ID of NIT table or 0b1 followed by BAT ID of BAT table	17
DVB-S/S2	0b0 followed by Network ID of NIT table or 0b1 followed by BAT ID of BAT table	17 17

7.2.2.5 Self-discovery installation

For this installation method the **CPE** should be able to self-discover the network parameters of the transport stream based network and thus be able to access all transport streams on the network.

Each service in each of the multiplexes will be tagged with the **ECI Platform_Operations tag** that can provide access to the service. This may be done in the SDT on a per service basis (see clause 7.2.2.6) or in the NIT or BAT (only for satellite networks) on a per multiplex basis (see clause 7.2.2.6).

The **CPE** shall offer the **user** the option to install any **ECI Client** of the **Platform_Operations** as part of the self-discovery installation process. In case a **user** decides to install an **ECI Client** of the **Platform_Operation** because he wishes to receive decrypted services via the related access network, the **CPE** default behaviour shall be to install all **services** tagged to that **Platform Operation** in the central service list of the **CPE**.

7.2.2.6 ECI service tag descriptor

The ECI_service_tag_descriptor is carried in the SDT. It tags each service with the **ECI** service providers that offer to descramble this service. The definition is given in Table 7.2.2.6-1.

Table 7.2.2.6-1: ECI service tag descriptor

Syntax	Number of bits	Mnemonic
`ECI_service_tag_descriptor(){`		
`  descriptor_tag`	8	uimsbf
`  descriptor_length`	8	uimsbf
`  platform_tag`	8	uimsbf
`}`		

Semantics:

descriptor_tag	**ECI** private tag value for descriptor_tag see ETSI GR ECI 004 [i.10].
platform_tag	This is the platform_tag value of **ECI Platform_Operation** as listed in the ECI_platform_operation_descriptor carries either in the NIT or the BAT of the network.

7.2.2.7 ECI platform list descriptor

The **ECI** platform list descriptor provides the list of **ECI Platform_Operations** that provide access to services of the different multiplexes in the network. The ECI_platform_list_descriptor carried in the NIT and/or BAT. The definition is given in Table 7.2.2.7-1.

Table 7.2.2.7-1: ECI_platform_list_desciptor

Syntax	Number of bits	Mnemonic
ECI_platform_list_descriptor(){		
descriptor_tag	8	uimsbf
descriptor_length	8	uimsbf
for (i=0;i<N;i++){		
platform_count	8	uimsbf
/* platform loop */		
for (j=1; j<M; j++){		
platform_tag	8	uimsbf
}		
service_count	16	uimsbf
/* service loop */		
for (j=0; j<M; j++){		
service_id	16	uimsbf
}		
}		
}		

Semantics:

descriptor_tag	ECI private tag value for descriptor_tag see ETSI GR ECI 004 [i.10].
platform_count	8-bit field is the number of provider tags in the following loop.
platform_tag	This is the platform_tag value of ECI Platform_Operations as listed in the ECI_platform_operation_descriptor, which is carried either in the NIT or the BAT of the network. The services to which the tagged Platform_Operation is associated follow in the service loop. Platform_tag values are permitted to appear multiple times in the outer loop of this descriptor.
service_count	16-bit field, representing the number of service_ids in the following loop.
service_id	DVB service ID of a service in the multiplex of the NIT or BAT that can be accessed using the access services of the platforms referred to in the preceding platform loop

7.2.3 IP network based client discovery

7.2.3.1 Manual installation

A **CPE** with access to IP networks shall offer a manual URL entry option to permit installation of a service provider. The URL will serve as base URL for the web-API.

NOTE: As part of the application functions of a **CPE**, some of which may be downloaded, the **CPE** may offer access to various online services. The **CPE** may offer a service provider plus client installation API interface so as to automate the client installation process for the **user**.

7.2.3.2 Web-page based installation

This type of solution for the installation of an **ECI-Client** is out of scope for the current specification and may subject to supplementary specifications.

7.3 Storage, Verification and Activation

7.3.1 General Update Policies

ECI supports frequent renewability of items to enable a high level of integrity. Therefore all downloaded items are regularly checked for updates. The following download-update policy shall apply to all **ECI Client** and **Platform Operation** data and accompanying revocation data.

ECI Hosts shall attempt to check for updates regularly and inform the user in case any action is required. Detailed requirements for the update policy are given in ETSI GR ECI 004 [i.10] Platform Operation Client Chain download.

The **ECI Host** shall store the **Platform OperationClient Chain** with the associated **ECI Client**. Storage and deletion shall be managed as part of installing and deletion of **ECI Clients**.

The **ECI Host** shall automatically update the platform provider **Certificate** and overwrite older versions.

7.3.2 ECI Client Image download and storage

As part of managing **ECI Client** related resources, the **ECI Host** shall store an **ECI Client Image** needed to access services or content from NV memory only after (implicit) **user** approval. Any automatic policy to install **ECI Clients** shall provide a **user** transparent method to deal with any resource limitation to manage **ECI Clients** in a manner that is transparent to the **user** and that does not lead to unexpected loss of access to content or services. In accordance, any **ECI Client Image** deletion shall be (implicitly) approved by the **user**.

The ECI Host shall store downloaded **ECI Clients** in NV memory with their original credentials on a **Platform Operation** basis. New **ECI Client** versions (including only new credentials) shall overwrite older versions (on a **Platform Operation** basis). Example: if two **Platform Operations** use the same **ECI Client**-type but use different versions, both versions shall be stored by the **ECI Host**.

The minimum image size a **CPE** can store per **ECI Client** slot is defined in ETSI GR ECI 004 [i.10].

7.3.3 ECI Client Validation and Activation

The **ECI Host** shall load the latest (by version number) **Platform Operation Client Chain** for the **Platform Operation Certificate** in the **Advanced Security System** and attempt to install the **Platform Operation** public key, in accordance with the generic rules for processing chains as defined in clause 5.4.2.

The **ECI Host** shall load the latest **ECI Client** in the **Advanced Security System**. It shall load the **Platform Operation** client co-signature in the **Advanced Security System**. It shall subsequently validate the **ECI Client**, in accordance with the generic rules for processing chains in clause 5.5 and verify the signature and co-signature for the **ECI Client Image**. If a revocation occurs the **ECI Host** shall notify the **user**.

A new **ECI Client** shall only be installed and activated if the validation process has been completed successfully.

7.4 ECI Client Chain structure formats

7.4.1 Introduction to ECI Client Chain structure formats

Figure 7.4.1-1 sketches the structure of the **ECI Client Certificate Chain**. The chain starts with the Vendor **Revocation List**, followed by **Security Vendor Certificate**, **ECI Client Revocation List** and finally the **ECI Client Image** file. In case of an **Image Series** an additional **ECI Client Image Certificate** is introduced. The **ECI Platform Operation** Client signature provides a second signature to the client image ensuring the applicability of the **ECI Client** to a platform operation. It is defined in clause 7.5.

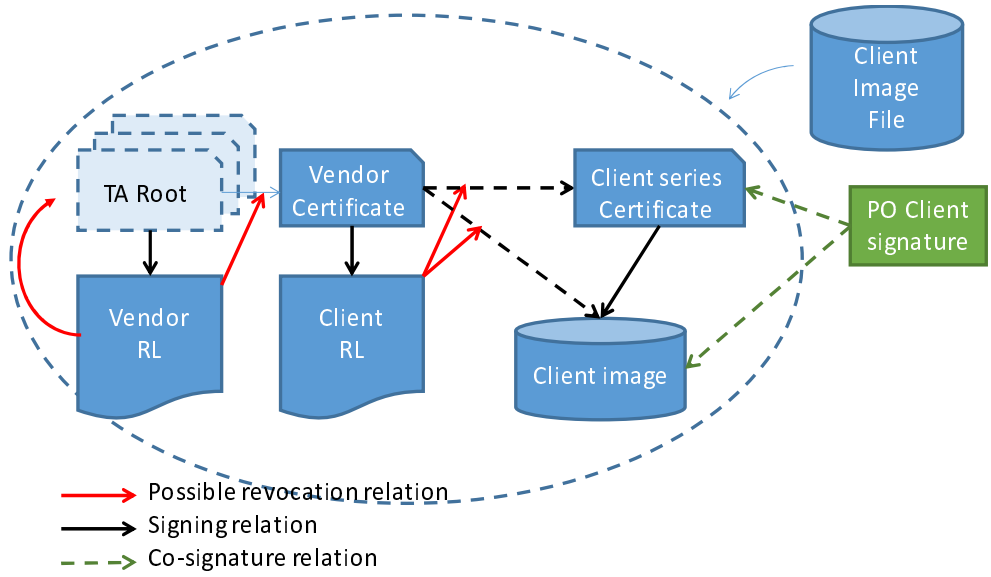


Figure 7.4.1-1: Client Authentication Chain

7.4.2 Security Vendor Certificate

Security Vendor Certificates are defined by the ECI_Certificate structure. The **Certificate** ID for the **Security Vendor Certificate** is defined in Table 7.4.2-1.

Table 7.4.2-1: Security Vendor ID definition

Syntax	No. of bits	Mnemonic
`ECI_Vendor_Id {`		
`  padding(4)`		
`  type /* see Table 5.2-2 */`	4	uimsbf
`  vendor_id`	20	uimsbf
`  vendor_version`	8	uimsbf
`}`		

Semantics:

type: integer	Value in accordance with Table 5.2-2.
vendor_id: integer	Vendor number assigned to Security Vendor, unique in the context of **ECI**.
vendor_version: integer	Id assigned incrementally to the version of the **Certificate** for the Security Vendor. Values 0x00 and 0xF0..0xFF are reserved.

7.4.3 ECI Client series Certificate and series target id

ECI Client series **Certificates** are defined by the **ECI_Certificate** structure. The certificate ID for the **Security Vendor certificate** is defined in Table 7.4.3-1.

Table 7.4.3-1: Client series ID definition

Syntax	No. of bits	Mnemonic
ECI_Client_Series_Id {		
padding(4)		
type /* see Table 5.2-2 */	4	uimsbf
client_type	12	uimsbf
client_version_major	8	uimsbf
client_version_minor	8	uimsbf
}		

Semantics:

type: integer	Value in accordance with Table 5.2-2.
client_type: integer	**ECI Client** type unique in the context of the **ECI Client's** Security Vendor id.
client_version_major: integer	Major version number of the **ECI Client** of an **ECI Client-type**. Versions increment for new major release (see note).
client_version_minor: integer	Minor version number of the **ECI Client. ECI Clients** can be revoked by minor version number comparison in **ECI Client Revocation Lists**, and be automatically replaced.
NOTE: **ECI Client** replacement on major release change is not automatic in **ECI** compliant **CPEs** as only minor version updates are triggered automatically.	

NOTE: The **ECI Client** type series **Certificates** are assigned to **ECI Clients** that require customized implementations per **CPE** but that are identical from a security and functionality perspective.

The client target ID is defined in the same way as for **ECI Hosts**, using the ECI_Host_Series_Image_Target_Id structure. This binds a client image to a specific **ECI Host**.

7.4.4 ECI Client Image signature

ECI Client signatures shall use the ECI_Data_Signature structure as defined in clause 5.6.

The **ECI Client** ID is defined in Table 7.4.4-1, and is identical in structure to the ECI_Client_Series_Id as defined in Table 7.4.3-1.

Table 7.4.4-1: Client ID definition

Syntax	No. of bits	Mnemonic
ECI_Client_Id {		
padding(4)		
type /* see Table 5.2-2 */	4	uimsbf
client_type	12	uimsbf
client_version_major	8	uimsbf
client_version_minor	8	uimsbf
}		

Semantics:

type: integer	Value in accordance with Table 5.2-2.
client_type: integer	Client type, as assigned by **ECI TA**.
client_version_major: integer	Major version number of the **ECI Client** of an **ECI Client-type**. Versions increment for new major release.
client_version_minor: integer	Minor version number of the **ECI Client. ECI Clients** can be revoked by minor version number comparison in **ECI Client** Revocation Lists.

7.5 ECI Platform Operation Chain Formats

7.5.1 Overview

In Figure 7.5.1-1 the authentication chain for the **Platform Operation Certificate** and the **Platform Operation** client signatures is presented. It starts with the **Operator Revocation List**, followed by the **Operator Certificate**, **Platform Operation Revocation List** and finally the **Platform Operation Certificate** containing the **Platform Operation** Public Key. This is used in combination with the **Platform Operation** Client **Revocation List** to validate the **ECI Client Images** permitted to operate for the platform.

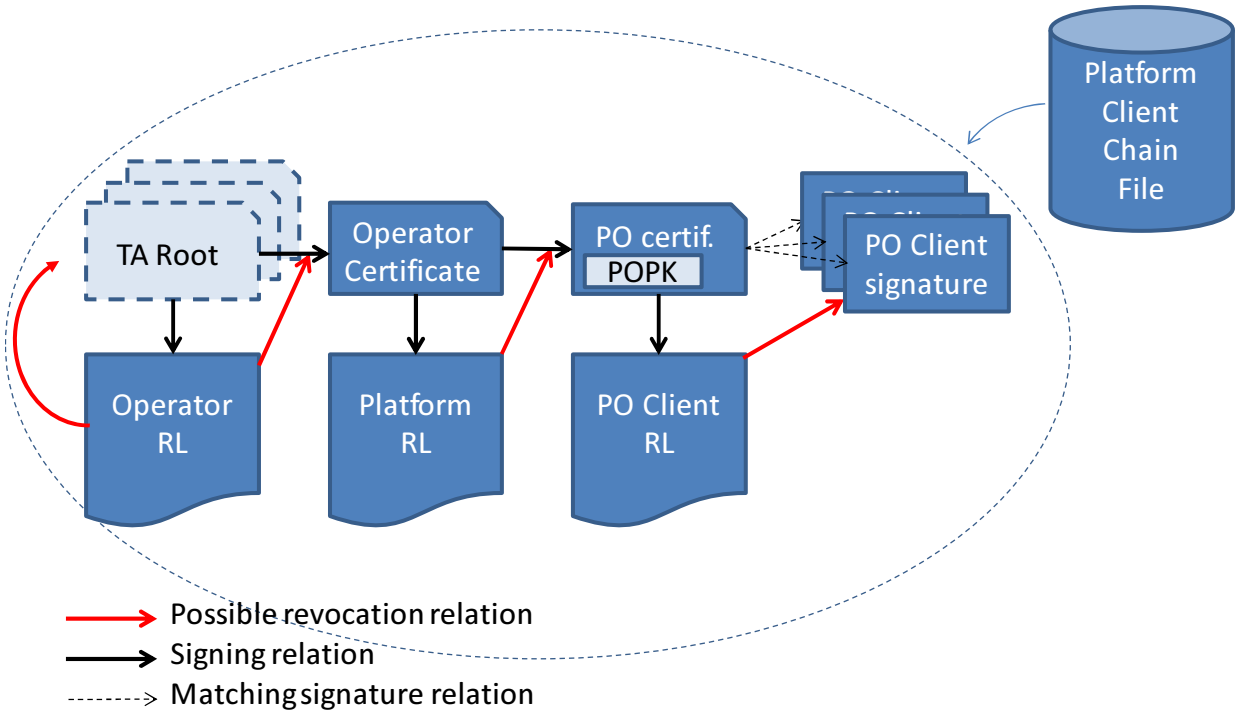


Figure 7.5.1-1: Authentication chain for the platform client chain

7.5.2 Operator Certificate

Operator Certificates are defined by the **ECI_Certificate** structure. The ids for the **Operator** are defined in Table 7.5.2-1.

Table 7.5.2-1: Operator ID definition

Syntax	No. of bits	Mnemonic
`ECI_Operator_Id {`		
`  padding(4)`		
`  type /* see Table 5.2-2`	4	uimsbf
`  operator_id`	20	uimsbf
`  operator_version`	8	uimsbf
`}`		

Semantics:

type: byte	Value in accordance with Table 5.2-2.
operator_id: integer	**Operator ID** assigned to an **Operator**, unique in the context of the **ECI** root.
operator_version: integer	Version number assigned incrementally to the version of the **Certificate** for the **Operator**. Values 0x00 and 0xF0..0xFF are reserved.

7.5.3 Platform Operation Certificate

Platform Operation Certificates are defined by the **ECI_Certificate** structure. The secret key for the **Platform Operation** is managed by the **Platform Operation**. The certificate ID for the **Platform Operation certificate** is defined in Table 7.5.3-1.

Table 7.5.3-1: Platform Operation ID definition

Syntax	No. of bits	Mnemonic
ECI_Platform_Operation_Id {		
padding(4)		
type /* see Table 5.2-2	4	uimsbf
platform_operation_id	20	uimsbf
platform_operation_version	8	uimsbf
}		

Semantics:

type: byte	Value in accordance with Table 5.2-2.
platform_operation_id: integer	**Platform Operation** number assigned to Security Vendor, unique in the context of the Operator **Certificate**.
platform_operation_version: integer	Incremented in case the **Platform Operation** changes its **Certificate**.

7.5.4 Platform Operation client revocation List

The **Platform Operation** client revocation list is a defined clause 5.3 using the identifier assignment as defined in Table 5.2-2. The entity_id fields in the revocation list refer to the cosignature_id field of the **Platform Operation** client signature data structure.

The minimum revocation list version number is defined as part of the initialization of the **ECI Client** and is validated using the Advances Security System.

7.5.5 Platform Operation client co-signature

The **Platform Operation** client co-signature provides the **Platform Operation** signature to verify that a Client image is permissible for providing access services for a platform. In addition is provides the vendor and client ID of the image for easy matching to the associated client image. The **Platform Operation** Client signatures have their own identifier enumeration; this permits independent revocation of previously permitted **ECI Client Images** using the **Platform Operation** client revocation list. The details are given in Table 7.5.5-1.

Table 7.5.5-1: Platform Operation Client cosignature definition

Syntax	No. of bits	Mnemonic
ECI_PO_Cosignature_Id {		
padding(4)		
type	4	uimsbf
entity_id	20	uimsbf
version	8	uimsbf
}		
ECI_PO_Client_Cosignature_Data {		
ECI_PO_Cosignature_Id **cosignature_id**	32	
client_tag	4	uimsbf
reserved	28	
ECI_Vendor_Id **vendor_id**	32	
if (/* image series cosignature */) {		
ECI_Client_Series_Id **client_series_id**	32	
format_version	8	uimsbf
if (format_version == 0x01){		
ECI_Signature_v1 **series_cosignature**		
}		
}		
if (/* image cosignature */){		
ECI_Client_id **client_id**	32	
ECI_Data_Signature **image_cosignature**		
}		
}		

Semantics:

type: byte	Value in accordance with Table 5.2-2.
entity_id: integer	Unique **identifier** assigned to the signature in the context of the **Platform Operation Certificate**. In conjunction with cosignature_version field assigned to only one **permitted client image**.
version: integer	Increased (e.g. incrementing most significant bits) in case the **Platform Operation** changes its public key. This field's lesser significant bits can be used to represent (part of) the version of the client image series or client image for convenience of the **Platform Operation** managing revocation by client version using the version field in the **Platform Operation** client revocation list.
cosignature_id: ECI_PO_Cosignature_Id	Identification of the identifier of the cosignature on a client image. This field is included in the co-signature calculation.
client_tag: integer	Short form identifier for installation purposes to designate a client_type in a **Platform Operation** context. Only clients that can replace each other from a **user**'s perspective shall have the same client_tag value. Typically minor versions of a client are equivalent.
vendor_id: ECI_Vendor_Id	Id of the vendor **Certificate** for the **ECI Client Image**. This field can be used to locate the client image series or client image for which the cosignature is provided in this data structure
client_series_id: ECI_Client_series_id	Id of the client series **Certificate** for verifying an image. The type field of the client_series_id field shall match the **Platform Operation Certificates** child-type for client_image_series: see Table 5.2-2, and thus defines the correct selection of the alternate interpretations of the data-structure.
format_version	Version of the format of the **Certificate** definition that applies for the co-signature (see Table 5.2-1). This shall match the client **Certificate** version definition. The only valid value of this field that is defined is 0x01.
series_cosignature: ECI_Signature_v1	This is the cosignature by the **Platform Operation** Secret Key of client_image_series certificate. The data that is input to the signature calculation shall be defined a being identical to the client_image_series certificate, replacing the client_image_series_id by the cosignature_id of this data structure and replacing the extension field by a 4-byte extension carrying the original client_image_series_id field of the **Certificate**.
client_id: ECI_Client_Id	Id of the client image. The type field of the client_id field shall match the **Platform Operation Certificates** child-type for client_image: see Table 5.2-2 and thus defines the correct selection of the alternate interpretations of the data-structure.
image_cosignature: ECI_Data_Signature	This is the cosignature by the **Platform Operation** Secret Key of the client image. The data that is input to the signature calculation shall be defined as: the cosignature_id field followed by the data of client image file input to the client image signature calculation as defined in clause 7.6.1.

7.6 File formats

7.6.1 ECI Client Image File Format

The **ECI Client** Credentials contain the data needed to verify the **ECI TA** authenticity of an **ECI Client**. It shall use the following format as defined in Table 7.6.1-1.

Table 7.6.1-1: Client credentials definition

Syntax	No. of bits	Mnemonic
ECI_Client_Credentials {		
ECI_Certificate_Chain **client_chain**		
if (client_chain.chain_length == 0x1) { /* no client series; regular image */		
ECI_RL **client_rl**		
}		
ECI_Data_Signature **client_signature**		
}		

Semantics:

header: ECI_Client_Chain_Header	Header of the **ECI Client** chain file.
client_chain: ECI_Client_Chain	**Certificate Chain** for validating an **ECI Client Image**, starting with the Security Vendor Root Revocation List, ending with the Security Vendor Certificate for non Image Series based **ECI Clients**, or ending with the **ECI Client** series **Certificate** for Image Series based **ECI Clients**.
client_rl: ECI_RL	Revocation List for **ECI Client Image** Ids.
client_signature: ECI_Data_Signature	Signature to validate the **ECI Client Image**, the public key for which is provided by the **ECI Client chain**.

The **ECI Client Image** file is defined in Table 7.6.1-2.

Table 7.6.1-2: ECI Client Image file definition

Syntax	No. of bits	Mnemonic
ECI_Client_Image_File {		
magic = 'ECI'	24	uimsbf
image_header_version	8	uimsbf
ECI_Client_Credentials **credentials**		
if (image_header_version == 0x01) {		
if (credentials.client_chain.chain_length == 0x1) { /* regular image */		
ECI_Client_Id **client_id**	32	uimsbf
}		
if (credentials.client_chain..chain_length == 0x2) { /* Image Series image*/		
ECI_ Image_Target_Id _Id **target _id**	64	uimsbf
ECI_Client_Series_Id **client_series_id**	32	
}		
vendor_id	20	uimsbf
image_encrypted_flag	14	uimsbf
online_flag	1	uimsbf
Reserved	10	
for (i=0; i<n; i++) {		
client_image_byte	8	uimsbf
}		
}		
}		

Semantics:

magic: byte[3]	Magic number used for verification of the format of the following data. It has the value of the three 8-bit ASCII representations of the characters 'ECI'. The **ECI Host** shall check the value of this field to verify if an **ECI** file has the expected format for additional data integrity.
image_header_version: byte	Format version of the image header. Value 0x01 is the presently defined version; all other values are reserved. The **ECI Host** shall ignore any image with a version number that is not recognized.
credentials: ECI_Client_Credentials	**ECI Client Credentials** for verifying the authenticity of the **ECI Client Image**.
series_image: Boolean	Series image is not a field but a function computed from credentials indicating the presence of an **ECI Client** type series **Certificate**.
series_id: ECI_Client_Series_Id	**ECI Client** series ID of the Image Series of the following image. The **ECI Host** shall check the value before loading the **ECI Client Image**.
series_image_id: ECI_Client-series_Image_Id	Image ID in Image Series of the following image. The **ECI Host** shall check the value before loading the **ECI Client Image**.
client_id: ECI_Client_Id	**ECI Client** ID of the **ECI Client Image**. The **ECI Host** shall check the value before loading the **ECI Client Image**.
vendor_id: ECI_Vendor_Id	Vendor ID of the Security Vendor of the **ECI Client Image** as defined in the ECI_Vendor_Id structure in clause 7.4.2. The **ECI Host** shall check this field before loading a (new) **ECI Client Image**.
image_encrypted_flag: integer	This flag signals if the image is encrypted. If the value of this field is 0b0 the image is not encrypted. If the value of this field is 0b1 the image is encrypted.
online_flag: integer	This flag signals if the protocol to retrieve a key to decrypt the image requires online interaction with the provisioning server using a nonce. See clause 7.8.3
client_image_byte: byte	Sequence of bytes containing the client image.

In Table 7.6.1-2 "**ECI Host** shall check" means that the **ECI Host** shall verify that the values which it expects will match the value in the field.

The **ECI Client Image** signature shall be computed over all data in the file following the credentials field.

7.6.2 Platform Operation Chain Data

The **ECI Client Image** file is defined in Table 7.6.2-1.

Table 7.6.2-1: Platform Operation Chain File Definition

Syntax	No. of bits	Mnemonic
ECI_Operation_Certificate_File {		
magic = 'EPC'	24	uimsbf
version	8	uimsbf
if (version == 0x01) {		
ECI_Certificate_Chain **operation_chain**		
ECI_RL po_client_rl		
client_image_count	16	uimsbf
for (i=0; i<client_image_count; i++) {		
ECI_PO_Client_Cosignature_Data **po_client_data**		
}		
ECI_RL po_client_rl		
}		
}		

Semantics:

magic: byte[3]	Magic number used for verification of the format of the following data. It has the value of the three 8-bit ASCII representations of the characters 'EPC'. The **ECI Host** shall check the value of this field to verify if an **ECI** file has the expected format for additional data integrity.
Image_header_version: byte	Format version of the image header. Value 0x01 is the presently defined version; all other values are reserved. The **ECI Host** shall ignore any image with a version number that is not recognized.
operation_chain: ECI_Client_Chain	**Certificate Chain** for validating an **ECI Client Image**, starting with the Operator Root revocation list, ending with the **Platform Operation Certificate**.
po_client_rl: ECI_RL	This is the **Platform Operation** client revocation list used to validate the client image cosignatures. The **ECI Host** shall check the cosignature_ids in the po_client_data as part of the verification of the cosignature.
client_image_count: integer	Number of signature data structures for client images in the following loop.

In Table 7.6.2-1 "**ECI Host** shall check" means that the **ECI Host** shall verify that the values which it expects will match the value in the field.

7.6.3 Revocation data files

On behalf of the **ECI Client** loader there are two types of revocation data files. Both files use the ECI_Revocation_Data_File format defined in Table 5.5-2.

The **ECI Client** revocation data file uses father_type equal 0x0 (R**oot Certificate**) and sub_type equal to the Vendor revocation list type. The revocation_data conforms to the constraint that the leaf revocation list in the trees are **ECI Client** revocation lists.

The **Platform Operation** revocation data file uses father_type equal 0x0 (R**oot Certificate**) and sub_type equal to the **Operator** revocation list type. The revocation_data conforms to the constraint that the leaf revocation list in the trees are **Platform Operation** revocation lists.

7.7 ECI Client resources transport protocols

7.7.1 General and profiling

This clause defines the application of protocols in **CPEs** and **Platform Operations**.

The broadcast protocol does not provide for an **Image Series** option. Series based images are only foreseen for IP-connected devices.

CPE supporting both broadcast and online access to **ECI Client** resources shall use broadcast access with a higher priority (unless indicated otherwise in the present document) so as to offload online traffic, but may use online access in case of urgency (**user** waiting), and shall use online access in case minimum access frequencies cannot be met over the broadcast network.

7.7.2 Broadcast transport protocol

7.7.2.1 Introduction

ECI requires support data for various functions on behalf of the **ECI Client** and/or the **ECI Host** to be able to initialize and support the **ECI Client**. The same transport protocol is used for all types of data, and is defined in this clause. It is closely related to the protocol used to download **ECI Host Image** files.

For broadcast delivery data is broken up in buckets using a hash function on the access index used by the **CPE** to determine if it needs the data. By using buckets the amount of data that the **CPE** needs to download is significantly reduced and the selectivity of monitoring changes in the data that are actually relevant for the **CPE** is improved.

The following separate carousel groups are defined (by content type):

- ECI Client Images (per Security Vendor).
- **ECI Client** revocation data, structured in buckets based on the <client_id,client-version_major> and vendor_id index.
- Platform operation **Certificate Chain**.
- **Platform Operation** revocation data, structured in buckets based on the provider_id and operator_id index.
- **ECI Host Image** revocation data, structured in buckets.
- **ECI** AS_setup **ECI Client** initialization data, structured inbuckets.
- Carousel groups are defined for import and export data structures (see clause 9.8).
- Carousel groups are defined for operator proprietary data.

All DSMCC carousel parameters shall comply with ETSI EN 301 192 [15].

An **Operator** may use multiple carousels on separate multiplexes to transmit all required data. However, for any specific **ECI Client** the **ECI Host** shall only have to monitor the updates of a single location DII of a data carousel.

7.7.2.2 Credential and revocation data handover to Operator

The data formats and protocols for transferring credentials and revocation lists to an operator do not form part of the **ECI** specification.

7.7.2.3 Security Vendor to Operator handover

The data formats and protocols for transferring content from **Security Vendor** to operator do not form part of the present document.

7.7.2.4 PSI signalling

The carousels shall use the stream_identifier_descriptor [19] in the PMT to tag the stream used for transmitting the carousel so as to permit referencing by the data_broadcast descriptor in the SI.

The carousels shall use a data_broadcast_id_descriptor with data_broadcast_id as defined in Table 7.7.2.4-1.

Table 7.7.2.4-1: Data Broadcast ID value for ECI specific carousels

Data_broadcast_id value	Meaning
Allocated by DVB project office, see broadcast-id value defined in ETSI TS 101 162 [20].	**ECI Operator** specific client support data carousel

The selector bytes of the data_broadcast_id_descriptor shall follow the structure as defined in Table 7.7.2.4-2.

Table 7.7.2.4-2: Carousel ID structure for ECI DVB DSMCC data carousels

Syntax	No. of bits	Mnemonic
ECI_carousel_id_structure {		
version	8	uimsbf
if (version == 0x01){		
operator_id	20	uimsbf
platform_**operation_id**	20	uimsbf
}		
}		

Semantics:

version: integer	Version of the structure; at present only 0x01 is defined. All other values are reserved. **CPEs** encountering a version other that 0x01 shall ignore this descriptor.
operator_id: ECI_Operator_Id	**ECI** ID of **Operator** (which is defined for any **Operator Certificate**) of the **Platform Operation** of the carousel.
platform_operation_id: ECI_Platform_Operation-Id	As per **Platform Operation Certificate**: ID of the **Platform Operation**.

7.7.2.5 SI signalling

7.7.2.5.1 Data Carousel location Signalling via Data Location Linkage Descriptor

The **ECI Client** data location linkage descriptor is an **ECI** private DVB linkage descriptor [20]. This linkage descriptor assists a **CPE** with the location of the multiplex carrying an **ECI Client** data carousel for a specific **Platform Operation**. This linkage descriptor is carried in the NIT or BAT. The **ECI Client** Data location linkage descriptor shall always be preceded in the table section by a DVB private data specifier descriptor [20] with private_data_specifier field value equal to "ECI" as defined in [20]. This descriptor may appear in the NIT or BAT multiple times. This linkage descriptor shall be carried in networks and bouquets with more than 4 multiplexes.

With reference to the definition of the linkage descriptor as defined in ETSI EN 300 468 [19] and ETSI TS 101 211 [21] the fields of the **ECI Client** data location linkage descriptor have the following specific application:

- **service_id:** may be set to 0x0000 to signal no specific service_id is signalled.
- **linkage_type:** value 0x80 signalling an **ECI** Client Data location linkage descriptor.

The private data byte field of the **ECI Client** Data location linkage descriptor shall carry the structure defined in Table 7.7.2.5.1-1.

Table 7.7.2.5.1-1: Private data structure for ECI Client Data carousel location linkage descriptor

Syntax	No. of bits	Mnemonic
ECI_client_data_location {		
version	8	uimsbf
if (version==0x01){		
for (i=0;i<n; i++){		
operator_id	20	uimsbf
platform_**operation_id**	20	uimsbf
}		
}		
}		

Semantics:

version: integer	Version of the structure; at present only 0x01 is defined. All other values are reserved. **CPEs** encountering a version other that 0x01 shall ignore this descriptor.
operator_id: ECI_Operator_Id	**ECI** ID of Operator (which is defined for any **Operator Certificate**) of the **Platform Operation** of the carousel. The value 0x00000 signals any operator.
platform_operation_id: ECI_Platform_Operation-Id	As per **Platform Operation Certificate**: ID of the **Platform Operation**. The value 0x0000 signals any **Platform Operation**.

Network and bouquet operators may use wildcard specifiers (value 0x00000) for operator_id or platform_operation_id to link to a multiplex carrying one or more **ECI Client** data carousels. For efficiency reasons it is recommended that such signalling is constrained to assist **CPEs** to inspect as few multiplexes as required in order to locate a specific **Platform Operation** carousel.

It is recommended only a single **ECI Client** data carousel location linkage descriptor to a multiplex is used in a NIT or BAT, and that all applicable carousels located in that multiplex are listed in one ECI_Client_data_location structure.

7.7.2.5.2 ECI Client emergency download descriptor

In order to indicate the need to urgently replace an **ECI Client Image** one or more ECI_client_emergency_download descriptors can be placed in NIT, BAT or in one of the SDT entries for a service for which the flagged **ECI Client** can provide access. The **ECI Host** shall be able to retrieve this descriptor from any of the tables in which it appears in any of the currently tuned multiplexes and perform the associated processing and use any spare tuner to access relevant multiplexes to acquire this descriptor with a worst case 30 minutes interval period.

The ECI_client_emergency_download_descriptor permits targeting specific operation platforms and specific host types in order to minimize the disturbance caused by emergency updates.

When the **ECI Host** finds a new ECI_client_emergency_download descriptor (verified by table-origin and emergency_id field) it shall match its Host and Client configuration against the targeting information in the descriptor. If a target match is found and the version of the currently installed client image requires an update, the Host shall perform this update in accordance with the emergency_indicator. This can cause a disruption for ongoing **user** activities in the **CPE** in case of resource conflicts.

The **ECI** operation descriptor is a DVB private descriptor and shall always be preceded in the table it appears in by the DVB private_data_specifier_descriptor using the **ECI** private_data_specifier_field (see ETSI EN 300 468 [19]). The syntax of the descriptor is defined in Table 7.7.2.5.2-1.

Table 7.7.2.5.2-1: ECI_Client_Emergency_Download_Descriptor

Syntax	No. of bits	Mnemonic
ECI_client_emergency_download_descriptor{		
descriptor_tag	8	uimsbf
descriptor_length	8	uimsbf
/* main loop */		
main_loop_nr	8	uimsbf
for (i=0; i<main_loop_nr; i++){		
/* target platform */		
platform_operation_tag	8	uimsbf
/* host target loop */		
host_nr	8	uimsbf
/* host id target loop */		
for (j=0; j<host_nr; j++){		
manufacturer_id	20	uimsbf
cpe_type_id	20	uimsbf
host_version	8	uimsbf
}		
/* client image loop */		
client_nr		
for (j=0; j<client_nr; j++){		
emergency_indicator	4	uimsbf
client_tag	4	uimsbf
min_client_version_major	8	uimsbf
min_client_version_minor	8	uimsbf
}		
}		
/* private data till end of descriptor*/		
for (i=0; i<n; i++){		
private_data_byte	8	
}		
}		

Semantics:

descriptor_tag	**ECI** private tag value for descriptor_tag: see ETSI GR ECI 004 [i.10].
descriptor_length	See ETSI EN 300 468 [19].
main_loop_nr	Number of entries in the main loop. The separate main loop entries shall be evaluated separately by the **ECI Host**, i.e. have OR semantics. The various elements in one loop entry shall have AND semantics.
platform_operation_tag	Tag value for the **ECI** platform as listed in the ECI_platform_operation_descriptor in the NIT/BAT. The **ECI Host** shall consider an emergency update if the platform_operation matches to the platform_operation of one of the installed **ECI Clients**.
host_nr	Number of entries in the host target loop; value 0 means all **ECI Hosts** are targeted. The loop entries shall have OR semantics; i.e. if any host target specification matches the target condition in the main loop has a matching state.
manufacturer_id	Manufacturer_id of the host targeted by an emergency update. The host shall consider an emergency update if the value of this field matches the manyufacturer_id of the **ECI Host**.
cpe_type_id	Value as defined by ECI_CPE_Type_ID in Table 6.2.2.1-2. The **ECI Host** shall consider an emergency update if the cpe_type_id of the Host matches to the value of this field. Cpe_type_id.cpe_type equal 0x000 shall mean any **ECI Host** cpe_types is a match (and cpe_model and host-version shall be ignored). cpe_type_id.cpe_model equal 0x00 shall mean any **ECI Host** cpe_model is a match (and host version shall be ignored).
host_version	The **ECI Host** shall consider an emergency update if and only if its host version is less or equal to the value in this field. See note.

client_nr	Number of entries in the client image loop. The loop entries shall have OR semantics and all client images that match shall be considered for an emergency update.
emergency_indicator	The **ECI Host** shall use the value of this field to select the appropriate behaviour for starting the download and the subsequent update of the client as defined in Table 7.7.2.5.2-2.
client_tag	Tag value identifying the **ECI Client** as listed in the ECI_platform_operation_descriptor in the NIT/BAT that matches the platform_operation_tag field in the same main loop. The **ECI Host** shall consider an emergency update if the referred vendor_id and client_id match with one of the installed clients in the **ECI Host**.
min_client_version_major	This field represents the minimally acceptable major version number for the client image. The **ECI Host** consider to perform an emergency update if a Client is installed matching with client_tag that has a major version less than the value of this field.
min_client_version_minor	This field represents the minimally acceptable minor version number for the client image. The **ECI Host** shall consider to perform an emergency update if an **ECI Client** is installed matching with client_tag that has a minor version less than the value of this field and the major version equal to min_client_version_major.
client_id	Client identifier of an **ECI Client** that provides decryption services for services with platform_operation_tag, as defined in Table 7.4.4-1.
private_data_byte	Private data: the content may be defined by the operator managing the broadcast of this descriptor.
NOTE: Field value equal 0xFF implies all host versions match.	

Table 7.7.2.5.2-1 defines a number of conditions in the main loop (having AND semantics) that shall be met in order for the **ECI Host** to consider performing an emergency update. If all these conditions are met the **ECI Host** shall perform an emergency download and installation of one or more client images in accordance with the emergency_indicator field for that client.

Table 7.7.2.5.2-2: ECI_Client_emergency_download_descriptor emergency_indicator field values

Name	Value	Description
System emergency	0x01	The **ECI Host** shall download the new client image and install it as quickly as possible interrupting ongoing **user** instigated activities if so required (see note 1).
Client emergency	0x02	The **ECI Host** shall download the new client image and install it before any media-handle session for that Client is opened. Any ongoing media-handle sessions for this Client shall first be terminated (see note 2).
Client urgency	0x03	The **ECI Host** shall download the new client image and install it on the first occasion this does not cause any disruption to any **user** instigated activity or latest during the next power-up event (see note 3).
RFU	other	Reserved for future use.
NOTE 1: Platform operators can use this for instance if the present Client can cause harm to the **ECI Host** and/or other Clients and has to be replaced immediately. NOTE 2: Platform operators can use this for instance if the present Client has very poor performance for decrypting services. NOTE 3: Platform operators can use this for instance if the present Client has serious deficiencies for decrypting services but can perform reasonably for regular use cases.		

7.7.2.6 Carousel compatibility descriptor

The compatibilityDescriptor used in DVB DSMCC data carousels [15] shall be used in the DSI DII messages.

The compatibilityDescriptor provides the information on the type of data transported in a carousel group. The specifierData() shall contain the **ECI** OUI. Table 7.7.2.6-1 defines the applicable fields of the compatibilityDescriptor in **ECI Client** data carousels.

Table 7.7.2.6-1: ECI Data Carousel content types

Descriptor type field	Group Purpose	Model field	Version field	Bucket Index to compute the module ID
0xA0	**ECI Client Images** and credentials files for one Vendor	Vendor_id of the **Security Vendor** of the images		Freely assigned
0xA2	**ECI Client** revocation data files (as buckets)	platform_operation_id		= Vendor_id + <Client_type, client_version_major> (see note)
0xA3	**Platform Operation** chain file	platform_operation_id, platform_operation_version		Freely assigned
0xA4	**Platform Operation** revocation data files (as buckets)	platform_operation_id		= Operator_id + provider_id
0xA5	**ECI Host** revocation data files (as buckets)	platform_operation_id		= Manufacturer_id + cpe_type_id
0xA6	AS_setup files (as buckets)	platform_operation_id		target_id for **CPE**
0xA7-0xAA	UI Application container (see clause 9.4.3.4.2	Defined by operator		Freely assigned
0xB0	Export tree file	platform_operation_id (of exporting **ECI Client**)		Freely assigned
0xB1	Import chains file	platform_operation_id (of importing **ECI Client**)		Freely assigned
0xB2	Import authentication chains file	platform_operation_id (of importing **ECI Client**)		Freely assigned
0xB8-0xBF	Operator proprietary format	Defined by operator		Defined by operator
Other values	reserved			
NOTE: Concatenation of the two fields, with most significant one as first argument, making up a 20-bit number.				

The bucket index computation shall use 32 bit modular integer arithmetic and is defined in clause 7.7.2.7.

7.7.2.7 Carousel DSI

In case the carousel is a two layer carousel, the DSI shall contain a complete index of the groups in the carousel (i.e. one loop entry per DII).

The compatibilityDescriptor is defined in Table 7.7.2.6-1. The DII non-loop fields shall meet the following constrains:

- block Size: at least 512 bytes, for groups with larger modules at least 2 kbyte is recommended;
- tCDownloadScenario: shall be at least 4-times the slowest DDB repeat message in the group. TCDownload shall also meet the maximum constraints in Table B.4-1;
- numberOfModules: reflects the number of modules for regular carousels and the number of buckets (each mapped onto one module) for bucketized data. For a **Platform Operation Certificate Chain** data the value shall be 1.

The values for tCDownloadScenario below reflect the timeout period for acquisition of a full data item by a **CPE**. It shall be at minimum 4-times the slowest DDB repeat time of any of the modules of the group. The values for the various items are defined in clause B.4.

The following module loop fields shall meet the following constraints:

- moduleId: bit 15 to 8 shall be the same as the LSB of the groupId in the corresponding groupInfo structure in the DSI. Bits 7 to 0 are assigned in accordance with Table 7.7.2.7-1.
- moduleVersion: application depends on carousel type, shall be in accordance with Table 7.7.2.7-1.
- moduleInfoLength: 0 for all **ECI** carousels.

Table 7.7.2.7-1: ECI Carousel group parameters

Group type	ModuleId bit 7..0	ModuleVersion	ModuleInfo
Client Images	client_type	client_version	None
Client Revocation Data	bucket_number	Incremented on every update	None
Platform Operation client chain	Assigned by **Operator**	Incremented on every update	None
Platform Operation Revocation Data	bucket_number	Incremented on every update	None
ECI Host Revocation Data	bucket_number	Incremented on every update	None
ECI AS_setup data	bucket_number	Incremented on every update	None

For bucketized number, the bucket number (equal to module_id bit [7..0]) shall be computed from the index with a simple modulo operation:

```
bucket_number = bucket Index % numberOfModules
```

7.7.2.8 Carousel DDB

No specific requirements.

7.7.2.9 Dynamic carousel behaviour

The carousel version numbering and DSI, DII updating shall be in accordance with ETSI TR 101 202 [16]. This implies that any update of a module shall be reflected in the version number of the module, its DII and cascading upwards to the DSI (if present).

CPE implementation can monitor changes in their target modules so as to follow any dynamic update during normal operation.

7.7.3 Web transport protocols

7.7.3.1 Introduction

The various required data items can be retrieved by the **ECI Host** from a server to be appointed by the **Operator**.

The interface shall use direct HTTPS requests as specified in clause 9.4.4.6 and follows RESTfull design principles [i.2] with the request encoded as a combination of URL extension and query parameters, and the response encoded as a binary file.

The HTTP server shall respond with one of the following status codes:

- 200: OK (file requested is returned).
- 302 FOUND: redirects to defer the request to another server; http request to be repeated on the URL returned.
- 404: Item not present on server.
- 500 .. 599: Server error.

The specification of the URLs used for the requests uses a 'Bachus Naur' style specification. The names of the symbols that correspond to fields in **ECI** data structures shall be represented as the hexadecimal representation (string of charachters '0' .. '9' , 'A' .. 'F') of its value with twice the number of digits as bytes used to represent the number in **ECI** internal binary data structures. The server shall ignore any additional query parameters it does not recognize.

7.7.3.2 ECI Web API overview

The **Operator** shall support an online server that responds to the following HTTP1.1 [18] GET request following the following URL syntax and semantics:

URL ::= base-url '/' 'eci' major '_' minor '/' tail.

major and **minor** shall reflect major and minor number of the protocol version in decimal representation without leading zeros. The current version is 1.0. The definition of tail is given in Table 7.7.3.2-1.

Table 7.7.3.2-1: Definition of tail

```
tail ::= host_version |
         host_images |
         host_image_version |
         host_image |
         po_check |
         po_client_check         po_certchain |
         po_revocation  |
         client_version |
         client_credential_version |
         client_image |
         client_revocation |
         as_request |
         tail_extension*.
```

The tail_extension indicates various extension options to the **ECI** Web API as defined in the present document.

7.7.3.3 Web API ECI Host related requests

The following **ECI Host** related web API requests are defined:

- host_version ::= 'host-version ' '?target-id=' target_id.
  This shall return the latest version of the **ECI Host Image** set for the **CPE** identified by **target_id**.

- host_images ::= 'hi-images' ' ?target-id=' target_id.
  This shall return the latest number of images for an **ECI Host** for the **CPE** identified by **target_id**.

- host_image_version ::= 'hi-version' ' ?target-id=' target_id '&image-id=' image_id .
  This shall return the latest version of the **ECI Host Image** file **image_id** for the **CPE** identified by **target_id**.

- host_image ::= 'host-image' ' ?target-id=' target_id '&image-id=' image_id.
  This shall return the latest **ECI Host Image image_number** for the **CPE** identified by **target_id**.
  image_number=="FF" shall return the **ECI Host** credential file for the **ECI Host Images**, including the latest revocation data.

For **ECI Host** related requests, the server for a **Platform Operation** can support **ECI Hosts** for any **CPE** type it desires. If it supports a **CPE** type it shall support the complete latest set of **ECI Host Images** and the corresponding **host_image_version**, **host_images** and **host_revocation** queries. The format of the file returned is the ECI_Host_Version_File as defined in Table 7.7.3.3-1.

Table 7.7.3.3-1: ECI Host version file definition

Syntax	No. of bits	Mnemonic
ECI_Host_Version_File {		
magic = 'RHVE'	32	uimsbf
host_version	8	uimsbf
}		

Semantics:

magic: byte[4]	8-bit ASCII representation of the string 'RHIM'.
host_version: integer	Version number of the **ECI Host Certificate**.

The format of the file returned is the ECI_Host_Images_File as defined in Table 7.7.3.3-2.

Table 7.7.3.3-2: Host Images file definition

Syntax	No. of bits	Mnemonic
ECI_Host_Images_File {		
magic = 'RHIM'	32	uimsbf
host_images	8	uimsbf
}		

Semantics:

magic: byte[4]	8-bit ASCII representation of the string 'RHIM'.
host_images: integer	Number of **ECI Host Images** supported by the **CPE** type identified in the request.

The format of the file returned is the ECI_Host_Image_Version_File as defined in Table 7.7.3.3-3

Table 7.7.3.3-3: Host Image Version File Syntax

Syntax	No. of bits	Mnemonic
ECI_Host_Image_Version_File {		
magic = 'RHIV'	32	uimsbf
host_image_version	16	uimsbf
}		

Semantics:

magic: byte[4]	8-bit ASCII representation of the string 'RHIV'.
host_image_version: integer	**ECI Host Image** version of the **ECI Host Image** identified by the request.

7.7.3.4 Web API Platform Operation related requests

The server of the **Platform Operation** shall support the following requests on behalf of the **Platform Operation** id's it supports:

```
po_check ::= 'po_check' '/' operator_id '/' platform_operation_id .
```

This shall return the revocation status of the **Certificate** issued for **operator_id**, **platform_operation_id** in the file format defined in Table 7.7.3.4-1. The server for a **Platform Operation** shall at minimum support its own **Platform Operation Certificates** in operation through this interface.

```
po_client_check ::= 'po-client-check' '/' operator_id '/'
                    platform_operation_id '?cosignature-id=' cosignature_id .
```

This shall return the platform revocation status of the **ECI Client Image** for cosignature_id according to the latest platform operation client revocation list. See Table 7.7.3.4-2.

```
po_certchain ::= 'po-chain' '/' operator_id '/' platform_operation_id .
```

This shall return the latest **ECI Client** chain for the **Platform Operation** identified by **operator_id**, **platform_operation_id** as defined in Table 7.6.2-1. The server for a **Platform Operation** shall at minimum support its own **Platform Operation Certificates** in operation through this interface.

```
po_revocation_ ::= 'po-revoc' '/' operator_id .
```

This shall return the latest **Platform Operation** revocation data file containing the revocation list for the **Operator** identified by **operator_id**. The server shall at least support the latest revocation data for the **Operator** of its own **Platform Operation**. **ECI Hosts** shall use this API to attempt to acquire the latest revocation data of all stored **ECI Clients**.

Table 7.7.3.4-1: Platform Operation Check File Syntax

Syntax	No. of bits	Mnemonic
ECI_PO_Check_File {		
magic = 'RPCH'	32	uimsbf
non_revoked_certificate_flag	8	uimsbf
}		

Semantics:

magic: byte[4]	8-bit ASCII representation of the string 'RHIV'.
non_revoked_certificate_flag: byte	Value 0x00 in case the **Certificate** of the **Platform Operation** ID identified by the request was revoked, 0x01 otherwise.

Table 7.7.3.4-2: Platform Operation Client Check File Syntax

Syntax	No. of bits	Mnemonic
ECI_PO_Client_Check_File {		
magic = 'RPCC'	32	uimsbf
non_revoked_certificate_flag	8	uimsbf
}		

Semantics:

magic: byte[4]	8-bit ASCII representation of the string 'RHIV'.
non_revoked_certificate_flag: byte	Value 0x00 in case the client image associated with cosignature_id field of the request was revoked according to the latest Platform Operation Client Revocation list of the Platform Operation, 0x01 otherwise.

7.7.3.5 Web API client requests

The **Operator**'s server shall support the following requests on behalf of the clients required by its **Platform Operation**'s id:

```
client_version ::= 'client-ver' '/' vendor_id '/'
                   client_type '/' client_version_major .
```

- This shall return a Client Version File (see Table 7.7.3.5-1) containing the latest version of **ECI Client Image** for a client identified by **vendor_id**, **client_type**. The server shall at minimum support the clients used to operate its own **Platform Operation** services.

```
client_credential_version ::= 'client-ver' '/' vendor_id '/'
                              client_type '/' client_version_major .
```

- This shall return a Client Credential Version File (see Table 7.7.3.5-2) containing the latest version of **ECI Client** Credentials for a client identified by **vendor_id**, **client_type**. The server shall at minimum support the clients used to operate its own **Platform Operation** services.

```
client_image ::= 'client-img' '/' vendor_id '/'
                 client_type '/'client_version_major
                 ['? &target-id=' image_target_id] .
```

- This shall return the latest **ECI Client Image** file for a client identified by <vendor_id, client_type, client_version_major>. In case of an **Image** image_target_id of type ECI_Image_Target_Id is provided as a query parameter. The server shall at least support the **Vendors** of **ECI Clients** used to operate their own **Platform Operation** services. **ECI Hosts** shall use this API to attempt to acquire the latest revocation data of all stored **ECI Clients**.

```
client_revocation_data ::= 'client-revoc' '/' vendor_id .
```

- This shall return the latest **ECI Client** revocation data file for a client identified by **vendor_id**. The server shall at minimum support the clients used to operate its own **Platform Operation** services.

Table 7.7.3.5-1: Client Version File Syntax

Syntax	No. of bits	Mnemonic
ECI_Client_Version_File {		
magic = 'RCVE'	32	uimsbf
client_version	16	uimsbf
emergency_download_descriptor		
}		

Semantics:

magic: byte[4]	8-bit ASCII representation of the string 'RCVE'.
client_version: integer	Latest client version of the client type identified in the request.
emergency_download_descriptor	An **ECI_client_emergency_download_descriptor** in which the **ECI Host** shall assume platform_operation_tag shall match the Platform Operation of the provider of the web-api of the client and the client_tag shall match the client image as requested in the web api parameters.

Table 7.7.3.5-2: Client Credential Version File Syntax

Syntax	No. Of bits	Mnemonic
ECI_Client_Credential_Version_File {		
magic = 'RCCV'	32	uimsbf
root_version	8	uimsbf
vendor_rl_version	24	uimsbf
eci_vendor_id	32	uimsbf
padding(4)		
client_rl_version	24	uimsbf
eci_client_id	32	uimsbf
}		

Semantics:

magic: byte[4]	8-bit ASCII representation of the string 'RCCV'.
root_version: integer	Root version (as defined in Table 5.3-1) of latest **ECI Client** credentials.
vendor_rl_version: integer	Security Vendor revocation list version number of latest **ECI Client** credentials.
eci_vendor_id: ECI_Vendor_Id	ECI_Vendor_Id (as defined in Table 7.6.1-2) of the latest **ECI Client** credentials.
client_rl_version: integer	Client revocation list version number of latest **ECI Client** credentials.
eci_client_id: ECI_Client_Series-Id	ECI_Client_Series_Id (as defined in Table 7.6.1-2) of the latest **ECI Client** credentials.

7.7.3.6 Web API AS_setup requests

In case the **Operator** supports online registration of encrypted-mode **ECI Clients** the following request shall be supported:

```
as_request ::= 'as_request' '/' vendor_id '/' eci_client_id
               '?&image-target-id=' target_id '&nonce=' nonce].
```

The request returns the as_setup file for the specified client (<vendor_id,eci_client_id>) and the for the **CPE** specified by ECI_Image_Target_Id target_id. The type of eci_client_id can be ECI_Client_Id or EI_Client_Series_Id. Nonce is the value for the nonce as specified by the **ECI Client Image** decryption protocol. See clause 7.8.4.2 for more details.

7.8 Platform Operation ECI Client installation

7.8.1 Scope and Profiling

The **Platform Operation** can select the security options for **ECI Client** installations and signal this using the image_encrypted_flag and the online flag in the **ECI Client Image** file (see Table 7.6.1-2):

- "**ECI Client** installation mode with unencrypted **ECI Client Image** file", in which the (latest version of the) **ECI Client** as proposed by the signalling is defined in clause 7.2 for decoding of services is downloaded and **ECI Client** initiation takes place.
- "**ECI Client** installation mode with encrypted **ECI Client Image** file ", which in addition to the first mode permits the **Platform Operation** to encrypt the **ECI Client Image** and authenticate as defined in ETSI GS ECI 001-5-1 [4] **ECI Client** decryption is **ECI Host** specific and encompasses **ECI Host** version verification, thus further ensuring that the confidentiality of the **ECI Client** after decryption is guaranteed by not permitting decryption on unknown or compromised **ECI Hosts**. An ECI_Image_Target_Id is required in case a **CPE** is not connected to an online network. In this use case the ECI_Image_Target_Id needs to be sent to the security headend manually.

The protocol for both versions of **ECI Client** initiation is defined in the rest of this clause.

Platform Operations operating online **CPEs** in installation mode with encryption can force the use of the latest **ECI Client** by using a AS generated nonce in the decryption protocol with the **Platform Operation** server for the **ECI Client** (see clause 7.7.3.6).

Profiling rules:

- In case online registration is offered by the **Platform Operation** (the signalling is defined in clause 7.2) and the **CPE** is able to access online services the **CPE** shall use the online registration protocol.
- Broadcast-reception capable **CPEs** shall be able to perform the broadcast registration protocol. Broadcast mode requires registration of the **CPE** on initial **Platform Operation** registration.
- Platform Operations supporting broadcast networks supporting **CPEs** without simultaneous online connectivity shall support broadcast mode registration. The details for the **user** entering the registration information for a **CPE** shall obey the applicable formatting rules.

7.8.2 ECI Client installation mode with unencrypted ECI Client Image file

At the start of **ECI Client** initialization the **ECI Host** reserves an **AS slot** for the **Platform Operation,** resets the AS-slot and loads the **Platform Operation** public key into the **AS slot** as defined in ETSI GS ECI 001-5-1 [4].

If required the **ECI Host** downloads the **ECI Client**, stores it in NV RAM for future retrieval and starts it up. The **ECI Client** will further guide the **user** through installation. Installation may involve the **user** manually sending the **CPE** ECI_Image_Target_Id value target_id to the headend in case the **CPE** does not have an online connection for broadcast system security registration.

On any subsequent reboot the **ECI Host** will re-initialize t the **ECI Client**.

7.8.3 ECI Client installation mode with encrypted ECI Client Image file

This mode of operation uses an encrypted download of the **ECI Client Image** using an **Operator** selected key. This operator selected key is encrypted and carried in an **as_setup** structure.

At the start of **ECI Client** initiation the **ECI Host** reserves an **AS slot** for the **Platform Operation** resets the AS-slot and loads the **Platform Operation** public key into the **AS slot**:

- The **ECI Host** shall distinguish two modes for **as_setup** retrieval: **Registration mode**: this mode is entered if the **ECI Client** is initiated for the first time or the POPK or the **ECI Client** version has changed or the client operates in online re-registration mode using a unique nonce for every re-registration. The **as_setup** structure for the **CPE** shall be retrieved from the **Platform Operation** network.

- **Registered mode**: the previous **as_setup** structure is retrieved from NV memory. In case of any pending **ECI Client** or **ECI Host** version change the **ECI Client** should warn the **user** to initiate or unblock such a download (under default download settings these should normally take place automatically within a reasonable time frame). Downloading of a new **ECI Client** will also require a new **as_setup** structure.

In the registration mode the **ECI Host** shall perform the following actions for retrieving a new **as_setup** structure:

1) The **ECI Host** initializes the AS-slot and retrieves:

   - The **CPE**'s ECI_Image_Target_Id value target_id;

   - A nonce (128 bit) retrieved from the **AS slot** through application of the **getAsSlotRk** function (see ETSI GS ECI 001-5-1 [4]) in case of online registration.

2) The **ECI Host** shall send the above information to retrieve an **as_setup** message from the **Platform Operation**:

   - In case of a **broadcast registration** the **ECI Host** shall present the target_id on the screen with the **Platform Operation** registration dialogue box. The **ECI Host** shall retrieve the **as_setup** structure from the AS setup carousel (see clause 7.7.2).

NOTE 1: In case a platform provides multiple **ECI Client** types the **Platform Operation** may request the **user** to also provide some additional information in order to provide the **as_setup** for the appropriate **ECI Client** type.

NOTE 2: The **Platform Operation** may assume the **CPE** has downloaded the latest **ECI Client Image** version, and provide the **as_setup** structure only for that **ECI Client Image**.

   - In case of an **online registration** the **CPE** shall register the client identification, the **CPE**'s target_id and nonce using the web-API in clause 7.3.3.

NOTE 3: The **Platform operation** can decide to apply the nonce to ensure renewed registration on every **ECI Host** re-initialization event.

Following the **as_setup** acquisition sequence in registration mode, or having recovered the **as_setup** structure from NV-memory in registered mode, the **ECI Host** shall initialize the AS and attempt to load the encrypted **ECI Client**:

1) Load the **as_setup** structure in the AS using the **reqAsClientImageDecrKey** message. Load the **ECI Client** certificate client chain into the AS. Load the **Platform Operation** Client Revocation list and the **Platform Operation** client co-signature. The following failure cases shall at least be reported to the **user** in an intelligible way or be handled automatically:

   a) Old **ECI Host** version - the **ECI Host** or its credentials need to be updated.

   b) Old **ECI Client** version - the **ECI Client** or its credentials need to be updated.

2) Decrypt the image using the AS computed Client **Image** Key if required and authenticate the **ECI Client Image** using the **ECI Client** Signature and **Platform Operation** co-signatures

3) Fail in case of validation error.

The as_setup structure and as_setup_file format shall comply with the definition in Table 7.8.3-1.

Table 7.8.3-1: AS-Setup Structure, File and Bucket File

Syntax	No. of bits	Mnemonic
ECI_As_Setup {		
as_version	8	uimsbf
if (as_setup_version == 0x01) {		
vendor_id	20	uimsbf
if (/* client image regular */){		
ECI_Client_id **client_id**		
}		
if (/* client image series */){		
ECI_Client_Series_Id **series_id**		
}		
ECI_Image_Target_Id **target_id**		
as_tag	16	uimsbf
online	1	uimsbf
padding(4)		
EciRootState **min_root_state**	32	
InputV inputV		
symKey eKey		
Extension **extension**		
}		
}		
ECI_As_Setup_File {		
magic file = 'AES'	24	uimsbf
as_setup_file_version	8	uimsbf
if (as_setup_version == 0x01){		
ECI_As_Setup **as_setup**		
}		
}		
ECI_As_Setup_Bucket_File {		
magic_bucket_file = 'AEB'	24	uimsbf
as_setup_bucket_version	8	uimsbf
if (as_setup_version == 0x01){ for (i=0; i<n; i++) {		
ECI_As_Setup **as_setup_item**		
}		
}		
}		

Semantics:

vendor_id: integer	Security Vendor of the **ECI Client** this as_setup is intended for.
client_id: ECI_Client_Id	ID of the **ECI Client** this as_setup is intended for. The preceeding if statement uses the type-field client_id: it should correspond to "regular client image".
series_id: ECI_Client_Series_Id	ID of the **ECI Client Series** this **as_setup** is intended for. The preceeding if statement uses the type-field client_id: it should correspond to "client image series".
target_id: ECI_Image_Target_id	ECI_Image_Target_Id identifying the **CPE** for which this message is intended.
as_tag: integer	Tag to indicate the version of the as_setup structure for the above target. The value should change on any change of the **as_setup** structure for this target; e.g. increment.
online: bool	If true this message requires the slot-nonce to be used in the AK mechanism; if false no nonce is required. Note: this bit shall only be set in case of a working online connection.
min_root_state: minEciRootState	Minimum Root state (minimum root version number, minimum root revocation list number) to be applied for validating loaded **ECI Host** and **ECI Clients**. The field is encoded as a byte sequence as defined in ETSI GS ECI 001-5-1 [4].
inputV: InputV	InputV message for the AS system. The field is encoded as a byte sequence as defined in ETSI GS ECI 001-5-1 [4].
eKey: SymKey	Encrypted symmetrical key for decrypting the image. The field is encoded as a byte sequence as defined in ETSI GS ECI 001-5-1 [4].
extension: Extension	Extension data, backward compatible. Should not exceed 256 bytes for broadcast applications in order to keep broadcast carousels compact. No application is defined for this data.
magic_file: byte[3]	8-bit ASCII representation of the string 'AES'.
as_setup_file_version: integer	Version of the ECI_AS_Setup_File format. Values 0 and 0x2..0xff are reserved. Value 0x01 is used for the format defined here.
as_setup: ECI_As_Setup	The **Platform Operation's as_setup** structure to load a specific encrypted **ECI Client** on a specific **ECI Host**.
magic_bucket_file: byte[3]	8-bit ASCII representation of the string 'AEB'.
as_setup_item: ECI_As_Setup	The as_setup structures in this bucket. Any new as_setup structures shall be added in front; so the ordering is oldest one first, and only be deleted at the end. This permits quicker inspection of updates by **CPEs**. I.e. after a first check the next check only needs to check as_setup structures until the first one of the previous check series is encountered.

The minimum checking frequency for updates of the **as_setup** structure shall be the same as for other **ECI Client** data as defined in clause 7.3.1. Note that an update typically implies an update of the **ECI Client** and/or the **ECI Host** software for the **CPE**; and therefore any updates of these shall also be downloaded to ensure a coherent **ECI Client** initialization sequence can be completed. If such a coherent new set is not available the previous coherent set can be used.

When the **ECI Host** is in a state trying to complete the broadcast mode (manual) registration of a new or updated **ECI Client** the **ECI Host** shall check for an as_setup file carousel update with the highest possible frequency.

7.8.4 Transport Protocol

7.8.4.1 Broadcast protocol

The broadcast protocol for **as_setup** structures shall be in accordance with clause 7.7.2.

The amount of as_setup structures that need to be updated on an **ECI Client** version change may be very substantial. In order to limit the number of new online as_setup messages on a **ECI Client** version change in a large broadcast-only operation, the **Platform Operation** may make a new **ECI Client** available, and *stage* the playout of new credentials, thereby replacing groups of **ECI Clients** on **CPEs**; and may repeat this a number of times to catch as many **CPEs** as possible before using the security system to enforce the use of the new **ECI Client**.

7.8.4.2 Online protocol

The online protocol relies on a straightforward request-response protocol between the **CPE** and the **ECI Client** is defined in clause 7.7.3, passing the **CPE target_id** and the **nonce** as part of the request, returning the **ECI_As_Setup_File**.

7.8.5 Target ID presentation to user

Both **ECI Host** and **ECI Client** have to be able to present the **CPE**'s target_id to the **user** on broadcast networks in case no online connection is available to permit the generation of **CPE** specific information required to decrypt the **ECI Client Image** if required and to permit the **ECI Client's** AS system InitV messages to be generated (the transport protocol for these messages is defined by the **ECI Client**). Also the target_id may be readable as a printed item on the **CPE**'s exterior or in accompanying documentation. This clause defines the presentation of the target_id to the **user**.

The target_id is a 64 bit integer. It shall be presented to the **user** following the rules thereto in clause 6.2.2. using a 9-bit checksum and adding 9-bit substrings instead of 5-bit substrings. The target_id thus is represented as sequence of six 4-digit numbers with digits between 0 and 7.

CPEs and **ECI Clients** are permitted to use customized representations in their **user** interface (e.g. based on a private **CPE** numbering scheme) but shall always offer **ECI Client** registration functions on the basis of the above presentation format.

8 Revocation

8.1 Introduction

All parties and the items with which they contribute to the **ECI Ecosystem** will be certified by the **ECI TA**. Through this certification it will be possible to provide a suitable base quality for both functionality as well as robustness of implementations, and appropriate renewal measures by the contributing parties. This certification process also prevents hacking and piracy operations using **ECI**'s ecosystem.

ECI provides functionality to selectively exclude delivery of services to **CPEs** based on the **ECI TA** status of the **CPE** hardware, the **ECI Host**, other **Platform Operations** and **ECI Clients** loaded.

The **ECI TA** can revoke a **Platform Operation** if these do not follow commonly agreed rules, among others on non-interference with other **Platform Operations** on shared **CPEs**, or on delivery of pirate services through **ECI**. Similarly the **ECI TA** can revoke **ECI Clients** if these do not follow commonly agreed rules, among others non-interference with other **ECI Clients** on shared **CPEs** or hacking practices. The **ECI TA** can further revoke **ECI Host** software versions if these have significant inadequacies exposing **ECI Clients** secrets or permitting manipulation.

In all of the above cases the organizations responsible for the revoked item can repair the deficiency, typically replacing the revoked item by a new item. So a **Security Vendor** can replace an **ECI Client** with a new version, a **CPE Manufacturer** can provide security patches for an **ECI Host** and an **Operator** can improve its operations effectuated through a new version of its **Platform Operation Certificate**. All these operations have a collaborative nature and are suggested to take place following contractual agreements between the parties affected and the **ECI TA**.

In case parties engaged in **ECI** cause systematic violations of **ECI TA** agreements which adversely affect other parties or the **users**, all of their contributed items may be revoked from **ECI TA**.

In case certain **CPEs** no longer possess a valid **ECI Host** and are not foreseen to receive an update from their **CPE Manufacturer**, they may be revoked as such. This also occurs in case a **CPE**'s boot loader is compromised and permits loading of non-compliant **ECI Host** software.

The **CPEs** shall attempt to automatically replace a revoked version with an updated version if available. However, new downloads and **Revocation Lists** can be blocked. In this case a **Platform Operation** can deny delivering services or rendering to of stored content on such a **CPE**.

8.2 CPE Revocation

ECI permits **Platform Operation**s to exclude providing services to specific **CPEs** by using the selective rights delivery functionality of CA or DRM system. The **Platform Operation** can examine the latest **ECI TA** state of a **CPE** from the **ECI TA**. In case the **ECI TA** deems it necessary to revoke a **CPE** the **Platform Operation** can disable providing services to a **CPE** on the basis of its registered chipset ID with the CA or DRM system delivering services.

The present document also facilitates **Platform Operations** to exclude providing services to **CPEs** running revoked **ECI Hosts**. The **Platform Operation** can use the Advanced Security system to require a minimum version number for the **ECI Host** in accordance with a recent **ECI Host** revocation list as defined in the clause 8.3.

The **ECI Host** revocation mechanism can also be used for **CPE** revocation if deemed appropriate, by specifying a minimum **ECI Host** version higher than has been issued so far.

8.3 Generic Revocation Process

This clause refers to the combination of minimum **Root** version and minimum **Root** revocation list version as "minimum **Revocation List** version".

The ultimate revocation enforcement mechanism for an **ECI Host** is service starvation: in case a revoked item is present on the **ECI Host** despite application of (presumably old) **Revocation Lists** the **Platform Operation** may decide to stop providing services to that **ECI Host**. The delivery of the minimum acceptable revocation list required by a **Platform Operation** is protected by the **AS System**: its manipulation will itself cause service starvation. A **Platform Operation** can thus force a check on the version of the credentials used to install the **ECI Host** and all other **Platform Operation**s and **ECI Clients**.

The **Platform Operation** shall provide a download service of the **Revocation List** for any of the above items. (**ECI Hosts**, **ECI Clients** and **Platform Operations**). This ensures the latest revocation lists for all **ECI Clients** and Platform Operations loaded on the **ECI Host** is available.

The **AS System** initialization [4] allows the **ECI Host** to specify this minimum expected **Revocation List** version for all items. It is used to validate the revocation list version used by the **ECI Host** retrospectively. The **ECI Host** shall use the minimum **Root Revocation list** value of the **ECI Client item**s it wishes to load and the **ECI Host Image** it has loaded.

NOTE: An **ECI Host is** suggested not to load items for which would cause revocation to take effect, and notify the **user** instead.

Preventing undue service starvation requires the latest credentials (and if necessary the latest versions) for all items to be loaded to be available in an **ECI Host**. To prevent that **ECI Clients** are unable to function properly due to security hazards caused by the presence of revoked **ECI Hosts Platform Operation Certificates** or **ECI Clients**, the **ECI Host** shall provide the following functionality to ensure the latest credentials and (if necessary) items are available to prevent undue service starvation to take effect:

- It shall keep the latest **ECI TA Revocation List** chain of each item that is verified in its present **ECI Host**, **Platform Operation and ECI Client** configuration using the credential and **Revocation List** download services of the **CPE Manufacturer** and of the **Platform Operation** of its **ECI Clients**.
  The default settings for all relevant **CPE** modes shall enable such downloading.

- The **CPE** shall have no mode of operation that permanently prevents downloading other than the power not being connected or download network access being inhibited (not due to a **CPE** state or mode of operation).

- It shall be possible to restore the default settings regarding downloading and default revocation of **ECI Clients** and **Platform Operations** with a simple **user** action.

The present document permits users to override the default Host behaviour to revoke items causing service starvation of others. In case users do this (e.g. keeping an old Client running) they may experience increasing difficulty rendering contemporary services.

8.4 Revocation Lists based ECI Host Revocation

A **CPE** not being maintained properly can have a revoked **ECI Host**. The **CPE Manufacturers** are to provide updated credentials including the latest applicable **ECI Revocation List**. In addition a **Platform Operation** that wishes to operate an **ECI Client** on an **ECI Host** is able to provide a download service for a **Revocation List** pertaining to the **ECI Host**'s credentials and can provide a download service for selected **ECI Hosts**. The **ECI Host** shall apply the **Revocation Lists** for the **ECI Host** credentials (**Root Certificate** and Manufacturer **Certificate**) in accordance with the generic **Revocation List** processing rules as defined in ETSI GS ECI 001-5-1 [4].

The format of the **ECI Host** revocation data file is defined in clause 5.3.

8.5 ECI Platform Operation Revocation

A **Platform Operation** that wishes to operate an **ECI Client** on an **ECI Host** can provide a download service for a **Revocation List** pertaining to other **Platform Operation** credentials. The **ECI Host** shall apply the **Revocation Lists** to all installed **Platform Operation** credentials in accordance with the generic **Revocation List** processing rules as defined in ETSI GS ECI 001-5-1 [4].

The format of the **ECI Platform Operation** revocation file is defined in clause 7.6.3.

8.6 ECI Client Revocation

A **Platform Operation** that wishes to operate an **ECI Client** on an **ECI Host** can provide a download service for a **Revocation List** pertaining to other **ECI Clients**. The **ECI Host** shall apply the **Revocation Lists** to all installed **ECI Client** credentials in accordance with the generic **Revocation List** processing rules as defined in ETSI GS ECI 001-5-1 [4].

The format of the **ECI Client** revocation file is defined in clause 7.6.3.

9 ECI Client Interfaces

9.1 Introduction

9.1.1 Architecture of the ECI Client interfaces

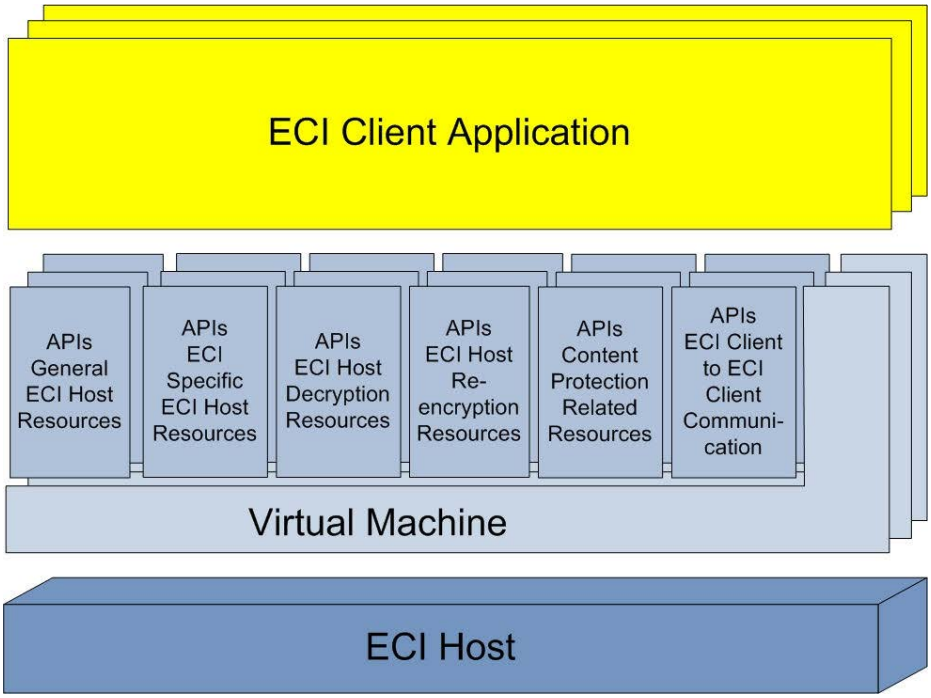


Figure 9.1.1-1: Structure of the APIs defined in clause 9

Figure 9.1.1-1 gives an overview of the structure of the APIs of the **ECI** System. It shows 6 blocks of APIs the **ECI Client** can use. Those blocks of APIs are specified in the clauses 9.4 up to 9.9. Table 9.1.1-1 lists the APIs defined in clause 9 of the present document.

Table 9.1.1-1: List of APIs defined in the present document

Clause No.	API category	Description
9.4	APIs for general **ECI Host** resources	APIs supporting general functionalities of the **ECI Client**
9.5	APIs for **ECI** specific **ECI Host** resources	APIs supporting **ECI** specific functionalities of the **ECI Client**
9.6	APIs for access to the **ECI Host** decryption resources	APIs allowing the **ECI Client** to utilize the decryption resources of the **ECI Host**
9.7	APIs for access to the **ECI Host** re-encryption resources	APIs allowing the **ECI Client** to utilize the re-encryption resources of the **ECI Host**
9.8	APIs for content property related resources	APIs supporting content protection functionalities of the **ECI Client**
9.9	APIs for **ECI Client** to **ECI Client** communication	APIs supporting the direct communication between **ECI Clients**

9.1.2 Media Handle

A **Media Handle** is an identifier of an object in the host environment that provides the context for all **ECI Host** interfaces provided to the **ECI Client** in terms of controlling the process of decryption of a content item. The **Media Handle** also permits the **ECI Client** to specify the data it requires from the content container in order to be able to descramble the content. In case of broadcast network delivery it also provides control over the selection of the program to be decoded and the stream selection from the delivery network (tuning function). An **ECI Client** can also request a **Media Handle** with access to a tuner in order to access data required for the operation of the **ECI Client** from network streams not accessed by the application/host for content acquisition purposes. For file and OTT-stream based delivery the **Media Handle** provides a means for the **ECI Client** to access security data in the file/stream not specified in a standardized location.

Media session descrambling operates directly under the control of the **ECI Client**. The synchronization of the CW application with the TS is based on scrambling control information in the TS. The synchronization of CW (commonly called keys in this context) to an ISOBMFF CENC file [6] shall be based on CENC KeyID identifiers.

Sessions making use of a **Media Handle** are listed in Table 9.1.2-1.

Table 9.1.2-1: Media Handle types

Name	Value	Description
MhDvbTs	0x01	TS shall comply with ISO/IEC 13818-1-1 [8]
MhIsobmffCenc	0x10	ISO BMFF file shall comply with ISO/IEC 23001-9 [37] and ISO/IEC 14496-12 [39]
RFU	other	Reserved for Future Use

9.2 ECI Virtual Machine Interface

9.2.1 Principles

A separate virtual machine instance shall be created for each **ECI Client**. Loading of the data and instructions for an **ECI Client** into a virtual machine is defined in clause 7.

The operation of the virtual machine is defined in ETSI GS ECI 001-4 [3].

All interaction of the **ECI Client** with the outside world shall be conducted using the message interface as defined in clause 9.2.3.

9.2.2 Instructions and data (static resources)

The VM will execute the instructions provided to it by the **ECI Client Loader** as part of the code segment(s) of the **ECI Client Image**.

The instructions are non-self-modifying, which is ensured by the VM. Any code which easily leads to non-desirable and/or easy to manipulate behaviour of an **ECI Client** (e.g. interpreters) is considered inappropriate and has to be ensured as part of the certification process of **ECI Clients**.

The maximum code and the static data space required by an **ECI Client** are defined in ETSI GR ECI 004 [i.10].

9.2.3 Interaction with ECI Host

All interactions of the **ECI Client** with the **ECI Host** are defined on the basis of the message model in this clause. There is no shared data between the **ECI Client** and the **ECI Host** other than:

- The data contained in messages;
- Any data stored in NV memory of the **ECI Host** on behalf of the **ECI Client**; or

- Any data in messages in communication channels to or from other **ECI Clients**.
  Note that this data is also exchanged through messages.

The message model is based on three different types of exchanges from **ECI Client** to **ECI Host**:

1) **Synchronous Client** initiated exchange: the **ECI Client** calls an **ECI Host Function** which reacts within a very short time. The **ECI Client**'s thread (execution flow) is blocked while the **ECI Host** processes the message and provides a return message.

2) **Asynchronous Client** initiated exchange: the **ECI Client** sends the **ECI Host** a **Client Request** message that will be queued and processed in due course by the **ECI Host**. The asynchronous call will provide an immediate **Return** with only a basic result (message identifier or error). The **ECI Host** will later provide a **Host Response** message reporting back with the status and results of the operation of the **ECI Host** initiated by the **ECI Client**.

3) **Asynchronous** Host initiated exchange: the **ECI Host** sends the **ECI Client** a message that will be queued and processed in due course by the **ECI Client**. The asynchronous call will provide an immediate return message with only a basic (standard) result. Type and format of this message, as it is represented in the **ECI Host,** is out of scope for the present document as this is an **ECI Host** internal issue:

4) Only the representation for the ECI Client is defined. The ECI Client will later provide a Response message reporting back with the status and results of the operation of the ECI Client initiated by the ECI Host.

The different types of message exchanges between **ECI Host** and **ECI Client** are shown in Figure 9.2.3-1.

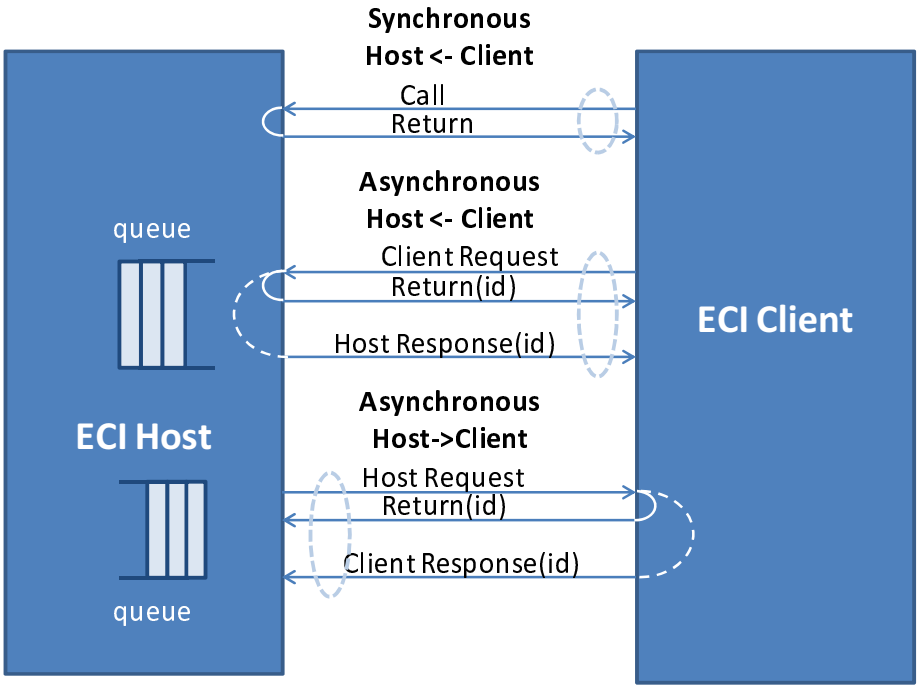


Figure 9.2.3-1: Message exchanges between Client and Host

The **ECI Client** has to ensure that the payload is protected as needed, e.g. control words and content properties. Furthermore the interface is not designed and intended for content exchanges.

The **ECI Client** shall implement Reponses to the **ECI Host Requests** it supports in accordance with the API definitions as presented in clause 9 using the identifier of the **Requests** in the **Response**.

The **ECI Host** shall implement Reponses to the **ECI Client Requests** it supports in accordance with the API definitions as presented in clause 9 using the identifier of the **Requests** t in the **Response**.

An asynchronous **Request** may optionally indicate that no **Response** is required. One example of usage is, when many data items are moved the initiator requires only a **Response** on the last **Request**, assuming all intermediate data items are processed correctly.

All Asynchronous **ECI Host Requests** and **ECI Host Reponses** are queued "in the order they occur".

9.2.4 Dynamic Resources provided for ECI Client's

Technical parameters of the minimum required dynamic resources of an **ECI Client** are specified in ETSI GR ECI 004 [i.10]. The following items are covered: Threads, stack space, heap space, execution time, NV storage and inter-client communication.

9.2.5 API version management

APIs defined in the present document are allowed to have multiple versions, e.g. for the purpose of offering enhanced functionality that replaces previous functionality or for resolving specification deficiencies. At initialization **ECI Clients** and their **ECI Host** need to establish which APIs are supported by their counterpart and select which is the version of each available API of the counterpart that will be used during the remainder of the **ECI Client**'s lifecycle. **ECI Clients** cannot use APIs other than the discovery API during the initialization phase since the message versions (i.e. their availability, length and syntax) are not defined until the discovery process is complete.

API versions are self-contained in their semantics: i.e. the message interaction between **ECI Client** and **ECI Host** via an API version is neither dependent on the support for other versions of that API in the **ECI Host** nor on interactions of the **ECI Host** with other **ECI Clients** using other versions of that API.

NOTE 1: For practical reasons the text in clauses defining new API versions can refer to text defining older API versions in the present document.

APIs are either mandatory, optional or conditional (i.e. mandatory subject to a condition). An example of conditionality is that PVR related API requires support on a **CPE** that supports PVR functionality. Future versions of the present document can define profiles of APIs to be supported by **ECI Hosts** and **ECI Clients** referencing profile name and specification version number.

For compliance to the present document and to ensure backward compatibility an **ECI Host** or **ECI Client** that support an API shall support all versions of that API (including the latest) unless older versions are explicitly deprecated in (future versions of) the present document or explicitly stated otherwise.

NOTE 2: The creation of a future version of the present document does not imply deployed or new **ECI Clients** and **ECI Hosts** have to be compliant. Any policies of field upgrade of **ECI Hosts** and **ECI Clients** to new specification versions or rules that mandate new specification versions, which apply to new **ECI Hosts** and **ECI Clients**, are out of scope of the present document.

ECI Clients should select the highest version number of an available APIs in **ECI Hosts** that they are able to handle, and vice versa **ECI Hosts** select the highest available version number of an API in **ECI Clients** that they are able to handle. This encourages forward version migration towards more mature APIs and stimulates avoidance of legacy issues in case of deprecating (older) API versions.

In view of the typically longer lifecycle of **ECI Hosts** and the relative ease of updating **ECI Clients, ECI Clients** should be able to support older **ECI Host** API versions reflecting the installed base situation (which can be subject to further agreements thereto outside the scope of the present document). Vice versa new **ECI Hosts** should support older **ECI Clients** reflecting the **ECI Client** deployment (which can be subject to further agreements thereto, outside the scope of the present document).

The **ECI Client**-**ECI Host** discovery API is defined in clause 9.4.2.

9.2.6 Responsiveness Monitoring

The **ECI Host** shall deploy some basic automatic **ECI Client** restart functions in order to provide additional robustness of the overall **CPE** functionality. The **ECI Host** shall detect fatal error conditions in the **ECI Client** and shall re-initialize the **ECI Client** automatically on such events. All resources used by the **ECI Client** will be released before re-initialization, including media handles, mmi sessions, files, IP connections, etc.

The following error conditions are defined:

- The **ECI Host** shall monitor the execution of any illegal instruction by the **ECI Client** code, like undefined instructions opcode, addressing illegal data or addressing non-existing code, overflowing or under flowing register stack, etc.
- The **ECI Host** shall use a timeout on the acceptance of a new message by the **ECI Client**. The required performance figure for this parameter is defined in ETSI GR ECI 004 [i.10].

In case of repeated re-initialization the **ECI Host** may use a policy, possibly involving user settings or user input, to decode to exclude the repeatedly failing **ECI Client** on a more permanent basis.

NOTE: Any execution of a sys_exit syscall (see ETSI GS ECI 001-4 [3]) by an **ECI Client** will be understood as a regular termination of the **ECI Client**. Typically this implies the **ECI Client** can be removed or is replaced by a later version. The **ECI Host** not automatically removes the **ECI Client** on the basis of such an event, but wait until an appropriate replacement or removal procedure is invoked through other management policies for **ECI Clients**.

9.3 Mechanism for ECI Client APIs

9.3.1 Asynchronous message syntax

All messages structures are defined in terms of their appearance in the **ECI** VM. In Table 9.3-1 the message buffer structure for all asynchronous messages is presented in terms of their appearance in the VM memory map. Note that all message buffers are 32-bit aligned.

Table 9.3-1: Asynchronous message syntax

C-style Syntax	No. of bits
struct messageBuffer {	
uint32 **msgTag**;	32
uint16 **msgId**	16
uint16 **payloadLen;**	16
uint32 **payload[];**	n*32
} MessageBuffer;	

msgTag:

This field represents the following values:

- Bits 0-15: **msgApiTag**. API identification for the message (for definition see annex C).
- Bits 16-23: **msgCallTag**. API call identification, to be interpreted by the receiver in the context of the **msgTag** value and the agreed API version.
- Bits 24-31: **msgFlags**: Additional flags to qualify a message. The following definitions apply:
  - Bit 24: **msgNoResFlag**: for **Request** and invoke messages: if 0b1 no **Response** or answer is required; if 0b0 a **Response** or answer is required. This bit has no meaning in answer and response messages.
  - Bit 25-31 are reserved for future use; these bits shall be set to 0b0 by the initiator of the message.

The message tag shall be identical for **Response**s to **Request** messages and answers to invoke messages.

msgId:

- The value of the message identifier of the message as assigned by the **ECI Host**. For a response message this shall correspond to the value of the original request message. This field can be left uninitialized by an **ECI Client** sending a request (the value will be assigned by the **ECI Host** and returned as a result value of the SYS_PUTMSG syscall.).

payloadLen:

- The payload length field represents the size the payload buffer in bytes. The actually allocated size of the payload field shall be this value rounded up to the next multiple of 4 or more. **ECI Hosts** shall verify when interpreting the **payload** field of a received message that data does not extend beyond the **payloadLen**; otherwise an error shall be returned. **ECI Clients** can assume that **ECI Hosts** provide properly dimensioned message buffers.

payload field:

- The payload field is used to carry message parameters. The structure of a payload is defined using the c-syntax for function call signature used with specific mapping rules defined in clause 9.3.2.3.

9.3.2 Asynchronous message layout definition convention

9.3.2.1 Syntax of message definitions

Asynchronous messages are defined using a c-style function signature declaration. This notation corresponds to the layout of the messages through rules defined in this clause. Below is an example of a function signature declaration:

```
reqSetTimer(uint32 time, uchar priority)
```

9.3.2.2 Basic message parameter types

The syntax shall use the basic types for parameter definitions as specified in Table 9.3.2.2-1.

Table 9.3.2.2-1: Basic types used for message parameter definitions

Basic types	Represent
uint8, uchar, byte:	8-bit unsigned integer
int8, char, bool:	8-bit signed integer
uint16, ushort:	16-bit unsigned integer
int16, short:	16-bit signed integer
uint32, uint:	32-bit unsigned integer
int32, int:	32-bit signed integer
uint64, ulong:	64-bit unsigned integer
int64, long:	64-bit signed integer
char *, … ,long * (client memory)	32-bit ; only permitted for synchronous messages

For parameters of type bool the symbolic values **True** and **False** are used. As per the c-language definition **False** is represented by 0x00, **True** by any value other than 0x00.

9.3.2.3 Message payload to message parameter mapping

The **payload** field contains all parameters for the message. The **msgId** message identifier parameter and the **msgResult** result parameters are implied in that sense that they are not exposed in the function signature declarative syntax description. Their presence is implicitly defined by the message type.

The ECI Host shall associate an **msgId** to **ECI Host** and **ECI Client Request** messages in order to associate **Request** with the corresponding reply. The type of msgId is uint32. Management of msgId values is in the responsibility **of the ECI Host**. msgId values shall not be reissued until the **Response** message is transferred.

The **Response** shall contain an **msgResult** parameter of type int32.

These implied parameters are the first parameters in the payload field of a message buffer. Table 9.3.2.3-1 presents the payload field parameter sequence for each message type from the **ECI Client** perspective (the **ECI Host** perspective is out of scope in **ECI**).

Table 9.3.2.3-1: Message types and "hidden" parameters (Client perspective)

Message type	Implied Parameters	Payload field
Client Request, C→H	*None*	p_1, .. , p_n
Host Response, H→C	msgId, result	msgId, result, p_1, .. , p_n
Host Request, H→C	msgId	msgId, p_1, .. , p_n
Client Response, C→H	msgId, result	msgId, result, p_1, .. , p_n

The following rules shall be used to convert parameters (be it structures, byte and short arrays, etc.) to the layout of the payload of the message buffer in the **ECI Client** memory space:

- Parameters are mapped into memory with their lowest address first, with exception of the data fields of variable length arrays.
- Any 8 or 16-bit data type is extended to 32-bit using the extension appropriate to its type (signed or unsigned).
- Structures (not including bit-fields): all fields shall be mapped in the order they are defined, field-size aligned (for 16 and 32 bit entities) first field on the lowest address, padding field preceding a larger field following it. The structure is always padded to the next 32-bit boundary. Union structures shall be padded to the largest size of the alternatives.
- Byte (8-bit), short (16-bit) and int (32-bit) arrays: shall be included in the message buffer (not as pointers to the memory of the **ECI Client**). Fixed length arrays shall use the following notation <type>, <array_identifier>, '[' <constant> ']'. These shall be mapped in the order they occur in the parameter list. Variable length arrays shall use the notation <type>, <array_identifier>, '[' ']'. All variable length arrays shall be mapped to two 32-bit fields. The first field contains the offset in the message buffer where the first element of the array is located. The second field contains the length of the array (in bytes).
- 64-bit entities shall be stored with most significant 32-bit first (following typical conventions for mapping 64-bit entities in 32-bit little endian machines).
- All 32 and 16 bit entities shall have natural (unknown - defined by the underlying CPU architecture) endianness representation in memory.
- Any (char *) pointing to printable characters shall use UTF-8 representation [11] for the actual "code points" unless explicitly defined otherwise. Character representation may be 1 to 4 bytes (depending on the code-point). This specification does not define which code-points shall be printable in a **CPE** (which may have different implementations for different regions).

NOTE: The **ECI Host** is responsible for interpreting message tag in combination with the API-version agreed with the **ECI Client** during discovery. Similarly the **ECI Client** is responsible for interpreting message tag in combination with the API-version agreed with the **ECI Host** during discovery.

9.3.2.4 Naming convention for asynchronous messages

Convention for function names:

All function names shall start with a three letter indication reflecting the message type. The <name> of the function shall start with a capital. The following defines the name convention of messages by their type:

```
req<name>():    request message; res<name>():   response message;
```

EXAMPLE 1: reqIpTcpSend().

Convention for message pair notation:

Request and **Response** messages are defined as a pair, and similarly for invoke and answer messages. The following notation is used to refer to such message pairs:

```
<requestMessage>    →  <responseMessage>
```

EXAMPLE 2: reqIpTcpSend(socket,buffer) → resIpTcpSend(socket).

Function signatures may appear in these and other notations without parameter typing for brevity purposes.

Table 9.3.2.4-1 provides some examples of practical message name mapping to possible c-functions in a using a procedure-style, javascript-like event subscription/callback type programming approaches or dispatch loops. The **subscr** function permits a function call on receiving a message with tag. Two examples are provided: one that is selective on **msgId** identifier and including a **cntxt** structure to the function. The second example does not filter on **msgId** and does not provide a **cntxt** structure at callback/dispatch.

Table 9.3.2.4-1: Parameters in payload field per message type with parameters p_1, .. ,p_n

Message	Procedure like notation	Client Event callback Subscription	Client Callback/Dispatch notation or Invocation
Req, C→H	id = reqName([tag],p_1,..p_n)		
Res, H→C	res = resName([tag],id,p_1,..p_n)	subscr(tag,id,resName,cntxt) subscr(tag,resName)	resName(cntxt,res,p_1,..p_n) resName(id,p_1,..p_n)
Req, H→C	[tag =] reqName([id],p_1,..p_n)	subscr(tag,invName)	invName(id,p_1,..p_n)
Res, C→H	resName([tag],id,res,p_1..p_n)		

9.3.3 Synchronous messages

Synchronous messages adopt the same notational convention using function names as asynchronous messages. Synchronous message parameters shall not be serialized to fit into message buffers, but shall use general c-conventions for function calls and use the VM application binary interface definition for procedure mapping to the VM memory and register state. This permits synchronous messages to map directly onto regular c-functions as part of an **ECI Client** library.

There are three predefined types: **get** to read a variable in the **ECI Host** domain, **set** to write a variable in the **ECI Host** domain and a general purpose function **call** with a negative error code or non-negative function value return as shown in Table 9.3.3-1.

Table 9.3.3-1: Synchronous function types

Type	Applies to	Notation	Result	Semantics
Get	Host variable	getVariable((**i1**..**in**)	variable type	Read a variable indexed by parameters **i1**..**in** in the **ECI Host** domain (for this **ECI Client**) (see note).
Set	Host variable	setVariable((**i1**..**in,** value)	void	Assign value to variable indexed by parameters **i1**..**in** in **ECI Host** domain (for this **ECI Client**) (see note).
Call	Host	callFunc(p1..pn)	int or void	Make a (general purpose) synchronous call to a function in the **ECI Host** domain. The return value is of the same type as the result value for asynchronous messages: i.e. negative values represent an error occurred. Some functions may have a void type - permitting no error signalling.
NOTE: The **ECI Host** may be triggered in performing actions in addition to returning the requested object as a consequence of as Get function invocation.				

Examples of synchronous message definition:

```
uint getClock();
void setPwrWakeup (int timeout);
void memcpy(char *p1, char *p2; int len) ;
```

Examples of use:

```
uint clock = getClock() ;       /* read clock */
setPwrWakeup (1000);               /* set wakeup timer; triggers invocations
*/
(void) memcpy(ptr1,ptr2,100*1000)  /* copy client memory efficiently */
```

9.3.4 Error codes in Return

The Return code parameter of **Responses**, **Answers** and (if applicable) **Calls** shall contain a single 32-bit signed integer. If the value returned is zero or positive, the execution of the code was successful. A negative value is returned in case of an error. Errors are generic (see Table 9.3.4-1) or **Request** specific (see specific error codes per **Request**).

Table 9.3.4-1: Error Codes for return messages

Name/Constant	Value	Description
	1..MaxInt	Successful **Request**, value defined by message definitions.
ErrReqOkNoId	0	Successful **Request**.
ErrReqApiErr	-1	API designated by msgApiTag not supported.
ErrReqCallErr	-2	Call within API designated by msgApiTag not supported.
ReqQueueErr	-3	Problem queuing the message, **ECI** buffer queue overflow.
ReqResource	-4	Resource problem occurred when processing the **Request** (e.g. memory problem due to excessive messaging).
RFU	-5..-15	Reserved for future use (generic error types).
ReqParam<N>Err	-16..-48	Error in parameter **N** = -Result-15.
Reserved for VM errors	-49..-64	Error codes are reserved for VM specific errors as defined in ETSI GS ECI 001-4 [3].
RFU	-65 .. -256	Reserved for Future Use.
API specific error	-256 .. -511	API specific error defined by API Error Code table.
RFU	-512.. MinInt	Reserved for future use.

NOTE: Typically an **ECI Client** can rely on the **ECI Host** to support a specific profile of APIs as defined in clause 9.2.5 and queuing buffers of messages to be liberal. Therefore intelligent error processing is typically not required; the error code typically serves only **ECI Client** debugging scenarios.

The API specific error codes or the ReqParamNErr cannot be returned as part of a Return but such error shall be signalled as part of a **Response** instead.

9.3.5 Secure Authenticated Channel (SAC)

Tools for the establishment of a **Secure Authenticated Channel (SAC)** between an **ECI Client** and any other appropriate device are available with the Advanced Security APIs (see clause 9.5.2). In case An **ECI-Client** needs a secure authenticated communication with another **ECI Client** or any external device, it needs to define a proprietary mechanism, which can utilize the available APIs, especially the advanced security APIs.

9.3.6 Message Verification by ECI Host

In order to avoid error conditions or inappropriate actions as a consequence of inappropriate **Requests** or **Responses** **ECI Hosts** shall perform full checking of any message received from an **ECI Client**. The following checks shall be performed:

- Support of the **msgApiTag.**
- Support of the **msgCallId** within the API message space (in the context of the API version established at discovery).
- Verify whether the constraints on the payload and specifically **msgLength** match with the syntax rules for the message and that the message buffer (for asynchronous messages) and any memory of the **ECI Client**'s address space to be read or written to by the **ECI Host** is constrained to defined portions of the **ECI Client**'s address space.
- Verify whether any message-specific **Pre condition** fails (in the sense of the **Pre condition** being essential to the integrity of the **Request** or **Response**).
- Verify whether any pointer or memory implicated in the message is memory allocated to the **ECI Client**.

9.3.7 Message Processing by ECI Clients

Any memory allocated for sending a **Request** can be reused upon return, unless explicitly indicated otherwise (typically large messages for which the avoidance of copying is important). Similarly any memory allocated for sending a **Response** can be reused immediately following the send event.

ECI Clients shall not rely on **ECI Hosts** to return a **Response** for every **Request**.

ECI Clients may verify the correct syntax of any **ECI Host Request** or **Response**. There is no obligation on the **ECI Client** to respond to provide any feedback to the **ECI Host** in case of a badly formatted **Request** or **Response**.

9.4 APIs for general ECI Host resources

9.4.1 List of APIs defined in clause 9.4

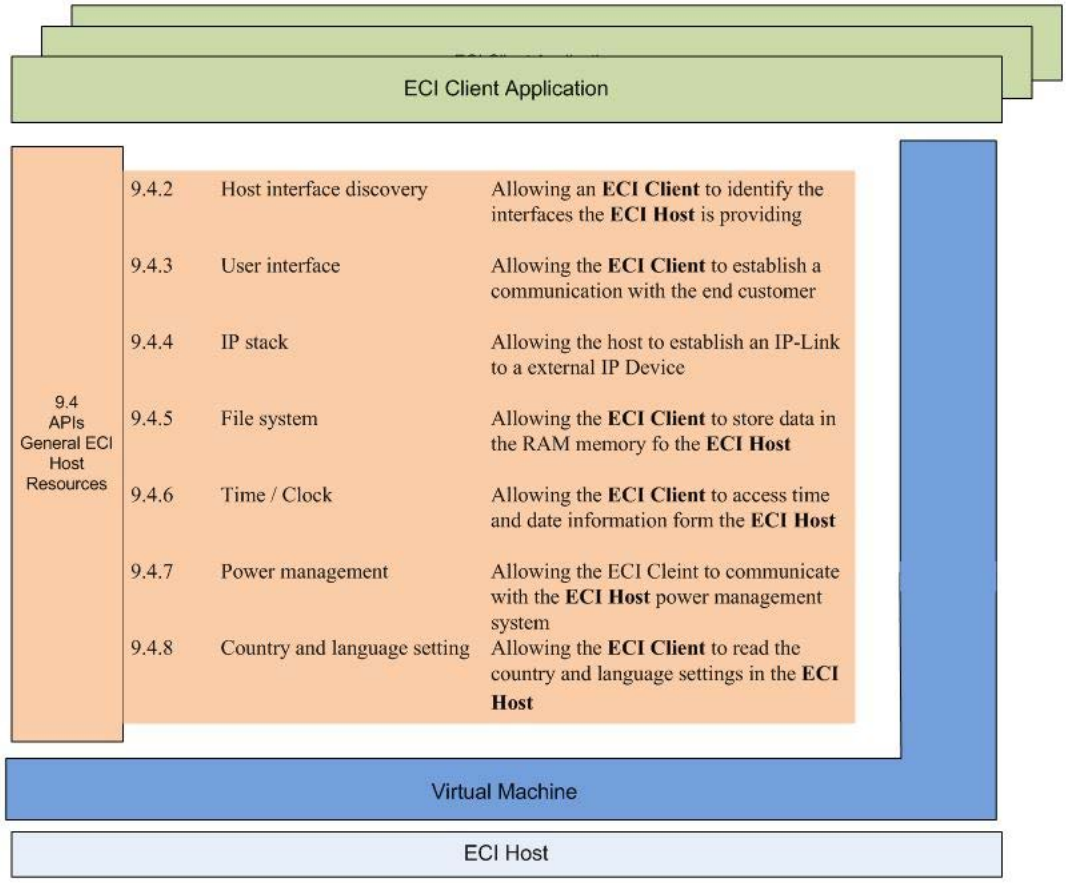


Figure 9.4.1-1: Block diagram of the APIs defined in clause 9.4

Table 9.4.1-1: List of APIs defined in clause 9.4

Clause	API name	Description
9.4.2	Host interface discovery	Allowing an **ECI Client** to identify the interfaces the **ECI Host** is providing
9.4.3	User interface	Allowing the **ECI Client** to establish a communication with the **user**
9.4.4	IP stack	Allowing the host to establish an IP-Link to an external IP device
9.4.5	File system	Allowing the **ECI Client** to store data in the RAM memory of the **ECI Host**
9.4.6	Time/Clock	Allowing the **ECI Client** to access time and date information from the **ECI Host**
9.4.7	Power management	Allowing the **ECI Client** to communicate with the **ECI Host** power management system
9.4.8	Country and language setting	Allowing the **ECI Client** to read the country and language settings in the **ECI Host**

Table 9.4.1-1 shows the APIs defined in clause 9.4 and Figure 9.4.1-1, which illustrates the location of the APIs defined in clause 9.4 with the **ECI architecture**.

An overview of the presentation Messages related to the different APIs is given per API in tables with the structure shown in Table 9.4.1-2.

Table 9.4.1-2: Structure of the table summarizing the functions of the individual API messages

Message	Type	Direction	Tag	Description
Name of the Message	See Table 9.4.1-3	C→H or H→C	Tag value	Short description of the function of the Message

The column Type in Table 9.4.1-2 gives the type of related Message, which can be either synchronous or asynchronous. More details are given in Table 9.4.1-3. A complete list of all API messages available for an **ECI Client** is given in annex E.

Table 9.4.1-3: Possible values for the Type column in

Category of Message	Notation in the Type column	Comment
Asynchronous Message	A	Possible message types: see Table 9.3.2.3-1
Synchronous Message	A	Possible message types: see Table 9.3.3-1
	Set	
	Get	
	Call	

9.4.2 API for the access to the ECI Host interface discovery resource

9.4.2.1 Introduction

This clause defines the API that can be used by an **ECI Client** to discover the APIs and the API versions that are supported by the **ECI Host** and select the most appropriate version for the duration of the **ECI Client's** session with the **ECI Host**. The API version management mechanism permits API selection on an API by API basis. Once an API version has been selected it will remain in use until the next **ECI Client** initialization event with the **ECI Host**.

Policies concerning the availability of APIs are discussed in clause 9.2.5. Mandatory APIs are defined in clause 10.

The **ECI Client** shall initiate version management as soon as it is initialized: no API can be used without a (mutually) established version.

The version of an API shall be represented by 16-bit number. API version numbering starts at 0x0000. Regular assignment of new versions is incremental (by 1).

Table 9.4.2.1-1 lists the API messages.

Table 9.4.2.1-1: ECI Host interface Discovery API

Message	Type	Dir.	Tag	Description
getApis	Get	C→H	0x0	Get available Host APIs
getApiVersions	Get	C→H	0x1	Get available versions of a host API
setApiVersion	Set	C→H	0x2	Set the version of the Host API to be used

9.4.2.2 getApis Message

C→H uint[] **getApis (**uint **maxNrApis**)

- This request returns a bit-array of maxNrApis that indicates the APIs supported by the **ECI Host**.

Property Definition:

- The Host API availability of API with tag **a** with (**a**< **maxNrApis)** is found as (**(*result*[a/32]>>(a%32))&0b1 == 0b1).**

Parameter definition:

maxNrApis: ushort	Highest APIs number for which to return the result plus one.

9.4.2.3 getApiVersions() Message

C→H uint[] **getApiVersions (**uhort **api,** ushort **maxNrVersions**)

- This request returns a bit-array of **maxNrVersions** that indicates the versions of **api** supported by the **ECI Host**.

Property Definition:

- The version availability of API with tag **api** for version **v** with (**v**< **maxNrVersions)** is found as (**(*result*[v/32]>>(v%32))&0b1 == 0b1).**

Parameter definition:

maxNrVersions: ushort	Highest version number for which to return in the result plus one.

9.4.2.4 setApiVersion() Message

C→H setApiVersion (ushort **api,** ushort **version**)

- This message sets the version of the API to be used between **ECI Client** and **ECI Host** for **api** to **version**. Should be called once only (subsequent calls have no effect).

Parameter definition:

api: ushort	Tag of the API for which the version shall be set.
version: ushort	Version number of **api** to be used in the subsequent session between Client and Host.

Detailed semantics:

- If **version** is not an existing API version supported by **api** then the API version shall be set to the first higher API version supported by the API if available or the highest API version otherwise.
- **ECI Clients** shall check the availability of a API version before performing initialization to that API version.

NOTE: Without checking explicitly unexpected API behaviour or error conditions may occur.

9.4.3 API for the access to the ECI Host user interface resource

9.4.3.1 Introduction

This clause defines the application environment for **ECI** applications, allowing the **ECI Client** to establish an interaction interface with the **User**. **ECI** applications are hosted by **ECI Clients**, and executed on an **ECI Host**. The applications use an HTML browser, which is available in TV devices for a number of platforms from device vendors and broadcasters.

Figure 9.4.3.1-1 depicts the individual entities in the **ECI** application environment. The **ECI Client** does not control and communicate directly with an **ECI** application it launched; it makes use of a proxy provided by the **ECI Host**. The proxy implements the API defined in clause 9.4.3.4 that allows **ECI Clients** to launch and stop **ECI** applications, and to communicate with running **ECI** applications, e.g. to process user input. Communication of the **ECI** application with the **ECI Client** is handled by the proxy by transcoding a browser HTTP GET request into either a resource from the application container or into an reqUiClientQuery API request to the **ECI Client** as defined in clause 9.4.3.4.8. The latter may provide the **ECI Client** with the user's input and permit the **ECI Client** to provide a response with dynamic content. The application container provides the (larger) static resources to build the UI screens; the **ECI Client** provides customized input to the UI screen and receives the user input.

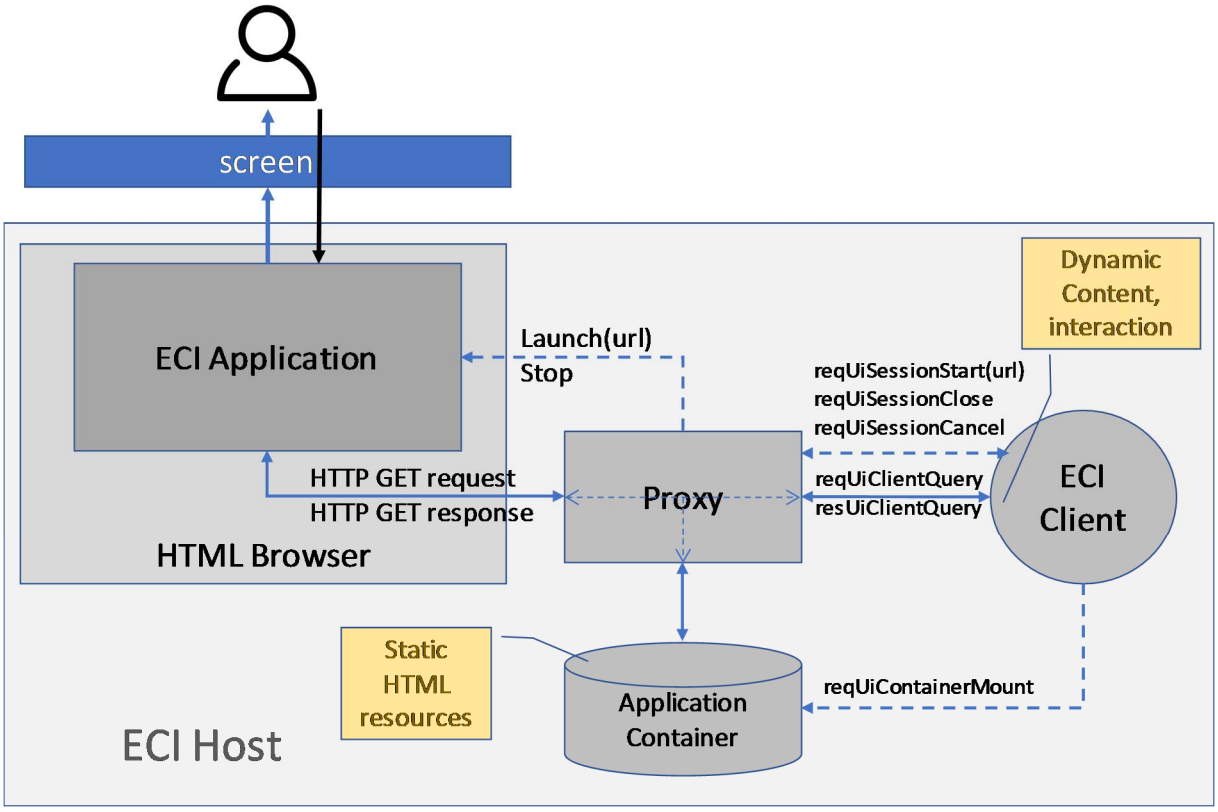


Figure 9.4.3.1-1: Block diagram of the User Interface API

9.4.3.2 User Interface environment

9.4.3.2.1 Browser Profile

The **ECI Host** shall provide an HTML browser that implements the Web Standards TV Profile as defined in [56] complying to constraints and extensions as defined in the present document. This profile is also adopted by the HbbTV system [i.9].

9.4.3.2.2 Constraints

The **ECI Host** shall deny HTTP requests to any resource of an **ECI Application** session that does not originate from this **ECI Application** session.

The URLs used to load the **ECI Application** resources into the browser shall be the concatenation of a base URL unique to the session and a relative URL to address either the **ECI Client** or the application container. E.g. if the session base URL is:

```
http://localhost:3000/session-x/
```

and a resource in the application container is:

```
main/pincode.html
```

then the browser URL is:

```
http://localhost:3000/session-x/main/pincode.html
```

When serving requests from the HTML browser the **ECI Host** should infer the content type of **ECI Application** resources from their file name extensions and should support at least:

- text/html - .html and .htm
- text/javascript - .js
- text/css - .css
- image/png - .png
- image/gif - .gif
- image/jpeg - .jpg and .jpeg

9.4.3.2.3 Browser Capabilities

9.4.3.2.3.1 Display Model

The browser window shall be full screen. The dimension of the browser window shall be at least 1 280 x 720 pixels. An **ECI** application should be authored such that it properly scales with larger dimensions.

The graphics plane that displays **ECI** applications shall be placed behind the graphics plane for terminal applications and it should be in front of any other graphics plane including those for video, subtitles and broadcast applications.

The plane for **ECI** applications fully covers any graphics plane except the terminal one. The background of the browser window should be transparent, i.e. if an area is not covered by any HTML element of the **ECI** application; the graphics planes below (one of which usually contains the broadcast video) should be visible. If the CSS property background-colour of the body element is set to transparent the background window of the browser shall be transparent.

When the terminal needs to temporarily overlay the **ECI** application, e.g. to show the system menu or channel info banner on a user action, the **ECI** application shall lose input focus. If the **ECI** application loses the input focus a blur event shall be sent with the Window object as target.

When the terminal closes its UI and the **ECI** application is still running it shall regain input focus. If the **ECI** application gains input, a focus event shall be sent with the Window object as target. The browser shall support RGBA32 as colour format.

9.4.3.2.3.2 Text and Fonts

The browser shall include an embedded proportional font. **ECI** applications can select the font using 'sans-serif' or 'default' as generic font family names to select the embedded font. The character set that is supported by the embedded font should be suitable for the region where the device is deployed. **ECI** applications may use the CSS3 Web Fonts as defined in [56] to use alternative fonts and character sets. The browser shall support at least one downloadable web font for each **ECI** application.

The browser shall support UTF-8 encoding for all text resources of an **ECI** application, i.e. HTML documents, scripts and style sheets.

9.4.3.2.3.3 Graphic Formats

The browser shall support graphics using the following formats: GIF [58], JPEG [57] and PNG [50].

9.4.3.2.3.4 User Input

The browser shall support user input by remote control using DOM3 KeyboardEvents. When an **ECI** application is running and it has the input focus, the **ECI Host** shall allow the user to initiate the following events:

- Numerical keys: 0-9
- Cursor keys: Left, Right, Up, Down, Enter and BrowserBack

Support for legacy attributes keyCode and charCode is not required.

9.4.3.2.3.5 Persistence

The browser shall support session storage for the WebStorage API and session cookies. An **ECI Client** should use its internal memory to keep information across browser sessions

9.4.3.2.3.6 ECI Application access to static HTML resources

The proxy receiving the HTTP requests from the **ECI Application** shall map the relative URL (i.e. the extension from the base URL of the session) to a relative path in the application container mounted by the **ECI Client**. The mapping from relative URL to file is direct: relative URL directoryname1/directoryname2/.. / directorynameN/filename is mapped to file filename in directory directorynameN contained in ... contained in director directoryname2 contained in directory directoryname1.

The application container directory structure and files shall comply with the following constraints:

- All file names and directories shall consist of alphanumerical characters and the characters '.' (dot) and '_' (underscore) and shall be no more than 40 characters.

More resource or performance requirements for the application container are defined in ETSI GR ECI 004 [i.10].

9.4.3.2.3.7 Communication between the ECI Client and ECI Applications

The browser supports XmlHttpRequest as required by clause 9.4.3.2.1 of the present document. Communication between **ECI** applications and **ECI Clients** is routed via the proxy of the **ECI Host**. The **ECI** application can perform an HTTP GET request using the XMLHttpRequest API as defined in this clause. The URL for the HTTP request shall be constructed from the base URL of the **ECI Application** session as defined in clause 9.4.3.2.2 and the relative URL '/client'. Any parameters shall be part of the query string as key - value pairs. Keys and values shall consist of ASCII characters only. Keys shall have a maximum length of 31 characters and values shall have a maximum length of 255 characters.

EXAMPLE: http://localhost:3000/ session-20170303-163100-01/client?id=e4f0&p2=v2.

When receiving the HTTP request proxy in the **ECI Host** shall send a reqUiClientQuery message to the **ECI Client** of the **ECI** application as defined in clause 9.4.3.4.5 with the parsed query string as key value pairs. The response from **ECI Client** to the host shall include the following parameter:

- type: a string conforming to media types as defined by IANA [30], e.g. application/json

- status code: an integer used in the response of the GET request, i.e. success should be 200
- body: a string of maximum 64 kByte

The **ECI Host** shall therefrom construct the HTTP GET response to the browser by setting the Content-Type header to the type parameter, the HTTP status to the error value and the response body to the value of the body parameter.

Communication with HTML applications not originating from the **ECI Client** is out of scope for this version of the present document.

9.4.3.3 Application Lifecycle

9.4.3.3.1 Launch of an ECI Application

The TV screen is a shared resource that is populated by terminal, broadcast, operator and third party applications. This version of the present document defines an application environment for basic user interfaces required to operate an **ECI** module, e.g. PIN entry, subscription information, etc.

Launching requests from **ECI Clients** on **ECI Hosts** shall be restricted to the following cases:

- The **ECI Host** is about to start the presentation of media (e.g. after tuning to a broadcast channel) that is being processed by the **ECI Client**.
- The **ECI Host** is presenting media that is processed by the **ECI Client**.
- The **ECI Host** requested the **ECI Client** to show its **Application Menu**.
- The **ECI Client** indicates it wishes to launch a non content stream related **ECI Application**, and the **ECI Host** can ensure the dialogue is on user request or does not conflict with content on the screen: i.e. there is no removal/blackout or screen overlay of third party content selected for viewing by the user.

For the above a launch request for performing a delegated parental authentication interaction as defined in clause 9.8.2.11 with the user is regarded as a launch request initiated by the **ECI Client** that issued the original parental authentication request as defined in clause 9.8.2.10.

A **Screen Conflict** is defined as a situation where the **ECI Client** requests the **ECI Host** to launch an **ECI Application** (open a UI session) but the above conditions for launching are not met.

In case the **ECI Host** has an ability to run interactive applications, the **ECI Host** shall be able to launch at least one **ECI Application** while running such interactive content related to media presented on the screen. Such **ECI Application** shall be directly related to the media presented on the screen. Launching the **ECI Application** shall not terminate the interactive content presented on the screen, and this content shall be able to resume interaction with the user when the **ECI Application** stops.

The **ECI Host** shall bring the desire of an **ECI Client** to launch a non content stream related **ECI Application** to the attention of the user or permit the **ECI Client** to launch such **ECI Application** without a **Screen Conflict** on a regular basis. This can be done for instance by launching such **ECI Applications** at power-up or standby entry, or using some user action in response to an attention icon in a banner or a **ECI Host** menu screen that is displayed regularly. **ECI Clients** should not assume the ability to frequently launch such **ECI Applications** and should restrain the purposes to matters that are important to the **ECI Client's** continued operation.

When launched by the **ECI Client** the **ECI Application** shall be loaded in browsing contexts that is not accessible from browsing contexts of broadcast or any other third party application.

The browser window shall be visible within one second, and should have fully loaded the **ECI Application**.

Future versions of the present document might provide extended lifecycle models and conflict resolution mechanisms as well as permit communication with externally launched HTML applications.

9.4.3.3.2 Termination of an ECI Application

To stop an **ECI Application** the **ECI Client** sends a reqUISessionStop message to the **ECI Host**. The request includes a uiSessionId that was returned by the **ECI Host** in the resUISessionOpen response. The **ECI** application shall be stopped. How this is achieved is implementation dependent, e.g. by stopping or minimizing the browser. In any case the **ECI** application shall loose the input focus and the browser shall not send anymore KeyboardEvents to the **ECI** application.

An **ECI** application shall also be stopped if any user action (like pressing P+/P-) brings the terminal into a state where an **ECI** application launch is prohibited. The **ECI Host** shall send a reqUiSessionCancel message to the **ECI Client**.

9.4.3.4 APIs related to the User communication

9.4.3.4.1 List of User communication API messages

The user interface API permits the **ECI Client** to mount a downloaded UI application container file to provide the bulk of the static HTML resources required for generating the user interface. The proxy automatically resolves all non client directed HTTP requests from the browser to the application container file.

The **ECI Host** can suggest the **ECI Client** to start an application, either in response to the user requesting to access the **ECI Client Application Menu** or by indicating to the **ECI Client** that there are no conflicts preventing it to present a non media handle related **ECI Application** to the user with the reqUiSessionCommence message. The **ECI Client** can indicate its interest to launch such a non media handle related dialogue via the setUiClientAttention message. Effectively this permits lower priority communication from the **ECI Client** to the **User** when there is no **Screen Conflict**.

All user interface sessions are opened by the **ECI Client** using the reqUiSessionOpen message. The relative URL to render the first UI screen is provided as a parameter. Both **ECI Client** and **ECI Host** can terminate the user interface session using the reqUiSessionClose and reqUiSessionCancel messages respectively.

The reqUiClientQuery message allows the **ECI Application** in the browser to send requests with parameters via the proxy to the **ECI Client** which can then respond with data for the HTML application. This communication permits the **ECI Application** to present data specific to the **ECI Client** and to provide the **ECI Client** with user input in the same way as an HTML application communicating with a dynamic HTTP server.

Table 9.4.3.4.1-1 lists all APIs defined in this clause.

Table 9.4.3.4.1-1: User Interface API Messages

Message	Type	Dir.	Tag	Description
reqUiContainerMount	A	C→H	0x0	Mounts a UI Application container with HTML resources to support UI sessions.
setUiClientAttention	S	C→H	0x1	**ECI Client** indicates a desire to start a UI session without association to a media handle.
reqUiSessionCommence	A	H→C	0x2	**ECI Host** suggests the **ECI Client** to open a UI session.
reqUiSessionOpen	A	C→H	0x3	The **ECI Client** requests to open a user interface session with the user and present content on the screen.
reqUiSessionClose	A	C→H	0x4	The **ECI Client** ends a user interface session.
reqUiSessionCancel	A	H→C	0x5	The **ECI Host** cancels a user interface session.
reqUiClientQuery	A	H→C	0x6	The **ECI Client** receives request from the HTML application in the browser and provides a (dynamic) response.

9.4.3.4.2 reqUiContainerMount Message

C→H reqUiContainerMount(fileName **filename**, PubKey **pk**) →
H→C resUiContainerMount (uint **indexFileLen**, uchar **indexFile**)

- This message permits the **ECI Client** to direct the **ECI Host** to appoint a file as the **ECI Client's** application container containing the HTML resources for its **ECI Application**. If successful it returns the content of the "EciIndex.txt" file in the main directory of the application container.

Request parameter definitions:

filename: fileName	Filename in the ECI Client file system that will be the designated application container.
pk: PubKey	Public key for verifying the signature of the application container.

Response parameter definitions:

indexFileLen: uint	Length of index file.
indexFile: uchar	Content of the index file.

Detailed semantics:

- The square brackets [and] with verbose text in between as used below signify the demarcation of fields and structures in ZIP file containers.
- The signature for verification of the container file is found in the [.ZIP file comment] field of the [end of central directory record] structure (see Zip File Format Specification version 6.3.3 of PKWARE® Inc. as referred in ISO/IEC 21320 [11].
- The [.ZIP Comment Field] is defined *ending* in the following string comprised of all ASCII characters: 'ECI_SIGNATURE="' followed by the value of the ECI_Data_Signature structure as defined in Table 5.6-1 encoded as a hex string using capitals, followed by a "" (a closing parentheses).

EXAMPLE:

ECI_SIGNATURE="01000000FCB1F60456719035"

The length of the encoded data signature string for an ECI_Data_Signature of type-1 is 520 characters.

- The **ECI Host** shall verify the signature as computed over the container file up until the [.Zip comment field] in the [end of central directory record] structure and setting the [.Zip comment length field] to 0x0000 using public key parameter pk and the process of clause 5.6 defined for computing signatures.
- The index file is defined as the file with the name "EciIndex.txt" in the main directory of the container file.
- The **ECI Client** shall mount a valid UI application container if this is required for UI sessions.
- The **ECI Client** shall be able to display a basic distress message to the user in case of failure to load and mount the UI application container.

Application remarks:

- Clients can load application container files to their file system from an online server using the HTTP(S) API (see clause 9.4.4.6) or from a broadcast transport stream using the Data Carousel API.
- The "EciIndex.txt" file can contain version information for the UI, verified by the public key signature.

Error codes regarding the reqUiContainerMount message are defined in Table 9.4.3.4.2-1

Table 9.4.3.4.2-1: reqUiContainerMount Error Codes

Name	Description
ErrUiContainerFileNot	See Table 9.4.3.4.9-1
ErrUiContainerNot	
ErrUiContainerSignature	
ErrUiContainerIndexTxtNot	

9.4.3.4.3 setUiClientAttention Message

C→H setUiClientAttention(uint **clientAttention**)

- This message indicates the desire of the **ECI Client** to open a UI session with the user without relation to a media handle (UI Session type equal EciUiSessionDiaReq, see clause 9.4.3.4.4).

Property definition:

clientAttention: uint	Defined values are: 0x0: no attention from the user is desirable. 0x1: attention from the user is desirable. All other values are reserved.

Post Conditions:

- If clientAttention=0x0 no reqUiClientSessionCommence(uiSessionType=EciUiSessionDiaReq) messages will be issued by the **ECI Host**.
- If clientAttention=0x1 a reqUiClientSessionCommence(uiSessionType=EciUiSessionDiaReq) message will be issued by the **ECI Host** if there is no pending message of this type.

9.4.3.4.4 reqUiSessionCommence Message

H→C reqUiSessionCommence (uint **uiSessionType**) →
C→H resUiSessionCommence ()

- This message permits the **ECI Host** to suggest the **ECI Cl**ient to open a UI session of a specific type.

Request parameter definitions:

uiSessionType: uint	Filename in the **ECI Client** file system that will be the designated application container. The values are defined in Table 9.4.3.4.4-1. Only the values EciUiSessionAppMenu and EciUiSessionDiaReq are permitted.

Table 9.4.3.4.4-1: ECI UI Session Types

Name	Value	Description
EciUiSessionDiaReq	0x00	**ECI Client** requested UI session with the end user through the setUiClientAttention message (not in association with a specific media handle) and the **ECI Host** can grant a reqUISessionOpen from the **ECI Client** thereto.
EciUiSessionAppMenu	0x01	**Application Menu** of the **ECI Client**. This permits user initiated access to all relevant settings, information and functions that can be initiated by the user.
EciUiSessionMh	0x02	**ECI Client** requested UI session in association with operations for a media handle.
EciUiSessionParAuthDel	0x03	**ECI Client** requested UI Session to perform a delegated parental authentication dialogue on behalf of processing content on a media handle.
RFU	Other	Reserved for Future Use.

NOTE: The values in Table 9.4.3.4.4-1are defined in a suggested priority order. This order can provide suggestions on resolving UI focus conflicts in **ECI Host** design.

Detailed semantics:

- An **ECI Client** shall be able to present an **Application Menu**. The **Application Menu** should at minimum permit the user to inspect the version of the **ECI Client**, a **Platform Operation** reference and the **ECI Client**'s operational state.

Preconditions Request:

- There shall be no previously issued pending reqUiSessionCommence message to the **ECI Client** for a UI session.

Postconditions Response:

- The **ECI Client** shall issue a reqUiSessionOpen message with the corresponding UI Session type or an error shall be reported.

Error codes regarding the reqUiSessionCommence message are defined in Table 9.4.3.4.4-2.

Table 9.4.3.4.4-2: reqUiClientSessionCommence Error Codes

Name	Description
ErrUiResourceError	See Table 9.4.3.4.9-1.
ErrUiClientError	

9.4.3.4.5 reqUiSessionOpen Message

C→H reqUiSessionOpen(uint **uiSessionType,** ushort **mH,** uint **relUrlLen,** char **relUrl[]) →**
H→C resUiSessionOpen(ushort **uiSessionId)**

- This message permits the **ECI Client** to requests a new UI session from **ECI Host**.

Request parameter definitions:

uiSessionType: uint	Type of UI session as defined in Table 9.4.3.4.4-1. If the value is EciUiSessionMh or EciUiSessionParAuthDel the mH parameter shall have relevance, otherwise not.
mH: ushort	Media handle of the content processing session the MMI is associated with.
relUrlLen: uint	Length of relUrl in bytes.
relUrl: char[]	Relative URL, null character terminated. Appended to the session base URL will form the URL for the browser to start the UI session. See clause 9.4.3.2.2.

Response parameter definitions:

uiSessionId: ushort	ID of the new UI session.

Detailed semantics:

- An **ECI Client** shall be capable of handling multiple UI sessions at once. However, only one simultaneous session of UI session type EciUiSessionAppMenu or EciUiSessionAppMenu is required to be supported and at most one UI session with UI Session type EciUiSessionMh is required per open media handle.
- An **ECI Client** shall be able to open UI sessions of a UI session of type EciUiSessionMh simultaneously.
- The **ECI Client** shall be capable of opening simultaneous UI sessions of UI session type EciUiSessionParAuthDel if the **ECI Client** supports the Parental Authentication Delegation API. Such UI sessions shall be able to proceed in parallel to other UI sessions of the **ECI Client**.
- An **ECI Host** can support one or more simultaneous UI sessions as fits its **CPEs** application modes.

Pre conditions Request:

1) If the uiSessionType value is EciUiSessionAppMenu or EciUiSessionDiaReq this message shall have been preceded by a reqUiClientCommence message with the same uiSessionType parameter.

2) If the uiSessionType value is EciUiSessionParAuthDel this message shall have been preceded by a reqParAuthDel message for media handle mH from the **ECI Host** to the **ECI Client**.

3) If the uiSessionType value is EciUiSessionMh, Mh shall be an open media handle session.

Pre condition Response:

1) If the uiSessionType value is EciUiSessionAppMenu or EciUiSessionDiaReq or EciUiSessionParAuthDel the **ECI Host** shall only accept the UI session request in case it previously requested it, the cause for the request has not been mitigated, and it is in a state that would not cause a **Screen Conflict**.

2) If the uiSessionType value is EciUiSessionMh the **ECI Host** shall grant the UI session request if it can establish a meaningful interaction with the user without initiating a screen priority conflict.

3) **ECI Hosts** shall not reject a second session from an **ECI Client** when the second session has a uiSessionType equal EciUiSessionParAuthDel. The **ECI Host** is permitted to cancel the first session.

Application notes:

1) If a media handle session is used for recording and there is no opportunity to initiate a dialogue with the user since this would cause a **Screen Conflict** or there is no active screen, the **ECI Host** shall refuse the session.

2) **ECI Host** applications are recommended to accommodate parental authentication UI sessions when e.g. programming future recordings that may require parental authentication using the reqParAuthCid message of the Parental Authentication API (see clause 9.8.2.10).

3) **ECI Hosts** can cancel a UI session with an **ECI Client** to permit a new session with uiSessionType equal EciUiSessionParAuthDel or EciUiSessionMh.

Error codes regarding the reqUiSessionOpen message are defined in Table 9.4.3.4.5-1.

Table 9.4.3.4.5-1: reqUiClientSessionStart Error Codes

Name	Description
ErrUiScreenConflict	See Table 9.4.3.4.9-1
ErrUiNoScreen	

9.4.3.4.6 reqUiSessionClose Message

C→H reqUiSessionClose(ushort **uiSessionId)** →
H→C resUiSessionClose(ushort **uiSessionId)**

- This message permits the **ECI Client** to close an existing UI session.

Request parameter definitions:

uiSessionId: ushort	ID of the UI session to close.

Response parameter definitions:

uiSessionId: ushort	ID of the UI session that was closed.

Pre conditions Request:

1) A UI session with uiSessionId shall be open.

2) No further messages referring to uiSessionId shall be sent to the **ECI Host**.

Pre condition Response:

1) No further messages referring to uiSessionId shall be sent to the **ECI Client**.

9.4.3.4.7 reqUiSessionCancel Message

H→C reqUiSessionCancel (ushort **uiSessionId,** uint **reason)** →
C→H resUiSessionCancel (ushort **uiSessionId)**

- This message permits the **ECI Host** to closing an existing UI session to an **ECI Client**. This message is intended to be used by the **ECI Host** in cases where the conditions for showing an **ECI Application** are no longer met, e.g. if a user switches to a different channel belonging to a different **ECI Client** causing a **Screen Conflict**.

Request parameter definitions:

uiSessionId: ushort	ID of the UI session to be cancelled.
reason: uint	Reason for cancelling the session. The values are defined in Table 9.4.3.4.9-1.

Response parameter definitions:

uiSessionId: ushort	ID of the UI session that was cancelled.

Pre conditions Request:

1) The session with uiSessionId shall be open.

2) No further messages shall be sent referring to uiSessionId.

Pre condition Response:

1) No further messages shall be sent referring to uiSessionId.

9.4.3.4.8 reqUIClientQuery Message

H→C reqUIClientQuery(ushort **uiSessionId,** uint **queryLen,** KeyValPair **query[])** →
C→H resUIClientQuery(ushort **uiSessionId,** uint **statusCode,** uint **typeLen,** char **type[],** uint **bodyLen,** uchar **body[])**

- This message conveys a HTTP request by the **ECI Application** running in the **ECI Host** Browser as described in clause 9.4.3.2.3.7 and permits the **ECI Client** to send a HTTP response back to the **ECI Application**.

Request parameter definitions:

uiSessionId: ushort	Id of the UI session from which the request is issued.
queryLen: uint	Length of the query parameter in bytes.
query[]: KeyValPair	Contains key value pairs of the query parameters of the HTTP request issued by the browser.

Type definitions for KeyValPair

```
#define MaxKeyLen 32
#define MaxValLen 256

typedef struct KeyValPair {
   char key[MaxKeyLen];  /* Key of the key value pair, null terminated*/
   char val[MaxValLen];  /* Value of the key value pair, null terminated */
} KeyValPair;
```

Response parameter definitions:

uiSessionId: ushort	Id of the UI session.
statusCode: uint	HTTP status code as defined in IETF RFC 7231 [44].
typeLen: uint	Length of type parameter in bytes.
type[]: char	Type of the response as a null terminated ASCII character string.
bodyLen: uint	Length of body parameter in bytes.
body[]: uchar	HTTP response-message.

Pre conditions Request:

1) The **uiSessionId** is open.

Detailed semantics:

- In case of a poorly formatted query string from the **ECI Application** the **ECI Host** may return HTTP status code 400 and not initiate a request to the **ECI Client**.
- The message parameter relation to the HTTP request and response from the browser are defined in clause 9.4.3.2.3.7.

9.4.3.4.9 Error codes for the User communication API

The error codes related to the user interface communication are listed in Table 9.4.3.4.9-1.

Table 9.4.3.4.9-1: User communication API Error Codes

Name	Value	Description
ErrUiContainerFileNot	-256	No UI application container file found.
ErrUiContainerNot	-257	File not a valid UI application container file.
ErrUiContainerSignature	-258	Signature check failure on application container file.
ErrUiContainerIndexTxtNot	-259	No "EciIndex.txt" file in application container top directory.
ErrUiResourceError	-260	**ECI Client** cannot mount the UI application container resource.
ErrUiClientError	-261	**ECI Client** is not in an operational state where it can present a UI.
ErrUiDiaNoMore	-262	The dialogue request from the **ECI Client** is no longer valid.
ErrUiScreenConflict	-263	The **ECI Host** has a **Screen Conflict** and cannot accommodate or sustain a session.
ErrUiNoScreen	-264	The **ECI Host** does not have or does not longer have access to a screen for the UI session presentation.
RFU	Other	Reserved for Future Use.

9.4.4 API for the access to the ECI Host IP stack resource

9.4.4.1 Introduction

In **CPEs** equipped with an IP-stack the **ECI Host** provides an internet access service on behalf of **ECI Clients**. **ECI Clients** can send messages using UDP/IP and open TCP/IP connections to peers in both **ECI Client** and server mode using the **ECI Hosts**. **ECI Host** names can be resolved to IP addresses using the available DNS services in the **ECI Host**.

The services provided are not secured beyond the generic software security of the **CPE** itself. I.e. if the **CPE** software outside the **ECI Host** is compromised, any IP traffic might be tampered with.

The **ECI Client** API for IP connectivity is based on the BSD socket paradigm as used in many contemporary Operating Systems.

The definition of the API is split into four parts:

1) Basic **ECI** IP Sockets and DNS functionality (clause 9.4.4.3).
2) UDP/IP communication using an **ECI** IP Socket (clause 9.4.4.4).

3) TCP/IP communication using an **ECI** IP Socket (clause 9.4.4.5).

4) HTTP(S) communication using the **ECI Host** HTTP services (clause 9.4.4.6).

9.4.4.2 Basic Specifications

An **ECI Host** that has an IP connection capability shall implement the IP protocol [23] including IPv6 [27] and applicable updates thereof. It shall provide a means to perform **ECI Host** name resolution to IP addresses in accordance using DNS in accordance with IETF RFC 1034 [25], IETF RFC 1035 [26] and applicable updates thereof.

For providing a simple short unreliable message protocol the **ECI Host** shall support UDP over IP in accordance with IETF RFC 768 [22] including applicable updates. For providing reliable connection-oriented message exchange with flow control the **ECI Host** shall support TCP over IP in accordance with IETF RFC 793 [24] and applicable updates.

The **ECI Host** does not have to provide support for UDP multicasting in either transmit or receive mode.

9.4.4.3 **ECI** IP Sockets

9.4.4.3.1 General

ECI Clients can open an **ECI** IP socket for the purpose of sending and receiving communication using TCP and IP.

NOTE: The term "socket" suggests a resemblance to the original BSD sockets as used in many operating systems. As a concept **ECI** IP sockets are similar but have specific properties that deviate from BSD sockets. Specifically the behaviour is fully asynchronous.

ECI IP sockets are endpoints for IP communication. **ECI Clients** can open a socket by identifying the local port number and the willingness to accept incoming connection **Requests** (operate as a TCP/IP server). The sockets can be closed in which case any associated connection or server behaviour is closed. The IP address of a peer host-name can be resolved using the DNS services of the **ECI Host**.

The available messages are listed in Table 9.4.4.3.1-1.

Table 9.4.4.3.1-1: IP Socket Messages

Message	Type	Dir.	Tag	Description
reqIpSocket	A	C→H	0x0	Opens an **ECI** IP Socket
reqIpClose	A	C→H	0x1	Closes **ECI** IP Socket
reqIpAddrinfo	A	C→H	0x2	Gets address of (remote) **ECI Host**

The structure type definitions for these APIs are defined in clause 9.3.

Type definitions for IP socket API:

```
typedef struct Addrinfo {
   ushort addressType;      /* IPv4 or IPv6 address*/
   uchar  ipAddress[16];    /* the IP address itself */
   ushort port;             /* port number - if relevant */
} Addrinfo;
```

Field definitions:

addressTyp: ushort.	See Table 9.4.4.3.4-1, only values ProtPrefIPv4 or ProtPrefIPv6 are permitted. This field defines the length of the hostAddress as 4 or 16 bytes (see note).
ipAddress: uchar[16]	4 or 16 bytes representing the byte wise representation (in network order) of an IPv4 or IPv6 address respectively. IPv4 addresses shall use the first 4 bytes of this parameter.
port: ushort	Port number of socket to connect to (field may be unused).
NOTE: **ProtPrefIPv4** or **ProtPrefIPv6** are defined in Table 9.4.4.3.4-1.	

9.4.4.3.2 reqIpSocket Message

C→H reqIpSocket(uchar **source,** ushort **sourcePort,** ushort **protocol)** →
H→C resIpSocket(uchar **socketId)**

- **The message** opens a socket for TCP or UDP based communication on a local IP address and port.

Request Parameter definitions:

source: uchar	See Table 9.4.4.3.2-1: specifies the ECI Host IP address to be used for the local socket (a preference in case multiple IP addresses are assigned). In case the specific IP address is not identifiable a suitable alternative shall be selected by the ECI Host.
sourcePort: ushort.	Port address of the local IP connection endpoint. Value equal 0x0000 shall mean that the ECI Host shall allocate a free port address for the socket. Other values below 1024 are not permitted.
Protocol: ushort	See Table 9.4.4.3.2-2: specifies the protocol used for the socket. The choice for IPv4 or IPv6 shall be specific.

Table 9.4.4.3.2-1: IP Source parameter

Name	Value	Description
IpSourceAny	0x00	Default IP Address of **ECI Host**.
IpSourceWan	0x01	**ECI Host** IP address used for WAN (internet) communication.
IpSourcePriv	0x02	**ECI Host** IP address used for private IP traffic on a proprietary IP protocol channel.
IpSourceLan	0x03	**ECI Host** IP Address used for local network communication.
RFU	Other	Reserved for Future Use.

Table 9.4.4.3.2-2: IP protocol parameter

Name	Value	Description
SockProtUdpIPv4	0x0001	UDP/IP using Ipv4.
SockProtUdpIPv6	0x0002	UDP/IP using Ipv6.
SockProtUdpIPany	0x0003	UDP/IP using IPv4 or v6.
SockProtTcpClientIpv4	0x0005	TCP/IP using Ipv4, client mode (only for initiating connection).
SockProtTcpClientIpv6	0x0006	TCP/IP using Ipv6, client mode (only for initiating connections).
SockProtTcpClientIpany	0x0007	TCP/IP using Ipv4 or v6, client mode (only for initiating connections).
SockProtTcpServerIpv4	0x0009	TCP/IP using Ipv4, server mode (for accepting incoming connections).
SockProtTcpServerIpv6	0x000A	TCP/IP using Ipv6, server mode (for accepting incoming connections).
SockProtTcpServerIpany	0x000B	TCP/IP using Ipv4 or v6, server mode (for accepting incoming connections).
RFU	other	Reserved for Future Use.

Response Parameter definitions:

SocketId: uchar.	Socket ID of opened socket.

Semantical description:

- Just after initialization the **Response** is allowed to be stalled until such time that **ECI Host** IP address initialization has been completed successfully. Performance figures are defined in ETSI GR ECI 004 [i.10].

Pre conditions Request:

1) The maximum number of sockets the **ECI Client** is allowed to request shall not be exceeded.

2) The source, sourcePort and protocol is a valid parameter configuration.

Post conditions Response:

1) Socket is opened or an error is returned in the **Response**.

Error codes regarding the opening of the sockets are listed in Table 9.4.4.3.2-3.

Table 9.4.4.3.2-3: resIpSocket Error Codes

Name	Description
ErrIpSourceProt	See Table 9.4.4.7-1.
ErrIpNoSockets	
ErrIpProtNotAvail	
ErrIpPortNotAvail	

9.4.4.3.3 reqIpClose Message

C→H reqIpClose(uchar **socketId)** →
H→C resIpClose(uchar **socketId)**

- Closes IP socket and any associated connection; all pending communication to and from the socket can be lost.

Request parameter definitions:

socketId: uchar	ID of socket to close.

Response parameter definitions:

socketId: uchar.	ID of socket that was closed.

Semantical description:

- This **Request** closes the socket and any IP connection associated with it. It will leave it to the **ECI Host** to send the proper disconnect messages to any communication peer if applicable. The successful completion of the latter is not required for sending the **Response**. A socket that has no associated connection will also be closed.

Pre conditions:

1) Socket exists and is in an opened state.

Post conditions:

1) Socket is closed and can no longer be used for any communication (unless reassigned on reqIpSocket).

Error codes regarding the closing of the socket are listed in Table 9.4.4.3.3-1.

Table 9.4.4.3.3-1: resIpClose Error Codes

Name	Description
ErrIpSocketNotOpen	See Table 9.4.4.7-1.

9.4.4.3.4 reqIpAddrInfo Message

C→H reqIpAddrinfo(uint **hostnameLenth,** char **hostname[],** uchar **protPref)** →
H→C resIpAddrinfo(Addrinfo **ipaddress)**

- This message provides the IP address information for addressing **ECI Host** using preferred protocol (protPref), returning the **ECI Host** address. The protocol shall use the DNS services of the **ECI Host** when required to resolve the **Request**.

Request parameter definitions:

hostNameLength: uint	Length of the name field (in bytes).
hostname: char[]	The name of the IP host to be resolved; be it in IPv4 dod notation [29], IPv6 colon notation [27] or actual host name [28].
protPref: uchar	Indicates the IP protocol preference as defined in Table 9.4.4.3.4-1.

Table 9.4.4.3.4-1: IP Protocol Preference parameter

Name	Value	Description
ProtPrefIpv4	0x1	An IPv4 address shall be returned.
ProtPrefIPv6	0x2	An IPv6 address shall be returned.
ProtPrefAny	0x3	Either an IPv4 or IPv6 address shall be returned.
RFU	other	Reserved for future use.

Response parameter definitions:

Ipaddress: Addrinfo	IP address of **ECI Host**. The port field is undefined.

Semantical description:

- This **Request** uses the **ECI Host** DNS services to translate the provided host name into a binary host address representation. Delays can occur due to temporary absence of DNS service access (e.g. during **CPE** start up); the **ECI Host** shall ensure an appropriate timeout is observed (i.e. the **Response** is always received by the **ECI Client**).

Post conditions Response:

1) Resolved host address or error.

Error codes regarding the closing of the socket are listed in Table 9.4.4.3.4-2.

Table 9.4.4.3.4-2: resIpAddrInfo Error Codes

Name	Description
ErrIpHostUnknown	See Table 9.4.4.7-1.
ErrIpHost	
ErrDnsOffline	

9.4.4.4 **ECI** UDP/IP

9.4.4.4.1 General

ECI Clients shall send and receive UDP datagrams using an open **ECI** UDP/IP socket. The related messages are defined in Table 9.4.4.4.1-1.

Table 9.4.4.4.1-1: UDP/IP Socket Messages

Message	Type	Dir.	Tag	Description
reqIpUdpSendMsg	A	C→H	0x3	Sends message to peer UDP port.
reqIpUdpRecvMsg	A	C→H	0x4	Receives a message from peer UDP port.

9.4.4.4.2 reqIpUdpSendMsg Message

C→H reqIpUdpSendMsg(uchar **socketId,** Addrinfo **peer,** uint **datagramLength,** byte **datagram[]) →**
H→C resIpUpdSendMsg(uchar **socketId)**

- This message sends an UDP datagram to a peer (IP address, IP port).

Request parameter definitions:

socketId: uchar	Length of the name field (in bytes).
peer: Addrinfo	Peer (IP address, IP port number) destination for the datagram.
datagramLength: uint	Length (in bytes) of the datagram.
datagram: byte[]	Datagram content (bytes in network order).

Response parameter definitions:

socketId: uchar	Socket on which the matching **Request** was issued.

Semantical description:

- The datagram is sent using UDP protocol and the socket's IP host address and port to the peer.

Pre conditions Request:

1) Socket had been opened for UDP using the same address structure as the peer's.

Post conditions:

1) Datagram is sent (but can be lost).

Error codes regarding the sending of UDP datagrams are listed in Table 9.4.4.4.2-1.

Table 9.4.4.4.2-1: resIpUdpSendMsg Error Codes

Name	Description
ErrIpUdpProtMismatch	See Table 9.4.4.7-1.
ErrIpUdpSocketNot	
ErrIpUdpTooLong	
ErrIpUdpIpOffline	

9.4.4.4.3 reqIpUdpRecvMsg Message

C→H reqIpUdpRecvMsg(uchar **socketId)** →
H→C resIpUdpRecvMsg(uchar **socketId,** Addrinfo **peer,** uint **datagramLength,** byte **datagram[])**

- This message allows the **ECI Client** to request the **ECI Host** to receive an UDP datagram from a peer (i.e. hostname, port) sent to the socket with **SocketId**.

Request parameter definitions:

socketId: uchar	Socket (implying port number and host address) on which an UDP datagram is anticipated to be received.

Response parameter definitions:

socketId: uchar	Length of the name field (in bytes).
peer: Addrinfo	IP address + port number of datagram source (peer).
datagramLength: uint	Length (in bytes) of the datagram.
datagram: byte[]	Datagram content (bytes in network order).

Semantical description:

- A datagram can be received on the socket in which case a **Response** is returned.

NOTE 1: Socket close will terminate any pending **reqIpUdpRecvMsg Request**s.

NOTE 2: Issuing multiple **reqIpUdpRecvMsg** before receiving corresponding **Response**s on the same socket is permitted, but the **ECI Host** has no obligation to support queuing of more than 5 of such **Request**s.

Pre conditions Request:

- Socket has been opened for UDP.

Post conditions Response:

- Datagram is sent (but can be lost).

Error codes regarding the receiving of UDP datagrams are listed in Table 9.4.4.4.3-1.

Table 9.4.4.4.3-1: resIpUdpRecvMsg Error Codes

Name	Description
ErrIpUdpSocketNot	See Table 9.4.4.7-1.

9.4.4.5 **ECI** TCP/IP

9.4.4.5.1 General

ECI Clients can send and receive messages over a TCP/IP connection opened on the creation of a socket creating an effective error-free bidirectional byte-stream sequence from the local **ECI Client** to a remote peer service, or vice versa. This allows the **ECI Client** to act as a server to **Requests** for channels from other parties (typically for LAN applications). The messages are listed in Table 9.4.4.5.1-1.

Table 9.4.4.5.1-1: TCP/IP Socket Messages

Message	Type	Dir.	Tag	Description
reqIpTcpConnect	A	C→H	0x5	TCP client connects to TCP server peer.
reqIpTcpSend	A	C→H	0x6	Sends data to connected peer.
reqIpTcpRecv	A	C→H	0x7	Receives data from connected peer.
reqIpTcpAccept	A	C→H	0x8	TCP server peer accepts connection from TPC client peer.

9.4.4.5.2 reqIpTcpConnect Message

C→H reqIpTcpConnect(uchar **socketId,** Addrinfo **peer)** →
H→C resIpTcpConnect(uchar **socketId)**

- This message requests the **ECI Host** to open a connection from an open TCP socket to the peer using the protocol of the socket.

Request parameter definitions:

socketId: uchar	Socket (inplying port number and host address) from which a TCP connections is to be established.
peer: Addrinfo	Peer IP address, IP port to which the connection is to be opened.

Response parameter definitions:

socketId: uchar	Socket ID of the socket on of the **Request**.

Semantical description:

- The local host will attempt to open a TCP connection from the local socket to the peer (IP address, IP port).

Pre conditions:

- Socket has been opened for TCP using the same IP address type (IPv4 or IPv6) as peerAddressType.

Post conditions:

- A TCP connection is established or an error condition is returned.

Error codes regarding the connection via TCP and IP are listed in Table 9.4.4.5.2-1.

Table 9.4.4.5.2-1: resIpTcpConnect Error Codes

Name	Description
ErrIpTcpProtMismatch	See Table 9.4.4.7-1.
ErrIpTcpSockNot	
ErrIpTcpIpOffline	
ErrIpTcpConnRefused	
ErrIpTcpConnTimeout	

9.4.4.5.3 reqIpTCPSend Message

C→H reqIpTcpSend(uchar **socketId,** bool **more,** uint **dataLen,** byte **data[])** →
H→C resIpTcpSend(uchar **socketId,** uint **actLen)**

- This message sends data using TCP on a TCP connected socket.

Request parameter definitions:

socketId: uchar	Socket (implying port number and host address) used for sending the data to the peer.
more: bool	Indicates .if the data and preceding data is to be forwarded to the peer immediately (**more**=**False**) or if more data follows in subsequent reqipTcpSend **Request**s (**more**=**True**).
dataLen: uint	Amount of data to be sent.
data: byte[]	Data to be sent.

Response parameter definitions:

socketId: uchar	Socket ID of the socket on which the send was issued.
actLen: uint	The actual number of bytes successfully sent.

Semantical description:

- The local host shall send the **data** to the peer over a connected TCP/IP socket with **socketID** to the connected peer.

Pre conditions Request:

1) Socket is in a connected TCP/IP mode.

Post conditions Response:

1) In case actLen is not equal to dataLen an error condition shall hold.

Error codes regarding the sending of TCP packets are listed in Table 9.4.4.5.3-1.

Table 9.4.4.5.3-1: resIpTcpSend Error Codes

Name	Description
ErrIpTcpSockNot	See Table 9.4.4.7-1.
ErrIpTcpIpOffline	
ErrIpTcpClosed	
ErrIpTcpConnTimeout	

9.4.4.5.4 reqIpTCPRecv Message

C→H reqIpTcpRecv(uchar **socketId,** uint **maxDataLen)** →
H→C resIpTcpRecv(uchar **socketId,** uint **dataLength,** byte **data[])**

- This message receives data using TCP on a TCP connected socket

Request parameter definitions:

socketId: uchar	Socket (implying port number and host address) used for receiving the data to the peer.
maxDataLen: uint	Maximum amount of data to be received.

Response parameter definitions:

socketId: uchar	Socket ID of the socket on which the receive message was issued.
dataLength: uint	Number of bytes of data received from peer.
data: byte[]	Data as received from peer.

Semantical description:

- The local host receives **data** from the peer over a connected TCP/IP socket with **socketID**.

Pre conditions Request:

1) Socket is a TCP socket.

Post conditions Response:

1) All available data up to length is returned up to the **maxDataLen** field in the **Request**. If no data is available the **Response** shall stall until the connection is closed, the TCP connection is deemed temporarily unavailable or the local connection to the IP network is lost.

Error codes regarding the receiving of TCP packets are listed in Table 9.4.4.5.4-1.

Table 9.4.4.5.4-1: resIpTcpRecv Error Codes

Name	Description
ErrIpTcpSockNot	See Table 9.4.4.7-1.
ErrIpTcpIpOffline	
ErrIpTcpClosed	
ErrIpTcpConnTimeout	

9.4.4.5.5 reqIpTCPAccept Message

C→H reqIpTcpAccept(uchar **socketId)** →
H→C resIpTcpAccept(uchar **socketId,** uchar **newSocketId,** Addrinfo **peer)**

- This message accepts an incoming connection **Request** on a TCP Server socket. Pending connection **Request**s shall be served; with a maximum as defined by the **ECI Host** implementation. Performance requirements for the TCP server are defend in ETSI GR ECI 004 [i.10].

Request parameter definitions:

socketId: uchar	Socket (implying port number and host address) used for receiving connection **Requests**.

Message field definitions:

socketId: uchar	Socket ID of the socket on which the request was issued.
newSocketId: uchar	Socket ID for the newly opened connection to the peer that issued a connection **Request**. The host address and port are inherited from the socket with **socketId**.
peer: Addrinfo	IP address + IP port of peer on the connection.

Semantical description:

- The local **ECI Host** waits for incoming TCP connection **Request**s on the IP-address/port as specified in the socket creation and opens a newly connected socket serving the incoming (or pending) connection **Request**. No **Response** can follow if there is no incoming **Request** or in case the server socket is closed.

Pre conditions Request:

1) Socket is a TCP server socket.

Post conditions Response:

1) A new socket with an opened TCP/IP connection is returned on any available connection **Request** to the server socket or an error is produced.

Error codes regarding the acceptance of TCP connections are listed in Table 9.4.4.5.5-1.

Table 9.4.4.5.5-1: resIpTcpAccept Error Codes

Name	Description
ErrIpTcpListSockNot	See Table 9.4.4.7-1.
ErrIpTcpNoMoreSockets	

9.4.4.6 API for HTTP(S) get services

9.4.4.6.1 General

The **ECI Host** shall provide basic HTTP(S) GET requests to retrieve resources from an IP based HTTP server on behalf of the client. This permits the **ECI Client** to retrieve web based resources (files) from internet servers. HTTPS may among others be used to retrieve Web-API based resources like import or export data as defined in clause 9.7.2 and clause 7.8.4.2.

The security is provided by HTTPS (TLS) of the underlying **CPE**'s TLS implementation.

NOTE: This security should in general not be used to ensure content protection integrity for **ECI Clients**, but may be used to ensure DDOS and other opportunistic attempts to manipulate **ECI Clients** are hampered.

The **ECI Host** shall support an **ECI Client** with a minimum amount of resources to issue HTTP Get requests as defined in ETSI GR ECI 004 [i.10].

The API messages for the HTTP(S) Get API are listed in Table 9.4.4.6.1-1.

Table 9.4.4.6.1-1: HTTP Get API Messages

Message	Type	Dir.	Tag	Description
reqHttpGetFile	A	C→H	0x0	Performs an HTTP Get request on a URL and stores the result in a file.
reqHttpGetData	A	C→H	0x1	Performs an HTTP Get request on a URL and passes the result as data to the Client.

9.4.4.6.2 Applicable Specifications

The HTTP and HTTPS protocol implementation for implementing the **ECI Client** API shall be compliant to HTTP1.1 [43] and [44].

The TLS implementation used for providing HTTP services to the **ECI Client** shall comply with:

1) TLS 1.2, see IETF RFC 5246 [45].
2) TLS AES-GCM, see IETF RFC 5288 [46].
3) TLS Extensions, see IETF RFC 6066 [47].
4) PKIX/X.509 [48] + Updates [49].

Additional extensions may be supported. All TLS implementations shall support the following cipher suites as defined in IETF RFC 5246 [45]:

1) TLS_RSA_WITH_AES_128_CBC_SHA256.
2) TLS_DHE_RSA_WITH_AES_128_GCM_SHA256.

Additional cipher suites for TLS may be supported.

Selection of TLS cipher suits has the following rules:

1) TLS_DHE_RSA_WITH_AES_128_GCM_SHA256 should be the default cipher suit.
2) AEAD cipher suites should be prioritized.
3) DHE based key exchange should be prioritized.
4) Keys longer than 128bit should not be prioritized.
5) 3DES should not be used.
6) RC4 shall not be used (as specified in [50]).
7) MD5 shall not be used (as specified in [51]).

The following processing rules apply:

1) TLS 1.2 shall be the minimum version required by all **ECI** Entities.
2) SSL 2.0 and 3.0 shall not be used.
3) Renegotiation shall not be used.
4) Compression should not be used (acceptable with GCM).
5) Primes for DH/DHE shall be at least 1 024 bit and shall be verified during TLS handshake.
6) Verification of **Certificates** and hosts shall comply with PKIX requirements [48] and [52].

Additional guidance for implementations can be found in the processing rules outlined by the CA/Browser Forum [i.5] and [i.6].

NOTE: In order to ensure interoperability, HTTP servers targeted to support **ECI Clients** with HTTP based services should support compatible modes and options and applicable recommendations as defined here for the HTTP client.

9.4.4.6.3 The reqHttpGetFile and reqHttpGetData Message

C→H **reqHttpGetFile(**filename **fname** ;char **url[]**, char **userAgent[]**; uint **redirs**, uint **timeout**) →
H→C **resHttpGetFile**(uint **httpStatus**)

C→H **reqHttpGetData(**char **url[]**, **userAgent[]**; uint **redirs**, uint **timeout**) →
H→C **resHttpGetData**(uint **httpStatus,** byte **data[]**)

- This **message** requests the **ECI Host** to perform a HTTP request to retrieve a file and return the HTTP status on completion.
- resHttpGetFile returns the resource as a file in the file system of the Client/.
- resHttpGetData returns the resource as message data with a limited size.

Request parameter definition:

fname: fileName	Filename of the file where the result (post data) of the request is stored by the **ECI Host**. Any existing data is overwritten.
url: char[]	URL in UTF-8 encoding [43]. Non-standard port numbers may be specified as part of the URL. TLS shall be used for URLs conforming to the "https URI Scheme" in IETF RFC 7230 [43].
userAgent: char[]	Specifies the User-Agent header field to be used as the HTTP header. **ECI Client**s may specify a specific value anticipated by the HTTP server of the **url** (see note).
redirs: unit	Maximum number of redirects that is permitted to complete the request. Minimum performance figures for **redirs** are defined in ETSI GR ECI 004 [i.10].
timeout: unit	Timeout in milliseconds for the HTTP request to complete. In case of a timeout the request will be aborted and a timeout error will be returned in the **Response**.
NOTE: It is not recommended to use the User-Agent as an access control or selection mechanism for the resource and follow the intended use as defined in IETF RFC 7231 [44].	

Response parameter definition:

httpStatus: uint	Value of the HTTP status.
data: byte[]	Data of the HTTP GET result in network order. The maximum size is limited by the message buffer size.

Detailed semantics:

- The **ECI Host** shall ensure the HTTP requests support a wide range of common file and media types. It is recommended not to include the Accept header field in the HTTP request header. In case an Accept header is added the following Content-Encoding MIME types shall be acceptable for retrieving the resource: application/octet-stream, application/json, image/jpeg, image/png, image/gif, text/plain, text/html, text/css, text/xml and text/javascript.
- The **ECI Host** shall ensure that the HTTP request header Accept-Encoding signals that the following Content-Encodings are acceptable: gzip.

Post Conditions Response:

1) The resource at **url** was retrieved and stored in a file names **fname** (for **resHttpGetFile)**or returned as data (for **rerHttpGetData**) or an error occurred.

The resHttpGetFile and resHttpGetData related error codes are listed in Table 9.4.4.6.3-1.

Table 9.4.4.6.3-1: resHttpGetFile and resHttpGetData Error Codes

Name	Description
ErrHttpGetNoSockets	See Table 9.4.4.6.4-1.
ErrHttpGetProtNotAvail	
ErrHttpGetPortNotAvail	
ErrHttpHostUnknown	
ErrHttpDnsOffline	
ErrHttpIpOffline	
ErrHttpTimeout	
ErrHttpGetFSFailure	
ErrHttpGetFSExceeded	
ErrHttpGetTlsAuth	
ErrHttpGetRedir	
ErrHttpGetData	

9.4.4.6.4 Error Codes for the HTTP Get API

The values are meanings of the API specific errors that can be returned by the **Response** messages for this API are listed in Table 9.4.4.6.4-1.

Table 9.4.4.6.4-1: Error codes for the HTTP Get APIs

Name	Value	Description
ErrHttpGetNoSockets	-257	See corresponding value for error codes in Table 9.4.4.7-1 for the IP Socket API.
ErrHttpGetProtNotAvail	-258	
ErrHttpGetPortNotAvail	-259	
ErrHttpHostUnknown	-261	
ErrHttpDnsOffline	-263	
ErrHttpIpOffline	-267	
ErrHttpTimeout	-270	The HTTP request could not finish within the timeout set in the request
ErrHttpGetFSFailure	-512	Value+256 corresponds to value of error codes in Table 9.4.5.5-1 for the File System API.
ErrHttpGetFSExceeded	-514	
ErrHttpGetTlsAuth	-768	Server or data could not be successfully authenticated by the TLS protocol.
ErrHttpGetRedir	-784	Number of redirects exceeded.
ErrHttpError	-785	The resource could not be retrieved from the server; the HTTP error code indicates the reason.
ErrHttpGetData	-786	Data of resource exceeded maximum length data field.

9.4.4.7 Error Codes for the IP Socket API

The values are meanings of the API specific errors that can be returned by the **Response** messages for this API are listed in Table 9.4.4.7-1.

Table 9.4.4.7-1: Error codes for IP Socket APIs

Name	Value	Description
ErrIpSourceProt	-256	Invalid combination of source and protocol.
ErrIpNoSockets	-257	No more sockets available.
ErrIpProtNotAvail	-258	Protocol not available.
ErrIpPortNotAvail	-259	Requested port is not available.
ErrIpSocketNotOpen	-260	Socket was not open.
ErrIpHostUnknown	-261	**ECI Host** unknown.
ErrIpHost	-262	**ECI Host** known but no address available (for the specified IP address type).
ErrDnsOffline	-263	DNS service is offline, possibly temporarily.
ErrIpUdpProtMismatch	-264	Peer address does not match socket protocol.
ErrIpUdpSockNot	-265	Socket is not a UDP socket.
ErrIpUdpTooLong	-266	Datagram to long for single UDP message.
ErrIpUdpIpOffline	-267	IP connection offline (peer cannot be reached).
ErrIpTcpProtMismatch	-268	Peer address does not match socket protocol.
ErrIpTcpSockNot	-269	Socket is not a TCP socket.
ErrIpTcpIpOffline	-258	No local IP internet connection at this time.
ErrIpTcpConnRefused	-259	Connection not accepted by peer host on this port.
ErrIpTcpConnTimeout	-260	Not able to get **Response** from peer **ECI Host**.
ErrIpTcpClosed	-261	TCP connection not or no longer available.
ErrIpTcpListSockNot	-262	Socket is not a TCP server socket.
ErrIpTcpNoMoreSockets	-263	Incoming connection **Request** was received but host out of sockets.
RFU	Other	Reserved for Future Use.

9.4.5 API for access to the file system

9.4.5.1 Introduction

The **ECI Client** has access to a private file system to store a limited amount of data that shall survive **ECI Client** lifecycles, **CPE** power cycles, system crashes, etc. under normal operating conditions. The reliability should at least be equal to the regular **CPE** file system; i.e. failures can occur under somewhat exceptional circumstances which can result in **user** discomfort. It is up to the security system managing the **ECI Client** to ensure no undue loss of rights to content access occurs for the **user**. The file system is not secure. Manipulation by entities other than the designated **ECI Client** and its supporting **ECI Host** under regular conditions (i.e. uncompromised **CPE** and **ECI Host**) shall not be possible.

The file system abstraction is that of a single flat directory. A basic directory service is available. The file system access functions are analogous to Unix/Linux/Posix file system calls, like open, close, write, read, lseek, opendir, readdir and lstat.

A minimum amount of file system storage shall be available to each **ECI Client** if it is stored by the **user**. This amount is defined in ETSI GR ECI 004 [i.10].

The file system API is partitioned in three subparts:

1) File opening and closing.

2) Reading and writing a file, random access and deleting selected data from a file.

3) Directory services.

Filenames shall consist of 8-bit ASCII character sequence of minimum 1 and maximum 8 of the following characters (comma separated): A-Z, a-z, 0-9, _ and shall be terminated with a NULL character. The filename definition is shown in Table 9.4.5.1-1.

Table 9.4.5.1-1: FileName structure

```
typedef char fileName[9];
```

Log files provide functionality permitting **ECI Clients** to write limited amounts of data in a buffered fashion; i.e. without halting execution. The number of log files per **ECI Client** is defined in xxx (min 2 per client). This makes such files suitable for application level logging, tracing and post mortem analysis.

9.4.5.2 File Opening and Closing

9.4.5.2.1 General

ECI Clients can open a file for reading and/or writing which delivers a fileHandle through which the subsequent read and write accesses can be performed. If a file does not exist it can be created. The file has a property "file location" which points to the present location for accessing the file.

FileHandles shall be managed by the **ECI Host**. A file handle that was closed shall not be reused immediately afterwards to ensure unsynchronized accesses to the file by an **ECI Client** will not result in accesses to the wrong file.

The messages listed in Table 9.4.5.2.1-1 are defined:

Table 9.4.5.2.1-1: File Open and Close messages

Message	Type	Dir.	Tag	Description
reqFileOpen	A	C→H	0x0	Opens an **ECI Client** private file.
reqFileClose	A	C→H	0x1	Closes an open file.

9.4.5.2.2 reqFileOpen Message

C→H reqFileOpen(fileName **fname,** uint **fileOpenOptions**) →
H→C resFileOpen(uchar **fileHandle)**

- This message allows the **ECI Client** to request the **ECI Host** to open a file with certain permissions for access.

Request parameter definitions:

fname: filename	Name of file to be opened.
fileOpenOptions: unit	Access mode in which to open file. The permitted values and their meaning are defined in Table 9.4.5.2.2-1.

Table 9.4.5.2.2-1: File Opening Options

Name	Bits	Value	Description
FileRead	0,1	0b00	File is opened for reading. The file location is set at the start of the file.
FileWriteAppend	0,1	0b01	File is opened for writing; subsequent writes are appended to existing file. The file location is set at the end of the file.
FileWriteOver	0,1	b11	File is opened for writing at any location. The file location is set at the end of the file.
Not in use	0,1	0b10	Not permitted.
LogFileNo	2	0b0	Regular file
LogFileYes	2	0b1	Special log file permitting synchronous writes.
Bits32-2		Other	Reserved for future use.

Response parameter definitions:

fileHandle: uchar	Reference (handle) to opened file.

Post conditions Request:

1) File opened in the desired access mode or an error will be returned. Error codes are listed in Table 9.4.5.2.2-2.

Table 9.4.5.2.2-2: resfileOpen error codes

Name	Description
ErrFileNameNotExist	See Table 9.4.5.5-1.
ErrFileQuotaExceeded	
ErrFileSystemFailure	

9.4.5.2.3 reqFileClose Message

C→H reqFileClose(uchar **fileHandle) →**
H→C resFileClose()

- This message closes the access to the file opened with **fileHandle**. Error codes regarding the closing of a file are listed in Table 9.4.5.2.3-1.

Request parameter definitions:

fileHandle: uchar	Handle of file to be closed.

Pre conditions Request:

1) fileHandle is in opened state.

Post conditions Request:

1) Subsequent accesses to fileHandle will fail with ErrFileNotOpen.

2) Any pending writes will be committed (unless an error occurs).

Table 9.4.5.2.3-1: resfileClose error codes

Name	Description
ErrFileHandleNotExist	See Table 9.4.5.5-1.
ErrFileSystemFailure	

9.4.5.3 File Access

9.4.5.3.1 General

The file access messages permit to read and write a file accessed through a file handle, and repositioning the present location in the file for reading/writing. The primitives defined have a direct correspondence to Linux/Unix conventions. The defined messages are listed in Table 9.4.5.3.1-1.

NOTE: reqFileWrite and reqFileRead have strong resemblance to reqTcpSend and reqTcpRecv.

Table 9.4.5.3.1-1: File Access messages

Message	Type	Dir.	Tag	Description
reqFileWrite	A	C→H	0x2	Writes consecutive bytes starting from the present file location.
reqFileRead	A	C→H	0x3	Reads consecutive bytes starting from the present file location.
reqFileSeek	A	C→H	0x4	Repositions the present file location.
reqFileRemoveData	A	C→H	0x5	Deletes data from a file at current location.
callFileDataLog	S	C→H	0x6	Appends data at the end of a buffered file.

9.4.5.3.2 reqFileWrite Message

C→H reqFileWrite(uchar **fileHandle,** bool **sync,** uint **dataLen,** byte **data[]) →**
H→C resFileWrite(uchar **fileHandle)**

- This message writes dataLen bytes to the file starting at the present file location.

Request parameter definitions:

fileHandle: uchar	Handle of file to be written to
sync: bool	If True write **Response** ensures the state of the file system is up to date with this and all preceding writes. If False the **ECI Host** can buffer write **Requests** (which can still be lost on a system failure).
dataLen: uint	Number of bytes to be written to the file.
data: byte[]	Data to be written to the file.

Response parameter definitions:

fileHandle: uchar	Handle of file that was written to.

Pre conditions Request:

1) File opened in write mode (FileWriteOver or FileWriteAppend mode).

2) File location can be written to: if file opened in FileWriteAppend mode the file location shall be at the end.

3) Amount of data to be written does not cause a file system quota problem.

Post conditions Request:

1) File state will be updated and the file location will be advanced from present (pending other buffered operations on the file) to present+dataLen, unless an error occurs.

2) In case of a successful write and **sync** the data is committed in NV state in the **ECI Host** file system.

Error codes are listed in Table 9.4.5.3.2-1.

Table 9.4.5.3.2-1: resFileWrite error codes

Name	Description
ErrFileHandleNotExist	See Table 9.4.5.5-1.
ErrFileQuotaExceeded	
ErrFileSystemFailure	
ErrFileWriteNot	

9.4.5.3.3 reqFileRead Message

C→H reqFileRead(uchar **fileHandle**, uint **dataLen) →**
H→C resFileRead(uchar **fileHandle**, uint **dataRead,** byte **data[])**

- This message reads the maximum dataLen bytes from the file starting at the present file location. Error codes regarding the reading of data from a file are listed in Table 9.4.5.3.3-1.

Request parameter definitions:

fileHandle: uchar	Handle of file to be read from.
dataLen: uint	Maximum number of bytes to be read.

Response parameter definitions:

fileHandle: uchar	Handle of file that was read from.
dataRead: uint	Number of bytes that was read and is stored in **data**.
data: byte []	Data that was read.

Pre conditions Request:

1) File is opened.

Post conditions Request:

1) An error occurred; or
2) Minimum of **dataLen** or remaining bytes in file from last file location is read from file; and
3) File location has been incremented with **dataRead**;
4) Unless an error occurs, file location will be advanced by **dataLen** or be located at the end of the file.

Table 9.4.5.3.3-1: resFileReade error codes

Name	Description
ErrFileHandleNotExist	See Table 9.4.5.5-1.
ErrFileSystemFailure	

9.4.5.3.4 reqFileSeek Message

C→H reqFileSeek(uchar **fileHandle,** int **offset,** uchar **seekPos) →**
H→C resFileSeek(uchar **fileHandle,** int **remOffset)**

- This message positions a pointer at a certain position within an opened file and returns parts of the file content.

Request parameter definitions:

fileHandle: uchar	Handle of file of which the file location is to be changed.
offset: int	Offset from seek reference location as specified by **seekPos** that the file location shall assume.
seekPos: uchar	See Table 9.4.5.3.4-1.

Table 9.4.5.3.4-1: File Seek reference location

Name	Value	Description
FileSeekSet	0x00	File reference location is at the start of the file.
FileSeekCur	0x01	File reference location is at the present file location.
FileSeekEnd	0x02	File reference location is at the end of the file.
RFU	Other	Reserved for future use.

Response parameter definitions:

fileHandle: uchar	Handle of file of which the file location was changed.
remOffset: int	Difference between the offset specified and the offset to which the file location is set.

Detailed semantics:

- The file location is repositioned and is defined in the parameter description of the **Request**. The file location will never be positioned beyond the end of the file or before the start of the file. The difference between the requested offset and the actual offset from the file reference location is returned in the **remOffset** result parameter. Error codes are listed in Table 9.4.5.3.4-2.

Pre conditions Request:

1) File is opened.

Post conditions Request:

1) An error occurred; or

2) The file location will be set as defined above; and

3) **remOffset** will reflect difference between offset and actual file location is defined above.

Table 9.4.5.3.4-2: resFileReade error codes

Name	Description
ErrFileHandleNotExist	See Table 9.4.5.5-1.
ErrFileSystemFailure	

9.4.5.3.5 reqFileRemoveData Message

C→H reqFileRemoveData(uchar **fileHandle,** bool **sync,** uint **dataLen) →**
H→C resFileRemoveData(uchar **fileHandle)**

- This message removes dataLen bytes from the file starting at the present file location.

Request parameter definitions:

fileHandle: uchar	Handle of file.
sync: bool	If True write **Response** ensures the state of the file system is up to date with this and all preceding writes. If False the **ECI Host** can buffer write **Request**s (which can still be lost on a system failure).
dataLen: uint	Number of bytes to be removed from the file. If this exceeds the end of the file only the bytes until the end of the file are removed.

Response parameter definitions:

fileHandle: uchar	Handle of file that was written to.

Pre conditions Request:

1) File opened in write mode (FileWriteOver mode).

Post conditions Request:

1) File state will be updated. The file location will remain the same.

2) In case of a successful removal and **sync** the data is committed in NV state in the **ECI Host** file system.

Error codes are listed in Table 9.4.5.3.5-1.

Table 9.4.5.3.5-1: resFileWrite error codes

Name	Description
ErrFileHandleNotExist	See Table 9.4.5.5-1.
ErrFileSystemFailure	
ErrFileWriteNot	

9.4.5.3.6 callFileDataLog Message

C→H callFileDataLog(uchar **fileHandle,** uint **dataLen**, byte **data[])**

- This message appends dataLen bytes (in data) at the end of the file using a system buffer.

Call parameter definitions:

fileHandle: uchar	Handle of file.
dataLen: uint	Number of bytes to be appended to the logfile.
data[]: byte	Data to be written.

Pre conditions Call:

1) File opened in write mode (FileWriteOver or FileWriteAppend mode).

2) File location is set at the end of the file.

3) Amount of data to be written does not cause a file system quota problem.

Post conditions Call:

1) File state is updated and the file location is advanced from present to present+dataLen, unless an error occurs.

2) The result will be committed to the **ECI Host** file system unless a system failure occurs.

Detailed Semantics:

1) The **ECI Host** shall buffer the data and append it to the end of the file as soon as expedient.

2) The maximum buffer space provided for a log for this purposes is defined in ETSI GR ECI 004 [i.10]. (64 kbyte per log file).

Error codes are listed in Table 9.4.5.3.6-1.

Table 9.4.5.3.6-1: resFileLog error codes

Name	Description
ErrFileHandleNotExist	Definition see Table 9.4.5.5-1.
ErrFileQuotaExceeded	
ErrFileSystemFailure	
ErrFileWriteNot	
ErrFileLogNot	

9.4.5.4 Directory services

9.4.5.4.1 General

The directory services offer functions to scan the available **ECI Client** files. Files are characterized by their unique name, and have size and last modified time attributes. The available messages are listed in Table 9.4.5.4.1-1.

NOTE: The time attribute has the same degree of integrity as the file system and file content itself.

Table 9.4.5.4.1-1: File directory service messages

Message	Type	Dir.	Tag	Description
reqFileStat	A	C→H	0x07	Return size and modification time of file.
reqFileCreate	A	C→H	0x08	Create a new file.
reqFileDelete	A	C→H	0x09	Delete a file.
reqFileDir	A	C→H	0x0A	Lists file names of files available in **ECI Clients** file system.

9.4.5.4.2 reqFileStat Message

C→H reqFileStat(fileName **filename)** →
H→C resFileStat(uint **size;** long **mtime)**

- This message allows the **ECI Client** to request the **ECI Host** to retrieve file size and last modification time of a stored file.

Request parameter definitions:

filename: filename	Name of the file for which the properties shall be retrieved.

Response parameter definitions:

size: uint	Size of file (in bytes).
mtime: long	Clock time of last synchronized file modification.

Pre condition Request:

1) Filename is an existing file in the file system.

Post conditions Request:

1) **size** and **mtime** reflect the properties of the file with name **filename** or an error occurred.

Error codes are listed in Table 9.4.5.4.2-1.

Table 9.4.5.4.2-1: resFileStat error codes

Name	Description
ErrFileNameNotExist	See Table 9.4.5.5-1.
ErrFileSystemFailure	

9.4.5.4.3 reqFileCreate Message

C→H reqFileCreate(fileName **filename)** →
H→C resFileCreate()

- This message allows the **ECI Client** to request the **ECI Host** to create a new empty file. Any existing file with the same name is deleted.

Request parameter definitions:

filename: filename	Name of the new empty file that shall be created.

Detailed semantics:

- The created file shall exist after a system failure unless the file system has been corrupted.

Post conditions Request:

1) Empty file with name filename exists in **ECI Client** file system with modification timestamp set to the current time or an error occurred.

Error codes are listed in Table 9.4.5.4.3-1.

Table 9.4.5.4.3-1: resFileCreate error codes

Name	Description
ErrFileQuotaExceeded	See Table 9.4.5.5-1.
ErrFileSystemFailure	

9.4.5.4.4 reqFileDelete Message

C→H reqFileDelete(fileName **filename)** →
H→C resFileDelete()

- This Message deletes a file with name **filename**.

Request parameter definitions:

filename: fileName	Name of the new empty file that shall be created.

Detailed semantics:

- The deleted file shall not exist after a system failure unless the file system has been corrupted.

Post conditions Request:

1) File with name **filename** does not exist in the file system.

Error codes are listed in Table 9.4.5.4.4-1.

Table 9.4.5.4.4-1: resFileDelete error codes

Name	Description
ErrFileNameNotExist	See Table 9.4.5.5-1.
ErrFileSystemFailure	

9.4.5.4.5 reqFileDir Message

C→H reqFileDir(ushort **maxNr)** →
H→C resFileDir(uint **listLen;** fileName **dirList[])**

- This message provides a list of filenames max. maxNr items. The list order is undefined.

Request parameter definitions:

maxNr: ushort	Maximum number of filenames that will be retrieved.

Response parameter definitions:

listLen: uint	Length of list in bytes.
dirList: filename []	Array of filenames of files available to the **ECI Client**.

Error codes are listed in Table 9.4.5.4.5-1.

Table 9.4.5.4.5-1: resFileDelete error codes

Name	Description
ErrFileSystemFailure	See Table 9.4.4.7-1.

9.4.5.5 Error Codes for the File System API

The values are meanings of the API specific errors that can be returned by the **Response** messages for this API are listed in Table 9.4.5.5-1.

Table 9.4.5.5-1: File System API Error Codes

Name	Value	Description
ErrFileSystemFailuret	-256	Corrupt or dismounted file system.
ErrFileNameNotExist	-257	File name does not exist in file system.
ErrFileQuotaExceeded	-258	File system resources for **ECI Client** have been exceeded.
ErrFileNameNotExists	-259	Filename does not exist in **ECI Client**'s file system.
ErrFileHandleNotExists	-260	File handle does not exist (might have been closed previously).
ErrFileAppendNot	-261	Attempt to write to file was not at the end of the file.
RFU	Other	Reserved for Future Use.

9.4.6 API for access to the Time/Clock resource

9.4.6.1 Introduction

The **ECI Client** has access to timer events and the time of day through a simple API. The clock is using the standard system clock; no specific secure or otherwise qualified time is available. The timer permits a message to be generated at some (delay) time in the future. The timer event can be cancelled.

NOTE: Using a combination of clock and timer API regularly occurring timer events can be created.

The Timer and Clock APIs are split into two parts:

1) Timer API.

2) Clock API.

9.4.6.2 Timer API

9.4.6.2.1 General

The timer API permits an **ECI Client** to set a timer that will send a **Response** at the set time. If required the event can be cancelled by the **ECI Client**. The number of outstanding timers at one time can be limited by implementation constraints. The minimum number of outstanding timers an **ECI Host** shall support for each **ECI Client** is defined in ETSI GR ECI 004 [i.10]. The messages for the timer API are listed in Table 9.4.6.2.1-1.

Table 9.4.6.2.1-1: Timer API messages

Message	Type	Dir.	Tag	Description
reqTimerEvent	A	C→H	0x0	Sets a timer event in the future.
reqTimerCancel	A	C→H	0x1	Cancels a previously set timer event.

9.4.6.2.2 reqTimerEvent Message

C→H reqTimerEvent(uint **timeInterval) →**
H→C resTimerEvent()

- This message sets timer in the future and receives a **Response** when the timer expires.

Request parameter definitions:

timeInterval: uint	Time in milliseconds in the future.

Post condition Request:

- After timeInterval milliseconds the resTimerEvent will be sent to the **ECI Client** unless reqTimerCancel is received first.

Pre conditions Response:

- The timer expired and no reqTimerCancel for the timer was received.

Error codes are listed in Table 9.4.6.2.2-1.

Table 9.4.6.2.2-1: resTimerEvent Error Codes

Name	Description
ErrTimerMaxExceeded	See Table 9.4.6.4-1.

9.4.6.2.3 reqTimerCancel Message

C→H reqTimerCancel(msgId **id) →**
H→C resTimerCancel()

- This message cancels previously set timer per message identifier of the original **Request**.

Request parameter definitions:

id: msgId	Cancel the timer which was set with an asynchronous message with message id.

Pre conditions Request:

1) Id was returned as result of a reqTimerEvent and timer did not yet expire.

Post conditions Response:

1) Timer is cancelled - no resTimerCancel will be sent - or an error is returned.

2) TimerExpired errors will occur if the timer was cancelled but **resTimerEvent** was received before the **resTimerCancel**.

9.4.6.3 Clock API

9.4.6.3.1 General

The Clock API permits the **ECI Client** to read the clock as an integer and to convert that to the local time representation. The Clock API messages are listed in Table 9.4.6.3.1-1.

Table 9.4.6.3.1-1: Clock API Messages

Message	Type	Dir.	Tag	Description
getTime	S	C→H	0x3	Reads the local system clock as integer value.
callLocaltime	S	C→H	0x4	Converts time integer value into localtime.

9.4.6.3.2 getTime Message

C→H long **getTime()**

- This message returns the time in seconds from January 1, 1970, 0:00 GMT.

9.4.6.3.3 callLocaltime Message

C→H callLocaltime(long **time;** tm ***tim)**

- This message converts **time** into human representation and is defined in the structure **tim**. Analogous to c-library function localtime from <time.h>.

Call parameter definitions:

time: long	Time as integer representation of seconds from January 1, 1970, 0:00 GMT to be converted to a local time.
tim: tm *	Pointer to tm structure that will be set to the local time. **tm** is defined in Table 9.4.6.3.3-1.

Table 9.4.6.3.3-1: Type definition for human time representation structure tm

```
typedef struct tm {
   int tm_sec;  // 0 .. 59 (seconds) or 60 in case of a leap second
   int tm_min;  // 0 .. 59 (minutes)
   int tm_hour     // 0 .. 23 (hours)
   int tm_mday; // 1 .. 31 (day of month)
   int tm_mon;     // 1 .. 12 (month)
   int tm_year; // year - 1900
   int tm_wday; // 0 .. 6 (day of week; 0=Sunday)
   int tm_yday; // 0 .. 365 (day of year, 0= 1jan)
   int tm_isdst;   // 1=daylight saving in effect, 0=no daylight saving
   char tm_zone[15]; // string for time zone: e.g. GMT, CET
   int tm_gmtoff;  // local timeoffset from GMT
} tm ;
```

9.4.6.4 Error Codes for the Time and Clock API

The values are meanings of the API specific errors that can be returned by the **Response** messages for this API are listed in Table 9.4.6.4-1.

Table 9.4.6.4-1: Time and Clock API error codes

Name	Value	Description
ErrTimerMaxExceeded	256	maximum timer duration exceeded.
RFU	other	Reserved for Future Use.

9.4.7 API for access to the Power management

9.4.7.1 Introduction

The **ECI Client** has access to the power management interface of the **ECI Host**. This interface permits the **ECI Client** to do either a simple power down or a negotiated power down on a system standby event and permits the **ECI Client** to restart **CPE** and the **ECI Client** at a later time from a standby power state in order to perform background functions. The **ECI Host** has the following power states:

- **PwrOn:** the **ECI Host** is functional and not intending to power down.
- **PwrToStby:** the **ECI Host** intends to go to standby state (but can return to PowerOn state). All **ECI Clients** are typically requested to power down.
- Standby: **ECI Host** and **ECI Client** state are non-functional. The **CPE** (and thus the **ECI Host** and **ECI Client**) can wake up from this state on pre-arranged events (typically a timer).
- Power-off: the **CPE** has no power. **ECI Host** and **ECI Client** are not in a functional state.

ECI Clients can act in a simple power management mode and simply be shut down as and when the **ECI Host** finds it appropriate to do so. Alternatively **ECI Clients** can request to be in managed mode by sending a **reqPwrInfo(PwrInfoOn)** message. In this mode they will be notified of the **ECI Hosts** intention to go to power down using the **reqPwrChange** message, which the **ECI Client** can acknowledge with **resPwrChange(PwrDown)** or postpone with an appropriate parameter to **resPwrChange(PwrUp)** until such time it has completed and it is ready to go to standby state. The **ECI Host** shall regularly re-iterate the **reqPwrChange** message. NOTE: no full guarantee can be provided that the **ECI Client** can always complete all activities (e.g. in the case of an uncontrolled power failure or prolonged deference of readiness to go to standby).

Figure 9.4.7.1-1 presents the **ECI Host** state with conditions for state transitions and actions/messages that are triggered on the transition to **ECI Clients** that are in managed mode.

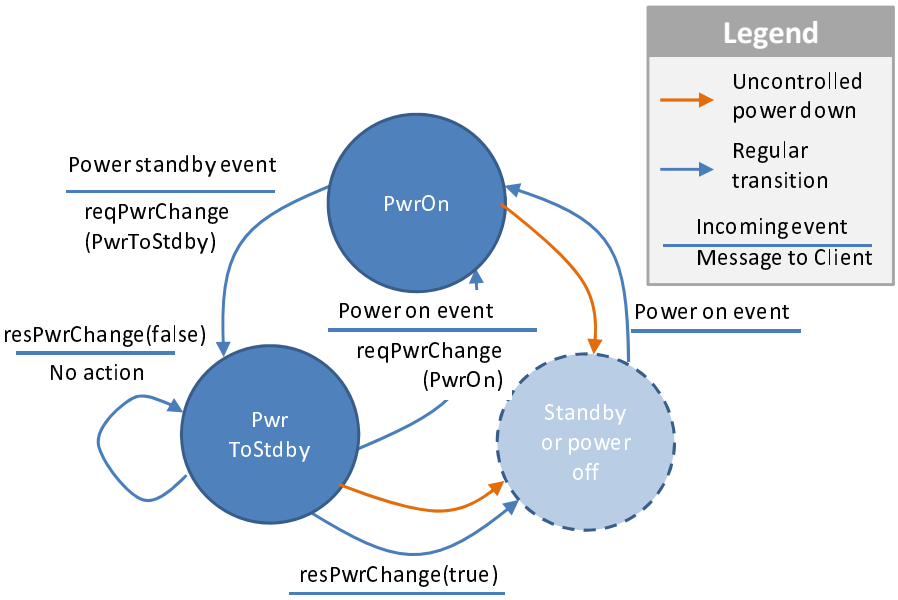


Figure 9.4.7.1-1: ECI Host power states and main interaction with a managed Client

ECI Clients and **ECI Hosts** shall be able to manage recovery from an *uncontrolled power down event*. In such cases it is permitted that regular **ECI Client** and **ECI Host** functionality is impeded in a non-permanent manner trying to minimize the problems that may occur to the **user**.

CPEs can have options to wake up from low-power state on network event features or other low-power modes. **ECI** does not define no specific behaviour for such power-modes and their interaction with the **ECI Host** or **ECI Clients** other than that **ECI Host** and **ECI Client** services shall continue to be functional if the **ECI Host** is in **PwrOn** or **PwrToStdby** state. In particular: there is no state specific to suspended execution.

ECI Clients shall be able to request the **ECI Host** to wake up from standby state at some time in the future and send a message to the **ECI Client**.

The Power Management API is split in the following groups of messages:

1) Power Transitions: managing orderly shutdown of **ECI Clients**. Details are defined in clause 9.4.7.2.

2) Timed power-wakeup functions on behalf of **ECI Clients**, Details are defined in clause 9.4.7.3.

9.4.7.2 Power Transition API messages definition

9.4.7.2.1 General

This clause of the Power Management API defines the functionality permitting **ECI Clients** to do an informed shutdown on an announced power down event in the **ECI Host** so as to provide an optimal service to the **user**. The defined messages are listed in Table 9.4.7.2.1-1.

Table 9.4.7.2.1-1: Power Transition Messages

Message	Type	Dir.	Tag	Description
getPwrStatus	S	C→H	0x0	Gets current value power status.
setPwrInfo	S	C→H	0x1	Requests event notices for changes in power status.
reqPwrChange	A	H→C	0x2	Notice of power status change.

ECI Clients shall not terminate after sending a **resPwrInfo(PwrDown)** but be ready to resume regular functions on receiving a **reqPwrChange(PwrOn)** message.

9.4.7.2.2 getPwrStatus Message

C→H uchar **getPwrStatus()**

- This message returns the current power status of the **ECI Host**.

Property definition: see Table 9.4.7.2.2-1.

Table 9.4.7.2.2-1: Host Power Status Values

Name	Value	Description
PwrOn	0x00	Default IP Address of **ECI Host**.
PwrToStdby	0x01	**ECI Host** IP address used for WAN (internet) communication.
RFU	other	Reserved for Future Use.

9.4.7.2.3 setPwrInfo Message

C→H setPwrInfo(bool **pwrInfo)**

- This message permits to enter and leave managed powerdown mode and control the **ECI Host** sending the **ECI Client resPwrChange** messages on power state change events.

Property definition:

- **pwrInfo** equal **true** is managed power mode; **pwrInfo** equal **false** is unmanaged power mode.

Semantical description:

- When **pwrInfo** is **True** the **ECI Host** will inform the **ECI Client** of power state changes and will not power down the **ECI Client** until the **ECI Client** confirms a reqPwrChange(PwrToStby).
  On **pwrInfo** is **False** the **ECI Host** will not inform the **ECI Client** of power state changes and will power down the **ECI Client** "at will".

- After start-up the state of **PowerInfo** for each **ECI Client** is **False**.

NOTE: **ECI Client**s that rely on a managed power down are suggested not commence with power-down cycle sensitive activities until they have sent the **ECI Host** the **reqPwrInfo(True)** message.

9.4.7.2.4 reqPwrChange Message

H → C reqPwrChange(uchar **hostPwrState)** →
C→H resPowerChange(bool **ready)**

- **This message** signals a change of power-state and if argument is **PwrToStdby Request**s the **ECI Client** to acknowledge or decline being ready to go to standby in a controlled way.

Request parameter definitions:

hostPwrState: uchar	New **ECI Host** power state. The possible values are defined in Table 9.4.7.2.2-1.

Response parameter definitions:

ready: bool	Indicates preparedness of **ECI Client** to enter standby state.

Semantical Description

- The **ECI Host** shall repeat sending this message in case the **ECI Client Response** is negative (not ready). Figures for the minimum repetition rate and a timeout are given in ETSI GR ECI 004 [i.10].

Pre conditions Request:

1) PwrInfo == True.

2) There was a (recent) power state change in the **ECI Host** and the **ECI Client** has not (yet) acknowledged to be ready to go to standby state.

Post conditions Response:

1) **ECI Client** ready to go to standby state if **ready** == **True**, not so if **ready == False.**

Error codes are defined in Table 9.4.7.2.4-1.

Table 9.4.7.2.4-1: ansPwrChange error codes

Name	Description
ErrPwrInfoNot	See Table 9.4.7.4-1.

NOTE: **ECI Hosts** take the **ErrPwrInfoNot error** for information only.

9.4.7.3 Wakeup from Standby Messages definition

9.4.7.3.1 General

This clause of the Power Management API defines the functionality permitting **ECI Client**s to resume execution on a pre-programmed time, waking up the **CPE** from Standby power state if required. The defined messages are listed in Table 9.4.7.3-1.

Table 9.4.7.3-1: Wakeup from standby Messages

Message	Type	Dir.	Tag	Description
setPwrWakeup	set	C→H	0x3	Sets wakeup time for **ECI Client**.
reqPwrWakeupEvent	A	H→C	0x4	Signals wakeup timer expiration.

9.4.7.3.2 setPwrWakeup Message

C→H setPwrWakeup(uint **time)**

- This message sets a timer: After **time** the **ECI Host** shall wakeup **ECI Client** from Standby if needed and send a **reqPwrWakeupEvent()**.

Property definition:

time: uint	Time in seconds until the **ECI Host** shall generate a wakeup event for the **ECI Client**. Value 0 means no wakeup event is required by the **ECI Client**.

Detailed semantics:

- In case an **ECI Host** is not impeded it shall wake up from standby and start an **ECI Client immediately**. In case it is impeded send the wakeup event at the soonest possible occasion thereafter. Time accuracy requiremnets are defined in ETSI GR ECI 004 [i.10].

9.4.7.3.3 reqPwrWakeupEvent Message

H→C reqPwrWakeupEvent() →
C→H resWakeupEvent()

- This message notifies **ECI Client** of its wakeup timer expiration with this message.
  The **ECI Client** shall acknowledge this **Request** with a **Response** when critical processing of wakeup event has been completed.

Detailed semantics:

- The **ECI Host** shall attempt to resend this message at successive **ECI Client** initialization events until it is acknowledged by the **ECI Client** with a **resPwrWakeupEvent()** message. The event is sent during **PwrOn** power state but delayed during **PwrToStdby**.

Pre conditions Request:

1) Power wakeup timer for **ECI Client** was previously set and has expired.

2) The event was not yet acknowledged with a **Response**.

3) **ECI Host** is in **PwrOn** power state.

Post conditions Response:

1) **ECI Host** shall stop sending **reqPwrWakeupEvent**() messages based on the power change event of the matching **Request**; refer **Pre condition** 2).

9.4.7.4 Error codes for the Power Transitions API

The values are meanings of the API specific errors that can be returned by the **Response** messages for this API are listed below in Table 9.4.7.4-1.

Table 9.4.7.4-1: Error codes for Power Transitions API

Name	Value	Description
ErrPwrInfoNot	-256	**ECI Client** indicates it did not request to be informed about power state change events.

9.4.8 API for access to the Country/Language setting resource

9.4.8.1 Introduction

The API for country and language settings permits an **ECI Client** or an **ECI Host** to request the actual country and language settings of the **user** from the **ECI Host** or an **ECI Client** respectively. The messages for the Country/Language setting API are listed in Table 9.4.8.1-1.

Table 9.4.8.1-1: Country/Language setting API messages

Message	Type	Dir.	Tag	Description
reqHCountry	A	C→H	0x0	Requests the actual **ECI Host** preferred country setting.
reqCCountry	A	H→C	0x1	Requests the actual **ECI Client** preferred country setting.
reqHLanguage	A	C→H	0x2	Requests the actual **ECI Host** preferred language setting.
reqCLanguage	A	H→C	0x3	Requests the actual **ECI Client** preferred language setting.

9.4.8.2 Country/Language API Message Definitions

9.4.8.2.1 reqHCountry setting Message

C→H reqHCountry() →
H→C resHCountry setting (uint **iso_3166_country_code)**

- This message allows the **ECI Client** to request the actual setting of the Country the **user** currently resides and receives a **Response** of the stored country setting from the **ECI Host**.

Response parameter definitions:

iso_3166_country_code: uint	This field contains the current **ECI Host** country setting. The country code is a 24-bit field that identifies the Host country using 3 uppercase characters as specified by ISO 3166-1 alpha 3 [54]. Each character is coded as 8-bits according to ISO 8859-1 [53].

Error codes are listed in Table 9.4.8.2.1-1.

Table 9.4.8.2.1-1: reqHCountry Error Codes

Name	Description
ErrCountryNotExists	**See** Table 9.4.8.2.5-1.

9.4.8.2.2 reqCCountry setting Message

H→C reqCCountry() →
C→H resCCountry setting (uint **iso_3166_country_code)**

- This message allows the **ECI Host** to request the actual setting of the Country the **user** currently resides and receives a **Response** of the stored country setting from the **ECI Client**.

Response parameter definitions:

iso_3166_country_code: uint	This field contains the current **ECI Host** country setting. The country code is a 24-bit field that identifies the Host country using 3 uppercase characters as specified by ISO 3166-1 alpha 3 [54]. Each character is coded as 8-bits according to ISO 8859-1 [53].

Error codes are listed in Table 9.4.8.2.2-1.

Table 9.4.8.2.2-1: reqCCountry Error Codes

Name	Description
ErrCountryNotExists	**See** Table 9.4.8.2.5-1.

9.4.8.2.3 reqHLanguage setting Message

H→C reqHLanguage(uint **iso_3166_language_code) →**
C→H resHLanguage setting()

- This message allows the **ECI Client** to request the actual setting of the language the **user** currently prefers and receives a **Response** of the stored language setting from the **ECI Host**.

Response parameter definitions:

iso_3166_language_code: uint	This field contains the current **ECI Host** language preference setting. This is a 24-bit field that identifies the language using 3 lowercase characters as specified by ISO 639-2 [55]. Both ISO 639-2/B and ISO 639-2/T can be used. Each character is coded into 8-bits according to ISO/IEC 8859-1 [53].

Error codes are listed in Table 9.4.8.2.3-1.

Table 9.4.8.2.3-1: reqHLanguage Error Codes

Name	Description
ErrLanguageNotExists	See Table 9.4.8.2.5-1.

9.4.8.2.4 reqCLanguage setting Message

H→C reqCLanguage(uint **iso_3166_language_code) →**
C→H resCLanguage setting()

- This message allows the **ECI Host** to request the actual setting of the language the **user** currently prefers and receives a **Response** of the stored language setting from the **ECI Client**.

Response parameter definitions:

iso_3166_language_code: uint	This field contains the current **ECI Host** language preference setting. This is a 24-bit field that identifies the language using 3 lowercase characters as specified by ISO 639-2 [55]. Both ISO 639-2/B and ISO 639-2/T may be used. Each character is coded into 8-bits according to ISO/IEC 8859-1 [53].

Error codes are listed in Table 9.4.8.2.4-1.

Table 9.4.8.2.4-1: reqCLanguage Error Codes

Name	Description
ErrLangageNotExists	**See** Table 9.4.8.2.5-1.

9.4.8.2.5 Error codes for the Country/Language setting API

The values of the API specific errors that can be returned by the **Response** messages for this API are listed below in Table 9.4.8.2.5-1.

Table 9.4.8.2.5-1: Error codes for Country/Language setting API

Name	Value	Description
ErrCountryNotExists	-256	**ECI Host** indicates that the **user** did not yet declare a country he is currently residing.
ErrLangageNotExists	-257	**ECI Host** indicates that the **user** did not yet declare a his preferred language for any User Interface communication.

9.5 APIs for ECI specific ECI Host resources

9.5.1 List of APIs for ECI specific ECI Host resources

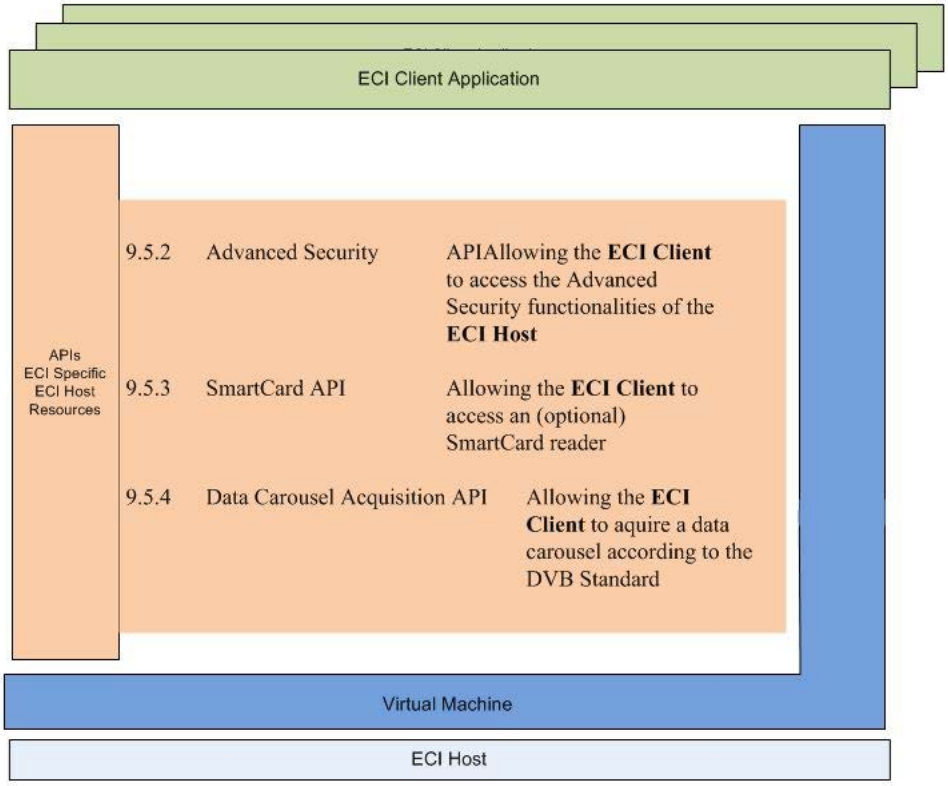


Figure 9.5.1-1: Block diagram of the APIs defined in clause 9.5

Table 9.5.1-1 lists the APIs covered in clause 9.8 and Table 9.5.1-1 illustrates the location of the APIs defined in clause 9.5 with the **ECI architecture**.

Table 9.5.1-1: List of APIs defined in clause 9.5

Clause	API name	Description
9.5.2	Advanced Security API	Allowing the **ECI Client** to access the Advanced Security functionalities of the **ECI Host**.
9.5.3	**Smart Card** API	Allowing the **ECI Client** to access an (optional) **Smart Card** reader.
9.5.4	Data Carousel Acquisition API	Allowing the **ECI Client** to acquire a data carousel according to the DVB Standard.

9.5.2 Advanced Security API

9.5.2.1 Introduction

On loading an **ECI Client** the **ECI Host** allocates an appropriate Advanced Security slot (an **ECI Client** type or a **Micro Server** type). This slot will be available during the lifecycle of that **ECI Client**. The **ECI Host** shall initialize the slot by loading the **Platform Operation Certificate Chain** containing the **Platform Operation** Public Key. This binds any meaningful further exchanges with the **AS slot** to the holder of the **Platform Operation** Secret Key.

The Advanced Security API allows an **ECI Client** to interact with the Advanced Security function in the **CPE**. There are several types of exchanges between the **ECI Client** and the AS function, which are typically initiated by the **ECI Client**. The **ECI Client** receives a signal on completion of longer AS operations.

The **AS slot** supports multiple sessions, permitting reuse of the stored information (state and configuration) in the **AS slot** for multiple media decryption and **Re-encryption Sessions**. The **AS slot** stores one intermediate key called the top level "link key" (LK_1) per session. New control words for sessions can be computed quickly on the basis of their LK_1.

The **AS slot** can also compute a secret "Authentication Key" that can be used for **ECI Client** application purposes, permitting highly secure delivery of secret information to the **ECI Client**.

The **AS slot** has a configuration which is initialized by the **ECI Client** and which defines its mode of operation. The **AS slot** permits the client to authenticate its configuration: there are two essential authentication modes:

1) **Key Ladder mode**. Authentication as part of the control word computation: the configuration of the slot was used in the computation to generate the control word that encrypted the content and the same information is required to compute the correct control word for decrypting the content, implicitly authenticating the configuration.

2) **Authentication Key mode**. Authentication is performed by an explicit validation function using verification data that can only be generated by the provisioner of the **ECI Client**. This function is practically required for an **AS slots** configured for re-encryption as this cannot be based on correct decryption as a verification means.

In addition to the above modes the **ECI Client** can require that a renewed verification is performed on each slot initialization by requiring "online authentication". Alternatively an "offline authentication" may be performed. For a successful authentication the selected authentication mode has to match with the data used to generate the authentication provided by the provisioner.

The overall AS API is split into separate APIs that permit the reflection of the capabilities of the **ECI Hosts** and **ECI Client** using it:

1) *AS General API*: this API defines generic AS functionality. All **ECI Hosts** and **ECI Clients** shall support it.

2) *AS Decryption API*: this API defines decryption specific AS functionality. All **ECI Hosts** and **ECI Clients** capable of decryption shall support it.

3) *AS Export API*: this API defines export specific AS functionality. All **ECI Hosts** and **ECI Clients** capable of supporting decryption and export shall support it. **ECI Hosts** supporting export shall also support encryption.

4) *AS Encryption API*: this API defines encryption specific AS functionality. All **ECI Hosts** and **ECI Clients** capable of encryption shall support it.

The following constraint shall apply:

- An **ECI Client** shall support either a decryption or encryption, and not require support for both at the same time.

The **ECI Host** and **ECI Client** shall use the **ECI Host** interface discovery resource to provide each other with information regarding each other's capabilities. The **ECI Host** shall allocate the appropriate slot in accordance with the discovery result: an encryption slot for **ECI Clients** requiring encryption and a decryption AS slot for **ECI Clients** requiring decryption.

NOTE: Functions that provide complementary functionality may exist in different APIs: the AS General API and a more specific AS API.

Messages in the AS General API only require support by the **ECI Host** is as far as this is necessary to reflect the **ECI Host** capabilities (decryption, export and encryption support).

The messages of the AS APIs are defined in terms of the AS functions defined in ETSI GS ECI 001-5-1 [4] clauses 6.2.4 and 7.9. In ETSI GS ECI 001-5-1 [4] clause 6.2.4.1 provides the AS function overview. The first parameter, the slotId parameter, is omitted from the definitions in ETSI GS ECI 001-5-1 [4]: this is supplied by the **ECI Host**.

Many of the type definitions and value definitions for parameters as used in this API definition are defined in ETSI GS ECI 001-5-1 [4]. The error codes for this API are defined in ETSI GS ECI 001-5-1 [4], and are not specifically listed here on a message by message basis. The error codes for parameter values correspond to the parameter sequence counting as defined by the referred functions in ETSI GS ECI 001-5-1 [4], which typically have one additional parameter (slotId).

9.5.2.2 Advanced Security General API Message Definitions

9.5.2.2.1 General

The Advanced Security General API provides the messages as listed in Table 9.5.2.2.1-1.

Table 9.5.2.2.1-1: Advanced Security General Messages

Message	Type	Dir.	Tag	Description
reqAsInitSlot	A	C→H	0x0	Initializes the **AS slot**.
callAsNextKeySession	S	C→H	0x1	Change to next random key for a session.
reqAsStopSession	A	C→H	0x2	Stop a session.
reqAsLoadSlotLk	A	C→H	0x3	Compute top level link key (LK1).
reqAsComputeAkClient	A	C→H	0x4	Compute Authentication Key for **ECI Client** applications.
reqAsClientChalResp	A	C→H	0x5	Apply **ECI Client** Authentication Key on data and return result.
getAsSlotRk	S	C→H	0x6	Get random key value for the **AS slot**.
getAsSessionRk	S	C→H	0x7	Get random key value for a session.
getAsSessionLimitCounter	S	C→H	0x8	Get current limit counter value for the session.
setAsSessionLimitEvent	S	C→H	0x9	Set limit value for sending a reqAsEventSessionLimit message to the **ECI Client**.
reqAsEventSessionLimit	A	H→C	0xA	On reaching a limit value for remaining units send event to **ECI Client**.
getAsClientRnd	S	C→H	0xB	Get a new random number for **ECI Client** applications.
getAsSC	S	C→H	0xC	Get current Scrambling Control field status of content in a session.
reqAsEventSC	A	H→C	0xD	Event message on change of scrambling control field in session.

9.5.2.2.2 reqAsInitSlot Message

C→H reqAsInitSlot(uint **slotVersion,** uint **slotMode** →
H→C resAsInitSlot()

- This message initializes the slot with various general parameters.

Request Parameter definitions:

slotVersion: uint	Version of the slot functionality as defined in ETSI GS ECI 001-5-1 [4].
slotMode: uint	Main Mode for the slot to operate in; see ETSI GS ECI 001-5-1 [4].

Semantical description:

- This message is equivalent to the AS function reqAsInitSlot as defined in ETSI GS ECI 001-5-1 [4]; with the **ECI Host** providing the value of the slotId and POPKchain parameters.

9.5.2.2.3 callAsNextKeySession Message

C→H callAsNextKeySession(uint **sessionId**)

- This message causes a change to the next random key for a session.

Request Parameter definitions:

sessionId: uint	Session for which to change top the next random key.

Semantical description:

- This message is equivalent to the AS message callAsNextKeySession as defined in ETSI GS ECI 001-5-1 [4]; with the **ECI Host** providing the value of the slotId parameter.

9.5.2.2.4 reqAsStopSession Message

C→H reqAsStopSession(uint **sessionId**) →
H→C resAsStopSession()

- This message stops an **AS slot** session.

Request Parameter definitions:

sessionId: uint	Id of Session to stop.

Semantical description:

- This message is equivalent to the AS function reqAsStopSession as defined in ETSI GS ECI 001-5-1 [4]; with the **ECI Host** providing the value of the slotId parameter.

9.5.2.2.5 reqAsLoadSlotLk Message

C→H reqAsLoadSlotLk(uint **sessId**, InputV **inputV**, ulong **spkUri,** uchar **spkIndx**) →
H→C resAsLoadSlotLk()

- This message computes the top level link key LK_1 that can be used subsequently to compute control words.

Request Parameter definitions:

sessId: uint	Id of session to be initialized.
inputV: InputV	Message containing Chip Set public key encrypted and Sender Secret Key signature protected link key LK_1.
spkUri: ulong	Usage rules for the SPK vector that is used subsequently to compute a control word, see ETSI GS ECI 001-5-1 [4].
spkIndx: uchar	Index defining the location of the AS slot's SPK in the SPK vector that is used subsequently to compute a control word, see ETSI GS ECI 001-5-1 [4] clause 5.

Semantical description:

- This message is equivalent to the AS function reqAsLoadSlotLk as defined in ETSI GS ECI 001-5-1 [4]; with the **ECI Host** providing the value of the slotId parameter.

- The **ECI Host** shall also issue a reqAsDeoupleDecryptSession function [4] if an AS decryption session is stopped that was previously coupled to another AS decryption session (see clause 9.5.2.3.1).

9.5.2.2.6 reqAsComputeAkClient Message

C→H reqAsComputeAkClient(InputV **inputV,** uint **nSpk** uchar **spkIndx,** PubKey **spk[16],** PubKey **popk[16],** SessionConfig **akCnf[16],** ulong **spkUri;** uchar **XT[32],** bool **online)** →
H→C resAsComputeAkClient ()

- This message computes an authentication key for use of the **ECI Client**.

Request Parameter definitions:

inputV: InputV	Message containing Chip Set public key encrypted and Sender Secret Key signature protected r value used to compute AK.
nSpk: uint	Number of values in SPK vector, see ETSI GS ECI 001-5-1 [4].
spkIndx: uchar	Index defining the location of the AS slot's SPK in the SPK vector, the AS slot's POPK value in the POPK vector and the AS slot's slotConfig in the clCnf vector that is used to compute the Client Authentication Key, see ETSI GS ECI 001-5-1 [4].
spk[16]: PubKey	**Sender Public Key** vector used to compute the Client Authentication Key; see ETSI GS ECI 001-5-1 [4].
popk[16]: PubKey	Platform Operator Public Key vector used to compute the Client Authentication Key; see ETSI GS ECI 001-5-1 [4].
akCnf[16]: SessionConfig	Client session configuration vector used to compute the Client Authentication Key; see ETSI GS ECI 001-5-1 [4].
spkUri: ulong	Usage rules for the SPK vector that is used subsequently to compute a control word, see ETSI GS ECI 001-5-1 [4].
XT[32]: uchar	Value of extension field used to compute the Client Authentication Key; see ETSI GS ECI 001-5-1 [4]. Default value is { 0x00 }.
online: bool	If true the slot random key is used for the Authentication Key computation forcing a fresh Authentication Key computation by the provisioner.

Semantical description:

- This message is equivalent to the AS function reqAsComputeAkClient as defined in ETSI GS ECI 001-5-1 [4]; with the **ECI Host** providing the value of the slotId parameter.

9.5.2.2.7 reqAsClientChalResp Message

C→H reqAsClientChalResp(uchar **challenge**[16]);→
H→C reqAsClientChalResp(uchar **response**[16])

- This message uses the Client Authentication Key, as computed by the reqAsComputeAkClient message (defined in ETSI GS ECI 001-5-1 [4]), to decrypt a 128-bit challenge parameter input to produce a 128-bit response parameter output.

Request Parameter definitions:

challenge[16]: uchar	128-bit input to be decrypted by the Client Authentication Key.

Response Parameter definitions:

response[16]: uchar	128-bit decrypted output.

Semantical description:

- This message is equivalent to the AS function reqAsClientChalResp as defined in ETSI GS ECI 001-5-1 [4]; with the **ECI Host** providing the value of the slotId parameter and the **Response** message carrying the "response" parameter result.

9.5.2.2.8 getAsSlotRk Message

C→H SymKey **getAsSlotRk**()

- This message reads the random key for the **ECI Client**'s **AS slot** session.

Semantical description:

- This message is equivalent to the **AS** function getAsSlotRk defined in ETSI GS ECI 001-5-1 [4]; with the **ECI Host** providing the value of the slotId parameter.

9.5.2.2.9 getAsSessionRk Message

C→H SymKey **getAsSessionRk**(uint **sessionId,** uint **rkIndx)**

- This message reads the current (rkIndx==0) or next (rkIndx==1) random key for the **ECI Client**'s session with the identifier sessionId.

Request Parameter definitions:

sessionId: uint	Id of the session for which to retrieve the random session key.
rkIndx: uint	Identified whether the current (rkIndx==0) or the next (rkIndx==1) random session key is to be retrieved.

Semantical description:

- This message is equivalent to the AS message getAsSessionRk defined in ETSI GS ECI 001-5-1 [4]; with the **ECI Host** providing the value of the slotId parameter.

9.5.2.2.10 getAsSessionLimitCounter Message

C→H ulong **getAsSessionLimitCounter**(uint **sessionId**)

- This message returns the limit counter value of the **ECI Client**'s sessionId.

Semantical description:

- This function is equivalent to the AS function getAsSessionLimitCounter defined in ETSI GS ECI 001-5-1 [4]; with the **ECI Host** providing the value of the slotId parameter.

Request Parameter definitions:

sessionId: uint	Id of the session for which to retrieve session limit counter.

9.5.2.2.11 setAsSessionLimitEvent Message

C→H ulong **setAsSessionLimitEvent** (uint **sessionId,** ulong **eventLimit**)

- This message sets the limit value eventLimit for the limitCounter of the **ECI Client**'s session with the identifier sessionId for a reqAsEventSessionLimit message to be returned to the **ECI Client**.

Request Parameter definitions:

sessionId: uint	Id of the session for which to set the session eventLimit.
eventLimit: ulong	Value of the event limit to be set.

Semantical description:

- This function is equivalent to the AS function setAsSessionLimitEvent defined in ETSI GS ECI 001-5-1 [4]; with the **ECI Host** providing the value of the slotId parameter.

9.5.2.2.12 reqAsEventSessionLimit Message

H→C reqAsEventSessionLimit (uint **sessionId**)
C→H res**AsEventSessionLimit ()**

- This message returns the limit counter value of the **ECI Client**'s sessionId.

Response Parameter definitions:

sessionId: uint	Id of the session that generated an eventLimit event.

Semantical description:

- This function is equivalent to the AS function reqAsEventSessionLimit defined in ETSI GS ECI 001-5-1 [4]; with the **ECI Host** providing removing the slotId parameter.

9.5.2.2.13 getAsClientRnd Message

C→H SymKey **getAsClientRnd**()

- This message returns a 128-bit random number.

Semantical description:

- This function is equivalent to the AS message getAsClientRnd defined in ETSI GS ECI 001-5-1 [4].

9.5.2.2.14 getAsSC Message

C→H uint **getAsSC**(uint **sessionId**)

- This message returns the current Scrambling Control field status of content in a session.

Request Parameter definitions:

sessionId: uint	Id of the session for which to retrieve the current scrambling control field.

Semantical description:

- This function is equivalent to the AS function getAsSC as defined in ETSI GS ECI 001-5-1 [4]; with the **ECI Host** providing the value of the slotId parameter.

9.5.2.2.15 reqAsEventSC Message

H→C reqAsEventSC(uint **sessionId**; uint **scramblingControlField**)
C→H res**AsEventSC()**

- This message indicates a change in the scrambling control field in the session with the identifier sessionId.

Response Parameter definitions:

sessionId: uint	Id of the session on which a scrambling status field change occurred.
scramblingControlField: uint	New value for the scrambling status field. See ETSI GS ECI 001-5-1 [4] clause 7.9 for the definition of the values and their semantics.

Semantical description:

- This message is equivalent to the AS function reqAsEventSC defined in ETSI GS ECI 001-5-1 [4]; with the **ECI Host** removing the value of the slotId parameter.

9.5.2.3 Advanced Security Decryption API Message Definitions

9.5.2.3.1 General

The Advanced Security Decryption API provides the messages as listed in Table 9.5.2.3.1-1.

Two decryption sessions can be coupled permitting different control words to be used to decrypt two streams of content that are to be treated as a single content item after decryption.

EXAMPLE: A sports channel may be broadcast with multiple sound channels, the sound channel for a specific language only being made available if a specific subscription is available for decrypting it. Only one session can be coupled to another session.

Table 9.5.2.3.1-1: Advanced Security Decryption Messages

Message	Type	Dir.	Tag	Description
reqAsAStartDecryptSession	A	H→C	0x0	Start a decryption session in the **ECI Client**'s **AS slot**.
reqAsComputeDecrCw	A	H→C	0x1	Compute a decryption control word.
reqAsAuthDecrSlotConfig	A	H→C	0x2	Authenticate the slot configuration with authentication mechanisms (decryption mode).

9.5.2.3.2 reqAsStartDecryptSession Message

C→H reqAsAStartDecryptSession(ushort **mh**, PubKey **spk**, SessionConfig **config,** ScrambleMode **sm**) →
H→C resAsASartDecryptSession(uint **sessionId)**

- This message starts a decryption session in the **ECI Client**'s **AS slot**.

Request Parameter definitions:

mh: ushort	**Media Handle** for which content is decrypted (to be used by **ECI Host** to associate the content to be decrypted to the decryption resource allocated to this session).
spk: PubKey	Sender Public Key for this session.
config: SessionConfig	Session configuration.
sm: ScrambleMode	Mode of descrambling to use. For definition see Table 9.5.2.3.2-1. See note.
NOTE: The information in the sm parameter should not contradict the cwUri parameter of a subsequent reqAsComputeDecrCw message.	

Table 9.5.2.3.2-1: ScrambleMode definition

```
typedef ScrambleMode {
  ucharmodeRef;
  ucharmode[16] ;
} ScrambleMode;
```

The definition of **modeRef** is given in Table 9.9.2.11-1.

Table 9.5.2.3.2-2: modeRef definition

Name	Value	Description
ScrambleModeHost	0x01	Host shall select (de)scramble mode based on standardized or proprietary information.
ScrambleModeDvb	0x02	DVB definition for scrambling mode is used. Byte 0 of the mode field contains a value with the same meaning as defined in the scrambling_mode field of the Scrambling_descriptor as defined in [56]. Byte 1 has the following meaning for byte 0 valing value 0x02, 0x03 and 0x10 (i.e. DVB CSA1/2, DVB CSA3 mode for descrambling and DVB-CISSA version 1 mode): Value==0x01: TS-mode (de)scrambling. Value==0x02: PES-mode (de)scrambling. All other values are reserved; all unused bytes of the mode field are reserved. See note 1.
ScrambleModeCencEnum	0x03	The scrambling mode is defined in [57] or Byte 0 of the mode field is defined as: Value==0x01: CENC CTR mode. Value==0x02: CENC CBC mode. Other values for byte 0 are reserved. For the above defined values of byte 0 byte 1 indicated the subscheme: Value==0x01: host defined, for encryption selected from one of the values defined below. Value==0x02: full segment encryption as defined in [58]. Value==0x03: subsample encryption as defined in [50]. Other values for byte 1 are reserved. For other values of byte 0 byte 1 is reserved. Bytes 2-15 are reserved. See note 2.
RFU	Other	Reserved for future use.
NOTE 1: The **ECI Host** shall at least support DVB CSA1/2, DVB CSA3 mode for descrambling and DVB-CISSA version 1 mode for scrambling and descrambling. NOTE 2: The **ECI Client** or (if so permitted) the **ECI Host** can select a scrambling mode for encryption that suitably fits the application; specifically taking into account streaming type applications typically use CBC full segment encryption and storage applications typically use CTR mode, and may benefit from subsample encryption. NOTE 3: All possible scrambling modes used for encryption ensure full security of the encrypted result. See ETSI GS ECI 001-5-1 [4].		

Response Parameter definitions:

sessionId: uint	Id of the session that was created.

Semantical description:

- This message is equivalent to [4] AS function reqAsASartDecryptSession; with the **ECI Host** providing the value of the slotId parameter, and the sessionId result returned in the **Response** message.

The **ECI Host** shall also issue a reqAsCoupleDecryptSession function [4] when a second as decryption session is started for the same media handle so as to couple these as decryption sessions, coupling the second session to the first session.

9.5.2.3.3 reqAsComputeDecrCw Message

C→H reqAsComputeDecrCw(int **sessionId**, ulong **cwUri,** uint **nSpk**, uint **nElk**, SymKey **elk**[24], PubKey **spk**[16], PubKey **popk**[16], SessionConfig **config**[16], uchar **XT**[32], uint **rkIndx,** Field2 **field2,** uint **cwIndx**) → **H→C resAsComputeDecrCw ()**

- This message computes a decryption control word.

Request Parameter definitions:

sessionId: int	Id of the session for which to compute a control word.
cwUri: ulong	cwUri defines the applications of the control word. cwUri values are defined in ETSI GS ECI 001-5-1 [4], clause 5.5.
nSpk: uint	Number of SPK values in the SPK vector.
nElk: uint	Number of Elk values in the ELK vector.
elk[24]: SymKey	Vector of symmetrically encrypted key values to be successively decrypted by the key ladder mechanism. Value elk[nElk-2] is the field1 input to the content property authentication as defined in ETSI GS ECI 001-5-1 [4], clause 6.3.
spk[16]: PubKey	Vector of sender public keys as defined in ETSI GS ECI 001-5-1 [4], clause 5.5.
popk[16]: PubKey	Vector of platform operator public keys as defined in ETSI GS ECI 001-5-1 [4], clause 5.5.
config[16]: SessionConfig	Vector of client session configurations as defined in ETSI GS ECI 001-5-1 [4], clause 5.5.
XT[32]: uchar	Spare input to control word mechanism as defined in ETSI GS ECI 001-5-1 [4], clause 5.5.
rkIndx: uint	Identified whether the current (rkIndx==0) or the next (rkIndx==1) random session key is to be used in the control word calculation.
field2: Field2	Larger content property content not authenticated in field1 as defined in ETSI GS ECI 001-5-1 [4], clause 6.3.
cwIndx: uint	Index of control word to be computed: 0 for even and 1 for odd control word; no meaning for file based decryption.

Semantical description:

- This message is equivalent to the AS function reqAsComputeDecrCw as defined in ETSI GS ECI 001-5-1 [4]; with the **ECI Host** providing the value of the slotId parameter.

9.5.2.3.4 reqAsAuthDecrSlotConfig Message

C→H reqAsAuthDecrSlotConfig(uint **sessionId**, InputV **inputV**; uchar **nSpk,** uint **spkIndx**, PubKey **spk**[16], PubKey **popk**[16], SessionConfig **cnf**[16], ulong **spkUri**, uchar **XT**[32], bool **online,** uchar **verifier[16]**) → **H→C resAsAuthDecrSlotConfig ()**

- This message authenticates the slot configuration with authentication mechanisms (decryption mode).

Request Parameter definitions:

sessionId: uint	Id of the session for which to authenticate the slot configuration.
inputV: InputV	Message containing Chip Set public key encrypted and Sender Secret Key signature protected r value used to compute AK used to authenticate the **AS slot** configuration.
nSpk: uchar	Number of SPK values in the SPK vector.
spkIndx: uint	Index defining the location of the AS slot's SPK in the SPK vector, the AS slot's POPK value in the POPK vector and the AS slot's slotConfig in the clCnf vector that is used to compute the Client Authentication Key, see ETSI GS ECI 001-5-1 [4].
spk[16]: PubKey	Vector of sender public keys as defined in ETSI GS ECI 001-5-1 [4], clause 5.5
popk[16]: PubKey	Vector of platform operator public keys as defined in ETSI GS ECI 001-5-1 [4], clause 5.5.
cnf[16]: SessionConfig	Vector of client configurations as defined in ETSI GS ECI 001-5-1 [4], clause 5.5.
spkUri: ulong	Usage rules for the SPK vector that is used subsequently to compute the authentication key AK, see ETSI GS ECI 001-5-1 [4].
XT[32]: uchar	Value of extension field used to compute the Client Authentication Key; see ETSI GS ECI 001-5-1 [4]. Default value is { 0x00 }.
online: bool	If true the slot random key is used for the Authentication Key computation forcing a fresh Authentication Key computation by the provisioner.
verifier[16]: uchar	Value with which reqAsAuthDecrSlotConfig authenticates the slot configuration.

Semantical description:

- This message is equivalent to the AS function reqAsAuthDecrSlotConfig as defined in ETSI GS ECI 001-5-1 [4]; with the **ECI Host** providing the value of the slotId parameter.

9.5.2.4 Advanced Security Export API

9.5.2.4.1 General

The general Advanced Security API provides the messages as listed in Table 9.5.2.4.1-1.

Table 9.5.2.4.1-1: Advanced Security Export messages

Message	Type	Dir.	Tag	Description
reqAsExportConnSetup	A	C→H	0x0	Setup an export connection from decryption to encryption session.
reqAsExportConnEnd	A	C→H	0x1	Terminate existing export session.

9.5.2.4.2 reqAsExportConnSetup Message

C→H reqAsExportConnSetup(uint **sessId,** ushort **expMh,** uint **grpIndx**; CertSerialChain **expCh,** CertSerialChain **impCh,** CertSerialChain **auth**[]) →
H→C resAsExportConnSetup()

- This message sets up an advanced security connection from the decryption session to the export media handle session.

Request Parameter definitions:

sessId: uint	Id of the export session of the **ECI Client**'s **AS slot**.
expMh: ushort	Id of the export media handle to be used for encryption of the decrypted content in the AS sessions.
grpIndx: uint	Index to store of the export session connection; permitted values are 0 or 1. This parameter can be used to alternate the export connection authentication to a Micro Server (e.g. for anticipating a forthcoming changeover of export group ID in a stream).
expCh: CertSerialChain	Export chain for **ECI Client**.
impCh: CertSerialChain	Import chain for encrypting/importing **ECI Client**.
auth[]: CertSerialChain	Authorization **Certificates** for Import chain.

Semantical description:

- This message is equivalent to thw **AS** function reqAsExportConnSetup as defined in ETSI GS ECI 001-5-1 [4]; with the **ECI Host** providing the value of the slotId, impSlotId and ImpSessId parameter. The **ECI Host** shall use the media handle of the export session to connect the AS decryption session to the corresponding AS encryption session, i.e. provide the impSlotId and impSessId parameters in the reqAsExportConnSetup AS function of ETSI GS ECI 001-5-1 [4].

9.5.2.4.3 reqAsExportConnEnd Message

C→H reqAsExportConnEnd(ushort **expMh)** →
H→C resAsExportConnEnd()

- This message terminate an existing export session.

Request Parameter definitions:

expMh: ushort	Export media handle session of the AS sessions for which the content exchange shall be terminated.

Semantical description:

- This message is equivalent to the AS function reqAsExportConnEnd as defined in ETSI GS ECI 001-5-1 [4]; with the **ECI Host** providing the value of the slotId and sessionId parameters associated with expMh.

9.5.2.5 Advanced Security Encryption API

9.5.2.5.1 General

The Advanced Security Encryption API provides the messages as listed in Table 9.5.2.5.1-1.

Table 9.5.2.5.1-1: Advanced Security Encryption messages

Message	Type	Dir.	Tag	Description
reqAsStartEncryptSession	A	C→H	0x0	Start an encryption session.
reqAsComputeEncrCw	A	C→H	0x1	Compute encryption control word.
reqAsAuthEncrSlotConfig	A	C→H	0x2	Authenticate the slot configuration and encryption parameters with authentication mechanisms (encryption mode).
reqAsLdUssk	A	C→H	0x3	Load micro server secret key.
reqAsMInikLk1	A	C→H	0x4	Compute asymmetrical **Micro Client** initialization message.
reqAsEventCpChange	A	H→C	0x5	Event message on content property change in imported content in an encryption session.
setAsPermitCPChange	S	C→H	0x6	Enable/disable imported content property CP changes taking effect on control word selection for encryption in an encryption session.
setAsSC	S	C→H	0x7	Set scrambling control field of encrypted content of an encryption session.

9.5.2.5.2 Target Client Chain Definition

Micro Servers can use the **Certificate Processing System** to provide a robust implementation of asymmetrical client authentication. **ECI** defines certificate chains to permit such **Micro Client** authentication. Such target chains are used as input to the reqAsMInikLk1 message.

The **Certificate Chains** shall be in accordance with clause 5.4.1. There are two types of **Certificates** involved:

- **A Micro Client Certificate** authenticates a single **Micro Client**; the Public Key of the **Certificate** shall be identical to the Chipset Public Key of the **Micro Client CPE** in case the **Micro Client** is an **ECI Client**.
- A Target Group **Certificate** authenticates one or more Target Groups or **Micro Client Certificates**.

Micro DRM system operators can use the **ECI Revocation List** mechanism to securely manage the evolution of authenticated Micro Clients for a server.

NOTE: The maintenance of **Revocation Lists** is a Micro DRM operator private issue.

The **Certificate** ID for the Target Group **Certificate** is defined in Table 9.5.2.5.2-1.

Table 9.5.2.5.2-1: Target Group ID definition

Syntax	No. of bits	Mnemonic
`ECI_Target_Group_Id {`		
`  padding(4)`		
`  type`	4	uimsbf
`  target_group_id`	20	uimsbf
`  target_group_version`	8	uimsbf
`}`		

Semantics:

type: integer	Value in accordance with Table 5.1.3-1.
target_group_id: integer	**Target Group** number, unique in the context of the **father**.
target_group_version: integer	Incremented in case the micro group changes its **Certificate**.

The **Certificate** ID for the **Micro Client Certificate** is defined in Table 9.5.2.5.2-2.

Table 9.5.2.5.2-2: Micro Client ID definition

Syntax	No. of bits	Mnemonic
ECI_Micro_Client_Id {		
padding(4)		
type	4	uimsbf
micro_client_id	20	uimsbf
micro_client_version	8	uimsbf
}		

Semantics:

type: integer	Value in accordance with Table 5.1.3-1.
micro_client_id: integer	**Micro Client** number, unique in the context of the **father**.
micro_client_version: integer	Incremented in case the micro group changes its **Certificate**.

9.5.2.5.3 reqAsStartEncryptSession Message

C→H reqAsStartEncryptSession(ushort **mh**, PubKey **spk**, SessionConfig **config**,
uint **nEncr**, PubKey **encrSpk[MaxSpkEncr]**, PubKey **encrPopk[MaxSpkEncr]**, ulong **encrCwUri**)→
H→C resAsStartEncryptSession()

- This message starts the encryption session.

Request Parameter definitions:

mh: ushort	Identifier of the media handle of the encrypted content for which to create an encryption session.
spk: PubKey	Public Key of Sender used to authenticate sender and LK1 encrypted message by the AS system.
config: SessionConfig	Configuration for the session.
nEncr: uint	Number of additional SPK (and POPK) values that are defined for encryption and possible subsequent decryption). The maximum value is MaxEncr (see ETSI GS ECI 001-5-1 [4]).
encrSpk: PubKey[]	Vector with additional SPK values for encryption.
encrPopk: PubKey[]	Vector with additional POPK values for encryption.
encrCwUri: ulong	CWURI value to use for encryption; see ETSI GS ECI 001-5-1 [4], clause 5.5.

Semantical description:

- This message is equivalent to the AS function reqAsStartEncryptSession as defined in ETSI GS ECI 001-5-1 [4]; with the **ECI Host** providing the value of the slotId parameter. The **ECI Host** shall derive the ,importSlotId and importSessionId parameters from the mh value.

NOTE: The **Response** message returns the new session ID created if no error occurred.

9.5.2.5.4 reqAsComputeEncrCw Message

C→H reqAsComputeEncrCw(int **sessId**, ulong **cwUri**, uint **nElk**, SymKey **elk**[24], uchar **XT**[32],
uint **rkIndx.** Field2 **field2.** uint **cwIndx)**→
H→C resAsComputeEncrCw()

- This message computes encryption control word.

Request Parameter definitions:

sessId: int	Id of the session for which to compute a control word.
cwUri: ulong	cwUri defines the applications of the control word. cwUri values are defined in ETSI GS ECI 001-5-1 [4], clause 5.5.
nElk: uint	Number of Elk values in the ELK vector.
elk[24]: SymKey	Vector of symmetrically encrypted key values to be successively decrypted by the key ladder mechanism. Value elk[nElk-2] is the field1 input to the content property authentication as defined in ETSI GS ECI 001-5-1 [4] clause 6.3.
XT[32]: uchar	Spare input to control word mechanism as defined in ETSI GS ECI 001-5-1 [4], clause 5.5.
rkIndx: uint	Identified whether the current (rkIndx==0) or the next (rkIndx==1) random session key is to be used in the control word calculation.
field2: Field2	Larger content property content not authenticated in field1 as defined in ETSI GS ECI 001-5-1 [4], clause 6.3.
cwIndx: uint	Index of control word to be computed: 0 for even and 1 for odd control word; no meaning for file based encryption.

Semantical description:

- This message is equivalent to the AS function reqAsComputeEncrCw as defined in ETSI GS ECI 001-5-1 [4]; with the **ECI Host** providing the value of the slotId parameter.

9.5.2.5.5 reqAsAuthEncrSlotConfig Message

C→H reqAsAuthEncrSlotConfig(uint **sessId**, InputV **inputV**, uchar **XT**[32], bool **online,** uchar **verifier[16]**) → **H→C resAsAuthEncrSlotConfig()**

- This message authenticates the slot configuration with authentication mechanisms (encryption mode).

Request Parameter definitions:

sessId: uint	Session ID for which the configuration shall be authenticated.
inputV: InputV	Message containing Chip Set public key encrypted and Sender Secret Key signature protected r value used to compute AK used to authenticate the **AS slot** configuration.
XT[32]: uchar	Spare input to control word mechanism as defined in ETSI GS ECI 001-5-1 [4] clause 5.5.
online: bool	If true the slot random key is used for the Authentication Key computation forcing a fresh Authentication Key computation by the provisioner.
verifier[16]: uchar	reqAsAuthDecrSlotConfig uses this value to authenticate the slot configuration.

Semantical description:

- This message is equivalent to the AS function reqAsAuthEncrConfig as defined in ETSI GS ECI 001-5-1 [4]; with the **ECI Host** providing the value of the slotId parameter.

9.5.2.5.6 eqAsLdUssk Message

C→H reqAsLdUssk(uint **sessId**, InputV **inputV,** uchar **XT**[32], bool **online,** uchar **mUssk**[NUSSK])→ **H→C resAsLdUssk()**

- This message loads the **Micro Server** secret key in case of asymmetrical authentication of the **ECI Clients** that will be able to decode the content.

Request Parameter definitions:

sessId: uint	Session ID for which the **Micro Server** secret key will be loaded.
inputV: InputV	Message containing Chip Set public key encrypted and Sender Secret Key signature protected r value used to compute AK used decrypt the **Micro Server** secret key to be loaded.
XT[32]: uchar	Spare input to control word mechanism as defined in ETSI GS ECI 001-5-1 [4], clause 5.5.
online: bool	If true the slot random key is used for the Authentication Key computation forcing a fresh Authentication Key computation by the provisioner.
mUssk[NUSSK]: uchar	Encrypted **Micro Server** secret key.

Semantical description:

- This function is equivalent to the AS function reqAsLdUssk as defined in ETSI GS ECI 001-5-1 [4]; with the **ECI Host** providing the value of the slotId parameter.

9.5.2.5.7 reqAsMInikLk1 Message

C→H reqAsMInikLk1(uint **sessId**, ECI_Certificate_Chain **ClCPK**) →
H→C resAsMInikLk1(InputV **inputV)**

- This message computes asymmetrical **Micro Client** initialization message.

Request Parameter definitions:

sessId: uint	Session ID for which the **Micro Server** secret key will be loaded.
ClCPK: ECI_Certificate_Chain	Target **Certificate Chain** as defined in clause 9.5.2.5.2 for loading the Micro Client Chipset Public key to be used to encrypt the secret session key between **Micro Server** and **Micro Client**.

Response Parameter definitions:

inputV: InputV	MicroDRM session key encrypted with the Micro Client Chipset Public key and signed by the Micro Server secret key. Can be used by the micro-client as a message to load the common session LK_1.

Semantical description:

- This function is equivalent to ETSI GS ECI 001-5-1 [4] AS function reqAsMInikLk1; with the **ECI Host** providing the value of the slotId parameter.

9.5.2.5.8 reqAsEventCpChange Message

H→C reqAsEventCpChange(int **sessionId**)

- This message requests an content property change of imported content in an encryption session

Request Parameter definitions:

sessionId: int	Encryption session on which a content property change event occurred on the imported content.

Semantical description:

- This message is equivalent to ETSI GS ECI 001-5-1 [4] AS function reqAsEventCpChange; with the **ECI Host** removing the slotId parameter.

9.5.2.5.9 setAsPermitCPChange Message

C→H setAsPermitCPChange(int **sessionId**; bool **permit**)

- This message initiates a content property change of imported content in an encryption session.

Request Parameter definitions:

sessionId: int	Encryption session for which to permit an automatic changeover of the control word on a content property change that will occur or which is pending.
permit: bool	Value true means permission is granted, False means no permission.

Semantical description:

- This function is equivalent to [4] AS function setAsPermitCPChange; with the **ECI Host** providing the value of the slotId parameter.

9.5.2.5.10 setAsSC Message

C→H setAsSC(int **sessionId**, uint **scramblingControlField**)

- This message sets the next value of scrambling control field in encryption session

Request Parameter definitions:

sessionId: int	Encryption session for which the scrambling control field is to be set that is to be used on the first possible point of change in the stream.
scramblingControlField: uint	Value of the scrambling control field; see ETSI GS ECI 001-5-1 [4], clause 7.9 for the permitted values and their meaning.

Semantical description:

- This function is equivalent to [4] AS function setAsSC; with the **ECI Host** providing the value of the slotId parameter.

9.5.2.5.11 Error Codes for the Advanced Security (AS) API

All error codes for the AS APIs are defined in ETSI GS ECI 001-5-1 [4], clause 6.2.4.15.

9.5.3 Smart Card API

9.5.3.1 Introduction

ECI permits **ECI Clients** to interface with a single detachable local security module (**Smart Card**). **ECI Clients** can create a secure channel from **ECI Client** to **Smart Card** or (security wise) directly from **Smart Card** to Advanced Security block so as to provide maximum robustness to the protection of control words. The details of the actual protocols to make exchanges for key management are not defined by **ECI** but are fully defined by the CA/DRM system on basis of the **Advanced Security** block API as defined in ETSI GS ECI 001-5-1 [4].

ECI compliant **CPEs** may have one or multiple card reader slots. The **ECI Host** manages card readers fully transparently for **ECI Clients**. The **ECI Host** matches any inserted **Smart Card** to the available **ECI Clients**. For this purpose **ECI Clients** publish a list of card specifiers to the **ECI Host**. The **ECI Host** manages any potential conflict between **ECI Clients** wishing to access the same **Smart Card**. The **ECI Host** further provides contention management for card readers.

9.5.3.2 Base specifications

This clause provides the basic standards and specifications that the **CPE** card reader hardware and associated drivers and **ECI Host** software shall comply with.

A **CPE**'s card reader physical characteristics can be based on relevant market requirements. The dominant format for conditional access cards is ID-1 (credit card size), but cards with ID-000 format (SIM) are also used. See ISO/IEC 7816-1 [31], ISO/IEC 7816-2 [32] and ISO/IEC 14496-12 [39] for reference.

A regular **CPE**'s card reader shall comply with ISO/IEC 7816-3 [33] clause 5 supporting at least class A (5V) and B (3V) operation. The following pins shall be supported: C1 (VCC), C2 (RST), C3 (CLK), C5 (GND) and C7 (I/O).

ECI Hosts may support card readers not compliant with the above. Such card readers shall be clearly marked as such and cannot be mistaken for regular **ECI** card readers by the **user**.

The **ECI Host** and **CPE** card reader hardware shall support the **ECI** relevant features defined in clauses 6 to 12 of ISO/IEC 7816-2 [32]. The **ECI Host** shall initialize any card inserted using the procedures as defined in ISO/IEC 7816-2 [32].

The **ECI Host** shall implement functionality of ISO/IEC 7816-3 [33] as required for implementing the specifications in the present document. The **ECI Host** shall support ISO/IEC 7816-5 [35] as far as required to support the AID retrieval functionality defined in the following clause 9.5.3.3.

9.5.3.3 Smart Card access management

Before initializing a connection to an **ECI Client** the **ECI Host** shall initialize the protocol and card reader in accordance with ISO/IEC 7816-3 [33], clauses 6 to 11. It shall select appropriate settings for the protocol, communication timing parameters and operating class of the **Smart Card**.

The **ECI Host** shall be able to retrieve the AID (Application IDentifier as defined in ETSI TS 103 205 [34], clause 8.2.1.2) as defined in ETSI TS 103 205 [34] clause 8.2.1 as retrieved from the card as defined in ISO/IEC 7816-3 [33] clause 8.2.2.1 from historical bytes, or initial data string. For multi-application **Smart Cards** the **ECI Host** shall be able to retrieve the list of AIDs as defined in ETSI TS 103 205 [34] clause 8.2, and specifically clause 8.2.1.1, clause 8.2.2 and there within specifically clause 8.2.2.3.

The **ECI Host** shall use the following list of Card Identifiers for a card:

1) If the card is a multi-application card in accordance with ISO/IEC 7816-3 [33] it shall use as the list of Card Identifiers the list of AIDs as retrieved from the EF.DIR's application templates and AIDs directly represented in the EF.DIR.

2) If the card is not a multi-application card in accordance with 1) above, the AID retrieved from the 'historical bytes' as defined in ETSI TS 103 205 [34], clause 8.1.1 or clause 8.1.2 shall be used as the single Card Identifier.

3) If no AID can be retrieved as defined in 1) or 2) above, the ATR as defined in ETSI TS 103 205 [34] clause 8.2 shall be used as the single Card Identifier. The ATR for the purpose of matching is defined from T0 up to Tk, excluding TCK (if present).

Based on the above Card Identifier list the **CPE** shall perform matching to **ECI Clients**.

ECI Clients shall provide the list of eligible Card Identifier Specifiers if it is ready to connect to a card. The Exclusive Card attribute shall be present per Card Identifier Specifier and indicates that the **ECI Host** shall signal a **Smart Card** access resolution conflict to the **user** in case multiple **ECI Clients** request access to a **Smart Card** matching the Card Identifier Specifier and such a **Smart Card** is inserted or present in one of the **CPE**'s **Smart Card** readers.

The **ECI Host** shall detect and when possible resolve any conflicts between card identification and matching **ECI Clients** according to the following rules:

- A **Smart Card** is considered to match to an **ECI Client** if one of the card identifiers in its card identifier list matches to one of the Card Identifier Specifiers of the **ECI Client**.

- In case a **Smart Card** matches to multiple **ECI Clients** and none of the **ECI Clients** wishes exclusive access a card session is granted in the following order:
  - A card session shall first be established for the **ECI** Client that had a session with the card most recently.
  - If none such **ECI** Client exists or the card is not recognized to have been inserted in a card reader of the **CPE** before a card session may be established by an algorithm to be elected by the **ECI Host**.
- An **ECI Client** shall disconnect a **Smart Card** session in case it cannot operate with the **Smart Card** so the **ECI Host** can match it with other **ECI Clients**, which can attempt to use it.

ECI Clients shall be able to handle **ECI Host**-generated "connect" and "disconnect" events on a **Smart Card** session.

9.5.3.4 Smart Card reader contention management

This clause defines the application conflict resolution functionalities of **ECI Hosts** for managing contention between Clients and available card readers for accessing **Smart Cards**.

When accessing **Smart Cards** through a card reader (**Smart Card** session) the **ECI Client** shall provide the **Smart Card** session priority. The values are:

- **Active:** used for a primary function which if interrupted creates discomfort for the **user**. An example is a viewing session requested by the **user** or a recording session previously programmed by the **user**.
- **Background:** in use for background processing which can be interrupted if necessary - this is the default state. An example is the processing of EMM messages for acquisition of future access rights.

An **ECI Client** shall be able to request a **Smart Card** to be inserted - implying active use - with references to one or more **Media Handle**s or a string indicating the application requiring the card in case this is not required for a specific **Media Handle**.

The **ECI Host** shall direct the **user** to an appropriate card reader in case an **ECI Client** requests a card using the following guidelines:

- It shall attempt to direct to a free card reader if available.
- It shall attempt to direct to a background mode reader if no free reader is available.
- If no background mode or free readers are available it should attempt to direct to an active mode reader that causes the **user** the least aggravation by using information from the application/**ECI Client** on the current active sessions of those readers.

The above process can involve the **ECI Host** to use additional information to match the card to the proper reader type (e.g. physical dimensions), e.g. by associating a reader-type to the **ECI Client** that fits the requirements of a successful connection to the **ECI Client** - assuming the same card type will be reinserted in the future. The **ECI Host** can use its own policies for this purpose.

9.5.3.5 Smart Card session management API

9.5.3.5.1 General

The **Smart Card** session management API shall provide Clients managed access to **Smart Cards** as defined in clauses 9.5.3.3 and 9.5.3.4.

For the **Smart Card** session management the available API messages are listed in Table 9.5.3.5.1-1.

Table 9.5.3.5.1-1: Smart Card session management API messages

Message	Type	Dir.	Tag	Description
setCardMatch	set	C→H	0x0	Set card identification specifier list for **ECI Client**.
callCardSessionPrio	call	C→H	0x1	Set **Smart Card** session priority.
getCardConnStatus	get	H→C	0x2	Provides status of card connection status.
reqCardConOpen	A	H→C	0x3	Informs **ECI Client** that a card session has been opened.
reqCardConClose	A	H→C	0x4	Informs **ECI Client** that a card session has been closed.
reqCardConClose	A	C→H	0x5	Informs **ECI Host** that **ECI Client** wishes to terminate a session with the connected card.

9.5.3.5.2 setCardMatch Message

C→H setCardMatch(uint **matchListLenth,** CardSpecifier **matchList[])**

- This message permits the **ECI Client** to indicate with which card identifiers the **ECI Client** wishes to connect to.

CardMatch Property Definition

matchListLength: uint	Length of matchList in terms of specifiers.
matchList: CardSpecifier[].	See Table 9.5.3.6.1-1: **Smart Card** Communication Messages. The **ECI Host** shall use this list to match connected **Smart Cards** to the **ECI Client** in accordance with clause 9.5.3.3. The Type definition is given in Table 9.5.3.5.2-1 and the values for the specifierType field are defined in Table 9.5.3.5.2-2.

Table 9.5.3.5.2-1: Type definitions for IP socket API

```
#define MaxAtr 32
#define MaxAid 16

typedef struct CardSpecifier {
    bool exclusiveFlag;
    uchar specifierType;
    union specifier {
        struct {
            uchar atrLen;
            byte atr[MaxAtr];
        } atrSpec;
        struct {
            uchar aidLen;
            byte aid[MaxAid];
        } aidSpec;
    }
} CardSpecifier;
```

Table 9.5.3.5.2-2: Smart Card Specifier Type

Name	Value	Description
CardSpecifierATR	0x01	Card specifier is of ATR type. A card matches to the specifier if the **atrLen** field is identical to the ATR length of the card and the ATR bytes of the card match the first **atrLen** bytes of the **atr** field. The ATR of a card is defined in clause 9.5.3.5.3, T0..TCK.
CardSpecifierAID	0x02	Card specifier is of AID type. A card matches to the specifier if the **aidLen** field is identical to the AID length of the card and the AID bytes of the card match the first **aidLen** bytes of the **aid** field. The AID of a card is defined in clause 9.5.3.3.
RFU	other	Reserved for Future Use.

Pre conditions:

1) The **ECI Client** is prepared to respond to invCardConOpen and invCardConClose messages if **matchListLength** > 0.

Post conditions

1) The **ECI Host** will match any card inserted in a card reader to the **ECI Client** as defined in clause 9.5.3.3. In case of a match it shall open a card session to the **ECI Client** is required is defined in clause 9.5.3.3.

2) The **ECI Host** will not drop a running card session in case the new matchList no longer provides a match to the currently connected **Smart Card**. The **ECI Client** shall use the reqCardConnClose message for that purpose.

9.5.3.5.3 callCardSessionPrio Message

C→H callCardSessionPrio(uchar **priority,** uint **nrMh,** ushort **mH[],** char ***clientApplication)**

- This message updates the card session priority and provides the **ECI Host** with the list of **Media Handles** mH and the **ECI Client** internal reason for requesting or having an Active card session.

Call Parameter Definition

priority: uchar	Priority of the card session required by the **ECI Client**. Values are defined in Table 9.5.3.5.3-1.
nrMh: uint	Number of **Media Handles** depending on an Active session to the card.
mH: ushort	List of **Media Handle**s that require an Active session to a **Smart Card**.
clientApplication: char *	Null char terminated string with the reason for the **ECI Client** to require an active session with a **Smart Card** not related to a **Media Handle** activity. If this pointer equals NULL there is no such requirement. If not NULL the string value shall have a meaningful value for the **user**. The maximum number of displayable characters is 40.

Table 9.5.3.5.3-1: Smart Card Session Priority Values

Name	Value	Description
CardPriorityBackground	0x01	**ECI Client** Card Priority requirement is Background and is defined in clause 9.5.3.4.
CardPriorityActive	0x02	**ECI Client** Card Priority requirement is Active and is defined in clause 9.5.3.4.
RFU	other	Reserved for Future Use.

Post conditions:

1) The **ECI Host** shall manage the card session as defined in clause 9.5.3.4 in accordance with **priority** and use **mH** and **clientApplicaiton** for resolving access conflicts to card readers through the **user** interface if so required.

9.5.3.5.4 getCardConnStatus Message

C→H uchar **getCardConStatus()**

- This message returns current session connection status to a **Smart Card**.

Property definition: see Table 9.5.3.5.4-1.

Table 9.5.3.5.4-1: Card Connection Status values

Name	Value	Description
CardConNo	0x00	**ECI Client** does not have a session with a **Smart Card**.
CardConYes	0x01	**ECI Client** has a session with a **Smart Card**.
RFU	other	Reserved for Future Use.

9.5.3.5.5 reqCCardConOpen Message

H→C reqCCardConOpen() →
C→H resCardConOpen()

- **This message allows the ECI Host** to inform the **ECI Client** on a new session connection event to a card; **ECI Client** responds confirming the event is being processed.

Pre condition Request:

1) A card session with the **ECI Client** is to be established in accordance with clause 9.5.3.3.

Post condition Response:

1) The **ECI Client** will manage the session priority in accordance with the requirements thereto given in clause 9.5.3.4.

2) The **ECI Client** shall close the session if it has no purpose for the card as defined in clause 9.5.3.3.

9.5.3.5.6 reqCCardConClose Message

H→C reqCCardConClose () →
C→H resCardConClose ()

- **This message allows the ECI Host** to inform the **ECI Client** that the session to the card was closed. **ECI Client** responds confirming the event was processed.

Pre condition Request:

1) Card was removed from reader or a major malfunction in the card reader subsystem caused the connection to be lost.

Post condition Response:

1) The **Response** of the **ECI Client** confirms that the **ECI Client** has processed the event and is ready to accept a new card connection as defined by the CardMatch property.

9.5.3.5.7 reqHCardConClose Message

C→H reqHCardConClose() →
H→C reqHCardConClose ()

- This message allows the **ECI Client** to indicate to the **ECI Host** that it has no further purposes for interaction with the connected **Smart Card**.

Post conditions Response

1) The **ECI Host** connect the **Smart Card** to another matching **ECI Client** as defined in 9.5.3.3 and shall not attempt to connect this card to the **ECI Client** (pending reboots and power cycles).

2) The **ECI Host** shall wait till reception of the **Response** before possibly reconnecting another matching **Smart Card** to the **ECI Client**.

9.5.3.6 Smart Card Communication API Message Definitions

9.5.3.6.1 General

The **Smart Card** Command **Response** API shall provide the communication session primitives between an **ECI Client** and a **Smart Card** in the context of an opened **Smart Card** session managed by the **ECI Host**. The **ECI Client** can perform ISO/IEC 7816-3 [33] Command/Response exchanges with the **ECI Host** at the level of APDUs (see note) as defined in ISO/IEC 7816-3 [33] clause 12. The **ECI Client** has access to all **Smart Card** management functions and can perform reset and re-initialization with custom parameter settings if so required and retrieves the communication settings. The **ECI** API messages are defined in Table 9.5.3.6.1-1.

NOTE: This also permits T=0 protocol exchanges at TPDU level through the use of short command and response exchanges at the APDU level interface.

Table 9.5.3.6.1-1: Smart Card Communication API messages

Message	Type	Dir.	Tag	Description
reqCardCmdRes	A	C→H	0x6	Send card command, get card response back.
reqCardReInit	A	C→H	0x7	Reset card (warm or cold) and reruns initialization sequence with the latest initialization preference setting.
callCardSetProp	set	H→C	0x8	Set card communication parameter.
callCardGetProp	get	H→C	0x9	Get card communication property/parameter.

9.5.3.6.2 reqCardCmdRes Message

C→H reqCardCmdRes(byte **nodeAddrByte,** uint **cmdApduLen,** byte **cmdApdu[]) →**
H→C resCardCmdRes(uint **resApduLen,** byte **resApdu[])**

- This message sends a command APDU to the **Smart Card** via the **ECI Host**, and gets a response APDU back is defined in clause 12 of ISO/IEC 7816-3 [33]. Related error codes are defined in Table 9.5.3.6.2-1.

Request Parameter Definition:

nodeAddrByte: byte	Node address byte for T=1 protocol setting of the established **Smart Card** protocol as defined in ISO/IEC 7810 [36] clause 11.3.2.1 This parameter is ignored in case the **Smart Card** protocol setting is T=0.
cmdApduLen: unit	Length of the cmd APDU in bytes. Note that the internal length coding of the **cmdApdu** shall not exceed the **cmdAduLen**.
cmdAdpu: byte []	The command APDU to be send to the card. Excess bytes in the cmdApdu field are ignored by the **ECI Host**.

Response Parameter Definition:

resApduLen: uint	Length of **Response** APDU in bytes.
resAdpu: byte []	The **Response** APDU received from the card.

Pre conditions Request:

1) The **ECI Client** has an open **Smart Card** session.

2) The previous reqCardCmdRes has resulted in a resCardCmdRes or the connection has (re-)initialized.

Table 9.5.3.6.2-1: resCardCmdRes Error Codes

Name	Description
ErrCardConnOpenNot	**See** Table 9.5.3.7-1.
ErrCardConnFail	

9.5.3.6.3 reqCardReInit Message

C→H reqCardReInit(uchar **resetMode) →**
H→C resCardReInit()

- This message requests the **ECI Host** to reset the **Smart Card** with resetMode, and reinitialized it with the latest card connection preference settings. The **Response** is returned when the process has completed (or failed). Related error codes are defined in Table 9.5.3.6.3-2.

Request Parameter Definition:

resetMode: uchar	See Table 9.5.3.6.3-1.

Table 9.5.3.6.3-1: Card resetMode values

Name	Value	Description
CardResetCold	0x01	A cold reset shall be performed and the card shall be re-initialized as if it was just powered up for the first time (see ISO/IEC 7816-1 [31] clause 6.2.3).
CardResetWarm	0x02	A warm reset shall be performed, the card communication timing parameters shall be re-initialized (see ISO/IEC 7816-3 [33] clause 6.2.3) and the "protocol and parameter selection" as defined in ISO/IEC 7816-3 [33] clause 9 shall be performed again, if applicable. This can be used specifically to attempt to switch the interface timing parameters to an **ECI Client** preferred value.
RFU	other	Reserved for Future Use.

Pre conditions Request:

1) The **ECI Client** has an open **Smart Card** session.

Post conditions Response:

1) The **Response** indicates the successful establishment of the interface protocol and parameter settings.

Table 9.5.3.6.3-2: resCardCmdRes Error Codes

Name	Description
ErrCardConnOpenNot	See Table 9.5.3.7-1.
ErrCardConnFail	

9.5.3.6.4 callCardSetProp Message

C→H callCardSetProp (ushort **propTag,** uin**t valueLen,** byte ***propValue)**

- This message sets the writable property indicated by **propTag** of the **Smart Card** interface to **propValue**.

Request Parameter Definition:

propTag: ushort	The tag of the Card Communication Protocol property to be changed. The values are defined in Table 9.5.3.6.5-2.
valueLen: uint	Length of the paramValue field in bytes.
propValue: byte *	Pointer to the property value to be written to the parameter indicated by paramTag.

Table 9.5.3.6.4-1: callCardSetProp Error Codes

Name	Description
ErrCardConnOpenNot	See Table 9.5.3.7-1.

9.5.3.6.5 callCardGetProp Message

C→H callCardGetPropf(ushort **propTag,** uint **valueLen,** byte ***propValue)**

- This message reads the accessible property indicated by **propTag** of the **Smart Card** interface into **propValue**. Related error codes are defined in Table 9.5.3.6.5-1.

Request Parameter Definition:

propTag: ushort	The tag of the Card Communication Protocol property to be changed. The values are defined in Table 9.5.3.6.5-2.
valueLen: uint	Maximum length of the paramValue field in bytes. Any excess bytes of property are not copied to propValue.
propValue: byte *	Pointer to the requested property value.

Table 9.5.3.6.5-1: callCardSetProp Error Codes

Name	Description
ErrCardConnOpenNot	See Table 9.5.3.7-1.

Table 9.5.3.6.5-2: Card API Tag Values and semantics for Card Protocol Properties

Name	Tag Value	Description
CardPropClass	0x0001	One byte. Value Class A=0x01, Class B = 0x02, Class= 0x03. Other values are reserved for future use. Read only.
CardPropAtrLen	0x0002	One byte. Length in bytes of the card's ATR in **CardPropAtr**. Read only.
CardPropAtr	0x0003	Byte string, max. 16 bytes. Card ATR on cold reset. Read only.
CardPropPpsExch	0x0004	Card and interface completed a successful PPS exchange if unequal 0x00.. Read only.
CardPropPpsVal	0x0004	One byte. Value of result of card PPS exchange of PSS1. Other values are not supported by the present document. Read only.
CardPropTAEff	0x0005	One byte. The effective value of TA applied for clock timing on the interface. Read only.
CardPropTCEff	0x0006	One byte. The effective value of TC applied for clock timing on the interface. Read only.
CardPropProt	0x0007	One byte. This indicates the protocol selected by the interface device to communicate with the card. The values are defined in ISO/IEC 7816-3 [33], clause 8.2.3, "T" field. The value 0x00 indicates the T=0 protocol, value 0x01 indicates the T=1 protocol. Other values can appear (up to 0x0E). Read only.
CardPropT1IFSC	0x0008	One byte. The current protocol value of IFSC (Information Field Size of Card) in the T=1 protocol encoded is defined in ISO/IEC 7816-3 [33], clause 11.4.2. Read only.
CardPropT1IFSD	0x0009	One byte. The current protocol value of IFSD (Information Field Size of Device = card reader) in the T=1 protocol encoded is defined in ISO/IEC 7816-3 [33], clause 11.4.2. Read only.
CardPropAidListLen	0x000A	One byte: length of list of card AIDs retrieved from the card during initialization. Read only.
CardPropAidList	0x000B	*(byte[MaxAid]): list of AIDs retrieved from the card during initialization. Read only.
CardPropClassPref	0x0011	Three bytes. Sequence of preferred Class values. The values for preference shall be attempted to be established (without violating safe=ty) in order. The values of the 3 bytes are in **CardPropClass,** with value 0x00 meaning "no more preference". Read and write.
CardPropImplClock	0x0012	One Byte **TA** value to be applied in case $\mathbf{TA}_2$ bit 5 in the ATR indicates implicit values for the clock frequency shall be applied. Read and write.
CardPropPps1SegLen	0x0013	One Byte. Value represents an unsigned binary number. Minimum value is 0, maximum value ix 0x08. Represents the number of PPS1 values to try in a PPS exchange negotiation in **CardPropPps1Seq** is defined in ISO/IEC 7816-3 [33], clause 9. See note.
CardPropPps1Seq	0x0014	One Byte sequence of maximum length 8 starting with the most desirable value for PPS1 to try to establish in a PPS exchange. Values are defined in ISO/IEC 7816-3 [33] clause 9.2. Read and write.
CardPropInfdPref	0x0015	One byte. Value indicates the preferred IFSD value to be established for the T1 protocol by the interface device. Read and write.
RFU	other	Reserved for Future Use.
NOTE: Values for PPS2 and PPS3 are not supported in this API and are not required to be supported by the **ECI Host**. Read and write.		

9.5.3.7 Error codes for the Smart Card API

The values are meanings of the API specific errors that can be returned by the **Response** messages for this API are listed in Table 9.5.3.7-1.

Table 9.5.3.7-1: Error codes of the Smart Card API

Name	Value	Description
ErrCardOpenNot	-256	No card session established.
ErrCardConnFail	-257	Card session established but no connection established (after reset).
RFU	**Other**	Reserved for Future Use.

9.5.4 Data Carousel Acquisition API

9.5.4.1 General

The Data Carousel Acquisition API permits an **ECI Client** to retrieve information from an **ECI** formatted broadcast carousel as defined in clause 7.7.2. An **ECI Client** can use this among others to retrieve possibly updated import export information.

NOTE: Data carousels are designed to carry quasi-static data and not a transport protocol of preference for transitory data.

An **ECI Client** can read from a carousel data directly or request the **ECI Host** to monitor updates of a carousel item module or group it is interested in. For monitoring this may either be during PwrOn power state or at some specified interval period during standby state. It is encouraged (for power consumption management reasons) to let these periods coincide with the **ECI Host** monitor periods.

The **ECI Host** will try to acquire the requested data and store it in a file for later access by the **ECI Client** through the file system API. The **ECI Host** provides for a minimum number of parallel channels of acquisition per **ECI Client** as defined in ETSI GR ECI 004 [i.10].

The messages of the Data Carousel Acquisition API are listed in Table 9.5.4.1-1.

Table 9.5.4.1-1: ECI Data Carousel Acquisition API messages

Message	Type	Dir.	Tag	Description
reqDCAcqGroupInfo	A	C→H	0x0	The **ECI Client** requests the **ECI Host** to read the GroupInfoIndication structure in the DSI message of the specified ECI data carousel.
reqDCAcqModule	A	C→H	0x1	The **ECI Client** requests the **ECI Host** to acquire a specific **ECI** data carousel module into a file using a module filter parameters and various modes

9.5.4.2 reqDCAcqGroupInfo Message

C→H reqDCAcqGroupInfo (uint **operatorId,** uint **platformId)** →
H→C resDCAcqGroupInfo (byte **gii[])**

- The **ECI Client** requests the **ECI Host** to read the GroupInfoIndication structure in the DSI message of the specified **ECI** data carousel. Related error codes are defined in Table 9.5.4.2-1.

Request Parameter definitions:

operatorId: uint	20-bit ID of the operator as found in ECI_carousel_id structure carried in the data_broadcast_id_descriptor() in the PSI (see clause 7.7.2.4).
platformId: uint	20-bit ID of the **Platform Operation** as found in ECI_carousel_id structure carried in the data_broadcast_id_descriptor() in the PSI (see clause 7.7.2.4).

Response Parameter definitions:

gii: byte[]	Byte array carrying the GoupInfoIndication structure as carried in the DSI of the carousel, as defined for DVB DSM-CC [15].

Detailed semantics:

- The **ECI Host** only provides access to carousels of clients that are loaded.

Table 9.5.4.2-1: reqDCGroupInfo Error Codes

Name	Description
ErrDCAcqNetwAccessResource	See Table 9.5.4.4-1
ErrDCAcqNetwAccessFail	
ErrDCAcqNoCarousel	

9.5.4.3 reqDCAcqModule Message

C→H reqDCAcqModule(uchar **aid,** fileName **fname**, uint **oId,** uint **pId,** byte **dType,** uint **model,** uint **version,** uint **index,** uint **mode)** →
H→C resDCAcqModule()

- This message allows the **ECI Client to** request the **ECI Host** to acquire a specific **ECI** data carousel module into a file using a module filter parameters and various modes.

Request Parameter definitions:

aid: uchar	Number of the acquisition filter. An **ECI Client** can have a maximum of 3 active acquisition filters (values 0 .. 2).
fname: fileName	Name of the file to which the data from the carousel module, which is to be acquired, shall be copied. Any existing data is overwritten.
oId: uint	20-bit ID of the operator as found in ECI_carousel_id structure carried in the data_broadcast_id_descriptor() in the PSI (see clause 7.7.2.4).
pId: uint	20-bit ID of the **Platform Operation** as found in ECI_carousel_id structure carried in the data_broadcast_id_descriptor() in the PSI (see clause 7.7.2.4).
dType: byte	This field should match to the Descriptor type field of the module group as defined in table 7.7.2-3.
model: uint	Carrying 16 bit unsigned value that should match to the model field in the compatibilityDescriptor of the group to be acquired. See Table 7.7.2.4-1.
version: uint	Carrying 16 bit unsigned value that should match (positive filter) or not match (negative filter) or be disregarded in matching, to the version field in the compatibilityDescriptor of the group to be acquired, depending **mode** parameter bit 0 and 1. See Table 7.7.2.4-1.
index: uint	Index of the module to be accessed in the group. This parameter shall be interpreted according to **mode** parameter bit 1.
mode: uint	Parameter is comprised of several fields: **bit 0:** signals the positive or negative filtering on **version**: 0b0 is positive filtering, 0b1 is negative filtering, **bit 1:** signals if filtering on **version** is to be ignored (value 0b1) or not (value 0b0), **bit 2:** signals if index is to be ignored (value 1) and any module is to be acquired (for single module carousels) or whether index needs to be used (modulo numberOfModules, see Table 7.7.2.6-1), **bit 29:** if set the **ECI Host** shall perform acquisition during standby by checking the carousel in accordance with its own acquisition requirements for this carousel and that such acquisition should continue until further notice in both standby and powerOn modes until such time the requested data was acquired, **bit 30:** signals if the acquisition shall assume the datacarousel is running and acquisition is to be completed within normal carousel schedule time (value 0b0) or whether the acquisition shall proceed as and when the carousel can be acquired and as and when the acquisition filter matches (0b1) (i.e. just wait till the data presents itself), **bit 31:** enable (value 0b1) or disable (value 0b0) acquisition with this filter **aid**.

Pre conditions Response:

1) The requested carousel module was acquired, a file system error was encountered or if **mode** bit 30 is set an acquisition problem was encountered.

2) The **ECI Host** is in PwerOn state. I.e. the **ECI Client** is not wakened on an acquisition during standby.

Post conditions Response:

1) The file contains the specified module or an error occurred.

2) When **mode** parameter bit 30 is set no acquisition errors can occur.

Detailed semantics:

- The **ECI H**ost only provides access to carousels of **ECI Clients** that are loaded and for which it is performing monitoring of the broadcast data carousel for **ECI Host** purposes.
- If not set no such standby acquisition will be performed. **ECI Clients** wishing to create their own acquisition scheduling can do so using the Wakeup API in clause 9.4.7.3.
- The **ECI Host** shall provide a "trivial" **Response** in case request with mode bit 31 cleared.

The related error codes are listed in Table 9.5.4.3-1.

Table 9.5.4.3-1: reqDCAcqModule Error Codes

Name	Description
ErrDCAcqNetwAccessResource	See Table 9.5.4.4-1
ErrDCAcqNetwAccessFail	
ErrDCAcqNoCarousel	
ErrDCAcqCarNoGroup	
ErrDCAcqCarNoModule	
ErrDCAcqCarTimeout	
ErrDCAcqFileSystemFailure	
ErrDCAcqFileQuotaExceeded	

9.5.4.4 Error Codes for the Data Carousel Acquisition API

The values are meanings of the API specific errors that can be returned by the **Response** messages for this API are listed in Table 9.5.4.4-1.

Table 9.5.4.4-1: Error codes media session API for TS media

Name	Value	Description
ErrDCAcqNetwAccessResource	-256	See Table 9.6.2.3.7-1.
ErrDCAcqNetwAccessFail	-257	See Table 9.6.2.3.7-1.
ErrDCAcqNoCarousel	-258	No carousel with matching operator and **Platform Operation**ID was found in the broadcast networks accessible to the **ECI Host**.
ErrDCAcqCarNoGroup	-260	The groupInfoIndication structure in carousel DSI was found but no matching group was found.
ErrDCAcqCarNoModule	-261	The carousel group (DII) was found but no matching module could be found.
ErrDCAcqCarTimeout	-262	A timeout occurred accessing the carousel DSI, DII or DDB.
ErrDCAcqFileSystemFailure	-263	See Table 9.4.5.5-1.
ErrDCAcqFileQuotaExceeded	-264	See Table 9.4.5.5-1.

9.6 APIs for access to the ECI Host decryption resource

9.6.1 ECI Host decryption API

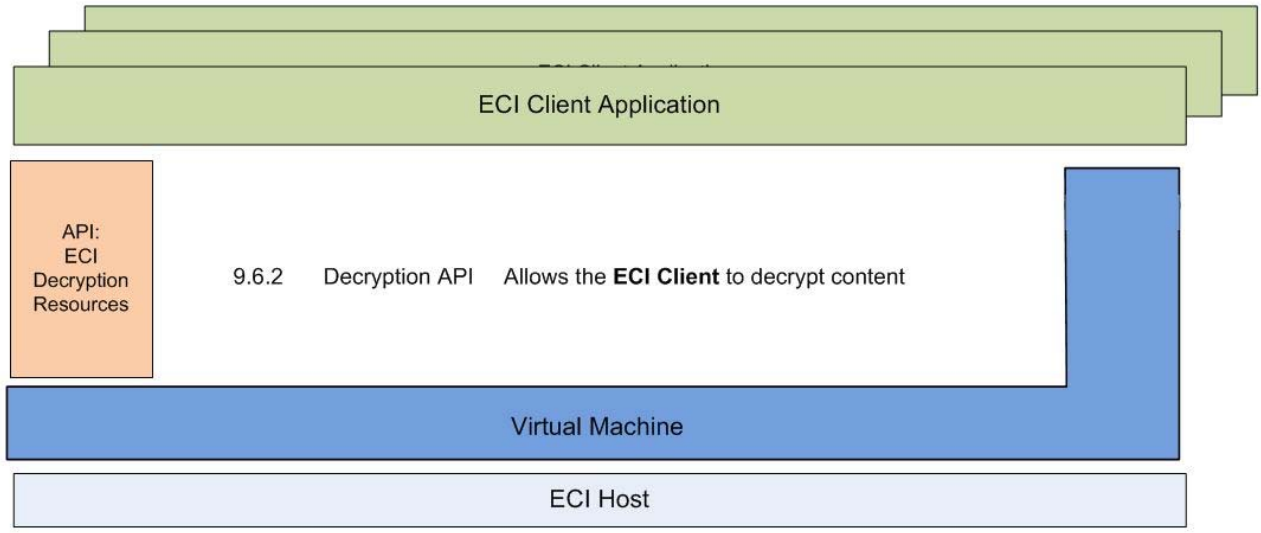


Figure 9.6.1-1: Block diagram of the APIs defined in clause 9.6

Table 9.6.1-1lists the APIs covered in clause 9.6 and Figure 9.7.1 illustrates the location of the APIs defined in clause 9.6 with the **ECI** architecture.

Table 9.6.1-1: List of APIs defined in clause 9.6

Clause	API name	Description
9.6.2	**ECI Host** decryption API	Allows the **ECI Client** to deliver Standard URI information related to a certain content element to the **ECI Host**.

9.6.2 Definition of the ECI Host decryption API

9.6.2.1 Introduction

The Decryption APIs allows the **ECI Host** (e.g. on request of resident or downloaded applications) to select an **ECI Client** matching the content decryption requirements and reques**t** it to be decrypted. All decryption messages between an **ECI Client** and a the **ECI Host** are exchanged in the context of a **Media Handle** which represents the content, any associated delivery network and resources required to decode it.

The following APIs make up the decryption APIs:

1) Generic media session API for all media types including matching function between content and **ECI Client**.

2) Transport Stream Decryption APIs.

3) File and stream Decryption APIs.

9.6.2.2 Media Session API

9.6.2.2.1 General

The **ECI Client** can announce the list of Match Specifiers by which the **ECI Host** can match it to content.

The **ECI Host** can request a matching **ECI Client** to open a descrambling session for a **Media Handle**. The opening of a session does not imply any decoding to commence. It merely ensures any resources needed for accessing content and/or metadata therein and for performing a descrambling session are available at both the **ECI Host** as well as on the **ECI Client** side. **ECI Clients** should ensure access to **Smart Cards** or other resources needed to actually descramble content are available before confirming a session. Table 9.6.2.2.1-1 lists the API functions.

Table 9.6.2.2.1-1: Media Handle Decryption Session API messages

Message	Type	Dir.	Tag	Description
setDcrMhMatch	Set	C→H	0x0	Signals to **ECI Host** under which Ids the **ECI Client** can be recognized for descrambling content.
reqDcrMhOpen	A	H→C	0x1	**ECI Host Request**s **ECI Client** to open a media session of a specified type using a **Media Handle**.
reqDcrMhClose	A	H→C	0x2	**ECI Host** closes a media session with an **ECI Client**.
reqDcrMhBcAlloc	A	C→H	0x3	**ECI Client Request**s **Media Handle** session for its own broadcast network access purposes.
reqDcrMhCancel	A	C→H	0x4	**ECI Client** cancels a media session with the **ECI Host**.

9.6.2.2.2 setDcrMhMatch API Message

C→H setDcrMhMatch(uint **matchListLength,** MatchSpecifier **matchList[])**

- This message permits the **ECI Client** to indicate to the **ECI Host** the decryption system Ids for which it is able to provide Transport Stream decryption services.

NOTE: The actual ability to decrypt content may depend on subscription, payment status or other conditions.

SetDcrMhMatch Property Definition

matchListLength: uint	Length of matchList in terms of specifiers.
matchList: MatchSpecifier[].	Table 9.6.2.2.2-1. The **ECI Host** shall use this list to match content to potential **ECI Client** decryption capabilities in accordance with clause 9.5.3.3. The match specifiers are defined by the MatchSpecifier type. All fields of MatchSpecifier shall match with the content in order to generate a match.

Table 9.6.2.2.2-1: Type definitions for MatchSpecifier

```
#define MaxMhSubFormat 16;
typedef struct MatchSpecifier {
   uchar decryptIdType; /*see Table 9.6.2.2.2-2 */
   union decryptId {
      bool     ECI Client ID;
      ushort   dvbCaId;
      byte     uuid[16];
   }
   byte  mhType;
   byte  subFormat[MaxMhSubFormat];
} MatchSpecifier;
```

Table 9.6.2.2.2-2: setDcrMhMatch decryptIdType definition

Name	Value	Description
None	0x00	Does not match to any content on an issued **Request**; indicates "no match" in case of opening a session.
ClientEciId	0x01	The identification of the **ECI Client** can be done based on of the **ECI Client** Id, composed of the 20-bit values (not including type and version fields) <<operator_id,platform_operation_id>,<vendor_id,client_id>> as specified in clause 7 of the present document.
ClientDvbCaId	0x02	The decryptId is a Conditional Access System Identifier as defined in CEN EN 50221 [13] and ETSI EN 301 192 [15]. This value indicates that dvbCaId is the used variant of the specifierType union. The actual+ values for dvbCaId are as defined in [13].
ClientUUID	0x03	The decrypted is a DRM ID as defined by CENC/Dash, specified as a UUID IETF RFC 4122 [12].
RFU	other	Reserved for Future Use.

mhType: unit	Type of the **Media Handle** (main decryption mode) supported by the **ECI Client** for this ClientEciID.
subFormat: byte[]	This parameter permits additional type specification to be defined for the **ECI Client**. The interpretation of these bytes depends on **mhType** is defined in Table 9.6.2.2.2-3.

Table 9.6.2.2.2-3: subFormat type definition

mhType value	Semantics of subFormat field
ISOBMFF	The subFormat field contains zero or more sequential 4CC definitions of the ISOBMFF ftyp or styp box brand values that are suited for decoding by the **ECI Client**. One (or more) of these 4CC values shall match to the `major_brand` or `compatible_brands[]` values of the ftyp or styp box of the ISOBMFF container. Value 0x0000 in subFormat shall mean no value (always mismatch), value 0xFFFF as the first entry shall mean any brand value (regardless of the following bytes).
Other	Reserved for Future Use.

Detailed semantics:

When trying to render Transport Stream based content the **ECI Host** shall try to match the content to the available **ECI Clients** using the following rules in priority order:

1) The **ECI Host** shall try to establish a set of applicable match specifiers using **ECI Client** IDs for that content as defined in clause 7.2.2. If any applicable **ECI Client** ID and associated match properties match to the **MatchSpecifier** of one **ECI Client** it shall offer content for decryption to that **ECI Client**. If multiple **ECI Clients** match the **ECI Host** shall use the following procedure:

   a) The **ECI Host** shall offer the content for decryption with the **ECI Client** that most recently successfully has delivered the CWs for decryption of content from the same "content source".

   b) If the first **ECI Client** fails to decrypt the content it shall attempt to use alternative **ECI Clients** that match whereby it should apply the **ECI Clients** order of most recent successful decryption history in relation to "content source".

2) If the **ECI Host** cannot establish any **ECI Client** ID for the content or if none of the **ECI Clients** under 1) above can decode the content, the **ECI Host** shall try to establish a set of other IDs for the content is defined in clause 9.5.4.3. If only one identifier and associated match properties match one **ECI Client** the **ECI Host** shall offer content for decryption to that **ECI Client**. If multiple **ECI Clients** match the **ECI Host** shall use the following procedure:

   a) The **ECI Host** shall offer the content for decryption with the **ECI Client** that most recently successfully decrypted content from the same "content source".

   b) If the first **ECI Client** fails to decrypt the content it shall attempt to use alternative **ECI Clients** that match whereby it should apply the **ECI Clients** order of most recent successful decryption history in relation to "content source".

The term "content source" in the above shall at minimum encompass:

1) A DVB broadcast network or bouquet therein that originates the TS.

2) A website used for browsing with a browser that offers references to content.

9.6.2.2.3 reqDcrMhOpen Message

H→C reqDcrMhOpen(ushort **mH,** MatchSpecifier **match)** →
C→H resDcrMhOpen(ushort **mH)**

- This message allows the **ECI Host** to request a decryption session with the **ECI Client**. **ECI Client** should reserve all resources normally required to perform decryption as identified by **mhType** and **match**. Related error codes are defined in Table 9.6.2.2.3-1.

Request parameter definition

mH: ushort	**Media Handle** of content to be decrypted.
match: MatchSpecifier	Copy of the matching specifier (also contains the medias Handle type of the session).

Response parameter definition

mH: ushort	**Media Handle** of content to be decrypted.

Pre conditions Request:

- The **ECI Host** has reserved all resources required to decrypt the content. For TS content this includes any tuning or other network access resources and applicable control there over, demultiplexing resources and descrambling resources for at minimum one cw-pair application.

Post conditions Response:

- In case of a successful result the **ECI Client** has reserved all resources typically required for decoding content for the requested session. This should include access to any external resources (DRM servers, **Smart Cards**, etc.) typically required for a decryption operation.

NOTE: Resources required by exception or resources that can normally be attained when required are excluded.

- In case ErrDcrUserDelay is returned the **ECI Client** is pending **user** input to open the session (e.g. to get access to a **Smart Card**). The **ECI Host** should repeat sending the reqDcrMhOpen Request (with the same parameters) until a positive result is returned or a definitive error is returned or alternatively may send a reqDcrMhClose to terminate the pending session. The **ECI Client** may cancel with reqDcrMhCancel in case it cannot attain the required **user** input.

Table 9.6.2.2.3-1: reqDcrMhOpen Error Codes

Name	Description
ErrDcrUserDelay	See Table 9.6.2.2.7-1
ErrDcrCardMissing	
ErrDcrServiceMissing	
ErrDcrResourceMissing	
ErrDcrMmiMissing	

9.6.2.2.4 reqDcrMhClose Message

H→C reqDcrMhClose(ushort **mH)** →
C→H resDcrMhClose(ushort **mH)**

- This message enables the **ECI Host** to close a decryption session with the **ECI Client**. The **ECI Client** may release the resources for this session.

Request parameter definition

mH: ushort	**Media Handle** of session to be closed

Request parameter definition

mH: ushort	**Media Handle** of session closed

Post conditions Request:

- The **ECI Client** releases any resources it required specifically for the session.

Post conditions Response:

- The **ECI Host** may release any resources related to the **Media Handle**.

9.6.2.2.5 reqDcrMhBcAlloc Message

C→H reqDcrMhBcAlloc(byte **networkType[2],** uchar **priority,** char **reason[80])** →
H→C resDcrMhBcAlloc(ushort **mH)**

- This message allows the **ECI Client** to request the connection to a broadcast network for security data acquisition purposes.

Request parameter definition

networkType: byte[2]	Broadcast network type to be accessed by an ECI Client; values in accordance with Table 9.6.2.2.5-1. DVB version 2 type network access shall imply; type 1 network access is also possible.
priority: uchar	Priority for accessing the network is defined in Table 9.6.2.2.5-1.
reason: char[80]	Null terminated string of maximum 80 characters that can be presented to the user to resolve resource conflict in the ECI Host for resolving this request.

Table 9.6.2.2.5-1: Broadcast Network Access Priority definition

Name	Value	Description
DcrAllocPrioBackground	0x01	Access is required for background processing which may not be granted or may be interrupted when a task with a higher priority requires access to the resources. An example is accessing EMM or security renewability data on a central multiplex.
DcrAllocPrioActivec	0x02	Access is required for a primary descrambling function and if not granted (or when interrupted) creates discomfort for the user. An example is a viewing session requested by the **user** or a recording session previously programmed by the **user**.
RFU	other	Reserved for Future Use.

Request parameter definition

mH: ushort	**Media Handle** of session opened

Detailed semantics:

- The **ECI Host** may cancel the session in case another task requires the network access resources with a higher priority using the reqDcrMhClose message.
- The **ECI Client** shall close the session using the reqDcrHmCancel message in case it no longer requires access to the network.

Post conditions Request:

1) The **ECI Host** has allocated all resources for accessing the requested network type.

Post conditions Response:

1) The **ECI Client** shall tune to acquire a Transport Stream using the reqDcrTsRelocate message before commencing section acquisition.

Table 9.6.2.2.5-2: reqDcrMhAlloc Error Codes

Name	Description
ErrDcrNetworkAccessCapability	See Table 9.6.2.2.7-1.
ErrDcrNetworkAccessResource	
ErrDcrPrioOverride	
ErrDcrResourceMissing	

9.6.2.2.6 reqDcrMhCancel Message

C→H reqDcrMhCancel(ushort **mH,** uchar **reason)** →
H→C resDcrMhCancel(ushort **mH)**

- This message allows the **ECI Client** to close a decryption session with the **ECI Host**. **ECI Client** has released all resources specifically needed for the session.

Request parameter definition:

mH: ushort	**Media Handle** of session to be closed.
reason: uchar	Reason for cancelling the decryption session. The values are defined in Table 9.6.2.2.6-1.

Table 9.6.2.2.6-1: reqDcrMhCancel reason values

Name	Value	Description
DrcMhUndefined	0x00	An undefined error occurred in the **ECI Client** requiring it to cancel the session.
DcrMhCardMissing	0x01	**Smart Card** is required for decoding but could not be successfully (re-)connected and assist in decrypting content within a reasonable time.
DcrMhServiceMissing	0x02	A service (external to the **CPE**) supporting the **ECI Client** in providing decryption services required to maintain a decryption session is not available in a reasonable time.
DcrMhResourceMissing	0x03	A resource (internal to the **CPE**) required for providing decryption services is not available to the **ECI Client** within a reasonable time (not including **DcrMhMmiMissing**).
DcrMhMmiMissing	0x04	The **ECI Client** was not successful in attaining an MMI session resource for **user** interaction required for maintaining the decryption session within a reasonable time.
DcrMhAllocTerminate	0x05	**Media Handle** was allocated on behalf of **ECI Client** through reqDcrMhAlloc and is no longer required by the **ECI Client**.
RFU	Other	Reserved for future use.

The reasonable time for the **ECI Host** to cancel a **Media Handle** session is defined in ETSI GR ECI 004 [i.10].

Response parameter definition:

mH: ushort	**Media Handle** of cancelled session

Pre condition Request:

- The **ECI Client** has released any resources it required specifically for the session.

Post conditions Request:

- The **ECI Host** may release any resources related to the **Media Handle**.

Post conditions Response:

- The **Media Handle** session is closed by the **ECI Host**.

9.6.2.2.7 Error codes for the Media Session API

The values are meanings of the API specific errors that can be returned by the **Response** messages for this API are listed in Table 9.6.2.2.7-1.

Table 9.6.2.2.7-1: Error codes media session API for TS media

Name	Value	Description
ErrDcrUserDelay	-256	Long delay waiting for input from **user** required to complete the operation occurred. Operation not completed.
ErrDcrCardMissing	-257	**Smart Card** required for session is not accessible/available
ErrDcrServiceMissing	-258	A service from outside the **CPE** required to support the **ECI Client** in decryption operations is not available.
ErrDcrResourceMissing	-259	A undefined resource inside the **CPE** required for accessing or decrypting content is not available.
ErrDcrMmiMissing	-260	**ECI Client** access to the MMI is not available.
ErrDcrDescrContinue	-261	**ECI Host** continues to attempt to descramble content in this TS.
ErrDcrNetworkAccessCapability	-262	The **ECI Host** does not have a network access resource for locating the requested TS.
ErrDcrNetworkAccessResource	-263	The **ECI Host** cannot acquire the network access resource for accessing the requested TS.
ErrDcrPrioOverride	-264	A higher priority task in the **CPE** required he resources for the **Media Handle** causing the **Media Handle** session to be terminated.
RFU	other	Reserved for Future Use.

9.6.2.3 Descrambling Transport Stream Data

9.6.2.3.1 Introduction

The **ECI Host** can request the **ECI Client** to perform a descrambling session (of a specific type: in this case the mpeg broadcast type) by providing it with a **Media Handle** (see clause 9.1.2). The **ECI Host** will provide the security data as specified by the **ECI Client** for descrambling the data.

For descrambling content in most Transport Stream formats **ECI** uses an implicit timing model for synchronization of the control words with the content offered to the descrambler. In this model the **ECI Host** provides the **ECI Client** with security control data from the Transport Stream as it is being demultiplexed and descrambled. The **ECI Client** provides the required control words (typically 2 per elementary stream, often identical for all elementary streams) at the appropriate time. The **ECI Client** typically decodes an ECM to CWs, and loads the CWs into the descrambler immediately. The application of these control words is synchronized with the stream through the signalling in the content stream using the scrambling control bits at TS packet level or at PES packet level.

The API is partitioned in the following clauses:

1) Starting, restarting and stopping Transport Stream decryption (clause 9.6.2.3).

2) Security data acquisition (clause 9.6.2.3.5).

3) Broadcast tuning functions (clause 9.6.2.3.6).

9.6.2.3.2 Transport Stream format and session versions

Transport Streams descrambled through a **Media Handle** with the media session type **MhDvbTsBroadcast** shall comply with the following specifications: ISO/IEC 13818-1-1 [8] (specifically the application of scrambling control bits to TS packets) and ETSI ETR 289 [40].

9.6.2.3.3 ECI Host Processing Requirements

9.6.2.3.3.1 Scrambling cipher detection

ECI Hosts shall signal the applicable cypher mode to the **ECI Client** based on the following rules:

1) For DVB streams it shall use the signalling using the scrambling descriptor in the PMT as defined in ETSI TS 103 127 [41] and ETSI TS 100 289 [42].

2) If no descriptor is found under 1) and the source is a DVB broadcast network the **ECI Host** shall assume CSA1 is used as specified in the definition of the scrambling descriptor.

9.6.2.3.3.2 CA Identification detection

For establishing the list of applicable DVB CA IDs for a scrambled service, scrambling being detected by either TS or PES packet scrambling bits, in a Transport Stream (originating from a broadcast network or otherwise) the **ECI Host** shall use the following acquisition rules:

1) It shall attempt to retrieve the CA_descriptors as carried in the PMT of the service. In case this is unsuccessful and the content is scrambled.

2) It shall attempt to retrieve the set of CA_system_ids as carried in the CA identifier descriptor as carries in any DVB bouquet, SDT or EIT table applicable for the content.

NOTE: For some sources of Transport Stream based content the applicable CA or DRM ID may be known through other means.

9.6.2.3.4 Starting and stopping Transport Stream decryption

9.6.2.3.4.1 General

The **ECI Host** can start the decrypting content on an open **Media Handle** using the **ECI Client** resources reserved. The **ECI Host** shall provide a "CA-PMT" table containing the specification of the elementary streams to be decrypted. Table 9.6.2.3.4.1-1 lists the available decryption API messages.

Table 9.6.2.3.4.1-1: Media Handle TS content decryption API

Message	Type	Dir.	Tag	Description
reqDcrTsDescrStart	A	H→C	0x08	**Requests ECI Client** to descramble or return the descramble status of a programme in a TS.
reqDcrTsDescrStop	A	H→C	0x09	**ECI Host** requests **ECI Client** to descramble a **Media Handle**.
reqDcrTsDescrQuit	A	C→H	0x0A	**ECI Client** terminates a descrambling session with the **ECI Host**.

9.6.2.3.4.2 reqDcrTsDescrStart Message

H→C reqDcrTsDescrStart(ushort **mH,** uint **caPmtLen,** byte **caPmt[]) →**
C→H resDcrTsDescrStart(ushort **mH,** unit **sizeofEsStat,** descrStat **esStat[])**

- This message allows the requests the **ECI Client** to commence decryption of a programme as defined by **caPmt** on the stream identified by **mH** or inquires as to the ability or conditions to do so.

Request parameter definition:

mH: ushort	**Media Handle** of TS stream.
caPmtLen: uint	Length in bytes of the **caPmt** parameter.
caPmt: byte[]	ca_pmt object is defined in [16] clause 8.4.3 in network byte order, with a modified interpretation of ca_pmt_list_management and ca_pmt_cmd_id parameters as defined in Table 9.6.2.3.4.2-1.

The ca_pmt_list_management parameter values and semantics shall comply to the definitions in Table 9.6.2.3.4.2-1.

Table 9.6.2.3.4.2-1: ca_pmt_list_management values

Name	Value	Description
DcrTsDescrStartOnly	0x03	A single programme should be descrambled in the service. This may be a new or an updated value.
DcrTsDescrStartUpdate	0x05	Same meaning as **DcrTsDescrStartOnly**.
RFU	Other	Reserved for future use.

The ca_pmt_cmd_id parameter values shall be identical to CEN EN 50221 [13] clause 8.4.3 with the following restrictions:

1) The value 0x02 (ok_mmi) is not permitted.

2) The values 0x01 (ok_scrambling) and 0x03 (query) shall not occur in the same ca_pmt structure. I.e. a **Request** shall either be a pure query or a pure descrambling request.

Response parameter definition:

mH: ushort	**Media Handle** of TS stream.
sizeofEsStat: uint	Number of bytes of esStat parameter.
esStat: descrStat	The descrambling status of the elementary streams as specified in the **caPmt** parameter of the **Request**. descrStat is defined in Table 9.6.2.3.4.2-2. A **descrStat.pid** value shall occur only once in **esStat**. Each elementary_PID parameter of the ca_pmt structure of [13] shall occur once unless it's corresponding ca_pmd_cmd_id is 0x04 (not_selected) in which case it shall not occur in esStat.

Table 9.6.2.3.4.2-2: Type definition for descrStat structure

```
typedef struct descrStat {
   ushort pid;
   uchar    caStatus
} descrStat;
```

pid: ushort	PID value of the stream to be descrambled.
caStatus: uchar	Values shall correspond to the definition of the CA_enable parameter of the ca_pmt_reply object in [13], clause 8.4.3.

Detailed semantics:

1) The **ECI Host** shall issue this command in case the set of elementary streams to be decoded has to change.

2) The **ECI Host** shall issue a **reqDcrTsDescrEnd Request** in case the media session is stopped. Failure to do so may mislead the **ECI Client** to registering ongoing content consumption by the **user** and associated charges.

3) Related error codes are defined in Table 9.6.2.3.4.2-3.

Pre conditions Request:

1) **mH** is open and has a TS format.

Post conditions Request:

1) **ECI Client** may start descrambling actions and use other mH TS related functions.

Table 9.6.2.3.4.2-3: reqDcrTsStart Error Codes

Name	Description
ErrDcrUserDelay	See Table 9.6.2.3.7-1.
ErrDcrCardMissing	
ErrDcrServiceMissing	
ErrDcrResourceMissing	
ErrDcrMmiMissing	

9.6.2.3.4.3 reqDcrTsDescrStop Message

H→C reqDcrTsDescr Stop(ushort **mH) →**
C→H resDcrDescr Stop(ushort **mH)**

- This message allows the **ECI Host** to indicate the **ECI Client** that it shall stop the TS descrambling operation related to the current **mH**.

Request parameter definition:

mH: ushort	**Media Handle** of TS stream

Response parameter definition:

mH: ushort	**Media Handle** of TS stream

Pre conditions Response:

1) Any **ECI Client** operation related to descrambling mH is terminated.

9.6.2.3.4.4 reqDcrTsDescrQuit Message

C→H reqDcrTsDescr Quit(ushort **mH, ushort** reason**) →**
H→C resDcrDescr Quit(ushort **mH)**

- This message allows the **ECI Client** to inform the **ECI Host** that it has stopped to process keys for the TS descrambling operation related to the current **mH**.

Request parameter definition:

mH: ushort	**Media Handle** of TS stream
reason: ushort	The reason why the **ECI Client** has terminated the key processing for the descrambling operation is defined in Table 9.7.2.5.9-1.

Response parameter definition:

mH: ushort	**Media Handle** of TS stream

Pre conditions Response:

1) All **ECI Host** activities related to descrambling **mH** has terminated or an error is returned.

Post conditions Response:

1) All **ECI Client** activity related to **mH** shall terminate immediately or an error was returned.

Table 9.6.2.3.4.4-1: reqDcrMhCancel Error Codes

Name	Description
ErrDcrDescrContinue	See Table 9.6.2.3.7-1.

9.6.2.3.5 ECI Client Decryption data acquisition in TS

9.6.2.3.5.1 General

The **ECI Client** can acquire in-band TS data required for decryption purposes in the form of sections from the Transport Stream associated with a **Media Handle**. The most straightforward form is setting a section filter. In order to speed up acquisition on channel changes it can set a default section filter including the PMT and ECM stream. It can also read other standard MPEG and DVB tables from the **ECI Host**. MPEG sections are data structures as defined in ISO/IEC 13818-1-1 [8], clause 2.4.4.1, private_section() structure. The functions of this part of the MPEG TS API are listed in Table 9.6.2.3.5.1-1.

Table 9.6.2.3.5.1-1: ECI Host TS Descrambling Control Messages

Message	Type	Dir.	Tag	Description
setDcrTsSectionAcqDefault	set	C→H	0x10	Sets a default filter for section acquisition.
setDcrTsSectionAcq	set	C→H	0x11	Sets a filter for section acquisitions.
reqDcrTsSection	A	H→C	0x12	Forwards a acquired section to **ECI Client**.
reqDcrTsTable	A	C→H	0x13	**ECI Client** acquires a table in the stream.

9.6.2.3.5.2 Section Filter Specification

MPEG sections as defined is defined in ISO/IEC 13818-1-1 [8] clause 2.4.4.11 can be extracted on specification from an **ECI Client** to the **ECI Host** from a Transport Stream. The **ECI Host** shall support 8 section filters for an **ECI Client**. A section filter setup permits the **ECI Client** to filter from one PID in the TS stream with a limited number of indirect specifiers (e.g. for PMT). It permits the **ECI Client** to setup positive filters (selected section fields match the **ECI Client**'s specification) and negative filters (the section data differ from the **ECI Client**'s filter specification). The filtered sections may be clustered and sent either when reaching the maximum buffer size or alternatively be forwarded as soon as acquired.

The filtering of section bytes shall skip the 2nd and 3rd byte of a section.

The specification for a section filter is given in Table 9.6.2.3.5.2-1.

Table 9.6.2.3.5.2-1: Type definition for DcrSectionFilterSpec structure#define DcrSectionFilterMaxlen 16

```
#define DcrSectionFilterMaxlen 16
typedef struct dcrSectionFilterSpec {
   ushort    pid;
   ushort    caId;
   ushort    bufferSize;
   uint      timeout;
   uint      modeFlags;
   byte      filter[DcrSectionFilterMaxlen];
   byte      mask[DcrSectionFilterMaxlen];
   byte      neg[DcrSectionFilterMaxlen];
} dcrSectionFilterSpec;
```

The semantics are as follows:

pid: ushort	PID of TS packets to be filtered. PID values shall be represented by their unsigned 13-bit value: i.e. between 0x0000 and 0x1FFF. The PID of the PMT of the stream to be acquired is represented by 0x8000. The PID for an associated ECM stream to be acquired is represented by 0x8001.
caId: ushort	This field is relevant only when the value of the **pid** field is 0x8001. In that case the value of this field is the MPEG/DVB CA ID of the conditional access system for which the ECM stream shall be acquired. The **ECI Host** shall parse the PMT of the service to be descrambled and match **caId** field to the CA_descriptors (as defined in ISO/IEC 13818-1-1 [8]) applicable to the video PID if present or the first ES in the PMT and use the CA-PID field in the matching descriptor to identify the ECM stream to be acquired and filtered.
bufferSize: ushort	Maximum size of the buffer. At minimum a single section shall be buffered. By setting this field to zero every section will be forwarded separately.
timeout: uint	Timeout in ms for the filtering of a single section. Restarts at every section successfully filtered. Value zero means no timeout.
modeFlags: uint	When bit 0 is set the **ECI Host** shall prevent sending the same section to the **ECI Client** twice. The **ECI Host** shall use a buffer of previously acquired sections of max. 64 kB for this purpose. All other bits are reserved and shall be set to 0 by the **ECI Client**.
filter: byte []	Value to match to corresponding section bytes.
mask: byte[]	If a bit in this set to zero the corresponding match to the section value is ignored.
neg: byte []	If a bit is set to one the corresponding match to the section bit is negative.

A section matches the filter if all positively filtered masked section bits match their corresponding filter value and no negatively filtered masked section bits match their corresponding filter value (provided there is at least one negatively filtered bit). A section match (represented by **data** for section bytes 1 and 3-18) is defined by the **sectionFilterMatch** function.

```
bool sectionFilterMatch(byte *data, *filter, *mask, *neg) {
    int i;
    bool posMatch, negMatch;

    posMatch = True;
    negMatch = True;

    /* if all neg bytes are 0; the negative filter is always fulfilled */
    for (i=0; i< DcrSectionFilterMaxlen; i++)
        negMatch &&= neg[i] == 0;

    /* match section data to positive and negative filtering criteria*/
    for (i=0; i< DcrSectionFilterMaxlen; i++) {
        posMatch &&= (data[i] & mask[i] & ~neg[i]) == (filter[i] & mask[i] & ~neg[i]);
        negMatch ||= (data[i] & mask[i] & neg[i]) != (filter[i] & mask[i] & neg[i]);
    }
    return posMatch && negMatch;
}
```

9.6.2.3.5.3 reqDcrTsSectionAcqDefault Message

C→H setDcrTsSectionAcqDefault(ushort **mH,** uchar **filterNr,** dcrSectionFilterSpec **sectionFilter)**

- This message sets the default section filters that will be used by the **ECI Host** to acquire information from the stream for the **ECI Client** after a resDcrTsDescrStart message is received. This function can for instance be used by the **ECI Client** to speed up section acquisition of ECMs by the **ECI Host** during channel change.

Request parameter definition:

mH: ushort	**Media Handle** of TS stream on which to set the default section filter.
filterNr: uchar	Number of the filter to be programmed. The value shall be between 0 and 7.
sectionFilter: dcrSectionFilterSpec	Section filter specification according to clause 9.6.2.3.5.2 dcrSectionFilterSpec.

Post condition:

- This section filter shall be put into effect by the **ECI Host** immediately following a successful **resDcrTsDescrStart** is received. The **ECI Host** should anticipate a successful **resDcrTsDescrStart** if reasonably possible.

9.6.2.3.5.4 reqDcrTsSectionAcq Message

C→H setDcrTsSectionAcq(ushort **mH,** uchar **filterNr,** dcrSectionFilterSpec **sectionFilter**

- This message sets the section filters that will be used by the **ECI Host** to acquire information from the **mH** stream for the **ECI Client**.

Request parameter definition:

mH: ushort	**Media Handle** of TS stream on which to set the default section filter.
filterNr: uchar	Number of the filter to be programmed. The value shall be between 0 and 7.
sectionFilter: dcrSectionFilterSpec	Section filter specification according to section clause 9.6.2.3.5.2 dcrSectionFilterSpec.

Detailed semantics:

- Using this message after setting a default section filter will modify the section filter until the next resDcrTsDescrStart is issued on the same **Media Handle** which will reset it to the default section filter (if a default is set).

Post condition Set:

- This section filter shall be put into effect by the **ECI Host**.

9.6.2.3.5.5 reqDcrTsSection Message

H→C reqDcrTsSection(ushort **mH,** uchar **filterNr,** uint **sectionDataLen**, byte **sectionData[]) →**
C→H resDcrTsSectionAcq (ushort **mH,** uchar **filterNr)**

- This message sends one of more sections acquired by the **ECI Host** in the context of the TS stream identified by **mH** and the filter identified by **filterNr** to the **ECI Client**.
- Related error codes are defined in Table 9.6.2.3.5.5-1.

Request parameter definition:

mH: ushort	**Media Handle** of TS stream on which to set the default section filter.
filterNr: uchar	Number of the filter to be programmed. The value shall be between 0 and 7.
sectionDataLen: uint	Number of bytes in **sectionData**.
sectionData: byte []	Sequence of private_sections (bytes in network order) is defined in ISO/IEC 13818-1-1 [8] section 2.4.4.11. Any section with a CRC error is not passed to the **ECI Client**.

Response parameter definition:

mH: ushort	**Media Handle** of TS stream
filterNr: uchar	Number of the filter that was programmed

Pre condition Request:

1) Sections shall have been acquired by the **ECI Host** in accordance with the section filter specification or the timeout for the filter expired.

2) The previous **reqDcrTsSection** message was acknowledged with **resDcrTsSection**.

Post condition Response:

1) The next **reqDcrTsSection** message from the same filter may be sent by the **ECI Host**.

Table 9.6.2.3.5.5-1: reqDcrTsSection Error Codes

Name	Description
ErrDcrTsSectionTimeout	See Table 9.6.2.3.7-1.
ErrDcrTsSectionCrcErr	

9.6.2.3.5.6 reqDcrTsTable Message

C→H reqDcrTsTable(ushort **mH,** uchar **tableId,** uint **timeout,** uint **maxLen)**
H→C resDcrTsTable(ushort **mH,** uint **tableDataLen,** byte **tableData[])**

- This message requests the **ECI Host** to send the sections composing a standard table or sub table as applicable to the programme being descrambled on **mH**.

Request parameter definition:

mH: ushort	**Media Handle** of TS stream on which to set the default section filter.
tableId: uchar	Number of the filter to be programmed. Valid values are listed in Table 9.6.2.3.5.6-1.
timeout: uint	Timeout in milliseconds. Value of 0 means no timeout.
maxLen: uint	Maximum number of sectionData bytes to be returned. The **ECI Host** shall round down to the highest number of sections within this limit.

Table 9.6.2.3.5.6-1: ca_pmt_list_management values

Name	Value	Description
DcrTsTableMpegPat	0x0000	PAT table in accordance with ISO/IEC 13818-1-1 [8].
DcrTsTableMpegCat	0x0001	CAT table in accordance with ISO/IEC 13818-1-1 [8].
DcrTsTableMpegPmt	0x0002	PMT table of selected program in accordance with ISO/IEC 13818-1-1 [8]. Result is empty in case a composite PMT is used by the application.
DcrTsTableDvbNit	0x0140	NITactual_current table as specified in ETSI EN 300 468 [19] and ETSI TS 101 211 [21]. On cable networks using NIT_{other} to carry tables associated with the regions of such a network the applicable NIT_{other} table applicable to the **CPE**'s region shall be designated.
DcrTsTableDvbSdt	0x0142	SDTactual_current table as specified in ETSI EN 300 468 [19] and ETSI TS 101 211 [21].
DcrTsTableDvbBat	0x014A	BATactual table as specified in ETSI EN 300 468 [19] for the bouquet actively in use by the **ECI Host** and/or its application.
DcrTsTableDvbEitPf	0x014E	EITactual present and following table as specified in ETSI EN 300 468 [19] and ETSI TS 101 211 [21].
DcrTsDescrStartUpdate	0x05	Same meaning as **DcrTsDescrStartOnly**.

Response parameter definition:

mH: ushort	**Media Handle** of TS stream
tableDataLen: uint	Number of bytes in tableData.
tableData: byte []	Sequence of private_sections (bytes in network order) representing the (sub)table is defined in ISO/IEC 13818-1-1 [8], section 2.4.4.11.

Detailed semantics:

- The **ECI Host** shall use section filters to acquire fresh data for all tables that may be requested by the **ECI Client** (as well as for its other purposes). The table sections shall be sent once by the **ECI Host**. The **ECI Host** shall stall the **Response** if it still needs to acquire the requested table. The table shall be "up to date" and use the latest complete data available to the **ECI Host**. Error codes are defined in Table 9.6.2.3.5.6-2.

NOTE: A table can always be superseded by a next version in a stream at any time in the future.

- Minimum repetition rates for updating relevant DVB SI tables are defined in ETSI GR ECI 004 [i.10].
- PAT, CAT, and PMT: data is older than 20 seconds.

Table 9.6.2.3.5.6-2: reqDcrTsTable Error Codes

Name	Description
ErrDcrTsSectionTimeout	See Table 9.6.2.3.7-1.
ErrDcrTsSectionCrcErr	

9.6.2.3.6 ECI Client Source Control

9.6.2.3.6.1 General

The **ECI Client** has the ability to read the type of source of the Transport Stream, control (redirect) the source of the Transport Stream and redirect the program and/or components that are decoded by the **ECI Host**. The messages are listed in Table 9.6.2.3.6.1-1.

Table 9.6.2.3.6.1-1: TS Client source Control API Messages

Message	Type	Dir.	Tag	Description
getDcrTsSource	get	C→H	0x18	**ECI Client** gets the source of the TS.
reqDcrTsRelocate	A	C→H	0x19	**ECI Clients** relocates the source of the TS.
reqDcrTsSelectPrg	A	C→H	0x1A	**ECI Client** selects program in TS by program number.
reqDcrTsSelectPmt	A	C→H	0x1B	**ECI Client** selects program in TS by PMT.
reqDcrTsSelectCancel	A	C→H	0x1C	**ECI Client** cancels its previous program selection.

9.6.2.3.6.2 getDcrTsSource Message

C→H tsSourceType **getDcrTsSource(**ushort **mH**)

- This message returns type of source of the **Media Handle** in terms of network type and locator in the network.

Parameter definition:

mH: ushort	**Media Handle** of the TS stream to get the type and location of the tuned stream.

Property definition:

The property definitions are given in Table 9.6.2.3.6.2-1.

Table 9.6.2.3.6.2-1: Type definition for tsSourceType structure

```
#define MaxTsSourceDescr 254

typedef struct tsSourceType{
   ushort tsSourceTag ;
   byte   tsSourceDescr[MaxTsSourceDescr] ;
} tsSourceType ;
```

tsSourceTag: ushort	The type of the TS source. The defined values are listed below, including the corresponding meaning of **tsSourceDescr**.
tsSourceDescr: byte[MaxTsSourceDescr]	The meaning depends on tsSourceTag as listed in Table 9.6.2.3.6.2-2.

Table 9.6.2.3.6.2-2: Meaning of tsSource Tag

Name	Value	Description
tsSourceDvbTuner	0x0001	Source of TS is a DVB tuner. The tsSourceDescr contains a single descriptor from Table 9.6.2.3.6.2-3 in network byte order.
tsSourceDvbFile	0x0002	Source of TS is a file or other non tuneable asset like an IP network (see ETSI TS 102 034 [i.1]). The tsSourceDescr field is undefined.
tsDvbDuplet	0x8003	Source or TS can be found using the original network ID and Transport Stream ID within the present network. tsSourceDescr shall contain the network byte order of `struct dvbDuplet {ushort onid; ushort tsid};` This value will not be returned by getDcrTsSource message (which will return a tsSourceDvbTuner instead) but may be used in a reqDcrTsRelocate message.
RFU	Other	Reserved for future use.

Values higher than 0x7FFF are not absolute locators and shall not be returned by **getDcrTsSource**.

Table 9.6.2.3.6.2-3: DVB Tuner source descriptors

DVB Delivery descriptor Name	DVB Descriptor Tag value
terrestrial_delivery_system_descriptor	0x5A
T2_delivery_system_descriptor	0x7F, 0x04
satellite_delivery_system_descriptor	0x43
S2_delivery_system_descriptor	0x79
cable_delivery_system_descriptor	0x44
C2_delivery_system_descriptor	0x7F, 0x0D

The descriptors shall be used as defined in ETSI EN 300 468 [19], and shall contain a single destination frequency.

9.6.2.3.6.3 reqDcrTsRelocate Message

C→H reqDcrTsRelocate(ushort **mH**, tsSourceType **tsLoc**) →
H→C resDcrTsRelocate(ushort **mH)**

- This message requests the **ECI Host** to relocate the source of the TS to **tsLoc**. Related error codes are defined in Table 9.6.2.3.6.3-1.

Request parameter definition:

mH: ushort	**Media Handle** of the TS stream to relocate/retune.
tsLoc: tsSourceType	Location to which to relocate the stream is defined in Table 9.6.2.3.6.2-1.

Response parameter definition:

mH: ushort	**Media Handle** of the TS stream that was relocated.

Detailed semantics:

- In case another network access resource (e.g. tuner/demodulator for broadcast) is required than is currently allocated to the **Media Handle** the **Request** may not be granted by the **ECI Host** for resource constraint reasons.
- On a successful retune any existing filtering and/or descrambling is terminated. Default acquisition shall commence once the TS is acquired.

Table 9.6.2.3.6.3-1: reqDcrTsRelocate Error Codes

Name	Description
ErrDcrTsNetworkAccessCapability	See Table 9.6.2.3.7-1.
ErrDcrTsNetworkAccessResource	
ErrDcrTsNetworkAccessFail	

9.6.2.3.6.4 reqDcrTsSelectPrg Message

C→H reqDcrTsSelectPrg(ushort **mH,** ushort **prgNumber) →**
H→C resDcrTsSelectPrg(ushort **mH)**

- This message sets the program selection for descrambling by the **ECI Host** in the current TS to **prgNumber**.

Request parameter definition:

mH: ushort	**Media Handle** of the TS stream
prgNumber: ushort	Program number in MPEG PAT & PMT tables (see ISO/IEC 13818-1-1 [8]) in the TS defining the service to be selected by the **ECI Host**.

Response parameter definition:

mH: ushort	**Media Handle** of the TS stream

Detailed semantics:

- The **ECI Host** shall locate the PAT in the TS indicated by **mH**. It shall locate the PID of the PMT by matching **prgNumber** to program_number. It shall acquire the PMT from the located PID and use the regular **ECI Host** functions for selecting the components of the program to be rendered. If this is completed successfully the **ECI Host** shall issue a **reqDcrTsDescrStart Request to commence** descrambling of the program.

Post condition request:

1) If the **ECI Host** was descrambling a program not selected by a **reqDcrTsSelectPrg** or **reqDcrTsSelectPmt Request** it shall store the program selection parameters so it can later return to the program on a **reqDcrTsSelectCancel**.

Post condition Response:

1) If no error is returned the **ECI Host** shall subsequently send a **reqDcrTsDescrStart**.

The error codes for the request starting decryption message are given in Table 9.6.2.3.6.4-1-1.

Table 9.6.2.3.6.4-1-: reqDcrSelectPrg Error Codes

Name	Description
ErrDcrTsPrgNumberNotInPsi	See Table 9.6.2.3.7-1.
ErrDcrTsComponentSelectError	

9.6.2.3.6.5 reqDcrTsSelectPmt Message

C→H reqDcrTsSelectPmt(ushort **mH,** uint **pmtLen,** byte **pmt[]) →**
H→C resDcrTsSelectPmt(ushort **mH)**

- This message selects a new program to be descrambled by the **ECI Host** by sending a MPEG PMT table defining the program's components in the transport stream identified by **mH**.

Request parameter definition:

mH: ushort	**Media Handle** of the TS stream.
pmtLen: uint	Number of bytes of the **pmt** parameter.
pmt: byte	private_section containing a PMT table according to ISO/IEC 13818-1-1 [8].

Response parameter definition:

mH: ushort	**Media Handle** of the TS stream.

Detailed semantics:

- This command permits an **ECI Client** to select components in a TS that do not have an appropriate PAT and PMT table. The **ECI Host** shall use **pmt** for selecting the components of the program to be rendered. If this is completed successfully the **ECI Host** shall issue a **reqDcrTsDescrStart Request to commence** descrambling of the program.

Post condition request:

1) If the **ECI Host** was descrambling a program not selected by a **reqDcrTsSelectPrg** or **reqDcrTsSelectPmt Request** then it shall store the program selection parameters so it can later return to the program on a **reqDcrTsSelectCancel**.

Post condition Response:

1) If no error is returned the **ECI Host** shall subsequently send a **reqDcrTsDescrStart**.

The error codes for the request starting decryption message are given in Table 9.6.2.3.6.5-1.

Table 9.6.2.3.6.5-1: reqDcrSelectPmt Error Codes

Name	Description
ErrDcrTsComponentSelectError	See Table 9.6.2.3.7-1.

9.6.2.3.6.6 reqDcrTsSelectCancel Message

C→H reqDcrTsSelectCancel(ushort **mH) →**
H→C resDcrTsSelectCancel(ushort **mH)**

- This message cancels a preceding **reqDcrTsSelectPrg** or **reqDcrTsSelectPmt** by the **ECI Client**, returning to the original program selected by the **ECI Host** in the TS identified by **mH**.

Request parameter definition:

mH: ushort	**Media Handle** of the TS stream.

Response parameter definition:

mH: ushort	**Media Handle** of the TS stream.

Post condition Response:

1) The **ECI Host** may subsequently send a **reqDcrTsDescrStart** to resume descrambling of the original program.

9.6.2.3.7 Error Codes for the Media Session API for TS media

The values are meanings of the API specific errors that can be returned by the **Response** messages for this API are listed below in Table 9.6.2.3.7-1.

All TS specific **Media Handle** requests return an error code for the **Media Handle** parameter in case they are applied on a non-TS **Media Handle**.

Table 9.6.2.3.7-1: Error codes of the media session APIs for TS media

Name	Value	Description
ErrDcrTsUserDelay	-256	Long delay waiting for input from **user** required to complete the operation occurred. Operation not completed.
ErrDcrTsCardMissing	-257	**Smart Card** required for session is not accessible/available.
ErrDcrTsServiceMissing	-258	A service from outside the **CPE** required to support the **ECI Client** in decryption operations is not available.
ErrDcrTsResourceMissing	-259	A undefined resource inside the **CPE** required for accessing or decrypting content is not available.
ErrDcrTsMmiMissing	-260	**ECI Client** access to the MMI is not available.
ErrDcrDescrContinue	-261	**ECI Host** continues to attempt to descramble content in this TS.
ErrDcrTsSectionTimeout	-262	A timeout for acquiring a section occurred.
ErrDcrTsSectionCrcErr	-263	Within timeout period sections were retrieved but with CRC errors. Typically this means the stream is heavily corrupted.
ErrDcrTsNetworkAccessCapability	-264	The ECI Host does not have a network access resource for locating the requested TS.
ErrDcrTsNetworkAccessResource	-265	The ECI Host cannot acquire the network access resource for accessing the requested TS.
ErrDcrTsNetworkAccessFail	-266	The network access resource failed to (reliably) acquire the requested TS.
ErrDcrTsPrgNumberNotInPsi	-267	A PMT with corresponding program number could not be located from the PAT.
ErrDcrTsComponentSelectError	-268	A component in the PMT could not be selected for demultiplexing/descrambling.
ErrDcrTsPidNotDescrambled	-269	A Pid was not selected by **ECI Host** for descrambling.
ErrDcrTsCwIdNotValid	-270	An invalid control word ID was referenced.
RFU	other	Reserved for Future Use.

9.6.2.4 Decrypting file and stream based content

9.6.2.4.1 Introduction

This clause defines an **ECI Client**/**ECI Host** API that permits **CPE** and downloaded applications to interact with a security **ECI Client** through the **ECI Host** so as to descramble content formatted as ISOBMFF [37] or any other files or streams in which the **ECI Host** (or the underlying **CPE** or downloaded application acting through it):

- can extract the required security control data from the file or stream and pass it to the **ECI Client**;
- permit the descrambling keys generated by the **ECI Client** to be correctly applied (synchronized) to the content by means of Key-IDs.

ISOBMFF files [37] are a common packaging format for many non-real-time and adaptive download methods. There is also a common encryption method defined for such file formats: CENC [6]. Also the adaptive streaming format standard MPEG-Dash [7] and [38] is based on ISOBMFF and different (sometimes legacy) DRM systems use their own proprietary ISOBMFF subformat (with signature "brand" identifier).

One section of the API permits the **ECI Client** to specify which data it requires from the ISOBMFF file in order to perform such decoding, thus permitting proprietary (non CENC compliant) DRM applications of ISOBMFF to be used by **CPE** applications. The specifics of sample descrambling should be managed by the **ECI Host**: i.e. either be CENC compliant or require proprietary extensions in the **ECI Host**.

The API has the following sections:

1) Starting and stopping descrambling.
2) Setting of **ECI Client** specific security data acquisition filters.
3) Decryption Key (control word) API.

9.6.2.4.2 Applicable specifications

ISOBMFF files as referred in this clause shall be compliant to ETSI TS 103 285 [38]. CENC compliant ISOBMFF files (as required for standard deciphering) shall be compliant to ISO/IEC 23001-7 [6].

Dash compliant streaming data shall comply with ISO/IEC 23009-1 [7]. **ECI Hosts** implementing Dash shall (at minimum) comply with ISO/IEC 23001-7 [6], ISO/IEC 23001-9 [37], and ETSI TS 103 285 [38] in as far as applicable to the functional scope of the **CPE**.

9.6.2.4.3 ECI Host Processing Requirements

9.6.2.4.3.1 Decryption System Identification detection

The **ECI Host** shall be able to acquire the list of applicable decryption systems from the content container based on the following rules:

1) For all ISOBMFF and MP4 files the **ECI Host** shall acquire the File Type Box ('ftyp') and Segment Type Box ('styp') and shall use the `major_brand` field and the `compatible_brands[]` field for matching content to **ECI Clients**.

2) For ISOBMFF CENC encoded files the **ECI Host** shall recover the Protection System Specific Header Boxes ('pssh') from any of the possible locations (see ISO/IEC 23001-7 [6]) and collect from the `SystemID` field the UUIDs of the DRM systems suitable for decrypting the content. These files can be recognized by a Protection Scheme Information Box ('sinf') containing Scheme Type Box ('schm') with `scheme_type` field equal to 'cenc' or 'cbc1' and major version of the `scheme_version` field set to 0x0001. The definition and location of the 'sinf' boxes is defined in ISO/IEC 23001-7 [6].

3) For MPEG-Dash content the **ECI Host** shall acquire all ContentProtection descriptors in the MPD containing a specific UUID (starting with "urn:uuid:xxxxx", with xxxxx the UUID) for @SchemeIdUri attribute for the purpose of matching to **ECI Client** DRM UUIDs or containing a Conditional Access system ID in accordance with ETSI TS 103 285 [38] in the @value attribute (see [i.4] for the definition of this generic identifier). The **ECI Host** shall acquire all ContentProtection descriptors for matching to the **ECI Client** capabilities. It shall convert any PSSH boxes there included to corresponding ISOBMFF binary representation.

The matching process for content to **ECI Clients** is described in clause 9.6.2.4.5.2.1.

9.6.2.4.3.2 Scrambling type detection

ECI Hosts shall signal the applicable descrambling mode to the **ECI Client** based on the following rules:

1) For ISOBMFF CENC encoded files it shall be able to apply the rules as defined in ISO/IEC 23001-7 [6] for detection of the cipher (AES-CTR or AES-CBC) including clear/scrambled byte selection, padding, and initialization vector extraction and application is defined in ISO/IEC 23001-7 [6].

2) For MPEG DASH content in ISOBMFF format AES-CTR (with key rotation) shall be applied for descrambling as defined in ETSI TS 103 285 [38].

9.6.2.4.3.3 Default content container security data filtering

The **ECI Host** shall pass any boxes containing (opaque) information in the container designated for the **ECI Client** at the time this is relevant for the process of descrambling. This specifically means that:

1) For ISOBMFF CENC encoded files and Dash content in ISOBMFF format:

   a) Protection System Specific Header boxes in 'moov' and 'moof' boxes matching the UUID of the DRM system ID of the **ECI Client**, relevant to the being decoded now or in the near future.

   b) Protection Scheme Information Boxes 'sinf' - in case the **ECI Client** requires access to 'sinf' boxes.

9.6.2.4.3.4 Descrambling of Content

The **ECI Host** shall be responsible for interpreting the scrambling mode, identifying the data to be descrambled and processing the data using the descrambler using the appropriate Key-IDs to identify the keys as made available by the **ECI Client**.

In order for the **ECI Client** to compute the associated keys the **ECI Host** shall pass the required security control data from the content container to the **ECI Client** in a timely fashion.

9.6.2.4.4 Media Session API for File-based and streamed media

9.6.2.4.4.1 General

The **ECI Host** can start the decrypting content on an open **Media Handle** using the **ECI Client** resources reserved. The **ECI Host** shall provide initialization data for the **ECI Client** to start evaluating the access rights.

Table 9.6.2.4.4.1-1: Media Handle TS content decryption API

Message	Type	Dir.	Tag	Description
reqDcrFileStart	A	H→C	0x01	**Request**s **ECI Client** to descramble or return the descramble status of a file or stream.
reqDcrFileStop	A	H→C	0x02	**ECI Host** requests **ECI Client** to stop the key processing for the descrambling operation for a **Media Handle**.
reqDcrFileQuit	A	C→H	0x03	**ECI Client** cancels a descrambling operation with the **ECI Host**.

9.6.2.4.4.2 reqDcrFileStart Message

H→C reqDcrFileStart(ushort **mH,** uchar **reqType,** uchar **dataType,** uint **initDataLen,** byte **initData[]) → C→H resDcrFileStart(**ushort **mH,** uchar **dcrStat)**

- This message requests the **ECI Client** to return the descrambling status of and/or start a descrambling session on the content associated with mH. The **ECI Host** supplies initial data for the **ECI Client** to commence with any license acquisition and evaluation in accordance with the container/encryption format.

Request parameter definition:

mH: ushort	**Media Handle** of the File.
reqType: uchar	Type of **Request** (descrambling commencing or license inquiry) is defined in Table 9.6.2.4.4.2-1.
dataType: uchar	Type of InitData.
initDataLen: uint	Length in bytes of the initData container.
initData: byte	The initialization data from the content as defined by dataType. He coding of initDat is defined in Table 9.6.2.4.2-2.

Table 9.6.2.4.4.2-1: reqType encoding

Name	Value	Description
ReqTypeDcr	0x01	Descramble start; enter in dialogue with user if required.
ReqTypeInq	0x02	Inquiry to descrambling options.
RFU	Other	Reserved for future use.

Table 9.6.2.4.4.2-2: initData coding

dataType	Value	iniData
FmtIsoCenc	0x04	ISOBMFF PSSH boxes (see ISO/IEC 23001-7 [6]) encountered matching the DRM ID in the MatchSpecifier of the **ECI Client**.
FmtIsoCencDash	0x05	ISOBMFF PSSH boxes (see ETSI GS ECI 001-5-2 [5]) encountered in MPD (see ETSI GS ECI 001-5-2 [5]) or Initialization segment (see ISO/IEC 23009-1 [7]) encountered matching the DRM ID in the MatchSpecifier of the **ECI Client**.
FmtIsoProp	0x06	The **ECI Host** may pass data to the **ECI Client** based on proprietary knowledge. The **ECI Client** shall be able to interpret this data based on the same common proprietary knowledge.
FmtIsoPropDash	0x07	As FmtIsoProp including.
RFU	Other	Reserved for Future Use.

Response parameter definition:

mH: ushort	**Media Handle** of the TS stream.
dcrStat: uchar	Status of descrambling; see Table 9.6.2.4.4.2-3.

Table 9.6.2.4.4.2-3: Descrambling status

Name	Value	Description
DcrStatNo	0x00	No descrambling possible (DRM system does have the capability of descrambling).
DcrStatOk	0x01	Descramble start; enter in dialogue with user if required.
DcrStatDialog	0x02	Dialog with user required.
DcrStatPay	0x03	Payment required, possibly also user dialog.
DcrStatDrmNok	0xFE	DRM system does not have the capability of descrambling this content.
RFU	Other	Reserved for future use.

Detailed semantics:

- On inquiries any **user** dialogs will not be started by the **ECI Client** but the **ECI Client** shall evaluate the ability to descramble the content by clearing the license conditions with the license server without **user** dialogue.

Pre conditions Request:

1) Media Handle pending.

Pre condition Response:

1) If **ECI Client** can descramble content and reqType is OK the **ECI Client** shall be ready to generate descrambling keys.

The error codes for the request starting decryption message are given in Table 9.6.2.4.4.2-4.

Table 9.6.2.4.4.2-4: reqDcrFileOpen Error Codes

Name	Description
ErrDcrFileUserDelay	See Table 9.6.2.4.7-1.
ErrDcrFileCardMissing	
ErrDcrFileServiceMissing	
ErrDcrFileResourceMissing	
ErrDcrFileMmiMissing	

9.6.2.4.4.3 reqDcrFileStop Message

H→C reqDcrFile Stop(ushort **mH)** →
C→H resDcrFile Stop(ushort **mH)**

- This message allows the **ECI Host** to stop the file decryption operation.

Request parameter definition:

mH: ushort	**Media Handle** of the File

Response parameter definition:

mH: ushort	**Media Handle** of the File

Pre conditions Response:

1) **ECI Client** has terminated any operations related to decrypting content.

9.6.2.4.4.4 reqDcrFileQuit Message

C→H reqDcrFileQuit(ushort **mH,** uint **reason)** →
H→C resDcrFile Quit(ushort **mH)**

- This message allows the **ECI Client** to inform the **ECI Host** that it has terminated the key processing for a file decryption operation. Related error codes are defined in Table 9.6.2.4.4.4-1.

Request parameter definition:

mH: ushort	**Media Handle** of TS stream.
reason: uint	Values as defined in Table 9.7.2.5.9-1.

Response parameter definition:

mH: ushort	**Media Handle** of file.

Pre conditions Response:

1) All **ECI Host** activities related to descrambling **mH** has terminated or an error is returned.

Post conditions Response:

1) All **ECI Client** activity related to **mH** shall terminate immediately or an error shall be returned.

Table 9.6.2.4.4.4-1: reqDcrMhCancel Error Codes

Name	Description
ErrDcrFileDescrContinue	See Table 9.6.2.4.7-1.

9.6.2.4.5 ECI Client Specific Security Data Acquisition

9.6.2.4.5.1 General

The **ECI Host** shall perform standard data acquisition on the data to be decoded for information required by the **ECI Client** to perform key computation. The **ECI Client** may indicate specific data acquisition beyond standard data provided by the **ECI Host**. The **ECI Host** shall maintain a limited number of filters for acquisition of such data.

Table 9.6.2.4.5.1-1: Data Filter API

reqDcrFileFilter	req	C→H	0x04	**ECI Client** requests the **ECI Host** to set a data filter for security data aquisistion.
reqDcrFileData	A	C→H	0x05	**ECI Client** requests **ECI Host** to acquire data via the File Filter.

9.6.2.4.5.2 File Filter specification

9.6.2.4.5.2.1 Generic File Filter Definition

The file data filter specification is based on an underlying specification of the file format. Within the context of a defined file format a filter is defined. The generic File Filter specification is defined in Table 9.6.2.4.5.2.1-1.

Table 9.6.2.4.5.2.1-1: Generic File Filter specification

```
typedef struct dcrFileFilterSpec {
   ushort filterType;        // is defined in Table 9.6.2.4.5.2.1-3
   ushort filterLen;
   byte   filter[filterLen]; //  shall be formatted according to filterType
} dcrFileFilterSpec;
```

Table 9.6.2.4.5.2.1-2: File Filter types

FileFilterIsobmff	0x0001	File filter for ISMBMFF formatted data is defined in clause 9.6.2.4.5.2.2.
RFU	Other	Reserved for future use.

9.6.2.4.5.2.2 ISOBMFF specific File Filter Definition

The filter specification for ISOBMFF formatted files is defined in Table 9.6.2.4.5.2.2-1.

Table 9.6.2.4.5.2.2-1: ISOBMFF File Filter specification

```
#define MaxFilterFile 16 // maximum number of bytes in box that are filtered
#define MaxContainers  4 // maximum number of container boxes for a box
#define MaxUuidLen    16 // Length in bytes of a UUID

typedef struct BoxSpec {
   uint   boxType                  // 4CC code of box type
   byte   extendedType[MaxUuidLen]
                                   // UUID for boxType=='uuid', otherwise no
significance
   byte   filter[MaxFileFilter];      // shall match bytes of box following
   byte   filterMask[MaxFilter];
   ushort dataLen;             // maximum amount of box data to be acquired
} BoxSpec;

typedef struct dcrFileFilterIsobmff {
   BoxSpec    container[MaxContainer];
   BoxSpec    box;
} dcrFileFilterIsobmff;

bool function boxMatch
       (byte *boxData, byte *filter, byte*filterMask; int boxLen) {
{
   bool match = true;
   int i;
```

```
   for( i=0; i<MaxFilterFile && i<boxLen && match; i++) {
       match &&= (boxData[i] & filterMask[i] == filter & filterMask[i]) ;
   }
   return match;
}
```

The **ECI Host** shall parse the file and shall acquire boxes that match the **box** field and that are contained in boxes that match of any of the **container** array. The **ECI Host** shall skip scanning boxes not defined in ISO/IEC 14496-12 [39] or ISO/IEC 23001-7 [6].

The **boxType** in the **container** field of **dcrFileFilterIsobmff** may be set to '****' to indicate a wildcard. In that case the other fields of **container** shall have no significance, and set to 0 to indicate no match.

The **filter** and **mask** field in **BoxSpec** shall be matched to the first bytes after the type field of a box to be processed. For full boxes" (see ISO/IEC 14496-12 [39]) this is the version and flag field. The match shall be done according to the boxMatch function, with the **boxLen** parameter set to the number of bytes following the boxtype and extended_type of the box, **boxData** parameter to the start of these bytes, **filter** parameter to the **boxSpec.filter** field and the **filterMask** parameter set to the **boxSpec.filterMask** field.

The data returned by the filter are the boxes (in sequence) that match the filter as the file is parsed by the **ECI Host**. The **ECI Host** may cluster the boxes as is convenient but should not delay passing boxes to the **ECI Client** unnecessarily since this may prevent the **ECI Client** from generating required descrambling keys.

9.6.2.4.5.2.3 reqDcrFileFilter Message

C→H setDrcFileFilter(ushort **mH,** uchar **filterNr,** dcrFilleFilterSpec ***dataFilter)**

- This message requests the **ECI Host** to set a data filter on basis of the dataFilter for the acquisition of security data for the **ECI Client**.

Parameter definition:

mH: ushort	**Media Handle** of TS stream.
filterNr: uchar	Number of the File filter in the **ECI Host**.
dataFilter: dcrFilleFilterSpec *	The filter specification for data extraction.

Post condition Request:

- This section filter shall be put into effect by the **ECI Host** until a **reqDcrFileEnd** or **reqDcrFileCancel** is effected or a reqDcrFileFilter is set with **dataFilter == NULL.**

9.6.2.4.5.2.4 reqDcrFileAcqData Message

H→C reqDcrFileAcqData(ushort **mH,** uchar **filterNr,** uint **dataLen**, byte **data[]) →**
C→H resDcrFileAcqData (ushort **mH,** uchar **filterNr)**

- This message requests the **ECI-Host** to acquire and to send one of more sections in the context of the media file or stream identified by **mH** and the filter identified by **filterNr** to the **ECI Client**.

Request parameter definition:

mH: ushort	**Media Handle** of the file on which to set the default section filter.
filterNr: uchar	Number of the filter to be programmed. The value shall be between 0 and 7.
dataLen: uint	Number of bytes in **data**.
data[]: byte	Sequences of private_sections (bytes in network order) are defined in ISO/IEC 13818-1-1 [8], section 2.4.4.11. Any section with a CRC error is not passed to the **ECI Client**.

Response parameter definition:

mH: ushort	**Media Handle** of media file or stream.
filterNr: uchar	Number of the filter that was programmed.

The related error codes are listed in Table 9.6.2.4.5.2.4-1.

Table 9.6.2.4.5.2.4-1: reqDerFileAcqData Error Codes

Name	Description
ErrDcrAcqDataTimeout	See Table 9.6.2.4.7-1.
ErrDcrAcqDataDataErr	

9.6.2.4.6 File descrambling control word API

9.6.2.4.6.1 General

The content descrambling API section permits key to be made available for descrambling by the **ECI Client**. The **ECI Host** has to first initiate the availability of a control word by passing the Key-ID to the **ECI Client**. Once the key is available the **ECI Host** can apply the computed control word to the (encrypted) content. The API messages related to the Media Handle File content descrambling API are listed in Table 9.6.2.4.6.1-1.

Table 9.6.2.4.6.1-1: Media Handle File content descrambling API

Message	Type	Dir.	Tag	Description
reqDcrFileKeyComp	A	H→C	0x20	Initiate any required computing or other activity of the **ECI Client** to make a control word with Key-ID available.

9.6.2.4.6.2 **ECI Host** Processing Requirements

9.6.2.4.6.2.1 ISOBMFF CENC format content

This clause defined **ECI Host** processing requirements for descrambling content in ISOBMFF + CENC format.

The **ECI Host** has the responsibility for timely passing any KeyID information to the **ECI Client** so that the **ECI Client** can derive/acquire the required control word in a timely fashion. Other constraints permitting this should be at least 30 seconds ahead of anticipated use of the control word.

The Key-ID information is contained in several boxes associated with the media samples (sequences of (partially) encrypted media data): see e.g. [i.3], clause 5.4. The data in these boxes permit the extraction of Key-IDs, IVs and permit the identification of clear and encrypted data in media samples.

9.6.2.4.6.2.2 MPEG DASH format content

The details of the MPEG DASH formats the **ECI Host** has to support are currently not covered in the **ECI** specifications.

9.6.2.4.6.3 reqDcrFileKeyComp Message

H→C reqDcrFileKeyComp(ushort **mh,** byte **keyId**[MaxUuidLen]**)** →
C→H resDcrFileKeyComp(ushort **mH)**

- This message initiates the computation and any other activity required by the **ECI Client** to compute a control word identified by KeyId and make it available for decrypting content.

Request parameter definition:

mH: ushort	**Media Handle** of the TS stream.
keyId[MaxUuidLen]**:** byte	KeyID as a UUID in network byte order.

Response parameter definition:

mH: ushort	**Media Handle** of the TS stream.

Pre condition Response:

1) The key is available or an error or timeout occurred.

Detailed semantics:

- The **ECI Client** shall report an error in case the requested control word cannot be made available in a timely fashion (60 seconds). **ECI Clients** may continue trying to acquire the requested key even after an error is reported.
- On a reported error the **ECI Host** may re-issue the **Request**. **ECI Hosts** may issue a maximum number of 10 **Requests**.

The related error codes are listed in Table 9.6.2.4.6.3-1.

Table 9.6.2.4.6.3-1: reqDcrFileKeyComp Error Codes

Name	Description
ErrDcrFileUserDelay	See Table 9.6.2.4.7-1.
ErrDcrFileCardMissing	
ErrDcrFileServiceMissing	
ErrDcrFileResourceMissing	
ErrDcrFileMmiMissing	
ErrDcrFileKeyIdUnknown	
ErrDcrFileKeyOverflow	

9.6.2.4.7 Error Codes for the Decrypting file and stream based content API

The values are meanings of the API specific errors that can be returned by the **Response** messages for this API are listed in Table 9.6.2.4.7-1.

All file specific **Media Handle** requests return an error code for the **Media Handle** parameter in case they are applied on a non-file **Media Handle**.

Table 9.6.2.4.7-1: Error codes for media session APIs for file and stream media

Name	Value	Description
ErrDcrFileUserDelay	-256	Long delay waiting for input from user required to complete the operation occurred. Operation not completed.
ErrDcrFileCardMissing	-257	Smart Card required for session is not accessible/available.
ErrDcrFileServiceMissing	-258	A service from outside the CPE (e.g. DRM server) required to support the ECI Client in decryption operations is not available.
ErrDcrFileResourceMissing	-259	An undefined resource inside the CPE required for accessing or decrypting content is not available.
ErrDcrFileMmiMissing	-260	ECI Client access to the MMI is not available.
ErrDcrFileDescrContinue	-261	ECI Host continues to attempt to descramble content in this File.
ErrDcrAcqDataTimeout	-262	A timeout for acquiring a data occurred.
ErrDcrAcqDataDataErr	-263	Within timeout period sections were retrieved but with errors. Typically this means the file is corrupted or does not comply with the applicable specifications.
ErrDcrFileKeyIdUnknown	-300	keyId unknown to ECI Client/security system for this content.
ErrDcrFileKeyOverflow	-301	Too many Key-ID Requests in a short period; await ECI Client Responses to previous processing Requests.
ErrDcrFileKeyWithdrawn	-302	Key no longer available; rights withdrawn by ECI Client.

9.7 APIs for access to the ECI Host re-encryption resources

9.7.1 Introduction to the re-encryption APIS

9.7.1.1 List of APIs defined in clause 9.7

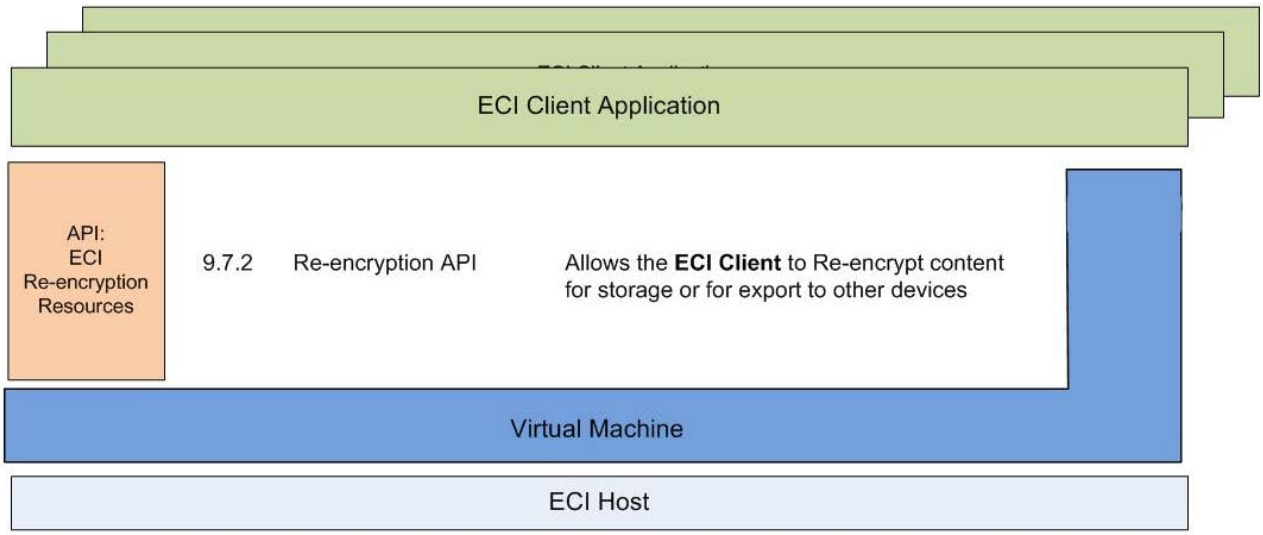


Figure 9.7.1-1: Block diagram of the APIs defined in clause 9.7

Table 9.7.1-1 lists the APIs covered in clause 9.7 and Figure 9.7.1-1 illustrates the location of the APIs defined in clause 9.7 with the **ECI architecture**.

Table 9.7.1-1: List of APIs defined in clause 9.7

Clause	API name	Description
9.7.2.3	**Export Connection** API	Allows the **ECI Client** to establish an **Export Connection** for imported content.
9.7.2.5	**Import Connection** API	Allows the **ECI Client** to import content, which was delivered encrypted via an access network and decrypted under control of an **ECI Client**.
9.7.2.6	**Micro Client** De-encryption API	Allows the **ECI Client** to decrypt imported and re-encrypted content.

9.7.1.2 General concept of re-encryption

Re-encryption in **ECI** allows an independent **Micro DRM System** to protect the content that is delivered by a CA or DRM **ECI Client** for further applications with our outside the **CPE**. The re-encryption system in an **ECI** compliant implementation is called a **Micro DRM System**. The applications of a **Micro DRM System** can be e.g. time shifting, PVR and streaming. The re-encryption **ECI Client** is called a **Micro Server**. The client, be it **ECI** or non-**ECI** compliant, that can decrypt the re-encrypted content, is called the **Micro Client**. The Client image and credentials for re-encryption can be downloaded as a regular **ECI Client**, provisioned by a Micro DRM master server. Figure 9.7.1.2-1 shows the overall system (excluding the micro DRM master server). In case of local storage the **Micro Server** and the **Micro Client** are implemented in a single device.

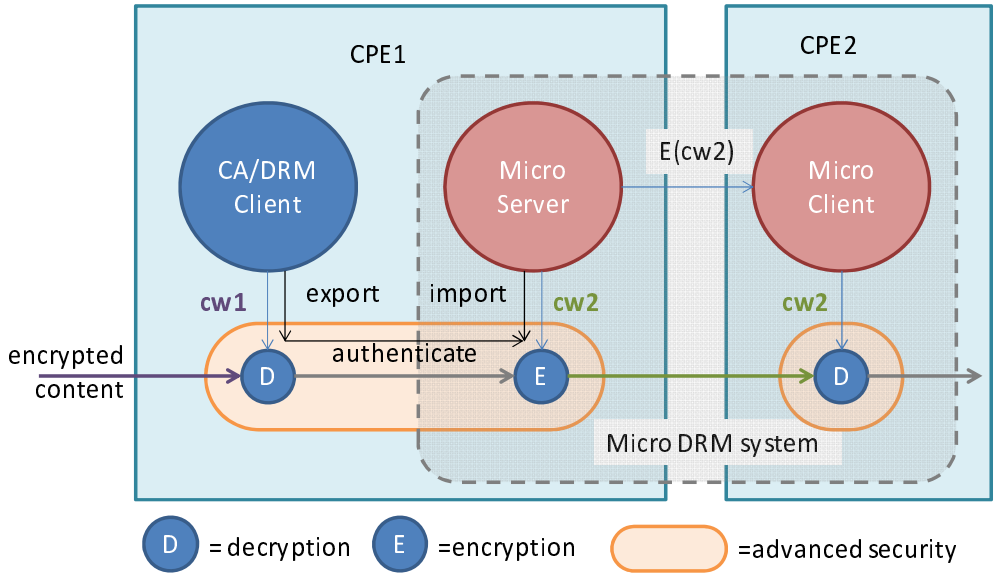


Figure 9.7.1.2-1: Micro DRM system diagram

The CA/DRM **ECI Client** initially decrypting the content can control whether it is permitted to export the content to the installed **Micro DRM Systems**. It authenticates the Micro Server for that purpose through the **Advanced Security** system; authentication being under the control of the CA/DRM Operator. Once the content is exported, the **Micro DRM System** has the responsibility for protection of the content. Decryption, re-encryption and the authentication for export is securely supported by the **Advanced Security** system. The principles are illustrated in Figure 9.7.1.2-1.

9.7.1.3 Re-encryption API structure overview

Figure 9.7.1.3-1 shows a more detailed diagram illustrating the role of the different APIs involved in re-encryption. The **ECI Host** provides the decoding **ECI Client** with all required information through the decryption API. The decoding **ECI Client** securely establishes the control word for decryption of the content through the Advanced Security API. The essential content properties (marks) are authenticated. The Export API permits **ECI Host** request the decoding **ECI Client** to establish an **Export Connection** to the desired Micro Server for re-encryption. The Advanced Security API allows the exporting **ECI Client** to authenticate the importing **Micro Server**. The **ECI Host** uses the Import API to establish the authorized **Export Connection** to a **Micro Server**. The re-encryption API permits the **ECI Host** to direct the **Micro Server** to a mode of operation corresponding to the content packaging format and the application (streaming, time-shifting or storage) and encrypt the content for the desired (authenticated) target **Micro Client**.

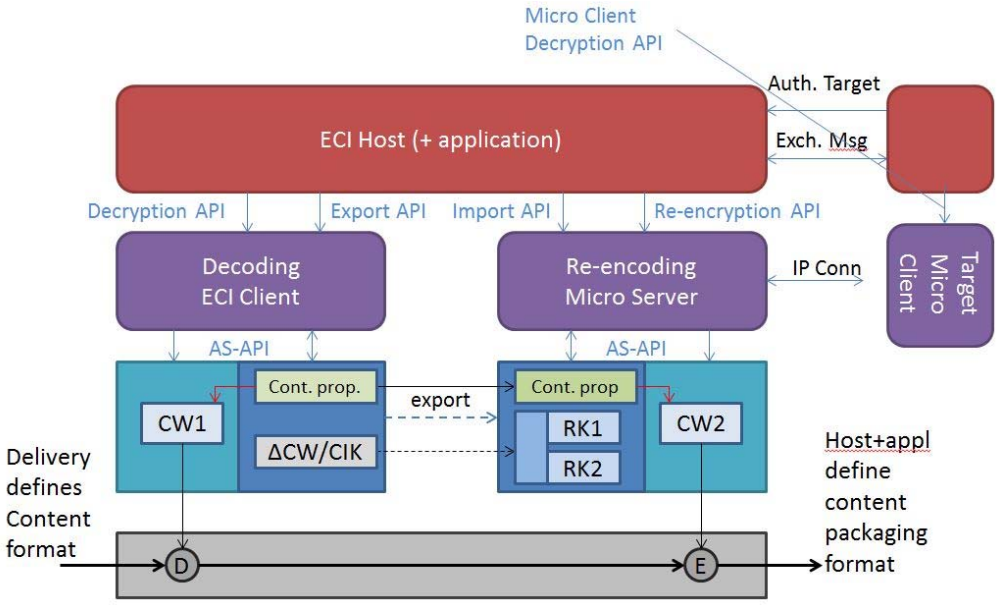


Figure 9.7.1.3-1: Architecture of the decryption and re-encryption functionalities

The scheme in Figure 9.7.1.3-1 and Figure 9.7.1.3-2 provide an overview of the main messages in the decryption, export control, import control, re-encryption and the Micro Client decryption APIs. It displays the content flowing from left to right: from a first CA/DRM delivery **ECI Client** over an **Export**/**Import Connection** to a Micro Server which encrypts the decrypted content to be finally decoded by a target Micro Client.

The four host-client APIs support the following processing steps:

- The *discovery phase* enables **ECI Clients** to publish their potential interworking options to the **ECI Hos**t (in collaboration with the application). This enables the **ECI Host** to match the requested content to a certain **ECI Client**. In cases the chosen **ECI Client** does not possess the appropriate rights to process this content; the **ECI Host** has to look for other **ECI Clients**. In home networking and distributed PVR applications this can involve application protocols like DLNA see [i.8].The *authentication step* permits the **ECI Host** to establish an authenticated connection between the desired **ECI Client** and the **Micro Server** or **the Micro Server and the Micro Client**. Authentications can be implicit: i.e. the cryptography proof for authentication can be embodied in the ability for the **ECI Client** to finally decrypt the content. Authentication always is always following the flow of the content. In some cases a reverse agreement is required. For business purposes an **Import Connection** may have to be approved by the **Micro Server**.

- The *session instantiation step* permits the **ECI Host** to reserve all resources needed to decrypt or encrypt content in a certain mode of operation associated with a **Media Handle**. The import and target connections are defined for reqEncrMhOpen on a **Micro Server**, or are implied in a regular CA/DRM **ECI Client**. Note that the **ECI Host** is responsible for allocating any complementary resources, like (de)scrambling, de-multiplexing and decoding processing resources for a total media application scenario to be able to proceed. The **ECI Client** ultimately requests the assignment of AS and decryption or encryption resources using the **Advanced Security** API.

- The *Session Control step* permits the **ECI Host** to start and stop the processing of content on **Media Handles**. For seamless processing of content on a path it is required to start the **ECI Clients** from destination to source: i.e. an **ECI Client** should be ready to process the content if it is presented as such.

Protocol Phase	Ca/DRM delivery Client		Micro Server		Micro Client
	Host->C	C<-Host	Host->C	C<-Host	Host->C
API:	***Decryption***	***Export Control***	***Import Control***	***Re-encryption***	***uC Decryption***
Discovery	setDcrMhMatch	reqExpConnNodes	reqImpConnNodes reqImpConnChain	reqEncrTargets	reqDcrTargets reqDcrTargetCred
Authentcation	*(provisioning procedure)*	reqExpConnSetup reqExpConnDrop *reqExpConnCancel*	reqImpConnSetup reqImpConnDrop *reqImpConnCancel*	reqEncrConnSetup reqEncrConnDrop reqEncrConnCancel	
Session Instantiation	reqDcrMhOpen reqDcrMhClose reqDcrMhCancel	reqExpMhOpen reqExpMhClose reqExpMhCancel	*(performed by re-encryption msg.)*	reqEncrMhOpen reqEncrMhClose reqEncrMhCancel	reqDcrMhOpen reqDcrMhClose reqDcrMhCancel
Session Control	reqDcrTsStart reqDcrTsStop reqDcrTsQuit			reqEncrMhStart reqEncrMhStop reqEncrMhQuit	reqDcrTsStart reqDcrTsStop reqDcrTsQuit
	reqDcrFileStart reqDcrFileStop reqDcrFileQuit				reqDcrFileStart reqDcrFileStop reqDcrFileQuit

Figure 9.7.1.3-2: En-/Decryption and Import/Export API overview

The messages use a certain systematic in their naming and semantics:

- The *discovery step* permits the **ECI Client** to publish its capabilities for connecting to another **ECI Client** or content. The messages setDcrMhMatch, reqExpConnNodes, reqImpConnNodes, reqEncrTargets, reqDcrTargets request the **ECI Client** to publish these (in the form of identities).
- The a*uthentication step* uses *setup*, *drop* and *cancel* messages for the creation of an (authenticated) connection, the de-allocation of a previous connection or the cancelling of such a connection by the **ECI Client**. The reference for a connection is an **Export Connection** (**ECI Client** exporting content), **Import Connection** (**ECI Client** importing content) or a target connection (**Micro Server** encrypting content for subsequent decryption by a target and vice versa e.g. a **Micro Client** decrypting content from a **Micro Server**).
- The *session instantiation step* uses *open*, *close* and *cancel* for the creation and termination of the sessions, all referring to a **Media Handle** as the common reference. Also MMI sessions and **Smart Card** resource management required by the **ECI Client** can refer to the **Media Handle** to permit the **ECI Host** to associate a **user** dialogue request within the context of its application.
- The *session control step* defines different messages for the decryption of two specific content formats: transport streams and file format. Processing can be *started*, *stopped* by the **ECI Host** and the **ECI Client** can *quit* processing in case of a lack of resources or a rights issue.

NOTE 1: For some protection systems it may not be necessary to perform significant processing for all phases. Their **ECI Clients** may perform just minor administrative processing for some of the messages.

NOTE 2: The nature of **ECI Clients** on an **Import/Export Connection** is different from the relation between a **Micro Server** and a **Micro Client**. On the **Import/Export Connection** with **ECI Clients** share the **ECI Host** and can exchange content through the AS export mechanism using **ECI** defined import/export **Certificate Chain**s. **Micro Server** and **Micro Client** can use a protocol of choice (characteristic for the micro DRM System) for establishing connections as long as it fits in the API framework and can use the **AS System** to establish authentication and common keys. Exchange of content on an **Export/Import Connection** is implicit (defined by the **ECI Host**); the authenticity (for export purposes) of the **Micro Server** will be validated by the **AS system**. Exchange of content between a **Micro Server** and **Micro Client** requires a **Media Handle** session and session control at both **Micro Server** and **Micro Client**.

9.7.2 ECI Export Control API

9.7.2.1 Introduction

ECI permits **ECI Clients** to export decoded content to **Micro Client** that will ensure for re-encryption for the purpose of (permitted) redistribution to other devices or (permitted) storage of the content for later playback. For this purpose **ECI** defines a **Certificate** structure that defines groups of permitted export **Micro DRM Systems**. Each content item decoded is accompanied by the identification of the appropriate **Export Group**. From the **Export Group** there has to be a chain of **Certificates** authorizing export to the selected **Micro Server**. The chain is processed by the Advanced Security System so as to provide a highly robust export authorization mechanism.

The exporting **ECI Client** is responsible for providing the **Export Group Certificates** and all direct descendants. The importing **Micro Client** is responsible for providing the complementary credential information to permit completion of the chain from exporting to importing **ECI Client**.

The **ECI Host** can set up a re-encryption connection from a decrypting **ECI Client** to an encrypting **Micro Client** (**Micro Server**). Once the connection is established the **ECI Host** can proceed with decrypting and re-encrypting content using Media **Handle** sessions. The **AS System** will ensure secure passing of content and associated protection information from the decoding **ECI Client** to the **Micro Client** based on the provided credentials through the **AS System**.

The **ECI Hosts** provides support for **ECI Clients** to access network services to receive up to date credentials for export and import, e.g. through the data Carousel API (clause 9.5.4) and the IP HTTP API (clause 9.4.4.6).

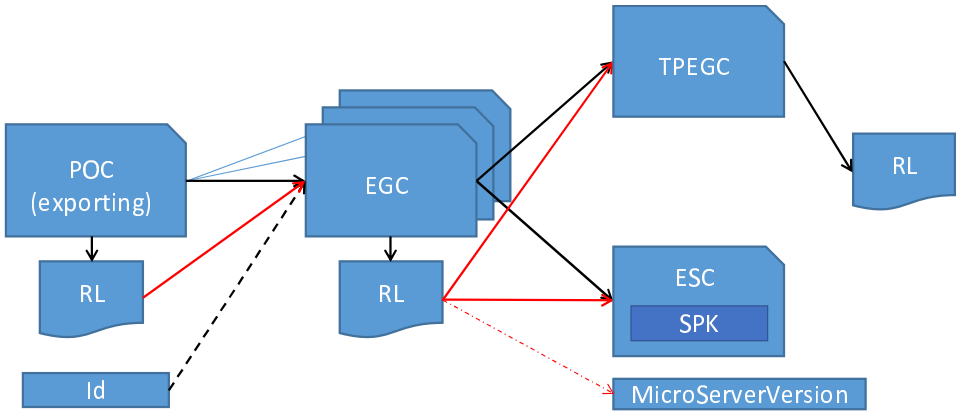
For targeting of re-encryption the **ECI Host** and application have to establish the authorized **Micro Clients** that are enabled to decode the content. This can be both an individual **CPE** (with a suitable Client) as well as a group (based on a shared key). The **ECI Host** then establishes an authorized connection between **Micro Server** and its matching **Micro Client** (one for each **Micro Client**). For time-shifting and recording applications the information needed by the **ECI Client** to be able to decode the content later can be stored (e.g. along with the re-encrypted content). For real-time streaming connections the session control messages needed by **Micro Server** and **Micro Client** can be passed either through the **ECI Host** of these two **Micro Clients** or may be communicated directly between the **Micro Clients** through an IP connection.

NOTE: The communication protocols and associated security aspects for **ECI Client** to **ECI Client** communication are out of scope for **ECI**.

9.7.2.2 Export Certificate Structures

9.7.2.2.1 Overall structure

The **ECI** export mechanism is based on **Certificates**. Most **Certificates** have an associated **Revocation List** to permit updates of the export permissions. Figure 9.7.2.2.1-1 presents the **Certificate** structure for immediate export control of a decoding **ECI Client**.



POC = Platform Operation Certificate
RL = Revocation List
Id = export group ID (defined by content rights)
SPK = Sender Public Key
EGC = Export Group Certificate
TPEGC = Third Party Export Group Certificate
ESC = Export System Certificate

Figure 9.7.2.2.1-1: ECI Certificate distribution structure

The **ECI Client Platform Operation** Certificate (POC) is the **Father** of the **Export Group** Certificates. The **ECI** POC has a special revocation list to permit the **ECI Client** to control the **Export Group** Certificate and their associated revocation list versions. Each **Export Group** Certificate is the **Father** of the actual Export Certificates or a further (descendant) **Export Group**. There are two types of Export Certificates:

1) An Export System Certificate (ESC) identifies the permitted export **Micro Server** by means of its **Sender Public Key**, permitting an immediate authentication. In addition the revocation-list version number of the ESC is used to define a minimum version number for the **Micro Server**.

2) A Third Party **Export Group** Certificate (TPEGC) refers to an **Export Group Certificate** managed by another organization. This permits larger heterogeneous groups of **Micro DRM Systems** to be authenticated with a single export **Certificate**.

The third party group export **Certificate** structure is further illustrated in Figure 9.7.2.2.1-2.

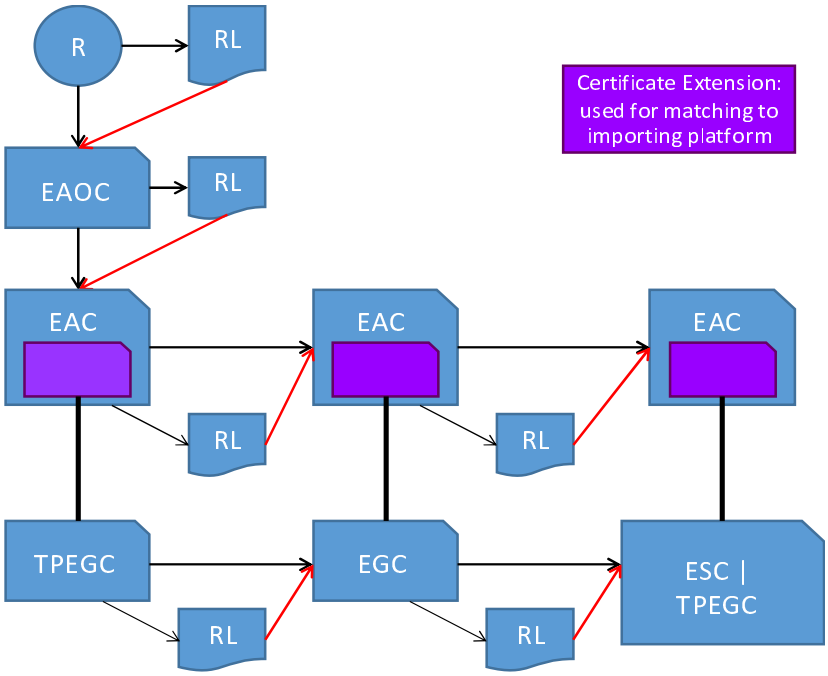


Figure 9.7.2.2.1-2: Third party group export Certificate structure

The **ECI Root Certificate** is the **Father** of an Export Authorization Operator Certificate (EAOC). The **ECI Root Certificate** maintains a special revocation list for such **Certificates**. The Export Authorization Operator Certificate (EAOC) is the **Father** of an Export Authorization Certificate (EAC). This **Certificate** is matched to a Third Part **Export Group** Certificate (TPEGC). Through this mechanism there is a dual authentication of a third party group in order to provide additional security.

A Third Party **Export Group** Certificate is the **Father** of either:

1) An **Export Group** Certificate (EGC), which itself can be the **Father** of another EGC or of any of the **Certificates** listed below. Each EGC has an associated **Revocation List**.

2) An Export System Certificate (ESC).

3) A (next) Third Party **Export Group** Certificate (TPEGC).

Each **Certificate** is additionally verified by a matching Export Authorization Certificate (EAC), which form a tree matching the TPEGC/EGC tree.

Table 9.7.2.2.1-1 provides an overview of the **Certificates** and their **Fathers**.

Table 9.7.2.2.1-1: Summary of the different Export certificates

Certificate Name	Abbr.	Description	Father
Export Group	EGC	This **Certificate** permits exporting **ECI Clients** to authenticate a set (group) of **Micro DRM Servers** and/or third party authenticated groups to which they permit export. The applicable **Export Group** is defined as part of an authenticated rights-attribute to the content.	POC, TPEGC, EGC
Third Party **Export Group**	TPEGC	A **Certificate** for authenticating a group of **Micro DRM Systems m**anaged by another (third) party.	EGC, TPEGC
Export Authorization Operator	EAOC	A **Certificate** that provides the basis for an operator that provides an authorization service for third party **Export Group**s. The **Certificate** is the **Father** of export authorization **Certificate** trees for third party **Export Group**s that it co-authenticates.	**ECI** Root
Export Authorization	EAC	This **Certificates** provides a co-authentication of a Third Party **Export Group** Certificate or an **Export Group** Certificate managed by a third party.	EAC, EAOC
Export System	ESC	This **Certificate** authenticates the **Platform Operation** Certificate of a **Micro Client**.	EGC, TPEGC

9.7.2.2.2 Export Certificate definitions

9.7.2.2.2.1 Export Group Certificate and Revocation List

The **Certificate** definitions for the **ECI Export Group Certificate** (EGC) shall be in accordance with the general ECI_certificate definition is given in clause 5.2. The EGC uses the identifier field of **ECI Certificates** with the following field definition as given in Table 9.7.2.2.2.1-1.

Table 9.7.2.2.2.1-1: ECI export group ID definition

Syntax	No. of bits	Mnemonic
`ECI_EGC_Id {`		
`  type  /* see` Table 5.3-1`*/`	4	uimsbf
`  export_group_id /* see` Table 5.3-1 `*/`	20	uimsbf
`  export_group_version`	8	uimsbf
`}`		

Semantics:

Type	Value in accordance with Table 5.2-1.
export_group_id: integer	Id assigned to **Export Group** by entity managing the **Export Group**. Values 0x00000 and 0xFFFFF0-0xFFFFF are reserved.
export_group_version: integer	Version of the **Export Group Certificate** with the identifier **export_group_id**.

For authentication purposes of **Child Certificates** the EGC shall be accompanied by a revocation list in accordance with clause 5.3 and especially Table 5.3-1.

9.7.2.2.2.2 Third party Export Group Certificate and Revocation List

The **Certificate** definitions for the **ECI** third party **Export Group Certificate** (TPEGC) shall be in accordance with the general ECI_certificate definition is given in clause 5.2. The TPEGC uses the identifier field of **ECI Certificates** with the following field definition as given in Table 9.7.2.2.2.2-1.

Table 9.7.2.2.2.2-1: TPEGC identifier field definition

Syntax	No. of bits	Mnemonic
ECI_TPEGC_Id {		
type /* **see** Table 5.2-1*/	4	uimsbf
tp_export_group_id /* **see** Table 5.3-1 */	20	uimsbf
tp_export_group_version	8	uimsbf
}		

Semantics:

Type	Value in accordance with Table 5.3-1.
tp_export_group_id: integer	Id assigned to the third party **Export Group** by entity managing the third party **Export Group**. Values 0x00000 and 0xFFFFF0-0xFFFFF are reserved.
tp_export_group_version: integer	Version of the **Export Group Certificate** with the identifier **tp_export_group_id**.

The extension field of the TPEGC, as defined in Table 9.7.2.2.2.2-, shall contain the following structure, using the definitions of **export_authorization_operator_id** in Table 7.5.2-1 and **export_authorization_id** in Table 7.5.3-1.

Table 9.7.2.2.2.2-2: TPEGC extension field definition

Syntax	No. of bits	Mnemonic
ECI_ TPEGC _Extension {		
export_authorization_operator_id	20	uimsbf
export_authorization_id	20	uimsbf
padding(4)		
Extension_field **extension**		
}		

Semantics:

export_authorization_operator_id: integer	**ECI** identifier of export authorization operator **Certificate** that co-authenticates this **Certificate**.
export_authorization_id: integer	**ECI** identifier of the export authorization **Certificate** that co-authenticates this **Certificate** (see clause 9.7.1.2.2.5).
extension: Extension_field	Extension of this structure.

For authentication purposes of **Child Certificates** the TPEGC shall be accompanied by a revocation list in accordance with clause 5.3 and Table 5.3-1.

9.7.2.2.2.3 Root Revocation List for Export Authorization Operator Certificates

For authentication purposes an export authentication chain has to start with a root revocation list in accordance with clause 5.3 and Table 5.3-1.

9.7.2.2.2.4 Export Authorization Operator Certificate

The **Certificate** definitions for the **ECI** export authorization operator **Certificate** (EAOC) shall be in accordance with the general ECI_certificate definition is given in clause 5.2. The EAOC uses the identifier field of **ECI Certificates** with the following field definition as given in Table 9.7.2.2.2.4-1.

Table 9.7.2.2.2.4-1: EAOC identifier field definition

Syntax	No. of bits	Mnemonic
ECI_EAOC_Id {		
type /* **see** Table 5.3-1*/	4	uimsbf
export_authorization_operator_id /* **see** Table 5.3-1 */	20	uimsbf
export_authorization_operator_version	8	uimsbf
}		

Semantics:

type	Value in accordance with Table 5.3-1.
export_authorization_operator_id: integer	Id assigned to the export authorization operator. Values 0x00000 and 0xFFFFF0-0xFFFFF are reserved.
export_authorization_operator_version: integer	Version of the export authorization operator **Certificate** with the identifier **export_authorization_operator_id**.

For authentication purposes of **Child Certificates** the EAOC shall be accompanied by a revocation list in accordance with clause 5.3 and Table 5.3-1.

9.7.2.2.2.5 Export Authorization Certificate and Revocation List

The **Certificate** definitions for the **ECI** export authorization **Certificate** (EAC) shall be in accordance with the general ECI_certificate definition is defined in clause 5.2, using a specific non-empty extension field. The EAC uses the identifier field of **ECI Certificates** with the following field definition as given in Table 9.7.2.2.2.5-1.

Table 9.7.2.2.2.5-1: EAC extension field definition

Syntax	No. of bits	Mnemonic
ECI_EAC_Id {		
type /* **see** Table 5.3-1*/	4	uimsbf
export_authorization_id /* **see** Table 5.3-1 */	20	uimsbf
export_authorization_version	8	uimsbf
}		

Semantics:

Type	Value in accordance with Table 5.3-1.
export_authorization_id: integer	Id assigned to the export authorization **Certificate** (in the context of its **Father**). Values 0x00000 and 0xFFFFF0-0xFFFFF are reserved.
export_authorization_version: integer	Version of the export authorization operator **Certificate** with the identifier **export_authorization_id**.

The extension field of the EAC shall contain the **Certificate** structure that is to be authorized for export (see clause 5.1.3) excluding the **signature** field, followed by an extension field.

For authentication purposes of **Child Certificates** the EAC shall be accompanied by a revocation list in accordance with clause 5.3, and Table 5.3-1, if they are required to authenticate **Child** Certificates.

9.7.2.2.2.6 Export System Certificate

The **Certificate** definitions for the **ECI** export system **Certificate** (ESC) shall be in accordance with the general ECI_certificate definition is defined in clause 5.2. The public_key field of the **Certificate** shall contain the SPK value used by the **Micro Server**. The ESC uses the identifier field of **ECI Certificates** with the field definition as given in Table 9.7.2.2.2.6-1.

Table 9.7.2.2.2.6-1: ESC extension field definition

Syntax	No. of bits	Mnemonic
ECI_ESC_Id {		
type /* **see** Table 5.3-1*/*	4	uimsbf
export_system_id /* **see** Table 5.3-1 */	20	uimsbf
export_system_version	8	uimsbf
}		

Semantics:

Type	Value in accordance with Table 5.3-1.
export_system_id: integer	Id assigned to the export system **Certificate** (in the context of its **Father**). Values 0x00000 and 0xFFFFF0-0xFFFFF are reserved.
export_system_version: integer	Version of the export system **Certificate** with identifier **export_system_id**.

9.7.2.2.3 Validation of Export Certificate Chains

The Exporting **ECI Client** with a pre-validated chain and with complementary export authorization chains in order to create the requested **Import/Export Connection**. The exporting **ECI Client** and the importing **ECI Micro Server, being** responsible for their part of the chains, shall provide the **user** with information in case of issues and/or attempts to acquire renewed chains. The **ECI Client** shall provide these chains for processing to the **AS System** in order to create the desired **Export/Import Connection**. In case the **AS System** finds validation errors in any chain or in the complementary export authorization the **ECI Client** will not be able to establish the required connection.

Export Authorization Certificates are used to co-authenticate an export **Certificate**. The processing rules for co-authentication are:

1) The export authorization **Certificate** and the **Certificate** to be co-authenticated have valid signatures (as defined by their respective **Father**s) and are not revoked.

2) All data in the **Certificate** to be co-authenticated except its signature are compared to the data in the corresponding extension field of the export authorization **Certificate**. In case of a miss-match the co-authentication is not successful.

For setting up an **Export/ Connection** the CPS shall follow the processing rules below:

1) All CPS processing rules for **Certificate Chain**s as listed in clause 5.4.2 shall apply.

2) The CPS shall verify the types of the **child** of a **Father Certificate** as being appropriate in accordance with Table 5.2-2.

3) The **Father** for the exporting **ECI Client**'s **Export Chain** shall be the **ECI** POC of the Client. The accompanying Revocation List for **Export Group**s shall be applied for validating **Child Export Group Certificates**. The version number of the POC Revocation List for **Export Group**s shall be greater than the minClientVersion (see ETSI GS ECI 001-5-1 [4]) of the Client.

4) The CPS shall accept max. 2 levels of EGC for the exporting **ECI Client**. I.e. a **Child** of a second level EGC shall be a TPEGC or ESC.

5) The CPS shall ensure that any TPGC is accompanied by an EAC co-authenticated through a chain (with accompanying Revocation Lists) from the Root to EAOC to the EAC. The version of the Root Revocation List for Export Authorization Operator Certificate shall be used to determine the maximum revocation list version number for "system integrity validation".

6) The CPS shall ensure that any EGC, ESC and TPEGC descending from a TPEGC is co-authenticated by an EAC that is the **Child** of the EAC that validated the **Father** of that **Certificate**.

Exporting **ECI Clients** and **Micro DRM Servers** should provide adequate pre-processing on their chains and provide the latest available versions so as to avoid revocation in the CPS.

9.7.2.2.4 Transport protocols for export credentials

9.7.2.2.4.1 General

Exporting **ECI Clients** and **Micro Servers** may define their own formats for transporting credential data. **ECI** defines a standardized file format for carrying such data. These standardized files are accessible for **ECI Clients** through the **ECI** Carousel Access API for broadcast media. For online provisioning of Clients **ECI** defines standardized web API calls for this purpose.

9.7.2.2.4.2 Export Tree File format

For the **Export Group**s tree the file format is defined in Table 9.7.2.2.4.2-1.

Table 9.7.2.2.4.2-1: ECI Export Tree File definition

Syntax	No. Of bits	Mnemonic
ECI_Export_Tree_File {	24	
magic = 'EET'		
image_header_version	8	uimsbf
if (image_header_version == 0x01) {		
ECI_Operator_Id **operator_id**	32	uimsbf
ECI_Platform_Operation_Id **platform_operation_id**	32	uimsbf
ECI_RL_Tree **export_group_tree**		
Extension_Field **extensions**		
}		
}		

Semantics:

magic: byte[3]	Magic number used for verification of the format of the following data. It has the value of the three 8-bit ASCII representations of the characters 'EET'. **ECI Clients** shall check the value of this field to verify if an **ECI** file has the expected format for additional data integrity.
image_header_version: byte	Format version of the image header. Value 0x01 is the presently defined version; all other values are reserved. **ECI Clients** shall ignore any image with a version number that is not recognized.
operator_id: ECI_Operator_Id	ID of **Operator** of the **ECI Client** of the export tree contained in the file. The operator_version field corresponds to the root of the export_group_tree.
Platform-operation_id: ECI_Platform_Operation_Id	ID of **Platform Operation** of the **ECI Client** of the export tree contained in the file.
export_group_tree: ECI_RL_Tree	The ECI_RL_Tree structure, starting with the **Export Group** revocation List for the **Export Group**s and terminated at the. For Certificates that do not require a complementary **Revocation List** this structure shall contain an empty **Revocation List** with a signature not required to match the **Certificate**.
extensions: Extension_field	Additional data as defined by the Operator.

9.7.2.2.4.3 Import Chains File format

For the **Import Chain**s of a **Micro Server** the file format is defined in Table 9.7.2.2.4.3-1.

Table 9.7.2.2.4.3-1: ECI Import Chains File definition

Syntax	No. of bits	Mnemonic
ECI_Import_Chain_File {	24	
magic = 'EIC'		
image_header_version	8	uimsbf
if (image_header_version == 0x01) {		
ECI_Operator_Id **operator_id**	32	uimsbf
ECI_Platform_Operation_Id **platform_operation_id**	32	uimsbf
nr_chains	16	uimsbf
padding(4)		
for (i=0; i<nr_chains; i++){		
ECI_Operator_Id **eaoc_id**	32	uimsbf
ECI_Platform_Operation_Id **eac_id**	32	uimsbf
ECI_Certificate_Chain **import_chain**		
}		
Extension_Field **extensions**		
}		
}		

Semantics:

magic: byte[3]	Magic number used for verification of the format of the following data. It has the value of the three 8-bit ASCII representations of the characters `'EIC'`. **ECI Clients** shall check the value of this field to verify if an **ECI** file has the expected format for additional data integrity.
image_header_version: byte	Format version of the image header. Value 0x01 is the presently defined version; all other values are reserved. **ECI Clients** shall ignore any image with a version number that is not recognized.
operator_id: ECI_Operator_Id	ID of **Operator** of the **Micro Server** for which this **Import Chain** is intended.
platform_operation_id: ECI_Platform_Operation_Id	ID of **Platform Operation** of **Micro Server** for which this **Import Chain** is intended.
nr_chains: integer	The number of **Import Chain**s in the file.
eaoc_id: ECI_Operator_Id	ID of the authorization **Operator** of the **Import Chain**.
eac_id: ECI_Platform_Id	ID of the EAC that co-authorizes the Platform **Operation** of the **Import Chain**.
import_chain: ECI_Certificate_Chain	The **ECI Certificate Chain** from the import **Platform Operation Certificate** to the ESG identifying the **Micro Client**. The chain may contain multiple TPEGCs. Each valid **Import Chain** shall be separately represented: i.e. if chain1 consists of two third party subchains and the second subchain can also be used separately as an **Import Chain** it shall be represented separately. For **Certificates** that do not require a complementary **Revocation List** this structure shall contain an empty **Revocation List** with a signature not required to match the **Certificate**.
extensions: Extension_field	Additional data as defined by the Operator.

9.7.2.2.4.4 Export Authorization File format

For the authorization of **Export Chain**s of a **Micro Server** the file format is defined in Table 9.7.2.2.4.4-1.

Table 9.7.2.2.4.4-1: ECI Export Authorization File definition

Syntax	No. of bits	Mnemonic
`ECI_Export_Authorization_File {`	24	
`  magic = 'EEA'`		
`  image_header_version`	8	uimsbf
`  if (image_header_version == 0x01) {`		
`    ECI_Operator_Id operator_id`	32	uimsbf
`    ECI_Platform_Operation_Id platform_operation_id`	32	uimsbf
`    nr_chains`	16	uimsbf
`    padding(4)`		
`    for (i=0; i<nr_chains; i++){`		
`      direct_flag`	1	uimsbf
`      padding(4)`		
`      ECI_Operator_Id o_id`	32	uimsbf
`      ECI_Platform_Operation_Id po_id`	32	uimsbf
`      ECI_Certificate_Chain chain`		
`    }`		
`    Extension_Field extensions`		
`  }`		
`}`		

Semantics:

magic: byte[3]	Magic number used for verification of the format of the following data. It has the value of the three 8-bit ASCII representations of the characters 'EEA'. **ECI Clients** shall check the value of this field to verify if an **ECI** file has the expected format for additional data integrity.
image_header_version: byte	Format version of the image header. Value 0x01 is the presently defined version; all other values are reserved. **ECI Clients** shall ignore any image with a version number that is not recognized.
operator_id: ECI_Operator_Id	ID of **Operator** of the **Micro Server** for which this **Import Chain** is intended.
Platform_operation_id: ECI_Platform_Operation_Id	ID of **Platform Operation** of **Micro Server** for which this **Import Chain** is intended.
nr_chains: integer	The number of export authorization chains in this file.
direct_flag: bit	If the value is 0b1 the following chain directly authorizes an ESC subchain and **o_id** and **po_id** are not relevant. If the value is 0b0 the following chain authorizes TPEGC subchain and **o_id** and **po_id** represent authorization **certificate** ids.
o_id: ECI_Operator_Id	ID of the third party **Operator** an interim third party export chain that is authenticated by the following export authentication chain.
po_id: ECI_Platform_Öperation_Id	ID of the third party **Platform Operation** an interim third party **Export Chain** that is authenticated by the following export authentication chain.
chain: ECI_Certificate_Chain	The **ECI Certificate Chain** from the **ECI Root Certificate** to the EAC authenticating the first TPEGC, ESG.
extensions: Extension_field	Additional data as defined by the Operator.

9.7.2.2.4.5 Broadcast Carousels carrying export credentials

Operators may deploy **ECI** defined carousels as defined in clause 7.7.2 to carry the export and/or import credentials of the **ECI Clients** they choose to support. However, for any specific **ECI Client** the **ECI Host** shall only have to monitor the updates of a single location DSI of a data carousel. I.e. for the purpose of carrying export or import credentials using the standard carousel format, an **Operator** shall use the same carousel that carries Client Image, **Platform Operation** credentials and revocation data, etc. for such an **ECI Client**. Also see clause 7.7.2.1.

The formats of the data of the carousel-modules shall follow Table 7.7.3-3. Modules designated by a compatibilityDescriptor with descriptorType field equal 0xB0 shall carry modules with a single ECI_Export_Tree_File structure, those with descriptorType field equal 0xB1 shall carry modules with a single ECI_Import_Chain_File structure and those with descriptorType field equal 0xB2 shall carry modules with a single ECI_Export_Authentication_File structure.

It is recommended that the **ECI Client** monitoring of updates in the carousel coincides with those to be performed by the **ECI Host** for the other **ECI Client** data so as to permit efficient power management.

9.7.2.2.4.6 Online provisioning of export credentials

The present document reserves the following web API URL structures for the purpose of permitting a standard structure for **ECI Clients** to access export credentials from an Operator's online server.

With reference to clause 7.7.3 for the definition of tail_extension and the notational conventions:

```
tail_extension* ::=
        client_export |
        client_import |
        client_exp_auth .
```

The notation tail_extension* indicates other extensions may be in future versions of the present documents.

The following web API requests are defined for import/export:

```
client_export ::= 'client-export/' operator_id '/' platform_operation_id .
```

This shall return the latest version of the export tree file with format ECI_Export_Tree_File for the **ECI Client** designated by **operator_id**, **platform_operatio_id**.

```
client_import ::= 'client-import/' operator_id '/' platform_operation_id .
```

This shall return the latest version of the **Import Chain** file with format ECI_Import_Chain_File for the **Micro Server** client designated by **operator_id**, **platform_operation_id**.

```
client_exp_auth ::= 'client-exp_auth/' operator_id '/' platform_operation_id .
```

This shall return the latest version of the export authentication file with format ECI_Export_Authentication_File for the **Micro Server** client designated by **operator_id**, **platform_operation_id**.

9.7.2.3 Export Connection API

9.7.2.3.1 General

ECI Clients can provide export information to the **ECI Host**. This permits the **ECI Host** to pair the exporting system with matching **Import Chain**s from **Micro Servers**. The **ECI Host** (and application) can define the actual connections to be created from all possible options. It can attempt to connect the exporting and matching importing **ECI Client** by sending the exporting **ECI Client** a connection request with the **Import Chain** for the target import **ECI Client**. The exporting **ECI Client** as well as the **ECI Host** may request to cancel the connection or to re-initialize the connection in case of updated import credentials. The available **Export Connection** messages are listed in Table 9.7.2.3.1-1.

Table 9.7.2.3.1-1: Export Connection API Messages

Message	Type	Dir.	Tag	Description
reqExpConnNodes	A	H→C	0x0	The **ECI Host** requests export option nodes from the **ECI Client**.
reqExpConnSetup	A	H→C	0x1	The **ECI Host** requests the **ECI Client** to initialize an **Export Connection** to an importing **ECI Client** based on an **Import Chain**.
reqExpConnDrop	A	H→C	0x2	The **ECI Hosts** cancels any previously initialized connection of an exporting **ECI Client** to an importing **ECI Client**.
reqExpConnCancel	A	C→H	0x3	The **ECI Client** terminates an initialized **Export Connection** with an importing **ECI Client**.
reqExpMhOpen	A	H→C	0x4	The **ECI Host** requests the **ECI Client** to create an export session based on a previously initialized **Export Connection**.
reqExpMhClose	A	H→C	0x5	The **ECI Host** closes an export session.
reqExpMhCancel	A	C→H	0x6	The **ECI Client** cancels an export session.

9.7.2.3.2 reqExpConnNodes Message

H→C reqExpConnNodes() →
C→H resExpConnNodes(ExpConnOption **conn Nodes [])**

- The message requests the **ECI Client** to return its list with possible **Export Connection**s; the **Response** message returns the list. Related error codes are listed in Table 9.7.2.3.2-2.

Response Parameter definitions:

connNodes: ExpConn Option[]	The list provides the **ECI** identities of either third party or **ECI Clients** that the **ECI Client** can connect to for export. Each option has a priority: the higher the priority the less chance that an export cannot be successfully completed. ExpConnNode is defined in Table 9.7.2.3.2-1.

Table 9.7.2.3.2-1: ExpConnNode type definition

```
typedef struct ExpConnNode {
   uint  targetType;
   uint  operatorId;
   uint  targetId;
   uint  targetPriority;
} ExpConnNode;
```

Field definitions:

targetType: uint	Type of the target: Value equal 1 is EAC (third party), Value equal 2 is POC (direct export). Other values are undefined.
operatorId: uint	Representing the 20-bit **ECI** Certificate ID of the operator of the target export: export_authorization_operator_id for the EAC target, and operator_id for the POC target.
targetId: uint	Representing the 20-bit **ECI** Certificate ID of the target export being the export_authorization_ id for the EAC target, and the platform_operation_id for the POC target.
targetPriority: uint	The priority to select a particular export is the sum of two parts: • Value in multiples of 1 024 that represents a specific (commercial) priority for the export to be connected to a particular **Micro Server**. • Value between 0 and 1 023 representing a fraction minus 1 of 1 024 of the anticipated content use cases that can be exported with this export **Micro DRM System**. **ECI Hosts** shall use this information to either automatically select the most suitable **Micro DRM System** (provided application requirements for the micro DRM application are met by the highest priority system) and/or to present the above as a preference towards the **user** in case of a manual selection.

Table 9.7.2.3.2-2: reqExpNodeInfo Error Codes

Name	Description
ErrExpConnNwAccess	See Table 9.7.2.3.9-1.
ErrExpConnAuthProblem	
ErrExpUninitState	

9.7.2.3.3 reqExpConnSetup Message

H→C reqExpConn Setup (CertChainSerial **Import,** CertChainSerial **Auth[],**ushort **connId)** →
C→H resExpConn Setup ()

- This message requests the **ECI Client** to initialize (or re-initialize) an **Export Connection** connId with the **ECI Client** with the identifier **clientId** using **Import Chain Import**, export authentication chains **Auth**, and **ECI Client** chain **Target**.

Request Parameter definitions:

Import: CertChainSerial	Import chain (from export TPEGC to ESC).
Auth: CertChainSerial[]	The export authentication chains from Root to the EAC that authenticates the first TPEGC in a single third party subchain. The chains in **Auth** are in order from exporting connecting TPEGC towards importing POC.
connId: ushort	ID of the **Export Connection**, assigned by the **ECI Host**.

CertChainSerial type and array type definition

CertChainSerial is the network order representation (big endian) of `ECI_Certificate_Chain` as defined in Table 5.4.1-1, padded to a multiple of 32-bits.

CertChainSerial[] is defined by the following (quasi-C) data structure:

```
typedef struct CertChainSerial {
   uint   numberElements;    /* the number of elements in the chain array*/
   uint   elementIndex[];    /* the index of the start of each element in
                                chainElements data container */
   uint   chainElements[];   /* data container with numberElements
                                SertChainSerial representations of the
                                successive chains in the array. */
  } CertChainSerial;
```

elementIndex and chainElements shall be represented inline data arrays in the certChainSerialArray data structure.

Detailed semantics:

- **ECI Hosts** can issue a reqExpConnSetup request on behalf of an existing connection so as to inform the exporting **ECI Client** of (potentially) new import credentials of the importing **ECI Client**. Exporting **ECI Client** are recommended to postpone renewal of the connection with the importing **ECI Client** until such time no active session is taking place, or if the present connection could be dropped instantly.

The related error codes are listed in Table 9.7.2.3.3-1.

Table 9.7.2.3.3-1: reqExpConnSetup Error Codes

Name	Description
ErrExpConnNwAccess	See Table 9.7.2.3.9-1.
ErrExpConnAuthProblem	
ErrExpUninitState	
ErrExpInvalidChain	

9.7.2.3.4 reqExpConnDrop Message

H→C reqExpConnDrop(ushort **connId)→**
C→H resExpConnDrop()

- The message requests the **ECI Client** to drop an **Export Connection** with the client identified by **connId**.

Request Parameter definitions:

connId: ushort	ID of the **Export Connection**.

Pre conditionPre condition Request:

1) An **Export Connection** (identified by **connId)** was previously established.

Post condition Response:

1) The **Export Connection** (if it existed) is closed.

The related error codes are listed in Table 9.7.2.3.4-1.

Table 9.7.2.3.4-1: reqExpConnDrop Error Codes

Name	Description
ErrExpConnNone	See Table 9.7.2.3.9-1.

9.7.2.3.5 reqExpConnCancel Message

C→H reqExpConnCancel(ushort **connId)** →
H→C resExpConnCancel()

- The message informs the **ECI Host** that the **Export Connection** identified by **connId** has been terminated by the **ECI Client**.

Request Parameter definitions:

connId: ushort	ID assigned to the connection.

Pre condition Request:

1) An **Export Connection** identified by **connId** was previously established.

9.7.2.3.6 reqExpMhOpen Message

H→C reqExpMhOpen(ushort **mhExp,** ushort **mhDcr,** ushort **connId)** →
C→H resExpMhOpen(ushort **mhExp)**

- The message requests the **ECI Client** to create an export session identified by **Media Handle mh** over **Export Connection connId**.

Request Parameter definitions:

mhExp: ushort	**Media Handle** assigned by the **ECI Host** to the **Export Connection**.
mhDcr: ushort	**Media Handle** of the decryption session to be exported.
connId: ushort	ID assigned to the **Export Connection**.

Response Parameter definitions:

mhExp: ushort	**Media Handle** assigned by the **ECI Host** to the **Export Connection**

Pre condition Request:

1) An **Export Connection connId** was previously established.

2) A Decryption session **mhDcr** was previously established.

Post condition Request:

1) An **Export Connection** is established or an error occurred.

Detailed semantics:

- The exporting **ECI Client** may suspend and resume export on an existing session e.g. based on **Export Group** inclusion of the connection.

The related error codes are listed in Table 9.7.2.3.6-1.

Table 9.7.2.3.6-1: reqExpMhOpen Error Codes

Name	Description
ErrExpConnNone	See Table 9.7.2.3.9-1.
ErrExpDcrMhNone	

9.7.2.3.7 reqExpMhClose Message

H→C reqExpMhClose(ushort **mhExp) →**
C→H resExpMhClose(ushort **mhExp)**

- This message requests the **ECI Client** to close an export session identified by **Media Handle mh** over **Export Connection connId**.

Request Parameter definitions:

mhExp: ushort	**Media Handle** assigned by the **ECI Host** to the **Export Connection**.

Response Parameter definitions:

mhExp: ushort	**Media Handle** assigned by the **ECI Host** to the **Export Connection**.

Pre condition Request:

1) An export session **mhExp** was previously established and did not yet terminate.

Post condition Request:

1) The export session **mhExp** has stopped.

The related error codes are listed in Table 9.7.2.3.7-1.

Table 9.7.2.3.7-1: reqExpMhClose Error Codes

Name	Description
ErrExpMhNone	See Table 9.7.2.3.9-1.

9.7.2.3.8 reqExpMhCancel Message

C→H reqExpMhCancel(ushort **mhExp) →**
H→C resExpMhCancel(ushort **mhExp)**

- This message informs the **ECI Host** that the **ECI Client** has stopped the export session **mhExp**.

Request Parameter definitions:

mhExp: ushort	**Media Handle** assigned by the **ECI Host** to the **Export Connection**.

Response Parameter definitions:

mhExp: ushort	**Media Handle** assigned by the **ECI Host** to the **Export Connection**.

Pre condition Request:

1) An export session **mhExp** was previously established.

2) The **ECI Client** terminated the session.

9.7.2.3.9 Error Codes for the Export Connection API

The values are meanings of the API specific errors that can be returned by the **Response** messages for this API are defined in Table 9.7.2.3.9-1.

Table 9.7.2.3.9-1: Error codes media session API for TS media

Name	Value	Description
ErrExpConnNwAccess	-256	Access to the network providing information on the requested information is not possible or is unexpectedly slow and could not be completed.
ErrErrConnAuthProblem	-257	Internal inconsistencies in the provisioned data were detected preventing the request to be completed.
ErrEcxpConnUninitState	-258	The **ECI Client** first requires provisioning and/or other perform functions in order to be able to respond to this request.
ErrExpConnInvalidChain	-259	A chain provided to the **ECI Client** was found to be invalid and/or it was not possible to authenticate it using the Authentication Chains.
ErrExpConnNone	-260	The connection did not exist.
ErrExpMhNone	-261	The export session indicated by **Media Handle** is not supported by the **ECI Client**.
ErrExpDcrMhNone	-262	The decryption session indicated by the **Media Handle** is not supported by the **ECI Client**.

9.7.2.4 Import Connection API

9.7.2.4.1 General

ECI Clients can provide their **Import Chain**s to the **ECI Host**. This permits the **ECI Host** to connect the importing **ECI Client** to matching export options from **Micro Servers**. The **ECI Host** and application can select to establish the connection(s) to be created from the available connection options. The **ECI Host** can commence to set up a connection between exporting and importing **ECI Client** by first requesting the importing Client permission to connect it to the exporting **ECI Client**. The importing Client can refuse such a connection on the basis of e.g. commercial considerations of its operator. In case a connection is established the importing **ECI Client** as well as the **ECI Host** may request to cancel the connection or to re-initialize the connection in case of updated import credentials.

Input chains are identified by their first node. i.e. the **ECI** ids of the EAOC and EAC for the TPEGC. This is referred to below in Table 9.7.2.4.1-1 as the *import node*.

Table 9.7.2.4.1-1: Import Connection API messages

Message	Type	Dir.	Tag	Description
reqImpConnNodes	A	H→C	0x0	**ECI Host** requests importing **ECI Client** to provide its import nodes.
reqImpConnChain	A	H→C	0x1	**ECI Host** requests importing **ECI Client** to provide input chain for a specific import node.
reqImpConnChainRenew	A	C→H	0x2	The **ECI Client** requests the **ECI Host** to reinitialize the connection using an updated **Import Chain**.
reqImpConnSetup	A	H→C	0x3	**ECI Host** requests importing **ECI Client** to initialize an **Import Connection** with a specific exporting **ECI Client** through an import node.
reqImpConnDrop	A	H→C	0x4	**ECI Host** drops the **Import Connection** with the specified exporting **ECI Client**.
reqImpConnCancel	A	C→H	0x5	**ECI Client** terminates the **Import Connection** with the specified exporting **ECI Client**.

9.7.2.4.2 reqImpConnNodes Message

H→C reqImpConnNodes () →
C→H resImpConnNodes(ImpConnNode **nodes[])**

- This message enables the **ECI Host** to request importing **ECI Client** to provide its import nodes.

Response Parameter definitions:

nodes[]: ImpConnNode	Array of import nodes and number of third party intermediaries. The structure of ImpConnNodes is defined in Table 9.7.2.4.2-1.

Table 9.7.2.4.2-1: ImpConnOption type definition

```
typedef str uct ImpConnNode {
   uint  targetType;
   uint  operatorId;
   uint  targetId;
   uint  intermediaries
} ImpConnNode;
```

Field definitions:

targetType: uint	Type of the target: 1 is EAC (third party), 2 is POC (direct export). Other values are undefined.
operatorId: uint	Representing the 20-bit **ECI** Certificate ID of the operator of the target import: export_authorization_operator_id for the EAC target, or operator_id for the POC target.
targetId: uint	Representing the 20-bit **ECI** Certificate ID of the target import: export_authorization_ id for the EAC target, or platform_operation_id for the POC target.
intermediaries: uint	Represents the number of intermediate third parties from input node to the POC of the importing **ECI Client**. **ECI Hosts** shall select the shortest **Import Chain** among alternatives for export options that have the same targetPriority for the exporting **ECI Client**.

The related error codes are listed in Table 9.7.2.4.2-2.

Table 9.7.2.4.2-2: reqExpConnInfo Error Codes

Name	Description
ErrImpConnNwAccess	See Table 9.7.2.4.7-1.
ErrImpConnAuthProblem	
ErrImpUninitState	

9.7.2.4.3 reqImpConnChain and reqImpConnChainRenew Messages

H→C reqImpConnChain(ImpConnNode **node)** →
C→H resImpConnChain(CertChainSerial **Import,** CertChainSerial **Auth[])**

- This message enables the **ECI Host** to request importing **ECI Client** to provide input chain for a specific import node.

C→H reqImpConnChainRenew(CertChainSerial **Import,** CertChainSerial**Auth[])** →
H→C resImpConnChainRenew()

- This message enables the **ECI Client** to request the **ECI Host** to reinitialize the connection using an updated **Import Chain**.

Request Parameter for reqImpConnChain:

node: ImpConnNode	Import node for which the **Import Chain** shall be returned to the **ECI Host**.

Request Parameter definitions for reqImpConnChainRenew and
Response Parameters definitions for reqImpConnChain:

Import: CertChainSerial	Import chain (from export TPEGC to ESC).
Auth: CertChainSerial[]	The export authentication chains from Root to the EAC that authenticates the first TPEGC in a single third party subchain. The chains in **Auth** are in order from exporting TPEGC towards importing POC.

Pre condition reqImpConnChainRenew Request:

1) An **Import Connection** was previously established with an **ECI Client** using an element in the provided chain.

Detailed semantics for reqImpConnChainRenew:

- The **ECI Host** shall pass the updated chain information forthwith to the affected exporting **ECI Clients**.
- It is recommended that Operators provide updated chains substantially ahead of deprecation the previous chain so as to ensure uninterrupted service provisioning.

The reqImpConnChain related error codes are listed in Table 9.7.2.4.3-1.

Table 9.7.2.4.3-1: reqImpConnChain Error Codes

Name	Description
ErrImpConnNwAccess	See Table 9.7.2.4.7-1.
ErrImpConnAuthProblem	
ErrImpConnUninitState	

The reqImpConnChainRenew related error codes are listed in Table 9.7.2.4.3-2.

Table 9.7.2.4.3-2: reqImpConnChainRenew Error Codes

Name	Description
ErrImpConnNoConn	See Table 9.7.2.4.7-1.

9.7.2.4.4 reqImpConnSetup Message

H→C reqImpConnStart (ImpConnNode **node,** ushort **exportClientId,** ushort **connId)** →
**C→H resImpConnStart(**)

- This message enables the **ECI Host** to request importing **ECI Client** to establish an **Import Connection** with a specific exporting **ECI Client** through an import node.

Request Parameters:

node: ImpConnNode	Import node through which the connection is established.
exportClientId: ushort	**ECI Host** identification of exporting **ECI Client**.
connId: ushort	**ID assigned to the import connection**.

Detailed semantics:

- The **ECI Client** may reject the **Import Connection** on the basis of commercial considerations of its operator.

The related error codes are listed in Table 9.7.2.4.4-1.

Table 9.7.2.4.4-1: reqExpConnStart Error Codes

Name	Description
ErrImpConnNwAccess	See Table 9.7.2.4.7-1.
ErrImpConnAuthProblem	
ErrImpConnUninitState	
ErrImpConnRefuseComm	
ErrImpConnUnknError	

9.7.2.4.5 reqImpConnDrop Message

H→C reqImpConnDrop (ushort **connId)** →
**C→H resImpConnDrop(**)

- This message enables the **ECI Host** to drop the **Import Connection** with the specified exporting **ECI Client**.

Request Parameters:

connId: ushort	**ECI Host** identification of the import connection to be dropped.

Pre condition Request:

1) An **Import Connection** (identified by **connId)** previously initialized.

Post condition Response:

1) The **Export Connection** (if it existed) is closed.

The related error codes are listed in Table 9.7.2.4.5-1.

Table 9.7.2.4.5-1: reqExpConnInfo Error Codes

Name	Description
ErrImpConnNwAccess	See Table 9.7.2.4.7-1.
ErrImpConnAuthProblem	
ErrImpConnUninitState	
ErrImpConnNoConn	

9.7.2.4.6 reqImpConnCancel Message

C→H reqImpConnCancel (ushort **connId)** →
**H→C resImpConnCancel(**)

- This message enables the **ECI Client** to terminate the **Import Connection** with the specified exporting **ECI Client**.\

Request Parameters:

connId: ushort	An **Import Connection** (identified by connId) previously initialized.

Pre condition Request:

1) An **Import Connection** was previously established with the Client with **ECI Host** Client ID **exportClientId** and is closed.

9.7.2.4.7 Error Codes for the Export Connection API

The values are meanings of the API specific errors that can be returned by the **Response** messages for this API are listed in Table 9.7.2.4.7-1.

Table 9.7.2.4.7-1: Error codes media session API for TS media

Name	Value	Description
ErrImpConnNwAccess	-256	Access to the network providing information on the requested information was unexpected slow.
ErrImpConnAuthProblem	-257	Internal inconsistencies in the provisioned data were detected preventing the request to be completed.
ErrImpUninitState	-258	The **ECI Client** first requires provisioning and/or other perform functions in order to be able to respond to this request.
ErrImpConnRefuseComm	-259	A chain provided to the **ECI Client** was found to be invalid and/or it was not possible to authenticate it using the Authentication Chains.
ErrImpConnRefuseComm	-260	The importing **ECI Client** declines to connect to the exporting **ECI Client** on basis of commercial conditions.
ErrImpConnUnknError	-261	The importing **ECI Client** encountered an unknown error.
ErrExpConnNone	-262	The connection did not exist.

9.7.2.5 Re-encryption API

9.7.2.5.1 General

The re-encryption API permits a **Micro Server** to re-encrypt content from an **Import Connection** specific to one of a group of clients for subsequent decoding by a **Micro Cli**ent. The decoding may have to be performed near instantaneously (streaming connection) and may not permit re-play on a subsequent session or alternatively the re-encrypted content may be stored or time shifted with associated decryption information for the decoding **Micro Client** and be decoded by the **Micro Client** later.

The discovery phase permits the application to match a **Micro Server** to a possible **Target (Micro Client** or a group of **Micro Clients)**, and to exchange the required authentication information from **Micro Client** to **Micro Server** to allow for the authentication of the **Micro Client** and to permit the basis for a trusted exchange of content. The **ECI Host** can select a bidirectional communication mode (IP based or message passing through the **ECI Host**) so as to support more elaborate protocols of authentication between Micro Server and **Micro Client**.

Based on a re-encryption connection to the **Target** and an **Import Connection,** the **ECI Host** can instantiate a **Media Handle** session of the mode (re-encryption, synchronization and data format mode) that is desired by the application and that can be supported by the **Micro Server**.

Once a re-encryption connection is established, the **ECI Host** can instantiate a **Media Handle** session with a **Micro Server** and start to re-encrypt content from an established **Import Connection** for the **Target** (**ECI Client** or the group of **ECI Clients**). Multiple simultaneous re-encryptions of the same content may be instantiated, each using its own **Media Handle** session. It is the responsibility of the **ECI Host** to ensure the content for the re-encryption **Media Handle** session is sourced from the authenticated export **Media Handle** on the **Export Connection**. An unauthorized wrong connection will lead to an export authentication failure.

The re-encryption control words are applied to the imported decrypted content and new markings (URI's, etc.) are applied to the re-encrypted content using the AS system.

There can be three main modes of *encryption-mode*:

1) Online Streaming mode: both **Micro Server** and **Micro Client** are active at the same time. They exchange messages directly (through an IP channel) or as explicit messages through their **ECI Hosts**.

2) Offline Streaming mode: the **Micro Server** encrypts content "on the fly" and regularly issues new data needed for decryption by the **Micro Client**. The result can be delayed (time offset mode) or stored.

3) Offline Storage mode: the **Micro Server** encrypts content and at the end produces the data required by the **Micro Client** at the start for decoding the content.

Figure 9.7.2.5.1-1 presents a schematic overview of the different encryption-modes.

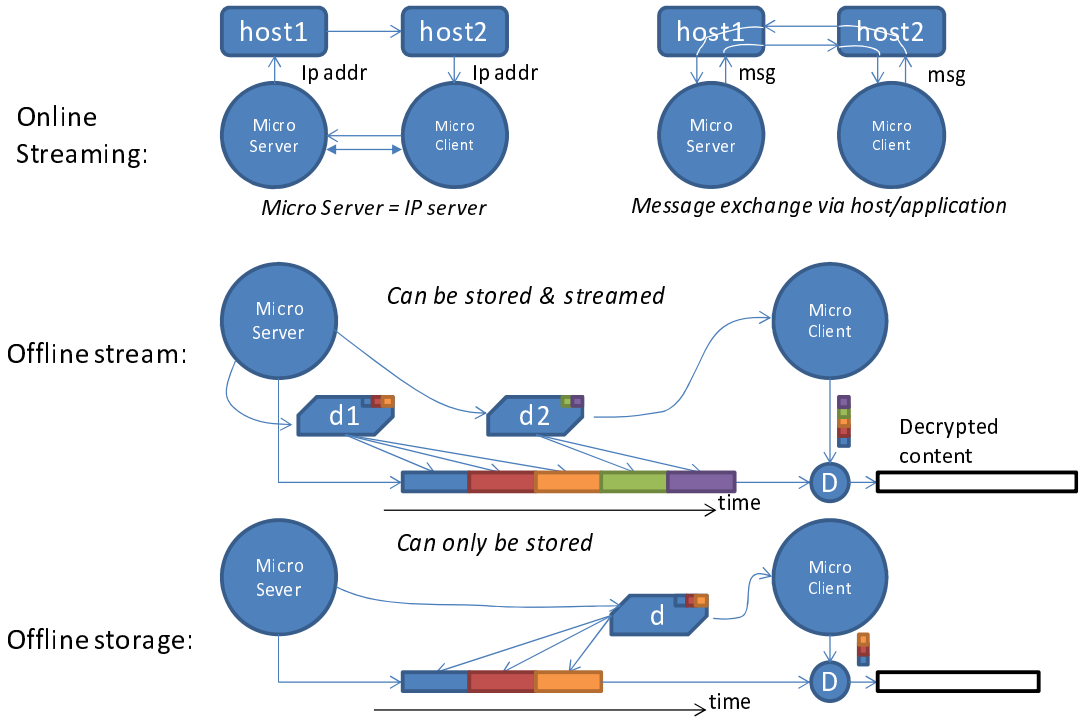


Figure 9.7.2.5.1-1: Encryption modes for micro DRM sessions

The data needed for decryption of content to be exchanged in the two offline encryption-modes between **Micro Server** and **Micro Client** may be passed in the following *data-format-modes*:

1) Generic mode: the **Micro Server** produces opaque data containers with information required for decrypting the content by the **Micro Client**.

2) ISOBMFF mode (only *for synchronization-mode* equal file mode): the **Micro Server** generates PSSH boxes for inclusion in an ISOBMFF file [39]. The **ECI Host** can use these to create ISOBMFF files by appropriate inclusion of the PSSH boxes in ISOBMFF MOOV or MOOF boxes.

There are two mechanisms supported for *synchronization-mode* to permit associating the correct control word to a section of content, applying to all of the above re-encryption modes:

1) In Transport Stream (alternating bit) mode the **Micro Server** produces ECM sections that can be packetized and inserted in the transport stream by the **ECI Host**. The ECM is inserted preceding the crypto-period for which it provides information to permit computation of the control word.

2) In File Mode the **Micro Server** produces encrypted control words that are referred to by explicit KeyID identifiers in the supplementary decryption information. The **ECI Host** should preserve the KeyID association of the content section encrypted with a specific control word so that the **Micro Client** can produce the correct control word for descrambling.

In offline mode, synchronization of the additional data needed for decryption or computation of the KeyId or ECMs explicitly referencing the time-dependency relation of the data relative to the KeyId or ECM number.

Not all **Micro Servers** have to support all modes of operation. At initialization, immediately after using the discovery API a **Micro Server** signals the modes (combination of encryption-mode, data-format-mode and synchronization-mode) it can support.

Once the **Media Handle** session has been instantiated it can be started and stopped by the **ECI Host** and cancelled by the **ECI Client**.

The messages for the re-encryption API are listed in Table 9.7.2.5.1-1.

Table 9.7.2.5.1-1: Re-encryption API messages

Message	Type	Dir.	Tag	Description
setEncrModes	set	C→H	0x0	The Micro Server informs the **ECI Host** about the modes (encryption-modes, data-format-modes and synchronization-modes) it supports.
reqEncrTargets	A	H→C	0x1	**ECI Host** requests the **Micro Server** to provide the target nodes it can authenticate for decryption.
reqEncrConnSetup	A	H→C	0x2	**ECI Host** requests the **ECI Client** to create a re-encryption target connection and to pre-authenticate the re-encryption target for subsequent reference in setting up a **Media Handle** session.
reqEncrConnDrop	A	H→C	0x3	**ECI Host** requests **the ECI Client** to drop any information on a previously pre-authenticated re-encryption connection.
reqEncrConnCancel	A	C→H	0x4	**ECI Client** cancels a previously established encryption target connection.
reqEncrMhOpen	A	H→C	0x5	**ECI Host** requests **ECI Client** to open a **Media Handle** session to re-encrypt content from an incoming **Import Connection** for an established re-encryption connection.
reqEncrMhClose	A	H→C	0x6	**ECI Host** closes the re-encryption session with the **ECI Client**.
reqEncrMhCancel	A	C→H	0x7	**ECI Client** terminates the **Import Connection** with the specified exporting **ECI Client**.
reqEncrMhStart	A	H→C	0x8	**ECI Host** requests the **ECI Client** to start the re-encryption operation for a **Media Handle** session.
reqEncrMhStop	A	H→C	0x9	**ECI Host** requests the **ECI Client** to stop a re-encryption operation for a **Media Handle** session.
reqEncrMhQuit	A	C→H	0xA	The **ECI Client** informs the **ECI Host** that the **Media Handle** re-encryption operation was terminated.
reqEncrIpServer	A	H→C	0xB	The **ECI Host** requests the IP server address of a Micro Server so as to permit **Micro Client**s to create IP connections.
reqEncrMsgSend	A	C→H	0xC	The **Micro Server** requests the **ECI Host** to forward a message to the target of a **Media Handle** session.
reqEncrMsgRecv	A	H→C	0xC	The **ECI Host** provides the **Micro Server** with a message from a target of a **Media Handle** session.
reqEncrTsData	A	C→H	0xE	The **Micro Server** provides the **ECI Host** with data to be forwarded to the target **Micro Client** of a **Media Handle** for decryption, including ECM related synchronization information.
reqEncrTsEcm	A	C→H	0xF	The **Micro Server** issues an ECM section that is required by the **Micro Client** to decrypt in the next crypto-period.
reqEncrFileData	A	C→H	0x10	The **Micro Server** provides the **ECI Host** with a message to be forwarded to the target **Micro Client** of a **Media Handle** for decryption, including KeyID related synchronization information.

9.7.2.5.2 setEncrModes Message

C→H setEncrModes(EciEncrModes **modes)**

- This message allows the **Micro Server** to inform the **ECI Host** about the modes (encryption-modes, data-format-modes and synchronization-modes) it supports.

Request Parameter definitions:

modes: EciEncrModes	Modes of encryption that are supported by the **Micro Server**. The type EciEncrModes is specified in Table 9.7.2.5.2-1.

Table 9.7.2.5.2-1: EciEncrModes type definition

```
Typedef uint EciEncrModes;
```

Bit definitions:

Name	Bit	Micro Server Mode Support on value equal 0b1
OnlineIpMode	0	Supports online IP mode.
OnlineMsgMode	1	Supports online Message mode.
OfflineStreamMode	2	Supports offline stream mode.
OfflineStorageMode	3	Supports offline store mode.
OfflineDataMode	4	Supports default data format containers for decryption data in offline mode. Not relevant if no offline mode is selected.
OfflineIsobmffMode	5	Supports ISOBMFF format PSSH boxes for decryption data in offline mode. Not relevant if no offline mode is selected.
SyncTs	6	Synchronizes control words to transport stream format alternating bit delimited crypto periods for the content.
SyncFile	7	Synchronizes to file-type formats using KeyID identification to associate content sections to their control word.
other	RFU	Reserved for future use.

9.7.2.5.3 reqEncrTargets Message

H→C reqEncrTargets() →
C→H resEncrTargets(EncrTarget **target[])**

- This message permits the **ECI Host** to request the **Micro Server** to provide the encryption targets it can authenticate.

Response Parameter definitions:

target: EncrTarget[]	List of encryption targets that the Micro Server can authenticate. The type definition of TargetClient is specified in Table 9.7.2.5.3-1.

Table 9.7.2.5.3-1: EncrTarget type definition

```
typedef struct EncrTarget {
   uint  targetType;
   byte  target[8];
} EncrTarget;
```

Field definitions:

targetType: uint	Type of the encryption target:Value equal 1 is individual client, Value equal 2 means group of clients, other values are reserved for future use.
target: byte[8]	ID representing the target. The value is defined within the scope of the **Micro DRM System**. Matching by the **ECI Host** is defined in terms of equality of **targetType** and **target** fields.

Detailed semantics:

- The **ECI Host** can match potential target **Micro Clients** based on **Target**. It is up to the application and/or **ECI Host** to locate potential candidate **Micro Clients**.

- **ECI Hosts** wishing to perform local PVR and time shift functions (using either an integrated or a connected/networked storage medium on which they can store encrypted content and associated data) can attempt to match a **Micro Server** being able to operate in **OfflineStreamMode** to **Micro Client**s installed on the same **ECI Host**.

9.7.2.5.4 reqEncrConnSetup Message

H→C reqEncrConnSetup(ushort **targetConnId**, EciEncrTarget **target**, ushort **credLen**, byte **cred[])**
C→H resEncrConnSetup(ushort **targetConnId)**

- This message permits the **ECI Host to** request **the Micro Server to** create a re-encryption connection to the target and to (pre-)authenticate the target. Error codes are defined in Table 9.7.2.5.19-1.

Request parameter definitions:

targetConnId: ushort	Id for further reference to the target between the **ECI Host** and Micro Server.
target: EciEncrTarget	ID representing the target for authentication. The value is defined within the scope of the Micro DRM system. Matching by the **ECI Host** is defined in terms of equality of **targetType** and **target** fields.
credLen: ushort	Length of the cred parameter in bytes.
cred: byte[]	The credential information from the target to be authenticated by the **Micro Server**.

Response parameter definitions:

targetConnId: ushort	Id for further reference to the target between the **ECI Host** and **Micro Server**.

Detailed semantics:

- In case **targetConnId** is equal to a **targetConnId** previously used by the **ECI Host,** but not dropped afterwards, the implication is that the previous target associated with targetId is replaced or updated.

Pre conditions Request:

1) The **target** should be equal to a T**arget** previously provided to the **ECI Host** by the **Micro Server** in a **resEncrTargets** message. Otherwise an error is returned for this parameter.

2) **Target** should match to a target provided by the **Micro Client** and permit authentication using **cred**.

Post condition Response:

1) The authentication status is returned. Note that the result is not necessarily conclusive and might e.g. provide the wrong credentials resulting in encrypted content that cannot be decoded.

2) The **ECI Host** can refer to the (pre-) authenticated target by **targetConnId**.

Table 9.7.2.5.4-1: reqEncrConnSetup Error Codes

Name	Description
ErrEncrAuthFail	See Table 9.7.2.5.19-1.
ErrEncrAuthInconclusive	

9.7.2.5.5 reqEncrConnDrop Message

H→C reqEncrConnDrop(ushort **targetConnId)** →
C→H resEncrConnDrop(ushort **targetConnId)**

- This message permits the **ECI Host to** request the **Micro Server** to drop any information on a previously pre-authenticated re-encryption connection.

Request parameter definitions:

targetConnId: ushort	Id for of the target connection to be removed by the Micro Server.

Response parameter definitions:

targetConnId: ushort	Id for of the target connection removed from the Micro Server.

Pre conditions Request:

1) The **targetConnId** should exist in the **Micro Server**.

Pre condition Response:

1) The **Micro Server** does not associate **targetConnId** to a pre-authenticated target connection any longer and has released any resources associated with the pre-authentication of **targetConnId**.

9.7.2.5.6 reqEncrConnCancel Message

C→H reqEncrConnCancel(ushort **targetConnId)** →
H→C resEncrConnDrop(ushort **targetConnId)**

- This message permits the **Micro Server** to inform in the **ECI Host** that it has cancelled a previously pre-authenticated re-encryption connection.

Request parameter definitions:

targetConnId: ushort	Id for of the target connection that was cancelled by the Micro Server.

Response parameter definitions:

targetConnId: ushort	Id for of the target connection that was cancelled by the Micro Server.

Pre conditions Request:

1) The **targetConnId** should exist in the **Micro Server**.

Pre condition Response:

1) TargetConnId value has been deallocated and may be reallocated by the **ECI Host** as part of a subsequent reqEcnrConnSetup message.

9.7.2.5.7 reqEncrMhOpen Message

H→C reqEncrMhOpen(ushort **mh,** ushort **impConn,** ushort **targetConnId,** EncrMode **mode)** →
C→H resEncrMhOpen(ushort **mh)**

- This message permits the **ECI Host** to request the **ECI Client** to open a **Media Handle** session to re-encrypt content under control of the **Micro Server** from an incoming **Import Connection** for forwarding to a pre-authenticated target. Error codes are defined in Table 9.7.2.5.7-1.

Request parameter definition:

mh: ushort	**Media Handle** for the encryption session to be opened, allocated by the **ECI Host**.
impConn: ushort	Id of the input connection from which the content is to be re-encrypted.
targetConnId: ushort	Id of the target connection for which the content is to be re-encrypted.
mode: EncrMode	Single mode (encryption-mode, data-format-mode, synchronization-mode) specification for the **Micro Server** to operate from, selected from the Micro Server modes capabilities as indicated with **setEncrModes**.

Response parameter definition:

mh: ushort	**Media Handle** for the encryption session to be opened, allocated by the **ECI Host**.

Pre conditions Request:

1) **The ECI Host** has reserved all required resources for the session to be created.

2) impConn and targetId are established by the **ECI Host** with the **Micro Server**.

Pre conditions Response:

1) In case of a successful result the **Micro Server** has reserved all resources typically required for re-encrypting content for the requested session. This should include access to any external resources (DRM servers, **Smart Cards**, etc.) typically required for a decryption operation.

NOTE: Resources required by exception or resources that can normally be attained when required are excluded.

2) In case **ErrEncrUserDelay** is returned the **Micro Server** is pending **user** input to open the session (e.g. to get access to a **Smart Card** or acquire a **user** authentication). The **ECI Host** can repeat sending the **reqEncrMhOpen Request** (with the same parameters) until a positive result is returned or a definitive error is returned or alternatively may send a **reqEncrMhClose** to terminate the pending session. The **Micro Server** may cancel with **reqEncrMhCancel** in case it cannot attain the required **user** input.

Table 9.7.2.5.7-1 reqEncrMhOpen Error Codes

Name	Description
ErrEncrUserMissing	See Table 9.7.2.5.19-1.
ErrEncrCardMissing	
EnrEncrServiceMissing	
ErrEncrResourceMissing	
ErrEncrMmiMissing	
ErrEncrClientAuthError	

9.7.2.5.8 reqEncrMhClose Message

H→C reqEncrMhClose(ushort **mh)** →
C→H resEncrMhClose(ushort **mh)**

- This message permits the **ECI Host** to close a **Re-encryption Session** with the **Micro Server**.

Request parameter definition:

mh: ushort	**Media Handle** for the encryption session to be closed.

Response parameter definition:

mh: ushort	**Media Handle** for the encryption session to be closed.

Pre conditions Request:

1) The **Media Handle** session is in an opened state (or an error will occur).

Pre conditions Response:

1) The resources required by the **Micro Server** for maintaining the session are released.

2) **mh** state is closed by the Client.

9.7.2.5.9 reqEncrMhCancel Message

C→H reqEncrMhCancel(ushort **mh,** uchar **reason)** →
H→C resEncrMhCancel(ushort **mh)**

- This message permits the **ECI Client** to close a **Re-encryption Session** with the specified exporting **ECI Client (Micro Server)**.

Request parameter definition:

mh: ushort	**Media Handle** for the encryption session that is cancelled by the Micro Server.
reason: uchar	Reasons for cancelling the decryption session. The values are defined in Table 9.7.2.5.9-1.

Table 9.7.2.5.9-1: reqEncrMhCancel reason values

Name	Value	Description
EncrMhUndefined	0x00	An undefined error occurred in the **Micro Server** requiring it to cancel the session.
EncrMhCardMissing	0x01	**Smart Card** is required for re-encryption but could not be successfully (re-)connected and assist in re-encrypting content within a reasonable time.
EncrMhServiceMissing	0x02	A service (external to the **CPE**) supporting the **Micro Server** in providing decryption services required to maintain a decryption session is not available in a reasonable time.
EncrMhResourceMissing	0x03	A resource (internal to the **CPE**) required for providing re-encryption services is not available to the **Micro Server** within a reasonable time (not including **DcrMhMmiMissing**).
EncrMhMmiMissing	0x04	The **Micro Server** was not successful in attaining an MMI session resource for **user** interaction required for maintaining the **Re-encryption Session** within a reasonable time.
RFU	Other	Reserved for future use.

Response parameter definition:

mh: ushort	**Media Handle** for the encryption session that is cancelled.

Pre condition Request:

1) The **ECI Client** has released any resources it required specifically for the session.

Post conditions Request:

1) The **ECI Host** may release any resources related to the **Media Handle**.

Post conditions Response:

1) The **Media Handle** session is closed by the **ECI Host**.

9.7.2.5.10 reqEncrMhStart Message

H→C reqEncrMhStart(ushort **mh)** →
C→H resEncrMhStart(ushort **mh)**

- This message permits the **ECI Host to** requests the **Micro Server** to start the re-encryption operation for a **Media Handle** session.

Request parameter definition:

mh: ushort	**Media Handle** for the encryption session to start.

Response parameter definition:

mh: ushort	**Media Handle** for the encryption session that was started.

Pre conditions Request:

1) The **Media Handle** session is in an opened state (or an error will occur).

Pre conditions Response:

1) The **Media Handle** session is started or (or an error occurred).

Detailed semantics:

- The encryption of content will proceed as the content is provided by the exporting **ECI Client**.
- Any URI conflicts or failures of the exporting **ECI Client** to authenticate the **Micro Server** for export of the content will not produce encrypted content, the output control URI status of the **Micro Server** set at OcAnyOther equal 0b1, all other output control bits shall be set to 0b0 (meaning no output permitted). The **Micro Server** will continue to attempt to encrypt content as and when this is permitted.
- Any initialization messages the **Micro Client** are made available through the respective messages for that purpose. For sessions with re-encryption mode equal **OfflineStreamMode** the first initialization data for decrypting the content is produced shortly following the **resEncrMhStart** message.
- Sending a second **reqEncrMhStart** before ending the encryption process will end the previous process and start the next process.

9.7.2.5.11 reqEncrMhStop Message

H→C reqEncrMhStop(ushort **mh)** →
C→H resEncrMhStop(ushort **mh)**

- This message permits the **ECI Host to** requests the **Micro Server** to stop the re-encryption operation for a **Media Handle** session.

Request parameter definition:

mh: ushort	**Media Handle** for the encryption session to end.

Response parameter definition:

mh: ushort	**Media Handle** for the encryption session that was ended.

Pre conditions Request:

1) The **Media Handle** session is in an started state (or an error will occur).

Pre conditions Response:

1) The **Media Handle** session is ended.

Post conditions Response:

1) The **Media Handle** session value can be reused by the **ECI Host**.

Detailed semantics:

- On sessions with encryption-mode equal **OfflineStorageMode** the final decryption data is produced before the **Micro Server** sends **resEncrMhStop**. This also holds for any final decryption data that may be needed for decryption in other types of sessions.

9.7.2.5.12 reqEncrMhQuit Message

C→H reqEncrMhQuit(ushort **mh,** uchar **reason)** →
C→H resEncrMhQuit(ushort **mh)**

- This message permits the **Micro Server to** inform the **ECI Host** that the **Media Handle** associated re-encryption operation was terminated.

Request parameter definition:

mh: ushort	**Media Handle** for the encryption session that has been terminated.
reason: uchar	Reason as given in Table 9.7.2.5.9-1.

Response parameter definition:

mh: ushort	**Media Handle** for the encryption session that has been terminated.

Pre conditions Request:

1) The **Media Handle** session was in an started state but is now terminated.

Pre conditions Response:

1) The **ECI Host** is aware of the non-started state of encryption of the session.

Detailed semantics:

- In case the error s of a quasi-permanent nature the **Micro Server** can also cancel the **Media Handle** session itself.
- In case the **Micro Server** can produce valid decryption data before terminating the **Re-encryption Session**, on sessions with encryption-mode equal **OfflineStorageMode** the final decryption data is produced before the **Micro Server** sends **resEncrMhQuit**. This also holds for any final decryption data that may be needed for decryption in other types of sessions.

9.7.2.5.13 reqEncrIpServer Message

H→C reqEncrIpServer(ushort **mh)** →
C→H resEncrIpServer(ushort **mh.** Addrinfo **addr)**

- This message permits the **ECI Host** to request the **Micro Server** to provide the target IP address for incoming IP connections from **Micro Clients**.

Request parameter definition:

mh: ushort	**Media Handle** for the encryption session for which an IP address for incoming messages or connections is required.

Response parameter definition:

mh: ushort	**Media Handle** for the encryption session for which an IP address for incoming messages or connections is required.
addr: Addrinfo	IP protocol/address/port for the incoming messages or connections of a **Micro Client**.

Pre conditions Request:

1) The **Media Handle** session is opened in mode **OnlineIpMode**.

Pre conditions Response:

1) The **ECI Host** is aware of the non-started state of encryption of the session.

Detailed semantics:

- The IP exchange between **Micro Client** and **Micro Server** is specific to the **Micro DRM System**. This includes protocol choice and any convention for terminating a connection or exchange on a content streaming session.
- This message may be issued on a **Media Handle** session on which the re-encryption process has not yet been started.

Table 9.7.2.5.13-1: reqEncrIpServer Error Codes

Name	Description
ErrEncrIpNone	See Table 9.7.2.5.19-1.

9.7.2.5.14 reqEncrMsgSend Message

C→H reqEncrMsgSend(ushort **mh,** uint **length,** byte **msg[]) →**
C→H resEncrMsgSend(ushort **mh)**

- This message permits the **Micro Server** to request the **ECI Host** to forward a message to the target **Micro Client** or **Micro Clients** (in case of a group target) associated with the **Media Handle**.

Request parameter definition:

mh: ushort	**Media Handle** for the encryption session for which a message has to be forwarded to the target **Micro Client**.
length: uint	Length of the **msg** field in bytes.
msg[]: byte	Message to be forwarded to the **Micro Client**.

Response parameter definition:

mh: ushort	**Media Handle** for the encryption session.

Pre conditions Request:

1) The **Media Handle** session is opened in mode **OnlineMsgMode**.

Pre condition Response:

1) The message has been forwarded to the **Micro Client**; the **ECI Host** is ready to accept a new **reqEncrMsgSend**.

Detailed semantics:

- The **ECI Host** shall be able to process and forward at least one message at a time to the **Micro Client**. The messages should be delivered in order. The **ECI Host** is not obliged to provide any specific buffering for more than one simultaneous outstanding **reqEncrMsgSend** request. A safe **Micro Server** implementation should use the **resEncrMsgSend** as a control flow handshake.
- The **ECI Host** forwarding mechanism shall have a reliability that is sufficient for regular applications not to fail (message loss, or disordering of one per 10 000). Applications in which essential access information for encrypted content may be permanently lost or during which and/or high value viewing may be impaired are recommended to take additional application level precautions.

9.7.2.5.15 reqEncrMsgRecv Message

H→C reqEncrMsgRecv(ushort **mh,** uint **length,** byte **msg[]) →**
C→H resEncrMsgRecv(ushort **mh)**

- This message permits that the **ECI Host** provides the **Micro Server** with a message from the target **Micro Client**.

Request parameter definition:

mh: ushort	**Media Handle** for the encryption session for which the **Micro Server** gets a message from the target **Micro Client**.
length: uint	Length of the **msg** field in bytes.
msg: byte[]	Message to be received by the **Micro Server**.

Response parameter definition:

mh: ushort	**Media Handle** for the encryption session for which an IP address for incoming messages or connections is required.

Pre conditions Request:

1) The **Media Handle** session is opened in mode **OnlineMsgMode**.

Pre condition Response:

1) The message has been processed by the **Micro Server** and it is ready to accept a new **reqEncrMsgRecv**.

Detailed semantics:

- The **Micro Server** shall process at minimum one message at a time. The **Micro Server** is not obliged to provide any specific buffering for more than one simultaneous outstanding **reqEncrMsgSend** request, though it should take care, it is ready to process a subsequent message respecting its other responsiveness requirements. A safe **ECI Host** implementation should use the **resEncrMsgRecv** as a control flow handshake.
- The reliability of the forwarding service between **Micro Client** and **Micro Server** is as defined for **reqEncrMsgSend** in clause 9.7.2.5.14.

9.7.2.5.16 reqEncrTsData Message

C→H reqEncrTsData(ushort **mh,** TsSync **sync,** uint **length,** byte **msg[]) →**
C→H resEncrTsData(ushort **mh)**

- This message permits the **Micro Server to** provide the **ECI Host** with data to be forwarded to the target **Micro Client** of a **Media Handle** to enable content decryption, including ECM related synchronization information.

Request parameter definition:

mh: ushort	**Media Handle** for the encryption session.
sync: TsSync	Synchronization of this information relative to an ecmId associated with the content. The details are given in Table 9.7.2.5.16-1.
length: uint	Length in bytes of the message to be forwarded.
msg: byte[]	Message to be forwarded to the **Micro Client**.

Table 9.7.2.5.16-1: TsSync typedef definition

```
typedef struct TsSync {
   uint  ecmId;
   uint  precTime;
} TsSync;
```

Field definitions:

ecmId: uint	Identification number of an ECM associated with the content that this data message for the **Micro Client** should precede.
precTime: uint	Real time in terms of 100 ms units, with a maximum of 300 seconds, in terms of content playback time that this message should precede the application of an ECM with ecmId to the content decoding process.

Response parameter definition:

mh: ushort	**Media Handle** for the encryption session for which an IP address for incoming messages or connections is required.

Pre conditions Request:

1) The **Media Handle** session is opened, the session is in *re-encryption-mode* **OfflineStream** or **OfflineStorage** mode**,** uses *data-format-mode* **OfflineDataMode** and *synchronization-mode* **SyncTs**.

Pre conditions Response:

1) The **ECI Host** is ready to receive a next data message.

Detailed semantics:

- The **ECI Host** should ensure that the **Micro Client** is provided with the data in line with the synchronization requirements along with the encrypted content.
- The **ECI Host** shall buffer the data of the message appropriately (as associated data to the content) and should respond to the next within the time period as defined in ETSI GR ECI 004 [i.10].
- The **Micro Server** may produce one or more data messages ahead of a started **Re-encryption Session** when operating in **OfflineStream** mode.
- The **Micro Server** shall produce at most one data message at the end of the encryption session in **OfflineStorage** mode. This data message may be preceded by the ECM it is supposed to synchronize to. Hence "offline Storage" mode. Typically this data message should be processed ahead of any content and ECMs by the **Micro Client**.

9.7.2.5.17 reqEncrTsEcm Message

C→H reqEncrTsEcm(ushort **mh,** uint **ecmId,** uint **length,** byte **ecm[]) →**
C→H resEncrTsEcm(ushort **mh)**

- This message permits the **Micro Server** to issue an ECM section that is required to decrypt in the next crypto-period.

Request parameter definition:

mh: ushort	**Media Handle** for the encryption session.
ecmId: uint	Identification number of the ECM assigned by the Micro Server for the purpose of synchronizing data messages.
length: uint	Length of the **ecm** parameter in bytes; the ecm has a single section format.
ecm: byte[]	ECM message to be inserted in the next crypto period.

Response parameter definition:

mh: ushort	**Media Handle** for the encryption session.

Pre conditions Request:

1) The **Media Handle** session is opened, the session uses *synchronization-mode* **SyncTs**.

Pre conditions Response:

1) The **ECI Host** is ready to insert the next ECM.

Detailed semantics:

- **ECI Host** shall insert the ECM in the Transport Stream within the time slot defined in ETSI GR ECI 004 [i.10] after receiving the message. The ECM shall be repeated at a reasonable interval (as defined in ISO/IEC 13818-1-1 [8]. The ECM PID shall be a free PID and is generated by the **ECI Host**.

- The **ECI Host** may update any PMT information in the stream may reflect the ECM PID or shall otherwise forward the ECM PID information to permit a **Micro Client** to later recover the required decryption information.

- On a content item change and/or another higher layer encryption change the **Micro Server** may issue two successive but different ECM messages for the same forthcoming crypto period. The **ECI Host** shall at minimum insert the last one for the remainder of the period. In time-shift/storage mode it shall insert the last ECM for the full crypto-period.

9.7.2.5.18 reqEncrFileData Message

H→C reqEncrMsgRecv(ushort **mh,** byte **syncKid[**MaxUuidLen]**,** uint **datalength,** byte **data[])**
C→H resEncrMsgRecv(ushort **mh)**

- This message permits the **ECI Host** provides the **Micro Server** with a message from the target **Micro Client**.

Request parameter definition:

mh: ushort	**Media Handle** for the encryption session.
syncKid [MaxUuidLen**]:** byte	KeyId that will be used to encrypt the next "fragment" of the file for which the associated data is required by the **Micro Client** for decryption.
datalength: uint	Length of the data in bytes.
data[]**:** byte	Data destined for the **Micro Client** for decryption purposes. The format of the data is opaque if the data-format-mode is **OfflineDataMode** and is a PSSH Box for inclusion in an ISOBMFF MOOV or MOOF box in case the data-format-mode is **OfflineIsobmffMode**.

Response parameter definition:

mh: ushort	**Media Handle** for the encryption session.

Pre conditions Request:

1) The **Media Handle** session is opened, the session is in *re-encryption-mode* **OfflineStream** or **OfflineStorage** mode and *synchronization-mode* **SyncFile**.

Pre conditions Response:

1) The **ECI Host** is ready to receive a next data message.

Detailed semantics:

- The **ECI Host** has to ensure that any target **Micro Client** is provided with the data in line with the synchronization requirements along with the encrypted content.

- The **ECI Host** shall create a valid ISOBMFF file including the provided PSSH box or otherwise ensure that the data is passed along with the file content to the **Micro Client** and provided to the **Micro Client** in line with the data synchronization requirements.

- **reqEncrMsgRecv** The **ECI Host** shall buffer the data of the message appropriately (as associated data to the content). Response time requirements are defined in ETSI GR ECI 004 [i.10].

- The **Micro Server** may produce one or more data messages ahead of a started **Re-encryption Session** when operating in **OfflineStream** mode.

- The **Micro Server** shall produce at most one data message at the end of the encryption session in **OfflineStorage** mode. Typically this data message has to be processed ahead of any content by the **Micro Client**.

9.7.2.5.19 Error Codes for the Re-encryption API

Table 9.7.2.5.19-1: Error codes for Re-encryption API

Name	Value	Description
ErrEncrAuthInconclusive	1	The authentication was only processed partially and was not conclusive, but no error occurred.
ErrEncrAuthFail	-256	It was not possible to identify the parental authentication status of the content item but the parental authentication was performed and found correct.
ErrEncrUserMissing	-257	**User** does not provide essential input to the **Micro Server** to proceed or continue with re-encryption of content.
ErrEncrCardMissing	-258	**Smart Card** is required for re-encryption but could not be successfully (re-)connected and assist in re-encrypting content within a reasonable time.
EnrEncrServiceMissing	-259	A service (external to the **CPE**) supporting the **Micro Server** in a decryption session is not available in a reasonable time.
ErrEncrResourceMissing	-260	A unspecified resource inside the **CPE** required for processing and/or re-encrypting the content is not available.
ErrEncrMmiMissing	-261	**Micro Server** access to the MMI is required but not available.
ErrEncrClientAuthError	-262	**Micro Server** fails to authenticate the target **Micro Client**.
ErrEncrIpNone	-263	**Micro Server** cannot provide an IP address for **Micro Client** communication.

9.7.2.6 Micro Client De-encryption API

9.7.2.6.1 General

The **Micro Client** decryption API permits a **Micro Client** to de-encrypt content from Micro Server.

The discovery phase permits a **Micro Client** to publish the decryption targets for which it can offer decryption services and can provide the credentials by which a **Micro Server** can create an authenticated connection to it as a target.

The **Micro Client** has to support modes of decryption that cover the encryption modes offered by its complementary **Micro Server**. Based on one of the commonly supported modes the **Micro Client** can decrypt services: this is based on the common decryption API.

Additional support messages to pass data for required decryption back and forth between the Micro Server and the **Micro Client** for the various modes are part of this API.

The messages for the **Micro Client** decryption API are listed in Table 9.7.2.6.1-1.

Table 9.7.2.6.1-1: Decryption API messages

Message	Type	Dir.	Tag	Description
setDcrModes	set	C→H	0x0	The **Micro Client** informs the **ECI Host** about the modes (encryption-modes, data-format-modes and synchronization-modes) it supports.
reqDcrTargets	A	H→C	0x1	The **ECI Host** requests the **Micro Client** to provide the encryption targets it can decrypt services for.
reqDcrTargetCred	A	H→C	0x2	The **ECI Host** requests the **ECI Client** to provide the initialization data for a Micro Server connection typically used for authentication of the target.
reqDcrIpServer	A	C→H	0xA	The **Micro Client** to request the **ECI Host** to provide the **Micro Server's** IP address for further communication related to the **Media Handle** session.
reqDcrMsgSend	A	C→H	0xB	The **Micro Client** requests the **ECI Host** to send a message to the **Micro Server** of a **Media Handle** session.
reqDcrMsgRecv	A	H→C	0xC	The **ECI Host** provides the **Micro Client** with a message from the **Micro Server** of a **Media Handle** session.
reqDcrTsData	A	C→H	0xD	The **Micro Server** provides the **ECI Host** with data to be forwarded to the target **Micro Client** of a **Media Handle** for decryption, including ECM related synchronization information.
reqEncrFileData	A	C→H	0xF0	The **Micro Server** provides the **ECI Host** with a message to be forwarded to the target **Micro Client** of a **Media Handle** for decryption, including KeyID related synchronization information.

9.7.2.6.2 setDcrModes Message

C→H setDcrModes(EciEncrModes **modes)**

- This message permits the **Micro Client** to inform the **ECI Host** about the modes (encryption-modes, data-format-modes and synchronization-modes) it supports.

Request Parameter definitions:

modes: EciEncrModes	Modes of decryption that are supported by the **Micro Client**. The type EciEncrModes is specified in Table 9.7.1.5.2-1.

9.7.2.6.3 reqDcrTargets Message

H→C reqDcrTargets() →
C→H resDcrTargets(EncrTarget **target[])**

- This message permits the **ECI Host** to request the **Micro Client** to provide the encryption targets for which it can decrypt.

Response Parameter definitions:

target[]: EncrTarget	List of encryption targets that the Micro Server can authenticate. The type definition of TargetClient is specified in Table 9.7.2.5.2-1.

Detailed semantics:

- The **ECI Host** can match potential target **Micro Clients** based on T**arget**. It is up to the application and/or **ECI Host** to locate potential candidate **Micro Clients**.

9.7.2.6.4 reqDcrTargetCred Message

H→C reqDcrTargetsCred(EncrTarget **target)** →
C→H reqDcrTargetsCred(uint **credLen**, byte **cred[]**)

- This message permits the **ECI Host** to request the **Micro Client** to provide credentials for encryption by a **Micro Server**.

Request Parameter definitions:

target: EncrTarget[]	The encryption target for which the **Micro Client** has to provide the actual credentials for encryption of the content by a **Micro Server**.

Response parameter definitions:

credLen: uint	Length of the cred parameter in number of bytes.
cred[]: byte	Credentials encoded in a format specific for the **Micro Server** that will encrypt the content to be decrypted by the **Micro Client**.

Detailed semantics:

- This message allows the **ECI Host** to request a **Micro Client** to provide credentials corresponding to the target parameter so that a **Micro Server** recognizing target can encrypt content for the **Micro Client**.

9.7.2.6.5 reqDcrIpServer Message

C→H reqDcrIpServer(ushort **mh)** →
C→H resDcrIpServer(ushort **mh,** Addrinfo **addr)**

- This message permits the **Micro Client** to request the **ECI Host** to provide the **Micro Server's** IP address for further communication related to the **Media Handle** session. Related error codes are defined in Table 9.7.2.6.5-1.

Request parameter definition:

mh: ushort	**Media Handle** for the decryption session for which a **Micro Server** IP address sending/receiving messages is requested.

Response parameter definition:

mh: ushort	**Media Handle** for the decryption session for which a **Micro Server** IP addresses sending/receiving messages is provided.
addr: Addrinfo	IP protocol/address/port for the Micro Server for this **Media Handle**.

Pre conditions Request:

1) The **Media Handle** session is opened in mode **OnlineIpMode**.

Pre conditions Response:

1) The **ECI Host** is aware of the non-started state of encryption of the session.

Detailed semantics:

- The IP exchange between **Micro Client** and **Micro Server** is specific to the **Micro DRM System**. This includes protocol choice and any convention for terminating a connection or exchange on a content streaming session.

- This message may be issued on a **Media Handle** session on which the re-encryption process has not yet been started.

Table 9.7.2.6.5-1: reqDcrIpServer Error Codes

Name	Description
ErrDcrIpNone	See Table 9.7.2.6.10-1.

9.7.2.6.6 reqDcrMsgSend Message

C→H reqDcrMsgSend(ushort **mh,** uint **length,** byte **msg[]) →**
C→H resDcrMsgSend(ushort **mh)**

- This message permits the **Micro Client** to request the **ECI Host** to forward a message to the target **Micro Server** associated with the **Media Handle**.

Request parameter definition:

mh: ushort	**Media Handle** for the decryption session for which a message has to be forwarded to the **Micro Server**.
length: uint	Length of the **msg** field in bytes.
msg[]: byte	Message to be forwarded to the **Micro Server**.

Response parameter definition:

mh: ushort	**Media Handle** of the encryption session.

Pre conditions Request:

1) The **Media Handle** session is opened in mode **OnlineMsgMode**.

Pre condition Response:

1) The message has been forwarded to the **Micro Server**; the **ECI Host** is ready to accept a new **reqDcrMsgSend**.

Detailed semantics:

- The **ECI Host** shall be able to process and forward at least one message at a time to the **Micro Server**. The messages should be delivered in order. The **ECI Host** is not obliged to provide any specific buffering for more than one simultaneous outstanding **reqDcrMsgSend** request. A safe **Micro Client** implementation should use the **resDcrMsgSend** as a control flow handshake.
- The reliability of the forwarding service between **Micro Server** and **Micro Client** is as defined for **reqEncrMsgSend** in clause 9.7.2.5.14.

9.7.2.6.7 reqDcrMsgRecv Message

H→C reqDcrMsgRecv(ushort **mh,** uint **length,** byte **msg[]) →**
C→H resDcrMsgRecv(ushort **mh)**

- This message permits that the **ECI Host** provides the **Micro Client** with a message from the target **Micro Server**.

Request parameter definition:

mh: ushort	**Media Handle** for the decryption session for which the **Micro Client** gets a message from the **Micro Server**.
length: uint	Length of the **msg** field in bytes.
msg[]: byte	Message to be received from the **Micro Server**.

Response parameter definition:

mh: ushort	**Media Handle** for the decryption session.

Pre conditions Request:

1) The **Media Handle** session is opened in mode **OnlineMsgMode**.

Pre condition Response:

1) The message has been processed by the **Micro Client** and it is ready to accept a new **reqDcrMsgRecv**.

Detailed semantics:

- The **Micro Client** shall process at minimum one message at a time. The **Micro Client** is not obliged to provide any specific buffering for more than one simultaneous outstanding **reqDcrMsgSend** request, though it should take care, it is ready to process a subsequent message respecting its other responsiveness requirements. A safe **ECI Host** implementation should use the **resDcrMsgRecv** as a control flow handshake.

- The reliability of the forwarding service between **Micro Client** and **Micro Server** is as defined for **reqEncrMsgSend** in clause 9.7.2.5.14.

9.7.2.6.8 reqDcrTsData Message

H→C reqDcrTsData(ushort **mh,** uint **length,** byte **msg[]) →**
C→H resDcrTsData(ushort **mh)**

- This message permits the **ECI Host** to provide the **Micro Client** with data required in the (near) future for decrypting the content on **Media Handle**.

Request parameter definition:

mh: ushort	**Media Handle** for the decryption session.
length: uint	Length in bytes of the message to be forwarded.
msg[]: byte	Message to be forwarded to the **Micro Client**.

Response parameter definition:

mh: ushort	**Media Handle** for the decryption session.

Pre conditions Request:

1) The **Media Handle** session is opened, the session is in *re-encryption-mode* **OfflineStream** or **OfflineStorage** mode**,** uses *data-format-mode* **OfflineDataMode** and *synchronization-mode* **SyncTs**.

Pre conditions Response:

1) The **ECI Host** is ready to receive a next data message.

Detailed semantics:

- The **ECI Host** should ensure that the **Micro Client** is provided with the data in line with the synchronization requirements provided by the **Micro Server** along with the encrypted content to be decrypted.

- The **Micro Client** shall receive at most one data message at the start of the decryption session in **OfflineStorage** mode. Hence "offline Storage" mode.

9.7.2.6.9 reqDcrFileData Message

H→C reqDcrFileData(ushort **mh,** uint **datalength,** byte **data[])**
C→H reqDcrFileData(ushort **mh)**

- This message permits the **ECI Host** to provide the **Micro Client** with a data from the target **Micro Server** required to decrypt content for the **Media Handle**.

Request parameter definition:

mh: ushort	**Media Handle** for the decryption session.
datalength: uint	Length of the data in bytes.
data[]: byte	Data destined for the **Micro Client** for decryption purposes. The format of the data is opaque if the data-format-mode is **OfflineDataMode** and is a PSSH Box for inclusion in an ISOBMFF MOOV or MOOF box in case the data-format-mode is **OfflineIsobmffMode**.

Response parameter definition:

mh: ushort	**Media Handle** for the encryption session.

Pre conditions Request:

1) The **Media Handle** session is opened, the session is in re-encryption-mode OfflineStream or OfflineStorage mode and synchronization-mode SyncFile.

Pre conditions Response:

1) The Micro Client is ready to receive a next data message.

Detailed semantics:

- The **ECI Host** has to ensure that the **Micro Client** is provided with the data in line with the synchronization requirements along with the encrypted content.
- The **ECI Host** might extract a PSSH box form a valid ISOBMFF file and provided to the **Micro Client** in line with the data synchronization requirements for decoding ISOBMFF files.
- The **ECI Host** shall provide at most one data message at the end of the encryption session in **OfflineStorage** mode. Typically this data message has to be processed ahead of any content by the **Micro Client**.

9.7.2.6.10 Error Codes for the Micro Client De-encryption API

- The error codes for the Micro Client De-encryption API are listed in Table 9.7.2.6.10-1.

Table 9.7.2.6.10-1 Micro Client De-encryption API related Error Codes

Name	Value	Description
ErrDcrIpNone	-256	The **ECI Host** has no IP address/port for communicating to the Micro Server.

9.8 APIs for content property related resources

9.8.1 List of APIs defined in clause 9.8

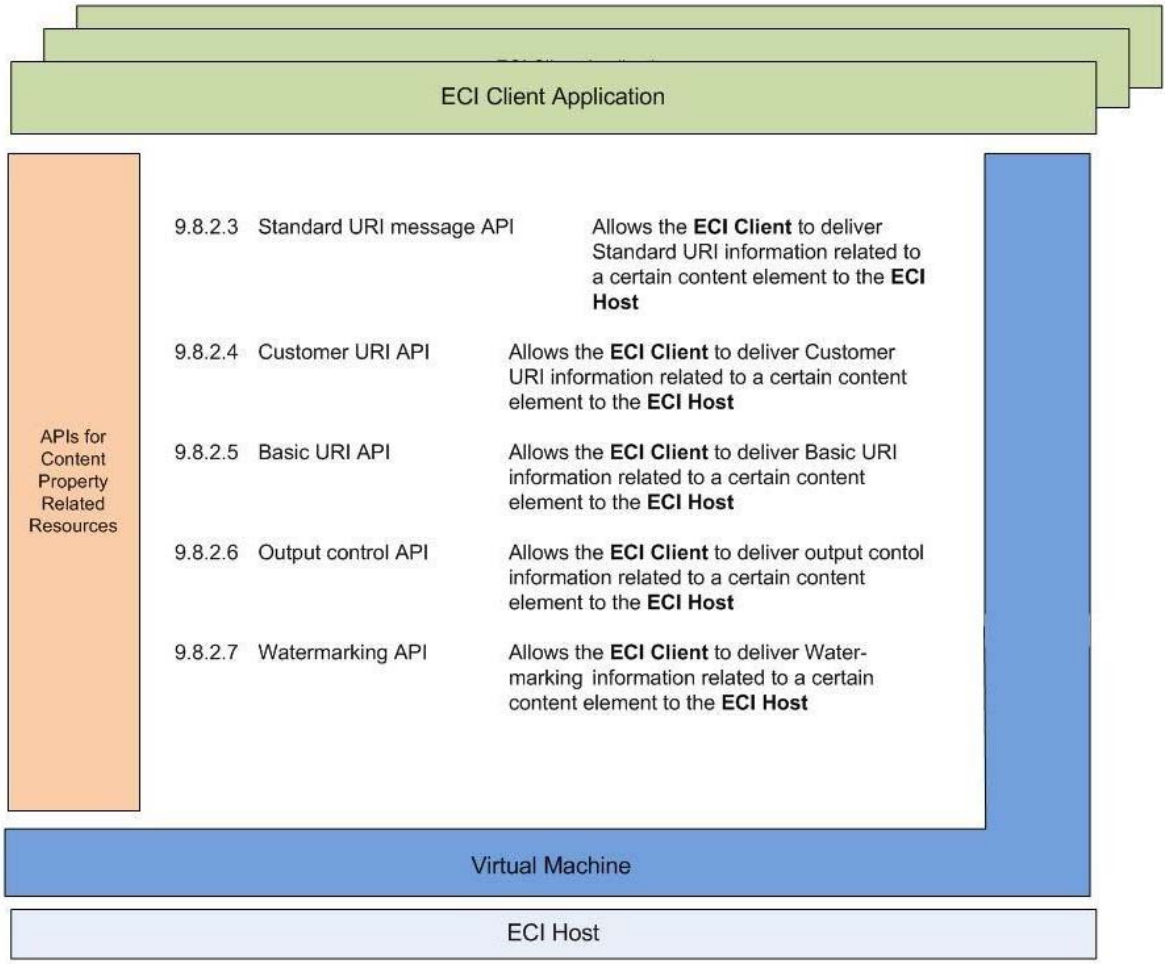


Figure 9.8.1-1: Block diagram of the APIs defined in clause 9.8

Table 9.8.1-1 lists the APIs covered in clause 9.8 and Figure 9.8.1-1 illustrates the location of the APIs defined in clause 9.8 with the **ECI architecture**.

Table 9.8.1-1: APIs for content protection related resources

Clause	API name	Description
9.8.2.3	Standard URI message API	Allows the **ECI Client** to deliver Standard URI information related to a certain content element to the **ECI Host** and vice versa.
9.8.2.4	Customer URI API	Allows the **ECI Client** to deliver **User** URI information related to a certain content element to the **ECI Host** and vice versa.
9.8.2.5	Basic URI API	Allows the **ECI Client** to deliver Basic URI information related to a certain content element to the **ECI Host** and vice versa.
9.8.2.6	Output control API	Allows the **ECI Client** to deliver output control information related to a certain content element to the **ECI Host** and vice versa.
9.8.2.7	Watermarking API	Allows the **ECI Client** to deliver Watermarking information related to a certain content element to the **ECI Host** and vice versa.
9.8.2.8	Parental Control API	Allows an **ECI Client** to deliver information on Parental Control Obligations associated to a certain content element to the **ECI Host**.
9.8.2.9	Content Property Sync API	Permits synchronization of various content property changes.
9.8.2.10	Parental Authentication API	Allows an **ECI Client** to delegate parental authentication to a standard parental authentication function in the **ECI Host**.
9.8.2.11	Parental Authentication delegation API	Allows an **ECI Client** to cancel a delegated parental authentification request

9.8.2 APIs for access to the usage rights and parental control resource

9.8.2.1 Introduction

This clause of the **ECI Client/Host** APIs permits the **ECI Client** to set the rights and conditions applying to decrypted content in a secure manner.

The rights and conditions API specifies the following aspects:

- Standard URI (Usage Rights Information): generated by the **ECI Client** and used by the **ECI Host** to control the applications of the content on industry standard outputs and applications.
- Basic URI: generated by the **ECI Client** and used by the **Advanced Security** and hardware subsystem of the **ECI Host** to set basic usage rights for the content. This permits the **ECI Client** to use robust hardware protection for basic rights properties that need to be in place on the content.
- Output Control: This permits the **ECI Client** to block outputs selectively that could be active under the conditions of the URI but which are nevertheless deemed inappropriate for use from a rights perspective.
- **ECI Host**-driven Watermarking Control: this permits the **ECI Client** to mark outgoing content with **ECI Client** specified marks through a **CPE** resident watermarking system.
- Parental Control conditions permit the **ECI Client** to forward the requirement to authenticate a parent to grant the access to the content to the protection system to which the content is exported.
- Content Property synchronization permits several content property changes occur simultaneously to be identified as such.
- The Parental Authentication function itself can be performed by an **ECI Client** itself or be delegated to a central industry standard function in the **ECI Host**. The **ECI Host** may in turn select a specific **ECI Client** to perform parental authentication on its behalf. The delegation options serve to permit one single parental authentication across multiple **ECI Clients** and the **ECI Host**.

The application of the new rights properties is securely linked to the application of a new control word to descramble the content. This ensures that rights are applied to the content they are associated with.

The content property APIs have a set and a get message. The set message is used by **ECI Clients** that decrypt content to signal the content properties associated with the next control word that is computed. The get function is used by **Micro Servers** that re-encrypt content to acquire the content properties of the incoming content for the purpose of constructing the appropriate authentication and signalling data for signalling content properties of the re-encrypted content.

The API version signalled as part of the Discovery API effectively aligns the version of the content properties that are used.

The **ECI Host** media handle context shall maintain at minimum two values for different content sections for each content property. So specifically for file based decryption it shall maintain at least two content sections each decoded with a separate KeyID for each content property. Table 9.8.2.1-1 lists the API functions. The rights API functions are grouped into separate APIs to permit independent version management.

Table 9.8.2.1-1: List of messages of the usage rights and parental control API

API	Message	Type	Dir.	Tag	Description
ApiStdUri	setDcrStdUri	set	C→H	0x0	Set standard URI for content to be descrambled.
ApiStdUri	getEncrStdUri	get	C→H	0x1	Get standard URI for content to be re-encrypted.
ApiCustUri	setDcrCustUri	set	C→H	0x0	Set custom URI for content to be descrambled.
ApiCustUri	getEncrCustUri	get	C→H	0x1	Get custom URI for content to be re-encrypted.
ApiBasicUri	setDcrBasicUri	set	C→H	0x0	Set Basic URI for content to be descrambled.
ApiBasicUri	getEncrBasicUri	get	C→H	0x1	Get Basic URI for content to be re-encrypted.
ApiOC	setDcrOutputCtl	set	C→H	0x0	Set Output Control restrictions for content to be descrambled.
ApOC	getEncrOutputCtrl	get	C→H	0x1	Get Output Control restrictions for content to be re-encrypted.
ApiDcrMark	getDcrMarkSyst	get	H→C	0x0	Get supported marking systems.
ApiDcrMark	setDcrMarkMeta	set	C→H	0x1	Set a marking system control value.
ApiDcrMark	getDcrMarkMeta	get	H→C	0x2	Read a marking system property.
ApiDcrMark	setDcrMarkBasic	set	C→H	0x3	Set basic marking payload for content to be descrambled.
ApiDcrMark	setDcrMarkExt	set	C→H	0x4	Set extended marking payload for content to be descrambled.
ApiPar	setDcrParCtl	set	C→H	0x0	Set Parental Control conditions for content to be descrambled.
ApiPar	getEncrParCtrl	get	C→H	0x1	Get Parental Control conditions for content to be descrambled.
ApiCpSync	setCpSync	set	C→H	0x0	**ECI Client** signals that the current set of content properties is coherent and can be applied to the content to be descrambled by forthcoming control word.
ApiCpSync	reqCpChange	req	H→C	0x1	**ECI Host** signals that a change is forthcoming in the content properties of content to be re-encrypted.
ApiParAuth	reqParAuthChk	req	C→H	0x0	Request to the **ECI Host** to perform a parental authentication on behalf of the **ECI Client**.
ApiParAuth	reqParAuthChkCan	req	C→H	0x1	Cancels a preceding parental authentication request to the Host.
ApiParAuth	reqParAuthCid	req	H→C	0x2	Requests parental pin code authorization for a (future) content item to be decoded. This may trigger a parental authentication dialogue.
ApiParAuthDel	reqParAuthDel	req	H→C	0x0	The **ECI Host** delegates a parental authentication to an **ECI Client**.
ApiParAuthDel	reqParAuthDelCan	req	H→C	0x1	The **ECI Host** cancels a preceding parental authentication request to the **ECI Client**.

9.8.2.2 Security Aspects & synchronization

The **ECI** specification permits content property information above to be authenticated by the **ECI Host** so as to prevent unauthorized manipulation of this information. This mechanism also ensures that the appropriate rights settings are applied to the content to which they are associated to. This is defined in ETSI GS ECI 001-5-1 [4].

For content property information the **ECI Host** can facilitate authentication of the rights information on behalf of the **ECI Client** using keys in the Advanced Security Block, thus ensuring the highest level of integrity for authentication. It is up to **ECI Clients** to use the **ECI Host**'s AS services appropriately for this purpose. This is defined in ETSI GS ECI 001-5-1 [4] as well.

9.8.2.3 Standard URI message API

9.8.2.3.1 setDcrStdUri Message

C→H setDcrStdUri(ushort **mh,** byte **keyId**[MaxUuidLen], StdUri **stdUri**)

- This message sets the Standard URI associated with **keyId** to **uri**.

Parameter definition:

mH: ushort	**Media Handle** of the content to be decoded
keyId[MaxUuidLen]**:** byte	KeyID as a UUID in network byte order to which the URI applies in case of file format decoding, byte 0 carrying 0x00 (even) or 0x01 (odd) for TS format streams to indicate applicability to the next CW.
stdUri: StdUri	Standard URI for content is defined in Table 9.8.2.3.1-1. The semantics of the fields correspond to those defined in ETSI TS 103 205 [34] and [i.7].

Table 9.8.2.3.1-1: Standard URI type specification

```
typedef struct StdUri {

   uint MajorVersion: 4;
   uint tmc: 1; /* trick_mode_control_info in [CI+ v1.4] */
   unit reserved1: 3;
   uint aps: 2; /* aps_copy_control_info in [CI+ v1.4] */
   uint emi: 2; /* emi_copy_control_info in [CI+ v1.4] */
   uint ict: 1; /* ict_copy_control_info in [CI+ v1.4] */
   uint rct: 1; /* rct_copy_control_info in [CI+ v1.4] */
   uint reserved2: 1; /* reserved bit */
   uint dot: 1; /* dot_copy_control_info in [CI+ v1.4] */
   uint rl: 8;     /* rl_copy_control_info in [CI+ v1.4] */

} StdUri;

The following rules shall apply (expressions over field shall evaluate to True)
in line with [CI+v1.4]
emi == 0b00 || rct == 0b0
emi == 0b11 || (dot == 0b0 && rl == 0x00)
emi == 0b01 || tmc == 0b0

The protocol_version field value 0x03 is defined for the definition above;
other values are reserved for future use.
```

StdUri Field semantics:

MajorVersion: uint: 4	Major version of this standard URI. **ECI Client**s shall set MajorVersion to 0b0000. **ECI Hosts** shall implement all versions up to their compliance level for this field, and shall interpret any high value as a URI not implemented and thus no usage rights shall apply.
reserved1: unit: 3	Reserved bits. Shall be set to 0b000 by the **ECI Client** and shall be ignored by **ECI Hosts** complying to this version of stdUri.
reserved2: unit: 3	Reserved bit. Shall be set to 0b0 by the **ECI Client** and shall be ignored by **ECI Hosts** complying to this version of stdUri.
Other fields	The semantics are as defined for the indicated fields of the CI Plus v1.4 URI [34] in the above structure definition.

Detailed semantics:

- For Transport Stream descrambling mode the URI shall apply to the content to be decoded with the keys applying to the next **reqDcrTsDescrKey**.
- **ECI Client** shall be in decryption mode.

9.8.2.3.2 getEncrStdUri Message

C→H StdUri getEncrStdUri(ushort **mh,** byte **keyId**[MaxUuidLen])

- This message sets the standard URI for forthcoming content.

Property definition:

- The standard URI is as defined in Table 9.8.2.3.1-1.

Parameter definition:

mH: ushort	**Media Handle** of the content to be encrypted.
keyId: byte[MaxUuidLen]	KeyID as a UUID in network byte order to which the URI applies in case of file format decoding, byte 0 carrying 0x00 (even) or 0x01 (odd) for TS format streams to indicate applicability to the next CW.

Detailed semantics:

- **ECI Client** shall be in encryption mode.

9.8.2.4 Customer URI API

9.8.2.4.1 setDcrCustUri Message

C→H setDcrCustUri(ushort **mh,** byte **keyId**[MaxUuidLen], unit **custUriLen**, byte ***custUri)**

- This message sets a custom URI associated with **keyId** to **uri**.

Parameter definition:

mH: ushort	**Media Handle** of the content to be decoded.
keyId: byte[MaxUuidLen]t	KeyID as a UUID in network byte order to which the URI applies in case of file format decoding, byte 0 carrying 0x00 (even) or 0x01 (odd) for TS format streams to indicate applicability to the next CW.
custUriLen: unit	Length in bytes of custom URI field.
custUri: byte *	Custom URI for content is defined in Table 9.8.2.4.1-1. Byte 0 and 1 shall act as msB, and lsB of the custom URI format. All values of byte 0 and 1 are reserved, except 0x80, 0x00 which shall mean an application specific meaning to the following bytes.

Table 9.8.2.4.1-1: Custom URI type specification

Name	Value byte 0, 1	Description
CustUriPrivate	0x80, 0x00	The meaning of the bytes following byte 1 this is private. The appropriate interpretation of the rest of the field is defined through other communication between **ECI Client** and **Micro Server** or protection system.
RFU	Other	Reserved for future use.

Detailed semantics:

- For Transport Stream descrambling mode the URI shall apply to the content to be decoded with the keys applying to the next **reqDcrTsDescrKey**.
- A maximum of 4 separate customs URIs are allowed to be set for one control word.
- **ECI Client** shall be in decryption mode.

9.8.2.4.2 getEncrCustUri Message

C→H custUri getEncrCustUri(ushort **mh,** byte **keyId**[MaxUuidLen], unit **custUriMaxLen)**

- This message gets the custom URI for forthcoming content.

Property definition:

- The custom URI is as defined in Table 9.9.1-1.

Parameter definition:

mH: ushort	**Media Handle** of the content to be encrypted.
keyId: byte[MaxUuidLen]	KeyID as a UUID in network byte order to which the URI applies in case of file format decoding, byte 0 carrying 0x00 (even) or 0x01 (odd) for TS format streams to indicate applicability to the next CW.
custUriMaxLen: uint	Maximum length (in bytes) of custom URI result; any additional content is to be truncated.

Detailed semantics:

- **ECI Client** shall be in encryption mode.

9.8.2.5 Basic URI API

9.8.2.5.1 setDcrBasicUri Message

C→H setDcrBasicUri(ushort **mh,** byte **keyId**[MaxUuidLen], BasicUri **basicUri**)

- This message sets the basic URI associated with **keyId** to **basicUri**. The basic URI provides simplified but highly robust rights management for decrypted content.

Parameter definition:

mH: ushort	**Media Handle** of the content to be decoded.
keyId[MaxUuidLen]**:** byte	KeyID as a UUID in network byte order to which the URI applies in case of file format decoding, byte 0 carrying 0x00 (even) or 0x01 (odd) for TS format streams to indicate applicability to the next CW.
basicUri: BasicUri	Basic URI for content is defined in Table 9.8.2.5.1-1. The semantics of the fields correspond to those defined in ETSI TS 103 205 [34].

Table 9.8.2.5.1-1: Basic URI type specification

```
typdef byte  BasicUri;
```

Name	Bits	Description
BasicUriVersion	7	Major version of basic URI. If the **ECI Host** has not implemented the version the **ECI Host** shall not permit the content to be decrypted and used. Value 0b0 defines version 0. Al;l other values are reserved and are not permitted.
BasicUriV0_0Ext	2..6	Reserved for future use, not used in v0.0. The only value defined for this field is 0b00000. Other values are not permitted. **ECI Hosts** implementing only basic Uri v0.0 shall ignore the values of this field: i.e. this may be used for future backward compatible extensions of v0.0; e.g. in the form of relaxations on v0.0 rights control.
BasicUriV0_0	0,1	Basic URI version 0.0. The values and meanings of this field are defined in Table 9.8.2.5.1-2.

Table 9.8.2.5.1-2: Basic URI V0.0 definition

Name	Value	Description
NoBasicProtection	0b00	No rights control through basic URI
RedistributionProtected	0b01	Encryption shall be on, replay prevention off
ViewOnly	0b10	Encryption shall be on, replay prevention shall be on
ViewOnlyStrict	0b11	Encryption shall be on; replay prevention shall be on, output shall be restricted to specifically qualified (secure) outputs.

Detailed semantics:

- For Transport Stream descrambling mode the URI shall apply to the content to be decoded with the keys applying to the next **reqDcrTsDescrKey**.
- The basic URI enables control by the **ECI Client** over rights implementation that is at the highest level of robustness supported by the **ECI Host**. It provides control over two protection mechanisms: encryption, which ensures the content is always scrambled on any output or storage medium, and replay prevention which ensures the encrypted content can only be descrambled in a live connection (i.e. cannot be stored). For specifics see ETSI GS ECI 001-5-2 [5].
- **ECI Client** shall be in decryption mode.

9.8.2.5.2 getEncrBasicUri Message

C→H BasicUri getEncrBasicUri(ushort **mh,** byte **keyId**[MaxUuidLen])

- This message gets the basic URI for forthcoming content.

Property definition:

- The basic URI is as defined in Table 9.8.2.5.1-1.

Parameter definition:

mH: ushort	**Media Handle** of the content to be encrypted.
keyId[MaxUuidLen]**:** byte	KeyID as a UUID in network byte order to which the URI applies in case of file format decoding, byte 0 carrying 0x00 (even) or 0x01 (odd) for TS format streams to indicate applicability to the next CW.

Detailed semantics:

- **ECI Client** shall be in encryption mode.

9.8.2.6 Output control API

9.8.2.6.1 setDcrOutputCtl Message

C→H setDcrOutputCtl(ushort **mh,** byte **keyId**[MaxUuidLen], uint **ocVector**)

- Set the Output Control settings associated with **keyId** to **ocVector**.

Parameter definition:

mH: ushort	**Media Handle** of the content to be decoded.
keyId[MaxUuidLen]**:** byte	KeyID as a UUID in network byte order to which the URI applies in case of file format decoding, byte 0 carrying 0x00 (even) or 0x01 (odd) for TS format streams to indicate applicability to the next CW.
ocVector: unit	Output Control vector for standard outputs as defined per Table 9.8.2.6.1-1.

Table 9.8.2.6.1-1: Output Control Vector specification

Name	Bits	Description
MajorVersion	7	Version of the ocVector parameter. Value 0b0 is defined for version 1. Any other value is reserved and is not permitted. If an **ECI Host** implementing **Major Version** 1 receives a value other than 0xb0 this shall mean no output is permitted.
OcAnyOther	6	Any other **ECI Host** output not covered by any of the output qualification criteria listed below. If the value is 0b0 output is permitted on these outputs, if the value is 0b1 output shall not be permitted. ***The value of this bit changes the encoding of the fields below.*** If the value is 0b0 the output restrictions shall be as listed below. If the value is 0b1 the encoding of shall be shall be bit-wise inverted. I.e. if OcAnyOther==0b1 and OcIP==0b1 output on the IP connection is permitted. See note 2.
OcIP	0	Output on any IP connection is permitted if value is 0b0, and shall not be permitted in value is 0b1.
OcUSB	1	Output on any USB connection is permitted if the value is 0b0, and shall not be permitted if the value is 0b1. Precondition for this is, that the decrypted content is not protected by any **ECI** recognized output protection system and/or an **ECI** micro DRM system under the control of the decrypting **ECI Client**.
OcDtcpIp	2,3	Output on a DTCP-IP protected connection is permitted if value is 0b0, and shall not be permitted in value is 0b1.
OcHdcp	3,4	Any HDCP protected output. For OcAnyOther equal to 0b0: • value 0b00: HDCP protected output permitted. • value 0b01: if the HDCP version is below 2.2 output shall not be permitted, if the HDCP version is 2.2 or higher output is permitted. • value 0b10: reserved; this value is not permitted. **ECI Hosts** shall interpret this value as equal to 0b11. • value 0b11: no HDCP protected output shall be permitted. For OcAnyOther equal to 0b1: • value 0b00: no HDCP output shall be permitted. • value 0b01: reserved, **ECI Hosts** shall interpret this value as equal to 0b00. • value 0b10: If the HDCP version 2.2 or higher output shall be permitted, if the HDCP output version is less than 2.2 no output shall be permitted. • value 0b11: any HDCP protected output is permitted.
OcWm	5	If the value of this bit is 0b1, output of the decoded content element is permitted only with the application of a watermark inserted by the **CPE** in the related content element. See note 3.
NOTE 1: Analogue output control is effectively provided by the standard URI **dot** and **ict** fields. NOTE 2: OcAnyOther effectively switches the output control field from a blacklist of outputs (when value equal 0b0) to a whitelist of outputs (when value equal 0b1). If an output field is 0b1 it means it is effectively "on the list". NOTE 3: Watermarking systems suitable for this application can be subject to approval.		

In case multiple ocVector fields apply to an output (e.g. an IP output protected by DTCP-IP) the most restrictive condition shall apply.

Detailed semantics:

- **ECI Client** shall be in decryption mode.

9.8.2.6.2 getEncrOutputCtrl Message

C→H uint getEncrOutputCtrl(ushort **mh,** byte **keyId**[MaxUuidLen])

- This message gets the output control for forthcoming content.

Property definition:

- The output control is as defined in Table 9.8.2.6.1-1.

Parameter definition:

mH: ushort	**Media Handle** of the content to be encrypted.
keyId[MaxUuidLen]**:** byte	KeyID as a UUID in network byte order to which the URI applies in case of file format decoding, byte 0 carrying 0x00 (even) or 0x01 (odd) for TS format streams to indicate applicability to the next CW.

Detailed semantics:

- **ECI Client** shall be in encryption mode.

9.8.2.7 Watermarking API

9.8.2.7.1 General

The marking API permits **ECI Clients** to discover embedded (water) marking systems available through the **ECI Host**, and then engage in a "setup" control dialogue with such systems. The marking systems may be able to engage in a dialogue with only a limited number of **ECI Clients** and may be able to mark only a limited number of **Media Handle** sessions simultaneously.

Marking systems can wish to engage with authorized **ECI Clients**. Such authorization can amongst others be established using the setMarkMeta and getMarkMeta messages using an authorization dialogue defined by the marking system.

ECI Clients may reserve access to a marking system by completing a successful engagement dialogue. This **ECI Client** (as identified by its ECI client id) shall remain engaged with the marking system until it is removed from the **CPE** or until it disengages.

9.8.2.7.2 getDcrMarkSyst Message

C→H MarkSystDescr **getDcrMarkSyst(**)

- This message allows the **ECI Client** to read the descriptors for the available marking systems.

Property definition:

The result type MarkSystDescr shall comply to the definition in Table 9.8.2.7.2-1.

Table 9.8.2.7.2-1: MarkSystDescr type definition

```
#define MaxMarkSystDescr 16;

typedef ushort MarkId;  /* ECI Marking ID allocated to a marking system */
// markId values: 0x8xxx are used for proprietary marking systems.
//                0x0000 shall mean no marking system
//                All other values are reserved by ECI, allocation of new
//                IDs and their publication is defined elsewhere.

typedef struct MarkSystDescrElem {
   MarkID markId;        /* ID of the marking system */
   uchar  nrClients;     /* number of Clients that can still be supported */
   uchar  markSystFlags  /* field as defined below */
} MarkSystDescr[MaxMarkSystDescr];
// Any available marking systems shall be listed as the first elements
// of MarkSystDescr. The remaining elements shall use markId==0x0000.
```

```
// markSystFlags:
//   bit 0 signals authorization required (0b1) or not (0b0)
//   bit 1 signals scrambled stream support (0b1) or not (0b0)
//   bit 2 signals multi simultaneous stream support (0b1) or not (0b0)
//   other bits are reserved and shall be ignored by Clients complying
//   to the present document
```

9.8.2.7.3 setDcrMarkMeta Message

C→H setDcrMarkMeta(MarkID **markId,** uchar **index,** byte **data[32])**

- **This message enables the ECI Host** to set control (meta) data for a marking system.

Parameter definition:

markId: MarkID	Marking system ID for which to set the Property Definition.
index: uchar	Sub property to be set for marking systems.
data[32]: byte	Value to apply to the sub property indicated by index.

9.8.2.7.4 getDcrMarkMeta Message

C→H byte[32] getDcrMarkMeta(MarkID **markId**, uchar **index)**

- This message enables the **ECI Client** to get control (Meta) data for a marking system.

Property Definition:

- Meta data for sub property **index** system with mark ID **markID**.

Parameter definition:

markId: MarkID	Marking system ID for which to read the Property definition: the result type MarkSystDescr shall comply with the definition in Table 9.8.2.7.4-1.
index: uchar	Subproperty of marking system to read.

9.8.2.7.5 SetDcrMarkBasic Message

C→H setDcrMarkBasic(ushort **mH,** byte **keyId**[MaxUuidLen], MarkID **markId**, byte **data**[16])

- This message enables the **ECI Client** to set max. 128 bits of data used to mark the content to be descrambled with the designated key.

Parameter definition:

mH: ushort	**Media Handle** of the content to be decoded.
keyId[MaxUuidLen]**:** byte	KeyID as a UUID in network byte order to which the URI applies in case of file format decoding, byte 0 carrying 0x00 (even) or 0x01 (odd) for TS format streams to indicate applicability to the next CW.
markId: MarkID	Marking system ID.
data[16]**:** byte	128 bit value.

9.8.2.7.6 SetDcrMarkExt Message

C→H setDcrMarkExt(ushort **mH,** byte **keyId**[MaxUuidLen], ushort **markId**, uint **dataLen**, byte **data**[]**)**

- **This message enables the ECI Client** to set an extended payload for marking system for marking content to be descrambled with the designated key.

Parameter definition:

mH: ushort	**Media Handle** of the content to be decoded.
keyId: byte[MaxUuidLen]	KeyID as a UUID in network byte order to which the URI applies in case of file format decoding, byte 0 carrying 0x00 (even) or 0x01 (odd) for TS format streams to indicate applicability to the next CW.
markId: ushort	Marking system ID to use for marking the content.
datalen: uint	Length of the data field.
Data[]: byte	Payload data for marking system.

9.8.2.8 Parental Control API

9.8.2.8.1 setDcrParCtl Message

C→H setDcrParCtl(ushort **mH,** byte **keyId**[MaxUuidLen], ParCond **pC)**

- **This message enables the ECI Client** to set the parental rating conditions (**pC**) for content of mH to be descrambled with the designated key.

Parameter definition:

mH: ushort	**Media Handle** of the content to be decoded.
keyId[MaxUuidLen]**:** byte	KeyID as a UUID in network byte order to which the parental control condition **pC** applies in case of file format decoding, byte 0 carrying 0x00 (even) or 0x01 (odd) for TS format streams to indicate applicability to the next CW.
pC: ParCond	Parental Control conditions to be applied on the content. See Table 9.8.2.8.1-1 for the definition of ParCond.

Table 9.8.2.8.1-1: Parental Condition type specification

```
typdef struct ParCond {
   byte basicCondition; /* see Table 9.8.2.8.1-2 */
   byte extendedQualifier[16];
} ParCond;
```

Table 9.8.2.8.1-2: Parental Condition basic condition definition

Name	Bits	Description
AuthRequired	7	0b1 means parental authentication is required before rendering the content. 0b0 means parental authentication may be required depending on extendedQualifier.
ToggleBit	6	This bit alternates in a stream to indicate a new parental authentication requirement on a change of the value of the bit.
Reserved	4,5	Shall be set to 0b00
QualifierFormat	0..3	Indicates the format of the extendedQualifier field. Value 0x0 indicates "no value", the extendedQualifier field shall be set to zeros, Value 0x1 indicates that the ExtendedQualifier field contains a DVB Parental Rating Descriptor as defined in ETSI EN 300 468 [19]. Remaining bytes shall have value zero. Parental authentication shall be required even if AuthRequired==0b0 in case the required rating for the applicable country exceeds the limit set by the parent (as defined by the semantics of the DVB parental rating descriptor). Values 0x2..0xF are reserved for future use.

Detailed semantics:

- **ECI** permit parental rating authentication conditions to be passed along with the content as an obligation to a system protecting the descrambled content.

- **ECI Client** shall be in decryption mode.

9.8.2.8.2 getEncrParCtrl Message

C→H ParCond getEncrParCtrl(ushort **mh,** byte **keyId**[MaxUuidLen])

- This message enables the **ECI Client** to get the parental control condition for forthcoming content.

Property definition:

- The parental control URI is as defined in Table 9.8.2.8.1-2.

Parameter definition:

mH: ushort	**Media Handle** of the content to be encrypted.
keyId[MaxUuidLen]**:** byte	KeyID as a UUID in network byte order to which the URI applies in case of file format decoding, byte 0 carrying 0x00 (even) or 0x01 (odd) for TS format streams to indicate applicability to the next CW.

Detailed semantics:

- **ECI Client** shall be in encryption mode.

9.8.2.9 Control Property Sync API

9.8.2.9.1 setCpSync Message

C→H setCpSync(ushort **mH,** byte **keyId**[MaxUuidLen]**)**

- This message signals to the **ECI Host** that the forthcoming content section indicated by keyId will have the content properties set through the standard URI, custom URI, basic URI, Output Control, Watermarking and Parental control APIs.

Parameter definition:

mH: ushort	**Media Handle** of the content to be decoded.
keyId[MaxUuidLen]**:** byte	KeyID as a UUID in network byte order to which the parental control condition **pC** applies in case of file format decoding, byte 0 carrying 0x00 (even) or 0x01 (odd) for TS format streams to indicate applicability to the next CW.

Detailed semantics:

- The message shall trigger the **ECI Host** to appropriately prepare for the forthcoming changes in content properties. This shall include sending a reqCpChange message to any Micro Server with an import/export connection to this media handle session.
- **ECI Client** shall be in decryption mode.

9.8.2.9.2 reqCpChange Message

H→C reqCpChange(ushort **mh,** byte **keyId**[MaxUuidLen])

- This message triggers the Micro Server to prepare a content property change based on the most recent future values for the content properties of the decrypted content that is re-encrypted by the **Micro Server**.

Property definition:

- The parental control URI is as defined in Table 9.8.2.8.1-2.

Parameter definition:

mH: ushort	**Media Handle** of the content to be encrypted.
keyId[MaxUuidLen]**:** byte	KeyID as a UUID in network byte order to which the URI applies in case of file format decoding, byte 0 carrying 0x00 (even) or 0x01 (odd) for TS format streams to indicate applicability to the next CW.

Detailed semantics:

- **ECI Client** shall be in encryption mode.
- The **ECI Client** shall get the content properties for the forthcoming content related to KeyId in the decrypted stream and prepare a new encryption setup for the new content (which can require a new CW).

9.8.2.10 Parental Authentication API

9.8.2.10.1 General

Authentication for parental approval may be performed directly by an **ECI Client** using an MMI session. Alternatively an **ECI Client** can request the **ECI Host** to perform (or have performed) the parental authentication, so as to harmonise the pin code management as well as improve the **user** interface experience by integrating the pin requests naturally in the **ECI Host** user interface. In turn the **user** through the **ECI Host** may select an **ECI Client** among available candidates to perform the parental authentication using the parental authentication delegation API ParAuthDel as defined in clause 9.8.2.11. This can be useful in case an **ECI Client** handling many content items cannot delegate its parental authentication but can perform parental authentication on behalf of the **ECI Host**.

This API also permits an **ECI Client** to start a parental authentication for a content item ahead of opening a media session e.g. for parental authentication of a future recording event.

9.8.2.10.2 Standard Parental Authentication Function

This clause defines a set of requirements for a standard parental rating function based on 4-character pin-codes that an **ECI Host** shall be able to perform if requested by an **ECI Client** or that an **ECI Client** shall perform on behalf of the **ECI Host** in case it offers such a service through the Parental Authentication Delegation API.

An **ECI Host** or **ECI Client** may provide an alternative authentication function than the one described in the sequel of this clause if such function provides at least the parental authentication integrity of the mechanism defined in this clause.

The following functionalities apply to the standard pin code based parental authentication mechanism:

1) Parental authentication is based on a pin code of at least 4 alphanumerical characters from a minimum set of at least 10 characters (e.g. digits).

2) The pin code setting shall be protected by the pin code itself or by a master authentication mechanism which protects access to assets or services of material value which are deemed highly inappropriate for access by minors from which content may need to be protected.

3) Any applicable parental rating limits setting shall be protected by the pin code or by a master authentication mechanism as per 2) here above.

4) Requirements on a potential master authentication mechanism shall create an authentication integrity of at least that of the pin code mechanism defined in this clause without being based on a master authentication mechanism.

5) At purchase of a Host the initial pin code for parental rating or the means to authenticate with the master authentication shall be passed to the owner only.

6) At installation of a new Client the platform operator shall pass the initial pin code or the means to authenticate with the master authentication to the owner only.

7) The **Manufacturer** or a custodian acting on its behalf may provide a means to reset the pin code to its initial value or provide a service by which the owner can have the pin code set to a new value which will be passed to the owner only.

8) The platform operator behalf may provide a means to reset the pin code to its initial value or provide a service by which the owner can have the pin code set to a new value which will be passed to the owner only.

9) In case of a 5 successively failed authentications within 15 minutes the parental authentication function shall refuse to perform a new authentication for at least 15 minutes.

10) It shall not be possible to recover or reset the pin code through application of regular **user** software, downloaded applications running on the **CPE** or any user interface or regular interfaces.

9.8.2.10.3 reqParAuthChk Message

C→H reqParAuthChk(ushort **mH) →**
C→H resParAuthChk(ushort **mH,** bool **ok)**

- This message allows the **ECI Client** to request the **ECI Host** to perform a parental authentication check using the standard parental authentication function of the **ECI Host** (see clause 9.8.2.10) and to return the result in a response message.

Request parameter definition:

mH: ushort	**Media Handle** of the content to be decoded.

Response parameter definition:

mH: ushort	**Media Handle** of the content to be decoded.
ok: bool	True corresponds to authentication succeeded, False otherwise including timeout.

Detailed semantics:

- Only one outstanding parental authentication check per **Media Handle** shall be distinguished by the **ECI Host**. Issuing a second request on the same **Media Handle** before the previous one was responded to or cancelled will result in two identical **Responses**.

- **reqParAuthChk** The **ECI Host** should use a timeout value for requesting parental authentication that will terminate within a reasonable period if there is no person present or willing to perform the authentication as defined in ETSI GR ECI 004 [i.10].

9.8.2.10.4 reqParAuthChkCan Message

C→H reqParAuthChkCan(ushort **mH) →**
H→C resParAuthChkCan(ushort **mH)**

- **ECI Client** cancels any previous requests to the **ECI** for parental authentication.

Request parameter definition:

mH: ushort	**Media Handle** of the content to be decoded.

Response parameter definition:

mH: ushort	**Media Handle** of the content to be decoded.

Post condition response:

1) The response to a previous **reqParAuthChk** message **may** be returned by the **ECI Host** to the **ECI Client** before the **resParAuthChkCan** message, but not thereafter.

9.8.2.10.5 reqParAuthCid Message

H→C reqParAuthCid(uint **cidLength,** byte **cid**[]**)** →
C→H resParAuthCid(bool **ok)**

- This message **allows t**he **ECI Host** to request the **ECI Client** to perform any required authentication for a future content item identified by **cid**.

Request parameter definition:

cidLength: uint	Length of the cid parameter.
cid[]: byte	Identification of the content to be subjected to parental authentication (if so required). The first byte indicated the format of the content identification parameter, as defined in Table 9.8.2.10.5-1.

Table 9.8.2.10.5-1: Content identification formats

Name	Value	Description
CidDvbEvent	0x01	DVB Event identification. The bytes following bytes in cid have the value of the sequence: original network id (2 bytes), transportstream id (2 bytes), service id (2 bytes), event id (2 bytes) as defined in the EIT table as defined in ETSI EN 300 468 [19]. All 2 byte fields in the sequence are represented in network order (most significant byte first).
RFU	other	Reserved for future use.

Response parameter definition:

ok: bool	True if parental authentication was successful or is not required.

Detailed semantics

- The **ECI Client** shall maintain a non-volatile record of content identifications that have been authenticated with this function. It may discard the oldest records and records that are no longer in the future in case it lacks storage space. The minimum requirements for this content identification buffering are defined in ETSI GR ECI 004 [i.10].

The related error codes are listed in Table 9.8.2.10.5-2.

Table 9.8.2.10.5-2: Error codes media session API for TS media

Name	Value	Description
ErrParAuthCidUnknOk	1	It was not possible to identify the parental authentication status of the content item but the parental authentication was performed and found correct.

The above error statuses may also be returned in case access to the required network resources was not available.

9.8.2.11 Parental Authentication delegation API

9.8.2.11.1 General

This API permits an **ECI Client** to indicate that it can perform a standard parental authentication function as defined in clause 9.8.2.10.2 and for the **ECI Host** to delegate pin code verifications to such an **ECI Client**.

An **ECI Client** may indicate support for the delegated authentication API using the configuration API at **ECI Client** initialization time. NOTE: at the same time an **ECI Client** may choose not to delegate its own parental authentication due to e.g. commercial, security or legal considerations.

The **ECI Host** shall offer a setup function to permit the **user** to select the **ECI Host** for standard parental control authentication or to delegate the standard parental control authentication to one of the **ECI Clients** offering this function.

9.8.2.11.2 reqParAuthDel Message

H→C reqParAuthDel(ushort **mh**) →
C→H resParAuthDel(ushort **mH,** bool **ok)**

- This message allows the **ECI Host** to request the **ECI Client** to perform a delegated parental authentication on its behalf for content on mH.

Request parameter definition:

mH: ushort	**Media Handle** of the content to be decoded.

Response parameter definition:

mH: ushort	**Media Handle** of the content to be decoded.
ok: bool	True if the parental authentication was successful, false if not so or if there was a timeout.

Detailed semantics:

- Only one outstanding parental authentication check per **Media Handle** shall be distinguished by the **ECI Client**. Issuing a second request on the same **Media Handle** before the previous one was responded to or cancelled will result in two identical responses.
- The **ECI Client** should use a timeout value for requesting parental authentication that will terminate within a reasonable period if there is no person present or willing to perform the authentication as defined in ETSI GR ECI 004 [i.10].

9.8.2.11.3 setParAuthDelCan Message

H→C reqParAuthDelCan(ushort **mH**) →
C→H resParAuthDelCan(ushort **mH)**

- This message allows the **ECI Host** to cancel a delegated parental authentication request.

Response parameter definition::

mH: ushort	**Media Handle** of the content to be decoded.

Response parameter definition:

mH: ushort	**Media Handle** of the content to be decoded.

Post condition response:

- The response to a previous reqParAuthDel message may be returned by the **ECI Host** to the **ECI Client** before the resParAuthDelCan message, but not thereafter.

9.9 APIs for ECI Client and Application communication

9.9.1 List of APIs defined in the this clause

Table 9.9.1-1: APIs for content protection related resources

Clause	API name	Description
9.9.2	Inter client communication API	Enables an **ECI Client** to establish a direct communication path to another **ECI Client**.

Table 9.9.1-1 lists the APIs covered in this clause.

9.9.2 Inter client communication API

9.9.2.1 General

The **ECI Host** offers an environment with standardized exchange of information between **ECI Clients** in the form of import/export information, URIs and content. **ECI Clients** may communicate amongst themselves in order to provide additional (at present not **ECI** defined) functionality. **ECI Clients** can register their principle ability and willingness to support inters client communication through the discovery resource (see clause 9.2). After system initialization, they can read the identities of other **ECI Clients** including the established import/export connections. The **ECI Clients** can open a communication channel (called pipe) to a potential counterpart and exchange messages over the pipe. Both sides can cancel the pipe. The pipe of an **ECI Client** is closed by the **ECI Host** on halting and/or re-initialization of its counterpart **ECI Client**.

The **ECI Host** provides **ECI Client** identities that are authenticated using the **ECI Certificate Chains** provided with the **ECI Clients**. **ECI Clients** shall provide an additional independent authentication mechanism in case communication with a counterpart can lead to security hazards.

In case of communication between an **ECI Client** decoding content and another **ECI Client** subsequently re-encrypting that content (a Micro Server) the recommendation for setting up a pipe is that the pipe is initiated (opened) by the Micro Server.

Table 9.9.2.1-1 shows the messages of the Inter client communication API.

Table 9.9.2.1-1: Inter Client Communication API Messages

Message	Type	Dir.	Tag	Description
getIccMaxClients	S	C→H	0x0	The **ECI Client** reads the maximum number of **ECI Clients** that the **ECI Host** may support.
reqIccSystemReady	A	H→C	0x1	The **ECI Host** informs the **ECI Client** that all **ECI Clients** are initialized.
getIccClientInfo	S	C→H	0x2	The **ECI Client** reads the identity and connection status of another **ECI Client** in the system.
reqIccPipeOpen	A	C→H	0x3	Request to open a pipe to another **ECI Client**.
reqIccPipeOpenReq	A	H→C	0x4	Incoming request from another **ECI Client** to open a pipe.
reqIccPipeCancel	A	C→H	0x5	**ECI Client** cancels the pipe.
reqIccPipeClose	A	H→C	0x6	**ECI Host** informs **ECI Client** that the pipe with the counterpart was closed.
reqIccPipeMsgSend	A	C→H	0x7	**ECI Client** sends a message to its counterpart of a pipe
reqIccPipeMsgRecv	A	H→C	0x8	**ECI Client** receives a message from its counterpart of a pipe.

9.9.2.2 getIccMaxClients Message

C→H uint getIccMaxClients()

- Gets the maximum number of **ECI Clients** that can be supported by the **ECI Host**.

Property Definition:

- Unsigned integer representing the maximum number of **ECI Clients** that the **ECI Host** can support.

9.9.2.3 reqIccSystemReady Message

H→C reqIccSystemReady()

- The **ECI Host** informs the **ECI Client** that all other **ECI Clients** are initialized.

Semantics:

- This message is provided at system initialization to indicate to all **ECI Client** registered to this API that it is possible to start reading the client Information Registry and attempt to open pipes to other **ECI Clients**.
- The ConnId field in the result reflects the latest status of the import/export connections of the **ECI Client** with a potential counterpart. These can be subject to change.
- No result message is required.

9.9.2.4 getIccClientInfo Message

C→H ClientInfo getIccClientInfo(ushort **clientId)**

- The **ECI Client** reads the identity and connection status of another **ECI Client** in the system.

Parameter Definition:

clientId: ushort	Id of the client for setting up pipes. This identifier does not change over the lifecycle of the system. It changes on reinitialization.

Property Definition:

- The connectionID is a dynamic property.
- ClientInfo is a structure providing the identity of the designated **ECI Client** and any import/export connections with that **ECI Client**. It is defined below.

Type definition for ClientInfo:

```
#define MaxConnId 32

typedef struct ClientInfo {
  ECI_Operator_Id operatorId;
  ECI_Platform_Operation_Id platformOperationId;
  ECI_Vendor_Id vendorId;
  union {
    ECI_Client_Series_Id clientSeriesId;
    ECI_Client_Id clientId;
  } client;
  ushort connId[MaxConnId];
}
```

Field definitions:

operatorId: ECI_Operator_Id	Operator ID of the **ECI Client**.
platformOperationId: ECI_Platform_Operation_Id	Platform Operation ID of the **ECI Client**.
client: union	Either an ECI_Client_Series_Id or a ECI_Client_Id. The type field of clientSeriesId and clientId define if this is a clientSeriesId or a clientId.
VendorId: ECI_Vendor_Id	Vendor ID of the **ECI Client**.
clientSeriesId: ECI_Client_Sesies_Id	Client series ID of the **ECI Client**.
clientId: ECI_Client_Id	Client ID of the **ECI Client**.
connId: ushort[MaxConnId]	Array of connection ids; value 0xFFFF signals an empty array entry. The empty array entries are all at the end of the array.

9.9.2.5 reqIccPipeOpen Message

C→H reqIccPipeOpen(ushort **clientId,** byte **protocolId[16]) →**
H→C resIccPipeOpen(ushort **clientId)**

- This message enables the **ECI Client** to request the **ECI Host** to open a pipe to another **ECI Client**.

Request parameter definition:

clientId: ushort	ID of the client to which a pipe is requested.
protocolId[16]: byte	ID for the message protocol to be used. This shall be a UUID [12] with octets in network order in the array.

Result parameter definition:

clientId: ushort	ID of the client to which a pipe was requested to be opened.

Pre conditions Response:

- The pipe is opened or an error code is returned. The related error codes are listed in Table 9.9.2.5-1.

Table 9.9.2.5-1 reqIccPipeOpen Error Codes

Name	Description
ErrIccPipeOpenReject	See Table 9.9.2.11-1.
ErrIccPipeOpenNoConn	
ErrIccPipeOpenProtocol	
ErrIccPipeOpenNotReady	

9.9.2.6 reqIccPipeOpenReq Message

H→C reqIccPipeOpenReq(ushort **clientId,** byte **protocolId**[16]**) →**
C→H resIccPipeOpen(ushort **clientId)**

- This message enables the **ECI Client** to receive an incoming request from another **ECI Client** to open a pipe via the **ECI Host**.

Request parameter definition:

clientId: ushort	ID of the client that requests a pipe.
protocolId[16]**:** byte	ID for the message protocol to be used. This shall be a UUID [12] with octets in network order as bytes.

Result parameter definition:

clientId: ushort	ID of the client that requested the pipe

Semantics:

- The response value of clientId shall be identical to the request value.

Pre conditions Response:

- The **ECI Client** may refuse the pipe. The error codes are equal to the ones for opening a pipe and are transparently conveyed to the requester. They are listed in Table 9.9.2.5-1.

9.9.2.7 reqIccPipeCancel Message

C→H reqIccPipeCancel(ushort **clientId)** →
H→C resIccPipeCancel(ushort **clientId)**

- This message enables the **ECI Client** to indicate the **ECI Host** that it wants to terminate the pipe.

Request parameter definition:

clientId: ushort	ID of the client of the pipe that is cancelled.

Result parameter definition:

clientId: ushort	ID of the client of the pipe that is cancelled.

Semantics:

- The response value of clientId shall be identical to the request value.

Pre conditions Response:

- The pipe is terminated: the **ECI Client** requesting the pipe-cancel will not receive any more messages from the pipe.

Detailed semantics:

- If the pipe was not open this handled without error condition.

9.9.2.8 reqIccPipeClose Message

H→C reqIccPipeClose(ushort **clientId,** uint **reason)** →
C→H resIccPipeClose(ushort **clientId)**

- This message enables the **ECI Host** to inform the **ECI Client** that the pipe with the counterpart was closed.

Request parameter definition:

clientId: ushort	ID of the client of the pipe that is closed.
reason: uint	Reason for closing of the pipe. The values are listed in Table 9.9.2.11-1.

Table 9.9.2.8-1: reqIccPipeClose reason values

Name	Value	Description
IccPipeCloseCancel	0x01	Pipe was closed by counterpart using an reqIccPipeCancel message.
IccPipeCloseStop	0x02	Pipe was closed by **ECI Host** as a consequence of a termination of the counterpart **ECI Client**. It is possible that the **ECI Client** is subsequently re-initialized.
RFU	Other	Reserved for future use.

Result parameter definition:

clientId: ushort	ID of the client of the pipe that is closed.

Precondition request:

- No more messages will be sent over the pipe.

Precondition response:

- The **ECI Client** will not attempt to send new messages over the (closed) pipe.

9.9.2.9 reqIccPipeMsgSend Message

C→H reqIccPipeMsgSend(ushort **clientId,** uint **msgId,** uint **dataLen,** byte **data[])→**
H→C resIccPipeMsgSend(ushort **clientId)**

- This message enables the **ECI Client** to send a message to its counterpart of a pipe. Related error codes are listed in Table 9.9.2.11-1.

Request parameter definition:

clientId: ushort	ID of the client to which the message is sent
msgId: uint	Id of the message. All negative values and zero are reserved; all positive values are application specific (meaning is defined in the context of the sender and recipient).
dataLen: uint	Length of the data parameter in number of bytes. This shall not exceed 32 768.
data[]: byte	Data field for the message.

Result parameter definition:

clientId: ushort	ID of the client of the pipe.

Precondition request:

- The next reqIccMsgSend message may only be sent after the previous resIccMsgSend message for the same pipe has been received.

Table 9.9.2.9-1: reqIccPipeMsgSend Error Codes

Name	Description
ErrIccPipeClosed	See Table 9.9.2.11-1.

9.9.2.10 reqIccPipeMsgRecv Message

H-→C reqIccPipeMsgRecv(ushort **clientId,** uint **msgId,** uint **dataLen,** byte **data[])→**
C→H resIccPipeMsgRecv(ushort **clientId)**

- This message enables the **ECI Client** to receive a message from its counterpart of a pipe.

Request parameter definition:

clientId: ushort	ID of the client from which the message is received.
msgId: uint	Id of the message. All negative values and zero are reserved; all positive values are application specific (meaning is defined in the context of the sender and recipient).
dataLen: uint	Length of the data parameter in number of bytes. This shall not exceed 32 768.
data: byte[]	Data field for the message.

Result parameter definition:

clientId: ushort	ID of the client of the pipe.

Precondition request:

- The next reqIccMsgRecv message will only be sent after the previous resIccMsgRecv message for the same pipe has been received.

9.9.2.11 Error Codes for the Inter client communication

Error codes for the inter client communication APIs are listed in Table 9.9.2.11-1.

Table 9.9.2.11-1: Error codes for Inter client communication

Name	Value	Description
ErrIccPipeOpenReject	-256	Counterpart rejected the pipe.
ErrIccPipeOpenNoConn	-257	Counterpart rejects pipe as a consequence of there not being an established import/export connection with the ECI Client.
ErrIccPipeOpenProtocol	-258	Counterpart rejects the protocol proposed for the pipe.
ErrIccPipeOpenNotReady	-259	Counterpart is not in a state where it is ready to accept a pipe. It is appropriate to re-attempt to establish a pipe later.
ErrIccPipeClosed	-260	Pipe is closed.

10 Mandatory and optional ECI Host functionalities

10.1 Introduction

The technical specifications of the **ECI** system support technical solutions for a very wide range of **CPEs** for media consumption. It is up to the decision of a **CPE** manufacturer which frontend, core and backend functions he implements in his device. For the frontend and backend functionalities of a device the manufacturer will most likely only implement only those **ECI** APIs which fit to his hardware / protocol stack. In order to provide flexibility for the **user**, Table 10.2-1 lists all mandatory (m), optional (o) and conditional (c) APIs for the different categories of **CPEs**.

10.2 List of mandatory and optional ECI functionalities for different types of CPE devices

Table 10.2-1 gives the list of mandatory and optional **ECI** functionalities for different types of **CPE** devices. The implementation of several APIs is conditional, depending on the availability of certain hardware/software components in the **CPE** device.

Table 10.2-1: List of mandatory and optional ECI functionalities

API	Clause	Host	Condition (if applicable)	Decr. Client	Micro server	Micro Client
Host interface discovery	9.4.2	M		M	M	M
MMI	9.4.3	M		O	O	O
IP	9.4.4	C	If IP connectivity is supported	O	O	O
HTTP(S)	9.4.4.6	M		O	O	O
File system	9.4.5	M		O	O	O
Timer and clock	9.4.6	M		O	O	O
Power management	9.4.7	M		O	O	O
Country and language setting	9.4.8	M		O	O	O
Advanced Security general	9.5.2.2	M		M	M	M
Advanced Security decryption	9.5.2.3	M		M	n.a.	M
Advanced Security export	9.5.2.4	C	For recording or gateway	O	n.a.	O
Advanced Security encryption	9.5.2.5	C	For recording or gateway	n.a.	M	na
Smart Card	9.5.3	C	For SC-reader supported	O	O	O
Data Carousel	9.5.4	C	For Broadcast network	O	O	O
Decryption (see note)	9.6.2	M		M	n.a.	M
Export connection	9.7.2.3	C	For recording or gateway	O	n.a.	O
Import connection	9.7.2.4	C	For recording or gateway	n.a.	M	n.a.
Re-encryption (see note)	9.7.2.5	C	For recording or gateway	n.a.	M	n.a.
Micro Client de-encryption	9.7.2.6	M		O	n.a.	M
Country and language setting	9.4.8	M		O	O	O
Standard URI	9.8.2.3	M		M	M	M
Customer URI	9.8.2.4	M		M	M	M
Basic URI	9.8.2.5	M		M	M	M
Output control	9.8.2.6	M		M	M	M
Watermarking	9.8.2.7	O		O	n.a.	O
Parental control	9.8.2.8	M		M/O	M/O	M/O
Content property sync	9.8.2.9	M		M	M	M
Parental authentication	9.8.2.10	M		O	n.a.	O
Parental authentication delegation	9.8.2.11	M		O	n.a.	O
Inter Client communication	9.9.2	M		O	O	O
NOTE: Slots can be designated specifically for micro servers and decryption clients. The slot itself is technically identical, but the required **AS** resources and associated descrambling functions are distinct.						

The discovery API does not offer a mechanism to permit an **ECI Host** to detect that an **ECI Client** can decrypt or encrypt file and/or transportstream format media data. Such signalling is provided by the mhType field of the decryptId parameter of the setDcrMhMatch message (see clause 9.6.2.2.2). For re-encryption such discovery is provided by the EciEncrModes parameter of the setEncrModes message (see clause 9.7.2.5.3).

- A consumption only **ECI** compliant device shall provide at least 2 VM instances and AS-slots.
- **ECI Hosts** that support PVR functionality shall support at least one additional container (**VM**-instance) and **AS-slot** for a Micro Server. If such **ECI Hosts** also provides playback functionality of the stored content it shall support at least one additional container (VM-instance) and **AS**-slot for a Micro Client that can decode the re-encrypted content.
- **ECI Hosts** that support networked gateway functionality shall support at least one additional container (**VM**-instance) and AS-slot for a **Micro Server**.

Annex A (normative): Crypotographical Functions of the ECI Host

A.1 Hash Function

The hash functions in the present document are all based on SHA256 as defined in [10].

Function *hash* clause 5.2 is equal to SHA-256() as defined in [10].

The c-function asHash(uchar *data, uint datalength, resultLength, uchar *result) uses the octets starting at data of length dataLength as *dataIn* octetstring and computes the octetstring resultOut as a resultLength/8 octet string, and stores it at result in accordance with:

resultOut = BS2OSP(truncate(SHA-256(OS2BSP(*dataIn*)),resultLength)))

resultLength shall be a multiple of 8. Truncate shall be the function that is the left truncation of a bit string (parameter 1) to the length (parameter 2) bits.

BS2OSP and OS2BSP are functions that convert a bit string to an octet string and vice versa as defined in clause 7 of GS ECI 001-5-1 [4].

A.2 Asymmetrical Cryptography

The asymmetrical encryption and decryption operations shall be defined in clauses 8.2 and 8.3 of ETSI GS ECI 001-5-1 [4].

A.3 Symmetrical Cryptography

AES cryptography as defined in the present document shall be as defined in [10] unless a specific application reference for an AES application is provided.

CBC applications of AES shall be as defined in [9] unless a specific application reference for CBC with AES is provided. If not defined otherwise initialization vector 0 shall be used.

CTR applications of AES shall be as defined in [9] unless a specific application reference for CTR with AES is provided. If not defined otherwise initialization vector 0 shall be used.

A.4 Random Number Generation

Random number generation as defined in the present document shall comply with the specification therefore set in ETSI GS ECI 001-5-1 [4] annex A.

Annex B (normative): Interoperability Parameters

B.1 Introduction

This annex defines parameters related to resource requirements in **CPEs**. The adherence to these requirements serves interoperability between **ECI Clients**, **ECI** security services delivered by networks and **CPEs**.

B.2 Revocation List length

CPEs shall reserve sufficient NV storage to store **Revocation Lists** of the following length for each item that can be revoked is defined in Table B.2-1. The **ECI TA** should ensure the issued **ECI TA** RLs adhere to these limits.

Table B.2-1: Revocation List maximum length

Revocation List	Max. number_of_ids
Manufacturer RL	500
Host RL	500
Vendor RL	500
ECI Client RL	500
Operator RL	500
Platform Operation RL	500

B.3 ECI Client Image size

An **ECI Host** shall have a minimum of 500 Kbyte **ECI Client Image** storage per **ECI Client** slot it supports.

B.4 Broadcast Carousel Configuration Parameters

ECI defines maximum acquisition times tCdownloadScenario for all items to be downloaded from a broadcast carousel so as to permit suitable design of **ECI Hosts**. The tCdownloadScenario parameter reflects the actual download time; therefore carousel repetition rate should be at least a three-fold multiple of this to ensure downloading by the **ECI Host** within these limits. Broadcasters should provide adequate bandwidth to support the required repetition rate.

ECI also defines a maximum module size for buffer allocation purposes.

Both tCdownloadScenario as well as the maximum module size that the **ECI Host** should be designed to handle are defined in Table B.4-1.

Table B.4-1: Maximum download scenario periods and module sizes for ECI carousels

Table type	tCdownloadScenario	Max. Module Size
ECI Client Images	5 minutes	500 Kbyte
ECI Client Revocation data	5 minutes	100 Kbyte per bucket
Platform Operation Certificate Chain	10 seconds	50 Kbyte
Platform Operation Revocation data	5 minutes	100 Kbyte per bucket
ECI Host revocation data	5 minutes	100 Kbyte per bucket
AS setup data	2 minutes	20 Kbyte per bucket

Annex C (normative): ECI Host API overview

Table C-1 defines the values for **MsgApiTag** as defined in clause 9.3.1.

Table C-1: Numbering scheme of the ECI APIs

API	Clause	MsgApiTag value	highest API version	deprecated API versions
Host interface discovery	9.4.2	0x0001	0x0000	none
MMI	9.4.3	0x0002	0x0000	none
IP	9.4.4	0x0003	0x0000	none
HTTP(S)	9.4.4.6	0x0004	0x0000	none
File system	9.4.5	0x0005	0x0000	none
Timer and clock	9.4.6	0x0006	0x0000	none
Power management	9.4.7	0x0007	0x0000	none
Country and language setting	9.4.8	0x0008	0x0000	none
Advanced Security general	9.5.2.2	0x0009	0x0000	none
Advanced Security decryption	9.5.2.3	0x000A	0x0000	none
Advanced Security export	9.5.2.4	0x000B	0x0000	none
Advanced Security encryption	9.5.2.5	0x000C	0x0000	none
Smart Card	9.5.3	0x000D	0x0000	none
Data Carousel	9.5.4	0x000E	0x0000	none
Decryption	9.6.2	0x000F	0x0000	none
Export connection	9.7.2.3	0x0010	0x0000	none
Import connection	9.7.2.4	0x0011	0x0000	none
Re-encryption	9.7.2.5	0x0012	0x0000	none
Micro Client de-encryption	9.7.2.6	0x0013	0x0000	none
Standard URI	9.8.2.3	0x0014	0x0000	none
Customer URI	9.8.2.4	0x0015	0x0000	none
Basic URI	9.8.2.5	0x0016	0x0000	none
Output control	9.8.2.6	0x0017	0x0000	none
Watermarking	9.8.2.7	0x0018	0x0000	none
Parental control	9.8.2.8	0x0019	0x0000	none
Content property sync	9.8.2.9	0x0020	0x0000	none
Parental authentication	9.8.2.10	0x0021	0x0000	none
Parental authentication delegation	9.8.2.11	0x0022	0x0000	none
Inter Client communication	9.9.2	0x0023	0x0000	none

Annex D (informative): List of all available API Messages in alphabetic order

The API messages listed in annex E are taken from the following tables of clause 9 of the present document and listed in Table D-1.

Table D-1: List of Tables giving the messages of the different APIs

API	clause	API category
Host Interface discovery API	9.4.2.1-1	General APIs
User Interface API	9.4.3.1-1	
IP socket API	9.4.4.3.1-1	
UDP socket API	9.4.4.4.1-1	
TCP socket API	9.4.4.5.1-1	
HTTP get API	9.4.4.6.1-1	
File Open/Close API	9.4.5.2.1-1	
File Access API	9.4.5.3.1-1	
File Directory Service API	9.4.5.4.1-1	
Timer API	9.4.6.2.1-1	
Clock API	9.4.6.3.1-1	
Power transition API	9.4.7.2-1	
Wakeup from standby API	9.4.7.3-1	
Country/Language setting API	9.4.8.1-1	
Advanced Security General API	9.5.2.2.1-1	**ECI** specific APIs
Advanced Security Decryption API	9.5.2.3.1-1	
Advanced Security Export API	9.5.2.4.1-1	
Advanced Security Encryption API	9.5.2.5.1-1	
Smart Card Session Management API	9.5.3.6.1-1	
Smart Card Communication API	9.5.3.6.1-1	
Data carousel acquisition API	9.5.4.1-1	
Media Handle decryption session API	9.6.2.2.1-1	
Export Connection API	9.7.2.3.1-1	
Import connection API	9.7.2.4.1-1	
Re-encryption API	9.7.2.5.1-1	
Decryption API	9.7.2.6.1-1	
usage rights and parental control API	9.8.2.1-1	
Inter Client Communication API	9.9.2.1-1	

Table D-2 lists all API messages in alphabetic order.

Table D-2: List of all API messages in alphabetic order

No.	Message	API	Clause	Type	Dir.	Description
1	callAsNextKeySession	Advanced Security General	9.5.2.2.3	S	C→H	Change to next random key for a session.
2	callCardGetProp	**Smart Card**	9.5.3.6.5	S	H→C	Get card communication property/parameter.
3	callCardSessionPrio	**Smart Card**	9.5.3.5.3	S	C→H	Set **Smart Card** session priority.
4	callCardSetProp	**Smart Card**	9.5.3.6.4	S	H→C	Set card communication parameter.
5	callFileDataLog	File System	9.4.5.3.6	S	C→H	Appends data at the end of a buffered file.
6	callLocaltime	Clock	9.4.6.3.3	S	C→H	Converts time integer value into local time.
7	getApis	Interface Discovery	9.4.2.2	S	C→H	Get available Host APIs.
8	getApiVersions	Interface Discovery	9.4.2.3	S	C→H	Get available versions of a host API.
9	getAsClientRnd	Advanced Security General	9.5.2.2.13	S	C→H	Get a new random number for the **ECI Client** applications.
10	getAsSC	Advanced Security General	9.5.2.2.14	S	C→H	Get current Scrambling Control field status of content in a session.
11	getAsSessionLimitCounter	Advanced Security General	9.5.2.2.10	S	C→H	Get current limit counter value for the session.

No.	Message	API	Clause	Type	Dir.	Description
12	getAsSessionRk	Advanced Security General	9.5.2.2.9	S	C→H	Get random key value for a session.
13	getAsSlotRk	Advanced Security General	9.5.2.2.8	S	C→H	Get random key value for the **AS slot.**
14	getCardConnStatus	**Smart Card**	9.5.3.5.4	S	H→C	Provides status of card connection status.
15	getDcrMarkMeta	Content Property	9.8.2.7.4	S	H→C	Read a marking system property.
16	getDcrMarkSyst	Content Property	9.8.2.7.2	S	H→C	Get supported marking systems.
17	getDcrTsSource	Decryption TS Source Control	9.6.2.3.6.2	S	C→H	The **ECI Client** gets the source of the TS.
18	getEncrStdUri	Content Property	9.8.2.3.2	S	C→H	Get standard URI for content to be re-encrypted.
19	getEncrBasicUri	Content Property	9.8.2.5.2	S	C→H	Get Basic URI for content to be re-encrypted.
20	getEncrCustUri	Content Property	9.8.2.4.2	S	C→H	Get custom URI for content to be re-encrypted.
21	getEncrOutputCtrl	Content Property	9.8.2.6.2	S	C→H	Get Output Control restrictions for content to be re-encrypted.
22	getEncrParCtrl	Content Property	9.8.2.8.2	S	C→H	Get Parental Control conditions for content to be descrambled.
23	getIccClientInfo	Inter Client Communication	9.9.2.4	S	C→H	The **ECI Client** reads the identity and connection status of another **ECI Client** in the system.
24	getIccMaxClients	Inter Client Communication	9.9.2.2	S	C→H	The **ECI Client** reads the maximum number of **ECI Clients** that the **ECI Host** may support.
25	getPwrStatus	Power Management	9.4.7.2.2	S	C→H	Gets current value power status.
26	getTime	Clock	9.4.6.3.2	S	C→H	Reads the local system clock as integer value.
27	reqAsAStartDecryptSession	Advanced Security Decryption	9.5.2.3.2	A	C→H	Start a decryption session in the **ECI Client**'s **AS slot**
28	reqAsAuthDecrSlotConfig	Advanced Security Decryption	9.5.2.3.4	A	H→C	Authenticate the slot configuration with authentication mechanisms (decryption mode).
29	reqAsAuthEncrSlotConfig	Advanced Security Encryption	9.5.2.5.5	A	C→H	Authenticate the slot configuration and encryption parameters with authentication mechanisms (encryption mode).
30	reqAsClientChalResp	Advanced Security General	9.5.2.2.7	A	C→H	Apply **ECI Client** Authentication Key on data and return result.
31	reqAsComputeAkClient	Advanced Security General	9.5.2.2.6	A	C→H	Compute Authentication Key for **ECI Client** applications.
32	reqAsComputeEncrCw	Advanced Security Encryption	9.5.2.5.4	A	C→H	Compute encryption control word.
33	reqAsEventCpChange	Advanced Security Encryption	9.5.2.5.8	A	H→C	Event message on content property change in imported content in an encryption session.
34	reqAsEventSC	Advanced Security General	9.5.2.2.15	A	H→C	Event message on change of scrambling control field in session.
35	reqAsEventSessionLimit	Advanced Security General	9.5.2.2.12	A	H→C	On reaching a limit value for remaining units send event to **ECI Client**.
36	reqAsExportConnEnd	Advanced Security Export	9.5.2.4.3	A	C→H	Terminate existing export session.
37	reqAsExportConnSetup	Advanced Security Export	9.5.2.4.2	A	C→H	Setup an export connection from decryption to encryption session.
38	reqAsInitSlot	Advanced Security General	9.5.2.2.2	A	C→H	Initializes the **AS slot**.
39	reqAsLdUssk	Advanced Security Encryption	9.5.2.5.6	A	C→H	Load micro server secret key.
40	reqAsLoadSlotLk	Advanced Security General	9.5.2.2.5	A	C→H	Compute top level link key (LK1).
41	reqAsMInikLk1	Advanced Security Encryption	9.5.2.5.7	A	C→H	Compute asymmetrical **Micro Client** initialization message.
42	reqAsStartEncryptSession	Advanced Security Encryption	9.5.2.5.3	A	C→H	Start an encryption session.
43	reqAsStopSession	Advanced Security General	9.5.2.2.4	A	C→H	Stop a session.
44	reqCardCmdRes	**Smart Card**	9.5.3.6.2	A	C→H	Send card command, get card response back.

No.	Message	API	Clause	Type	Dir.	Description
45	reqCardReInit	**Smart Card**	9.5.3.6.3	A	C→H	Reset card (warm or cold) and reruns initialization sequence with the latest initialization preference setting.
46	reqCCardConClose	**Smart Card**	9.5.3.5.6	A	H→C	Informs **ECI Client** that a card session has been closed.
47	reqCCardConOpen	**Smart Card**	9.5.3.5.5	A	H→C	Informs **ECI Client** that a card session has been opened.
48	reqCCountry	Country	9.4.8.2.2	A	H→C	The **ECI Host** requests the actual **ECI Client** preferred country setting.
49	reqCLanguage	Language	9.4.8.2.4	A	H→C	The **ECI Host** requests the actual **ECI Client** preferred language setting.
50	reqCpChange	Content Property	9.8.2.9.2	A	H→C	The **ECI Host** signals that a change is forthcoming in the content properties of content to be re-encrypted.
51	reqDCAcqModule	Data Carousel Acquisition	9.5.4.3	A	C→H	The **ECI Client** requests the **ECI Host** to acquire a specific **ECI** data carousel module into a file using a module filter parameters and various modes.
52	reqDCAcqGroupInfo	Data Carousel Acquisition	9.5.4.2	A	C→H	The **ECI Client** requests the **ECI Host** to read the GroupInfoIndication structure in the DSI message of the specified **ECI** data carousel.
53	reqDcrFileQuit	Decryption Media File	9.6.2.4.4.4	A	C→H	The **ECI Client** cancels a descrambling session with the **ECI Host**.
54	reqDcrFileData	Request Data via File Filter	9.6.2.4.5.2.4	A	C→H	The **ECI Client** requests the **ECI Host** to acquire data via the File Filter.
55	reqDcrFileStop	Decryption Media File	9.6.2.4.4.3	A	H→C	The **ECI Host** requests the **ECI Client** to stop descrambling of a **Media Handle**.
56	reqDcrFileFilter	Request File Filter	9.6.2.4.5.2.3	A	C→H	The **ECI Client** requests the **ECI Host** to set a data filter for security data acquisition.
57	reqDcrFileKeyComp	Request key computing	9.6.2.4.6.3	A	H→C	Initiate any required computing or other activity of the **ECI Client** to make a control word with Key-ID available.
58	reqDcrFileStart	Decryption Media File	9.6.2.4.4.2	A	H→C	Requests **ECI Client** to descramble or return the descramble status of a file or stream.
59	reqDcrIpServer	Re-encryption	9.7.2.6.5	A	C→H	The **Micro Client** to request the **ECI Host** to provide the **Micro Server's** IP address for further communication related to the **Media Handle** session.
60	reqDcrMhBcAlloc	MediaHandle Decryption	9.6.2.2.5	A	C→H	The **ECI Client** requests a **Media Handle** session for its own broadcast network access purposes.
61	reqDcrMhCancel	MediaHandle Decryption	9.6.2.2.6	A	C→H	The **ECI Client** cancels a media session with the **ECI Host**.
62	reqDcrMhClose	MediaHandle Decryption	9.6.2.2.4	A	H→C	The **ECI Host** closes a media session with an **ECI Client**.
63	reqDcrMhOpen	MediaHandle Decryption	9.6.2.2.3	A	H→C	The **ECI Host** requests the **ECI Client** to open a media session of a specified type using a **Media Handle**.
64	reqDcrMsgRecv	Re-encryption	9.7.2.6.7	A	H→C	The **ECI Host** provides the **Micro Client** with a message from the **Micro Server** of a **Media Handle** session.
65	reqDcrMsgSend	Re-encryption	9.7.2.6.6	A	C→H	The **Micro Client** requests the **ECI Host** to send a message to the **Micro Server** of a **Media Handle** session
66	reqDcrTargetCred	Re-encryption	9.7.2.6.4	A	H→C	The **ECI Host** requests the **ECI Client** to provide the initialization data for a Micro Server connection typically used for authentication of the target.
67	reqDcrTargets	Re-encryption	9.7.2.6.3	A	H→C	The **ECI Host** requests the **Micro Client** to provide the encryption targets it can decrypt services for.
68	reqDcrTsData	Re-encryption	9.7.2.6.8	A	C→H	The **Micro Server** provides the **ECI Host** with data to be forwarded to the target **Micro Client** of a **Media Handle** for decryption, including ECM related synchronization information.
69	reqDcrTsDescrquit	TS content Decryption	9.6.2.3.4.4	A	C→H	The **ECI Client** reqests the **ECI Host** to terminate the descrambling of a **Media Handle session**.

No.	Message	API	Clause	Type	Dir.	Description
70	reqDcrTsData	Micro-Client De-encryption	6.7.2.6.7	A	H→C	The **ECI Host** to provide the Micro Client with data required in the (near) future for decrypting the content on Media Handle.
71	reqDcrTsDescrStop	TS content Decryption	9.6.2.3.4.3	A	H→C	The **ECI Host** requests the **ECI Client** to stop the descrambling of a **Media Handle session**.
72	reqDcrTsDescrStart	TS content Decryption	9.6.2.3.4.2	A	H→C	Requests The **ECI Client** to descramble or return the descramble status of a programme in a TS.
73	reqDcrTsRelocate	Decryption TS Source control	9.6.2.3.6.3	A	C→H	The **ECI Clients** relocates the source of the TS.
75	reqDcrTsSection	Decryption TS data acquisition	9.6.2.3.5.5	A	H→C	Forwards a acquired section to **ECI Client**.
76	reqDcrTsSelectCancel	Decryption TS Source control	9.6.2.3.6.6	A	C→H	The **ECI Client** cancels its previous program selection.
77	reqDcrTsSelectPmt	Decryption TS Source control	9.6.2.3.6.5	A	C→H	The **ECI Client** selects program in TS by PMT.
78	reqDcrTsSelectPrg	Decryption TS Source control	9.6.2.3.6.4	A	C→H	The **ECI Client** selects program in TS by program number.
79	reqDcrTsTable	Decryption TS data acquisition	9.6.2.3.5.6	A	C→H	The **ECI Client** acquires a table in the stream.
80	reqEncrConnDrop	Re-encryption	9.7.2.5.5	A	H→C	The **ECI Host** requests **the ECI Client** to drop any information on a previously pre-authenticated re-encryption connection.
81	reqEncrConnSetup	Re-encryption	9.7.2.5.4	A	H→C	The **ECI Host** requests the **ECI Client** to create a re-encryption target connection and to pre-authenticate the re-encryption target for subsequent reference in setting up a **Media Handle** session.
82	reqEncrFileData	Re-encryption	9.7.2.5.18	A	C→H	The **Micro Server** provides the **ECI Host** with a message to be forwarded to the target **Micro Client** of a **Media Handle** for decryption, including KeyID related synchronization information.
83	reqEncrIpServer	Re-encryption	9.7.2.5.13	A	H→C	The **ECI Host** requests the IP server address of a Micro Server so as to permit **Micro Clients** to create IP connections.
84	reqEncrMhCancel	Re-encryption	9.7.2.5.9	A	C→H	The **ECI Client** terminates the **Import Connection** with the specified exporting **ECI Client**.
85	reqEncrMhClose	Re-encryption	9.7.2.5.8	A	H→C	**ECI Host** closes the **Re-encryption Session** with the **ECI Client**.
86	reqEncrMhOpen	Re-encryption	9.7.2.5.7	A	H→C	The **ECI Host** requests The **ECI Client** to open a **Media Handle** session to re-encrypt content from an incoming **Import Connection** for an established re-encryption connection.
87	reqEncrMhQuit	Re-encryption	9.7.2.5.12	A	C→H	The **ECI Client** informs the **ECI Host** that the **Media Handle** re-encryption operation was terminated.
88	reqEncrMhStart	Re-encryption	9.7.2.5.10	A	H→C	The **ECI Host** requests the **ECI Client** to start the re-encryption operation for a **Media Handle** session.
89	reqEncrMhStop	Re-encryption	9.7.2.5.11	A	H→C	The **ECI Host** requests the **ECI Client** to stop a re-encryption operation for a **Media Handle** session.
90	reqEncrMsgRecv	Re-encryption	9.7.2.5.18	A	H→C	The **ECI Host** provides the **Micro Server** with a message from a target of a **Media Handle** session.
91	reqEncrMsgSend	Re-encryption	9.7.2.5.14	A	C→H	The **Micro Server** requests the **ECI Host** to forward a message to the target of a **Media Handle** session.
92	reqEncrTargets	Re-encryption	9.7.2.5.3	A	H→C	The **ECI Host** requests the **ECI Client** to provide the target nodes it can authenticate.
93	reqEncrTsData	Re-encryption	9.7.2.5.16	A	C→H	The **Micro Server** provides the **ECI Host** with data to be forwarded to the target **Micro Client** of a **Media Handle** for decryption, including ECM related synchronization information.
94	reqEncrTsEcm	Re-encryption	9.7.2.5.17	A	C→H	The **Micro Server** issues an ECM section that is required by the **Micro**

No.	Message	API	Clause	Type	Dir.	Description
						Client to decrypt in the next crypto-period.
95	reqExpConnCancel	Export Connection	9.7.2.3.5	A	C→H	The **ECI Client** terminates an initialized **Export Connection** with an importing **ECI Client**.
96	reqExpConnDrop	Export Connection	9.7.2.3.4	A	H→C	The **ECI Hosts** cancels any previously initialized connection of an exporting **ECI Client** to an importing **ECI Client**.
97	reqExpConnNodes	Export Connection	9.7.2.3.2	A	H→C	The **ECI Host** requests export option nodes from the **ECI Client**.
98	reqExpConnSetup	Export Connection	9.7.2.3.3	A	H→C	The **ECI Host** requests the **ECI Client** to initialize an **Export Connection** to an importing **ECI Client** based on an **Import Chain**.
99	reqExpMhCancel	Export Connection	9.7.2.3.8	A	C→H	The **ECI Client** cancels an export session.
100	reqExpMhClose	Export Connection	9.7.2.3.7	A	H→C	The **ECI Host** closes an export session.
101	reqExpMhOpen	Export Connection	9.7.2.3.6	A	H→C	The **ECI Host** requests the **ECI Client** to create an export session based on a previously initialized **Export Connection**
102	reqFileClose	File System	9.4.5.2.3	A	C→H	Closes an open file.
103	reqFileCreate	File System	9.4.5.4.3	A	C→H	Create a new file.
104	reqFileDelete	File System	9.4.5.4.4	A	C→H	Delete a file.
105	reqFileDir	File System	9.4.5.4.5	A	C→H	Lists file names of files available in The **ECI Clients** file system.
106	reqFileOpen	File System	9.4.5.2.2	A	C→H	Opens an **ECI Client** private file.
107	reqFileRead	File System	9.4.5.3.3	A	C→H	Reads consecutive bytes starting from the present file location.
108	reqFileRemoveData	File System	9.4.5.3.5	A	C→H	Deletes data from a file at current location.
109	reqFileSeek	File System	9.4.5.3.4	A	C→H	Repositions the present file location.
110	reqFileStat	File System	9.4.5.4.2	A	C→H	Return size and modification time of file.
111	reqFileWrite	File System	9.4.5.3.2	A	C→H	Writes consecutive bytes starting from the present file location.
112	reqHCardConClose	**Smart Card**	9.5.3.5.7	A	C→H	Informs **ECI Host** that **ECI Client** wishes to terminate a session with the connected card.
113	reqHCountry	Country	9.4.8.2.1	A	C→H	Requests the actual **ECI Host** preferred country setting.
114	reqHLanguage	Language	9.4.8.2.3	A	C→H	Requests the actual **ECI Host** preferred language setting.
115	reqHttpGetData	HTTP Get	9.4.4.6.3	A	C→H	Performs an HTTP Get request on a URL and passes the result as data to the Client.
116	reqHttpGetFile	HTTP Get	9.4.4.6.3	A	C→H	Performs an HTTP Get request on a URL and stores the result in a file.
117	reqIccPipeCancel	Inter Client Communication	9.9.2.7	A	C→H	The **ECI Client** cancels the pipe.
118	reqIccPipeClose	Inter Client Communication	9.9.2.8	A	H→C	The **ECI Host** informs **ECI Client** that the pipe with the counterpart was closed.
119	reqIccPipeMsgRecv	Inter Client Communication	9.9.2.10	A	H→C	The **ECI Client** receives a message from its counterpart of a pipe.
120	reqIccPipeMsgSend	Inter Client Communication	9.9.2.9	A	C→H	**ECI Client** sends a message to its counterpart of a pipe.
121	reqIccPipeOpen	Inter Client Communication	9.9.2.5	A	C→H	Request to open a pipe to another **ECI Client**.
122	reqIccPipeOpenReq	Inter Client Communication	9.9.2.6	A	H→C	Incoming request from another **ECI Client** to open a pipe.
123	reqIccSystemReady	Inter Client Communication	9.9.2.3	A	H→C	The **ECI Host** informs the **ECI Client** that all **ECI Clients** are initialized.
124	reqImpConnCancel	Import Connection	9.7.2.4.6	A	C→H	The **ECI Client** terminates the **Import Connection** with the specified exporting **ECI Client**.

No.	Message	API	Clause	Type	Dir.	Description
125	reqImpConnChain	Import Connection	9.7.2.4.3	A	H→C	The **ECI Host** requests importing **ECI Client** to provide input chain for a specific import node.
126	reqImpConnChainRenew	Import Connection	9.7.2.4.3	A	C→H	The **ECI Client** requests the **ECI Host** to reinitialize the connection using an updated **Import Chain**.
127	reqImpConnDrop	Import Connection	9.7.2.4.5	A	H→C	The **ECI Host** drops the **Import Connection** with the specified exporting **ECI Client**.
128	reqImpConnNodes	Import Connection	9.7.2.4.2	A	H→C	The **ECI Host** requests importing **ECI Client** to provide its import nodes.
129	reqImpConnSetup	Import Connection	9.7.2.4.4	A	H→C	The **ECI Host** requests importing **ECI Client** to initialize an **Import Connection** with a specific exporting **ECI Client** through an import node.
130	reqIpAddrinfo	IP sockets	9.4.4.3.4	A	C→H	Gets address of (remote) **ECI Host**.
131	reqIpClose	IP sockets	9.4.4.3.3	A	C→H	Closes **ECI** IP Socket.
132	reqIpSocket	IP sockets	9.4.4.3.2	A	C→H	Opens an **ECI** IP Socket.
133	reqIpTcpAccept	TCP/IP Socket	9.4.4.5.5	A	C→H	TCP server peer accepts connection from TPC client peer.
134	reqIpTcpConnect	TCP/IP Socket	9.4.4.5.2	A	C→H	TCP client connects to TCP server peer.
135	reqIpTcpRecv	TCP/IP Socket	9.4.4.5.4	A	C→H	Receives data from connected peer.
136	reqIpTcpSend	TCP/IP Socket	9.4.4.5.3	A	C→H	Sends data to connected peer.
137	reqIpUdpRecvMsg	UDP/IP Socket	9.4.4.4.3	A	C→H	Receives a message from peer UDP port.
138	reqIpUdpSendMsg	UDP/IP Socket	9.4.4.4.2	A	C→H	Sends message to peer UDP port.
139	reqParAuthChk	Content Property	9.8.2.10.3	A	C→H	Request to the **ECI Host** to perform a parental authentication on behalf of the **ECI Client**.
140	reqParAuthChkCan	Content Property	9.8.2.10.4	A	C→H	Cancels a preceding parental authentication request to the Host.
141	reqParAuthCid	Content property	9.8.2.10.5	A	H→C	Requests parental pin code authorization for a (future) content item to be decoded. This may trigger a parental authentication dialogue.
142	reqParAuthDel	Content Property	9.8.2.11.2	A	H→C	The **ECI Host** delegates a parental authentication to an **ECI Client**.
143	reqParAuthDelCan	Content Property	9.8.2.11.3	A	H→C	The **ECI Host** cancels a preceding parental authentication request to the **ECI Client**.
144	reqPwrChange	Power Management	9.4.7.2.4	A	H→C	Notice of power status change.
145	reqTimerCancel	Timer	9.4.6.2.3	A	C→H	Cancels a previously set timer event.
146	reqTimerEvent	Timer	9.4.6.2.2	A	C→H	Sets a timer event in the future.
147	reqUiClientQuery	User Interface	9.4.3.4.8	A	H→C	The **ECI Client** receives request from the HTML application in the browser and provides a (dynamic) response.
148	reqUiContainerMount	User Interface	9.4.3.4.2	A	C→H	Mounts a UI Application container with HTML resources to support UI sessions.
149	reqUiSessionCancel	User Interface	9.4.3.4.7	A	H→C	The **ECI Host** cancels a user interface session.
150	reqUiSessionClose	User Interface	9.4.3.4.6	A	C→H	The **ECI Client** ends a user interface session.
151	reqUiSessionCommence	User Interface	9.4.3.4.4	A	H→C	**The ECI Host** suggests the **ECI Client** to open a UI session.
152	reqUiSessionOpen	User Interface	9.4.3.4.5	A	C→H	The **ECI Client** requests to open a user interface session with the user and present content on the screen.
153	reqPwrWakeupEvent	Power Management	9.4.7.3	A	H→C	Signals wakeup timer expiration.
154	setApiVersion	Interface discovery	9.4.2.4	S	C→H	Set the version of the Host API to be used.

No.	Message	API	Clause	Type	Dir.	Description
155	setAsPermitCPChange	Advanced Security Encryption	9.5.2.4	S	C→H	Enable/disable imported content property CP changes taking effect on control word selection for encryption in an encryption session.
156	setAsSC	Advanced Security Encryption	9.5.2.4	S	C→H	Set scrambling control field of encrypted content of an encryption session.
157	setAsSessionLimitEvent	Advanced Security General	9.5.2.5.11	S	C→H	Set limit value for sending a reqAsEventSessionLimit message to the **ECI Client**.
158	setCardMatch	**Smart Card**	9.5.3.5.2	S	C→H	Set card identification specifier list for **ECI Client**.
159	setCpSync	Content Property	9.8.2	S	C→H	**ECI Client** signals that the current set of content properties is coherent and can be applied to the content to be descrambled by forthcoming control word.
160	setDcrBasicUri	Content property	9.8.2.5.1	S	C→H	Set Basic URI for content to be descrambled.
161	setDcrCustUri	Content property	9.8.2.4.1	S	C→H	Set custom URI for content to be descrambled.
162	setDcrMarkBasic	Content property	9.8.2.7.5	S	C→H	Set basic marking payload for content to be descrambled.
163	setDcrMarkExt	Content property	9.8.2.7.6	S	C→H	Set extended marking payload for content to be descrambled.
164	setDcrMarkMeta	watermarking	9.8.2.7.3	S	C→H	Set a marking system control value.
165	setDcrMhMatch	MediaHandle Decryption	9.6.2.2.2	S	C→H	Signals to **ECI Host** under which Ids the **ECI Client** can be recognized for descrambling content.
166	setDcrModes	Re-encryption	9.7.2.6.1	S	C→H	The **Micro Client** informs the **ECI Host** about the modes (encryption-modes, data-format-modes and synchronization-modes) it supports.
167	setDcrOutputCtl	Content property	9.8.2.6.1	S	C→H	Set Output Control restrictions for content to be descrambled.
168	setDcrParCtl	Content property	9.8.2.8.1	S	C→H	Set Parental Control conditions for content to be descrambled.
170	setDcrStdUri	Content property	9.8.2.8.1	S	C→H	Set standard URI for content to be descrambled.
171	setDcrTsSectionAcq	Decryption TS data acquisition	9.6.2.3.5.4	S	C→H	Sets a filter for section acquisitions.
172	setDcrTsSectionAcqDefault	Decryption TS data acquisition	9.6.2.3.5.3	S	C→H	Sets a default filter for section acquisition.
173	setEncrModes	Re-encryption	9.7.2.5.2	S	C→H	The Micro Server informs the **ECI Host** about the modes (encryption-modes, data-format-modes and synchronization-modes) it supports.
176	setPwrInfo	Power Management	9.4.7.2.3	S	C→H	Requests event notices for changes in power status.
177	setUiClientAttention	User Interface	9.4.3.4.3	S	C→H	**ECI Client** indicates a desire to start a UI session without association to a media handle.
178	setPwrWakeup	Power Management	9.4.7.3	S	C→H	Sets wakeup time for **ECI Client**.

Annex E (normative): Forward Compatibility of Content Property Definitions

Content properties have to be implemented in a highly robust way using hardware or low level firmware and may be complex, costly or impossible to change or update after SOC production. The approach to creating an evolutionary path for such content properties despite such upgrade limitations is explained in this clause.

New content properties and/or expanded functionality on existing content properties may be required in the future. This may include an extension of the number of bits representing the value of a content property. The Content property implementation in an older **ECI Host** is not aware of new functionality and it is often not feasible to update it. The definition of content properties in **ECI Hosts** is such that a maximum forward compatibility with respect to new content property functionality is achieved.

ECI Hosts will have a defined behaviour for all input values and ignore any field extension they are not designed for also creating defined behaviour. I.e. each value of a future Content Property will have a *single defined behaviour* on all **ECI Hosts** not implementing all extensions, including **ECI Hosts** complying with the first content property version. Using this principle new Content Property values can be assigned with full awareness on what behaviour will result on preceding versions of **ECI Host** implementations. In case a new content property should have two (or more) different options for backward compatible interpretation by older **ECI Hosts** two (or more) reserved values can be assigned having the same new content property semantics in the new content property definition, but each with a suitable (but different) backward compatible interpretation.

An example of a field extension is for instance a new output control field to be defined for a new output type X in the output control API. This is assigned to bit-5 which is reserved in version 1. It can use the semantical equivalent of the OcIP field. Any previous implementations of **ECI Clients** will assign this field 0. The interpretation by an older **ECI Host** will be the following:

- if OcAnyOther==0b0 OutputX is permitted;
- if OcAnyOther==0b1 OutputX is not permitted.

This corresponds perfectly to the semantics in a new **ECI Host** implementation when OcX==0b0. However, when OcX==0b1 the output permission will be the reverse of before, thus permitting new functionality in the combination of a new **ECI Host** and a new **ECI Client**. Note that the reverse interpretation of the field values depending on OcAnyOther ensures that the value 0 for any undefined field takes its natural meaning: maximum permission for OcAnyOther==0b0 (other outputs permitted) and minimum permission for OcAnyOther= 0b1 (other outputs not permitted).

Vice versa it is important that **ECI Clients** not using the latest content property definition do not inadvertently address new content property functionality of later content property definitions that they are not aware of, or worse, use such presumably non-assigned values for private purposes based on the fact that such values have defined behaviour in all **ECI Hosts**. Such inappropriate use will typically create a serious obstacle for future incorporation of these values for **ECI** defined purposes. Therefore this specification explicitly forbids **ECI Clients** application of unassigned content property values.

Specifically: for fields which can have multiple values the reserved values will all have a defined behaviour in **ECI Hosts**, but the reserved values shall not be used by **ECI Clients**.

Any unassigned subfield in a content property definition shall have a defined behaviour defined in an **ECI Host**, which corresponds to one of the defined content property values. Typically an **ECI Host** shall ignore such subfields, i.e. the **ECI Host** interprets the content property value simply in terms of the fields that are defined. Typically **ECI Clients** shall assign the value 0 to such a subfield. Any deviation of the unassigned subfield equal zero policy shall be predefined by a version of the content property definition.

Any field extension shall be ignored by **ECI Hosts** compliant to the corresponding content property definition and **ECI Clients** assigning values shall assign the value 0 to such field extensions.

Annex F (informative): Authors & contributors

The following people have contributed to the present document:

Rapporteur:
Christoph Schaaf, Vodafone Kabel Deutschland

Klaus Illgner, IRT (from 02/2017 onwards)

Other contributors:

Joakim Larsson Edström. Zenterio

Robert Esterer, IRT

Karsten Höllerer, BNetzA

Jens Johann, Deutsche Telekom

Han-Seung Koo, ETRI

Peter Mann, BNetzA

Marnix Vlot, Vodafone Kabel Deutschland

Annex G (informative): Bibliography

- Klaus Illgner, Christoph Schaaf, Marnix Vlot: "Embedded Common Interface (ECI) for Digital Broadcasting Applications: Security and Interoperability combined", Broadband Journal of the SCTE, Vol. 38, No. 3, August 2016.
- Menezes, A., van Oorschot, P. and Vanstone, S: "Handbook of Applied Cryptography", CRC Press, 1996.
- ETSI GS ECI 002: "Embedded Common Interface (ECI) for exchangeable CA/DRM solutions; System validation".

History

Document history		
V1.1.1	July 2017	Publication