ETSI GS ECI 001-5-1 V1.1.1 (2017-07)

Embedded Common Interface (ECI) for exchangeable CA/DRM solutions; Part 5: The Advanced Security System; Sub-part 1: ECI specific functionalities

Disclaimer

The present document has been produced and approved by the Embedded Common Interface (ECI) 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-001-5-1

Keywords
authentication, CA, DRM, encryption, 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 5, sub-part 1 of a multi-part deliverable covering the ECI specific functionalities of an advanced security system, 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:

Sub-part 1: "ECI specific functionalities";

Sub-part 2: "Key Ladder Block".

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, including content, services, networks and customer premises equipment (CPE), to protect business models of content owners, network operators and PayTVoperators. It is also essential for consumers that they are able to continue using the CPEs they bought e.g. after a move or a change of network provider or even utilize devices for services of different commercial video portals. This can be achieved by the implementation of interoperable CA and DRM mechanisms inside CPEs, based on an appropriate security architecture.

As part of a security architecture the present document defines a security processing system for the authentication and verification of protected media content and of software images to be processed inside an ECI-compliant CPE. The core of the security architecture is built by a **Key Ladder Block** that supports secure processing with secret keys, targeting of keys to specific chips and authentication of the origin of key material.

Clause 4 gives an overview about the system architecture, defines robustness rules to fight attacks and describes the relation between the elements of the security architecture, **ECI Host** and **ECI Clients**.

Clause 5 describes the applications the **Key Ladder Block** can be used for, together with the associated functions.

For proper operations, the security processing system needs information about the state of each loaded **ECI Client.** This state information, as some of it needs to be secret, is handled with the help of an advanced security slot. The **ECI Host** assigns to each **ECI Client** such a slot that needs to be protected against malicious modifications. The definition of a slot and its configuration for several operations like decrypting or exporting content is described in clause 6.

In an **ECI**-compliant CPE content can be decrypted, it can be forwarded to standard outputs if permitted and it can be re-encrypted for export. The usage of an advanced security slot for these operations is specified in clause 7.

A **Certificate Processing Subsystem** that is realized as a special function of an advanced security slot is responsible for the authentication of items. Clause 8 specifies the rules that are applied for authentication.

The **ECI** system uses a loader mechanism that permits **ECI Clients** to securely verify the version of the **ECI Host** and **ECI Client** credentials that are loaded so as to detect any known security issue. The loader mechanism relies on robustness principles that are described in clause 9.

Clause 10 contains timing constraints for the operations described in the present document.

1 Scope

The present document defines a robust security processing subsystem for **ECI** called the **Advanced Security System**. The **Advanced Security System** provides a secure basis for software elements to be authenticated and loaded, performs security computations and verifications, manages the encryption and decryption of content and the exchange of content with associated rights and obligations. As such the **Advanced Security System** is part of a "secure video path" as it is referred to in contemporary specifications. The **Advanced Security System** applies the **ECI Key Ladder Block [**5**]** to perform secure calculations.

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 001-3: "Embedded Common Interface (ECI) for exchangeable CA/DRM solutions; Part 3: CA/DRM Container, Loader, Interfaces, Revocation".

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

[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 9899:2011: "Information technology - Programming languages - C".

[7] NIST FIPS PUB 180-4: "Secure Hash Standard (SHS)".

[8] NIST Special Publication 800-90A revision 1: "Recommendation for Random Number Generation Using Deterministic Random Bit Generators", June 2015.

NOTE: Available at http://dx.doi.org/10.6028/NIST.SP.800-90Ar1.

[9] ETSI ETR 289 (CSA1/2): "Digital Video Broadcasting (DVB); Support for use of scrambling and Conditional Access (CA) within digital broadcasting systems".

[10] ETSI TS 100 289 (V1.2.1) (CSA3): "Digital Video Broadcasting (DVB); Support for use of the DVB Scrambling Algorithm version 3 within digital broadcasting systems".

[11] ETSI TS 103 127 (V1.1.1) (CISSA): "Digital Video Broadcasting (DVB); Content Scrambling Algorithms for DVB-IPTV Services using MPEG2 Transport Streams".

[12] ISO/IEC 23001-7 (2016) (CENC): "Information technology - MPEG systems technologies - Part 7: Common encryption in ISO base media file format files".

[13] ISO/IEC 23009-4 (2013): "Information technology - Dynamic adaptive streaming over HTTP (DASH) - Part 4: Segment encryption and authentication".

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: "Using the DVB CSA algorithm" (licencing arrangement).

NOTE: Available at http://www.etsi.org/about/what-we-do/security-algorithms-and-codes/csa-licences.

[i.2] ETSI: "Using the DVB CSA3 algorithm" (licensing conditions).

NOTE: Available at http://www.etsi.org/about/what-we-do/security-algorithms-and-codes/csa3-licences.

[i.3] ETSI GR ECI 004: "Embedded Common Interface (ECI) for exchangeable CA/DRM solutions; Guidelines for the implementation of ECI".

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): robust secure processing system providing basic and highly secure processing functions for **ECI Clients**

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

AS-API: application programming interface between the **ECI Client** and its **ECI Host** permitting the **ECI Client** to exchange information with and perform operations on its **AS Slot**

Authentication Mechanism: Key Ladder Block function as defined in [5] that permits an **AS Slot** to provide secure key applications for purposes other than content decryption and encryption, like authentication

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: sequence of **Certificates** where the next **Certificate** can be authenticated by the public key of the preceding one

NOTE: Typically, in **ECI Certificates** are accompanied by a **Revocation List** that excludes **Certificates** that cannot be validated.

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

Content Properties (CP): properties of the content that provide information on rights and obligations associated with subsequent applications or transformations of the content, like usage rights information, selective output control and parental control information

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 Clients** in customer premises equipment and thus provides interoperability of CPE devices with respect to this system

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

ECI Client Loader: functionality of the **ECI Host** that uses the AS system to exclusively provide the function , verify and install a new **ECI Client** software image in an **ECI** container of the **ECI Host**

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

ECI Host Loader: CPE bootloading functionality that uses the AS system to exclusively provide the function to verify and install **ECI Host** software into a CPE

ECI Root Key: public key providing the origin of authentication for **ECI** certified entities and **Certificates**

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 connection: relation between an **AS Slot** decrypting content and an **AS Slot** subsequently re-encrypting the decrypted content indicating such re-encryption is permitted

Key Ladder: function of the **Key Ladder Block** as defined in ETSI GS ECI 001-5-2 [5] for computing control words and associated control word usage information for application in the content decryption or re-encryption function of a CPE

Key Ladder Block: robust secure mechanism to compute decryption, encryption and authentication keys as defined in ETSI GS ECI 001-5-2 [5], both **Key Ladder** and **Authentication Mechanism**

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

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: ECI Client that encrypts such that it can only be decoded by the targeted **Micro Client** or group of **Micro Clients**

operator: organization providing **Platform Operations** that is enlisted with the **ECI TA** for signing the **ECI** eco system

NOTE: An **Operator** may operate multiple **Platform Operations**.

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

Provisioning Server: server, typically located in a secure back office location, that provisions keys and other secure information to facilitate an encryption or decryption function through an **AS Slot**

revocation: status of exclusion of an entity in accordance with its enumeration in a **Revocation List**

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

robustness: property of the implementation of a specified secure function representing the effort and/or cost involved to compromise the security of the implemented secure function

root certificate: trusted certificate that is the origin of authentication for a chain of certificates

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

3.2 Abbreviations

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

ACF	Advanced Security Control Field
AD	Input of the Key Ladder Block
AES	Advanced Encryption Standard
AK	Authentication Key
API	Application Programming Interface
ARK	Advanced Security Random Key
AS	Advanced Security
CA	Conditional Access
CBC	Cypher Block Chaining
CENC	Common Encryption
CISSA	Common IPTV Software-oriented Scrambling Algorithm
CP	Content Properties
CPE	Customer Premises Equipment
CPS	Certificate Processing Subsystem
CSA	Common Scrambling Algorithm
CTR	Counter Mode
CW	Control Word
DRM	Digital Rights Management
EAC	Export Authorization Certificate
EAOC	Export Authorization Operator Certificate
ECI	Embedded Common Interface
ECM	Entitlement Control Message
EGC	Export Group Certificate
ERC	Export Revocation Certificate
ESC	Export System Certificate
LK	Link Key
MPEG	Moving Picture Experts Group
MSCSK	Micro Server Chipset Secret Key
PES	Packetized Elementary Stream
PO	Platform Operator
POC	Platform Operation Certificate
POPK	Platform Operation Public Key
REAOC	Revocation Export Authentication Operator Certificate
RFU	Reserved for Future Use
RK	Random Key
RL	Revocation List
SPK	Sender Public Key
TA	Trust Authority
TPEGC	Third Party Export Group Certificate
TS	MPEG 2 Transport Stream
URI	Usage Rights Information
XT	eXTension field

4 Principles

4.1 Overview

The present document is part of the Multipart ISG Group Specifications ECI 001, based on the **ECI** architecture ETSI GS ECI 001-1 [1] and **ECI** basic requirements ETSI GS ECI 001-2 [2].

Figure 4.1-1 presents the main principles of the **Advanced Security System**. The core of the **Advanced Security System** is formed by the **Key Ladder Block** as defined in [5], allowing secure processing with secret keys, targeting of keys to specific chips and authentication of the origin of key material.

The basis for loading images is embodied in the **loader core**. It uses the **Certificate Processing Subsystem** to verify the **ECI** status of **ECI Host** images, **ECI Client** images and Platform Operator (PO) credentials using a recent **ECI Root Key** and **ECI** root **Revocation List**. The version numbers of the **ECI Root Key** and **ECI** root **Revocation List** used by the **ECI Host** and other **ECI Clients** can be checked by **ECI Clients** that are loaded. These can refuse to descramble content on detecting unacceptable versions in accordance with the **ECI** revocation enforcement principle. Encrypted **ECI Client** images are decrypted upon loading.

Each **ECI Client** uses an Advanced Security slot. The **AS Slot** is identified by the **Platform Operation** Public Key of the **ECI Client.** The **ECI Host** ensures that **ECI Client** interactions through the **AS-API** are directed to the **AS Slot** allocated to that **ECI Client**. Each **AS Slot** is described by a slot state and a session state per encryption/decryption operation. The **AS Slot** can be used either for decryption purposes or for encryption purposes. The **AS Slot** session state also includes a configuration (config) defining the details of the operation and how the session state should be authenticated. The **ECI Client** provides the configuration information and input for other state information. The **Key Ladder Block** is used to authenticate SPK, POPK and the configuration information. The **AS Slot** can supply random numbers to selected **Key Ladder Block** inputs so as to generate random keys or to use as a nonce to ensure freshly computed **Key Ladder Block** inputs. This mechanism can be used to prevent replay of encrypted content and to ensure always online provisioning of an **AS Slot** by a **Provisioning Server**.

When decrypting content the **Content Properties** can be authenticated along with computing the control words, thus creating a strong link with the decrypted content. **Content Properties** are forwarded with the content to any standard output to ensure the proper setting of protection mechanisms for such an output. These properties are compared to those with which the content is re-encrypted on an **Export Connection**. An **Export Connection** is permitted only through the appropriate export/import **Certificate Chain**s. These are verified by the **Certificate Processing Subsystem** on behalf of the **AS Slot**. Standard outputs can be disabled through the output control mechanism.

Computed control words can be synchronized to MPEG Transport Stream formatted content using the alternating bit protocol. For this purpose the content processing system uses a double buffering mechanism with a current/next control word for stream processing.

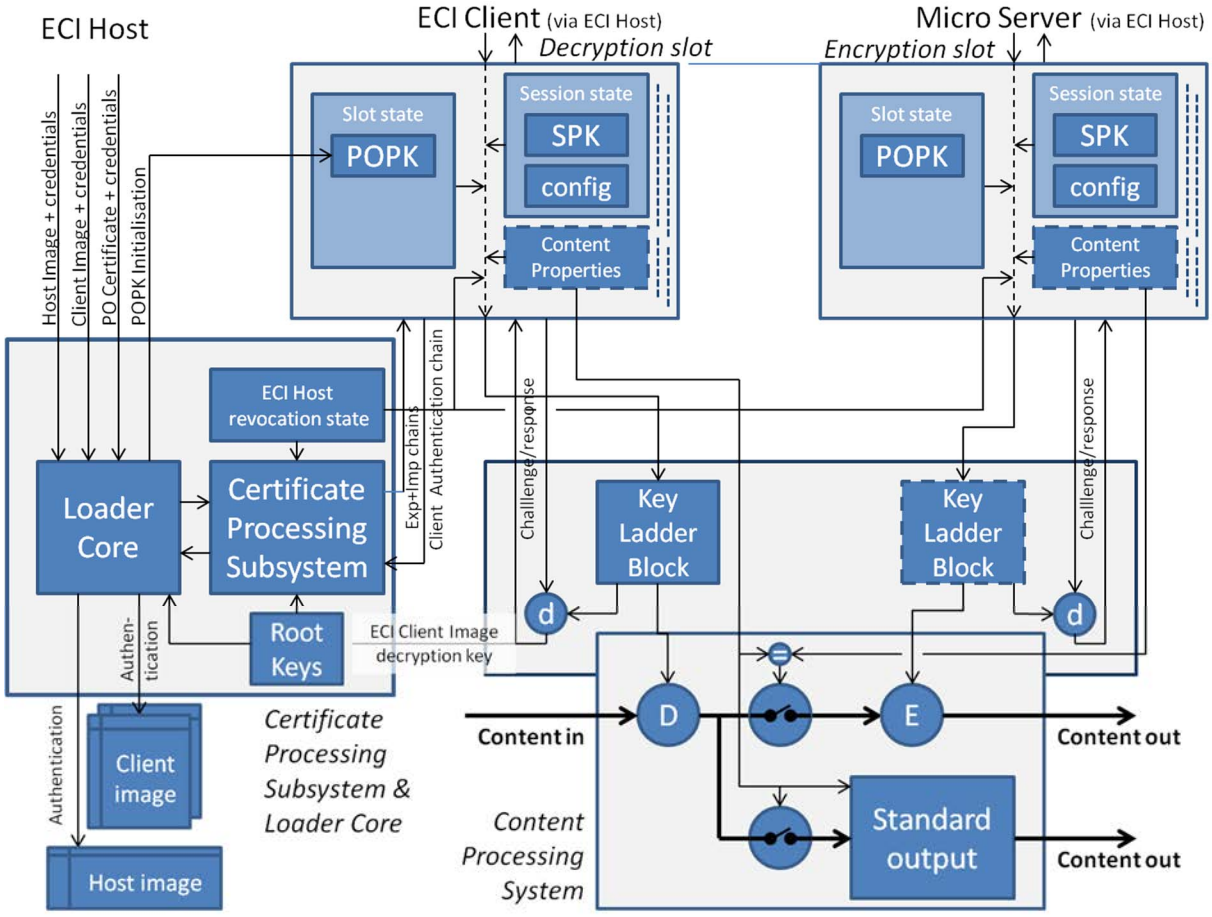

Figure 4.1-1: Block diagram of Advanced Security System

4.2 System Robustness Model

The **AS System** requires a robust implementation. **Robustness** is typically measured in terms of the effort and/or cost required to circumvent any security measures: i.e. observe secret values or manipulate state or values in a secure system.

The present document does not define a specific **Robustness** regime for various **ECI** functions. Nevertheless, the **ECI Robustness** architecture is based on the premise that some functions are more robust than others. This is illustrated in Figure 4.2-1.

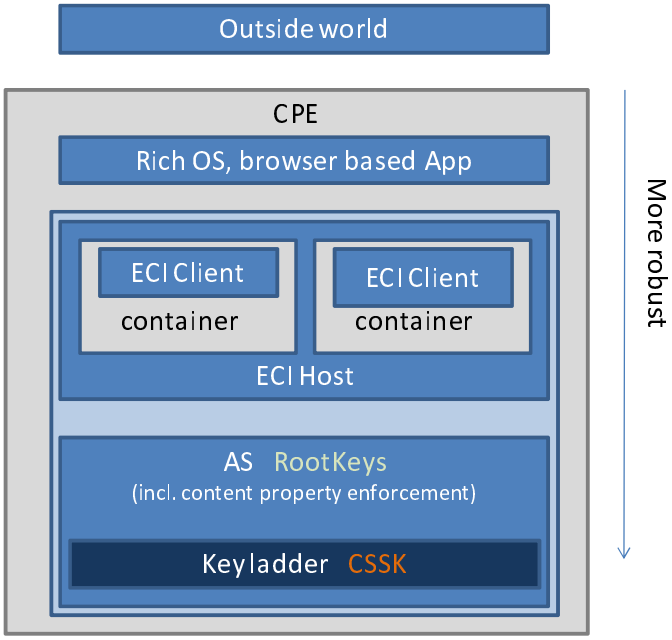

Figure 4.2-1: System robustness premise for ECI

The least robust environment is the outside world where any threat may exist. Data passing through that environment should be protected against manipulation and unauthorized inspection using authentication and encryption techniques. The rich operating system (typically including a browser) can be somewhat robust against manipulation and invasion, but is typically not able to withstand user-assisted or aggressive external hacking attacks. The **ECI Clients** and **ECI Host** security sensitive functions operate in an environment that is well protected from such attacks. In case the **ECI Host** is compromised also **ECI Clients** are compromised. In addition to resilience towards outside attacks the **ECI Clients** are sandboxed using the **ECI** virtual machine [4]: i.e. they neither can access information in the **ECI Host** nor can access any other **ECI Client**, apart from through defined **ECI** API interfaces. The **ECI Host** also ensures the **ECI Clients** have access to the **Advanced Security System** and the **Key Ladder Block**. At the core of the **Key Ladder Block** is the **Chip Set Secret Key** with allows each **ECI CPE** to be addressed uniquely. Typically the **Key Ladder Block** and major parts of the **Advanced Security System** are implemented in hardware and/or highly robust firmware.

4.3 Specification Principles

4.3.1 Implementation Freedom

The present document defines states and functions that operate on the AS System which results in a new state. The specific representation of the state of an implementation is not defined by the present document and may be completely defined by the implementation as long as the behaviour of the implementation can be reconstructed to states and state transition sequences using the transition functions as defined in the present document.

NOTE: In many cases the **Key Ladder** function as defined in [5] is a substantial part of the state transition function.

For example, an AS implementation may have a fast CPS that can re-authenticate POPK from the Platform Operation **Certificate Chain** for every application of the **Key Ladder Block**. The **AS Slot** in this case does not have to store POPK as an authenticated value in a tamper proof way. Similarly, some implementations may decide to compute LK_1 (the top level symmetrical key in the key ladder) once using two asymmetrical cryptography operations for an **AS Slot**, while others, that can perform the asymmetrical cryptography operations sufficiently fast, may recompute LK_1 from its original inputs for every **Key Ladder** application.

4.3.2 Specification Style and relation to AS-API

There is no direct API between the **ECI Client** and the **Advanced Security System**. The **ECI Host** acts as a conduit. Nevertheless, the definitions of the operations on an **AS Slot** map directly to the messages of the **AS API** as defined in [3], with exception of the slotId parameter, which is not required for **ECI Client** AS API messages. The **ECI Host** provides the slotId parameter to the **AS system**.

The transactions by the **ECI Host** (on behalf of an **ECI Client**) on an **AS Slot** are defined as C-functions declarations. These describe an atomic transaction on the state of the **AS Slot**. This can result in a new slot state. The specific representation of function parameters is not of direct consequence to functionality specified in the present document, except when cryptography functions are concerned. However, the representation is significant for the definition of the **AS API** in [3].

5 **Key Ladder** Application and Associated Functions

5.1 General

The **Key Ladder** and **Authentication Mechanism** defined in [5] play a central role in all robustly implemented secret key computations in an **ECI CPE**. The **Advanced Security System** shall apply these functions as defined in [5] with inputs and outputs as defined in the present document. All inputs to the **Key Ladder Block** shall be controlled by the **AS System**; any observation or manipulation shall not be possible in accordance with applicable AS **Robustness** rules and the **Key Ladder Block** specification [5].

5.2 **AS System** and client data authentication

The **Advanced Security System** can be provided with data from an **ECI Client**. The **AS System** provides a means to verify the authenticity of this data using the AD input of the **Key Ladder Block.**

The **AS System** computes the AD input of the **Key Ladder Block** as the hash of additional data to be authenticated in conjunction with a CW or AK computation. Following the bit-string notation of [5] AD shall be computed as:

$$AD = hash(ACF \| lm \| ARK \| P_1 \|...\| P_m \| C_1 \|...\| C_m \| XT)$$

Function *hash* is defined in clause A.1. lm is an 8 bit input containing the binary representation of m. The value of m corresponds to the value of m in the **Key Ladder Block** definition. The length of each P_i is 2 048 bits for the purpose of carrying public keys (POPK values). The length of C_i is defined as equal to the length of the SessionConfig structure in clause 6.2.2.5, the length of XT is 256, and serves a general purpose extension mechanism. It shall be set to 0. ARK is a 128-bit number, intended to represent a random value or all 0's in case no random input is needed. ACF is a control value defining the mode of operation.

5.3 Asymmetrical **Micro Server** mode

The **AS System** applies the **Authentication Mechanism** for the purpose of loading a **Micro Server** sender secret key into an **AS Slot** for the purpose of asymmetrical authentication between **Micro Servers** and **Micro Clients**.

Figure 5.3-1 shows the basic principle of the overall computation of asymmetrical authentication mode.

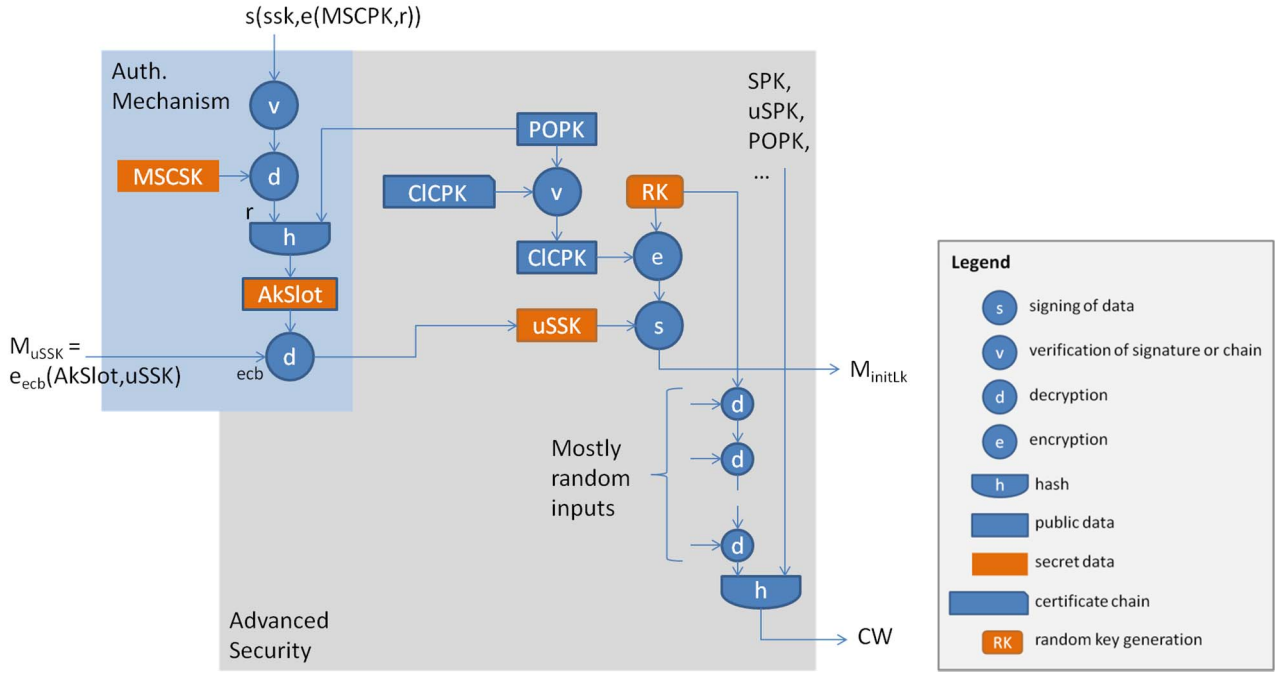

Figure 5.3-1: Computation of the Asymmetrical Micro Server mode

The **Authentication Mechanism** is used to compute the **AS Slot** authentication key AkSlot, using a.o. the ChipSet Key here called MSCSK. AkSlot is used to load the **Micro Server** Secret Key uSSK. The **Certificate Processing Subsystem** is used to authenticate the Chipset Public Key ClCPK of the target **Micro Client** using POPK as a root and a **Certificate Chain** containing ClCPK in the last **Certificate** of the chain. A random key RK is generated and ClCPK and uSSK are used to generate the **Micro Client Key Ladder** initialization message M_{initLk}. The random key is also used as the top level symmetrical key of a **Key Ladder** with the same structure and hash function as defined in [5], clause 5.1. The computed control word CW is used for encryption in by the **Micro Server**. The regular **Key Ladder** can be used in a **Micro Client** to compute CW for the purpose of decryption.

Clause 6.2.4.10 defines the specifics of the uSSK computation (the **ldUssk** function).

The computation of M_{initLk} shall be as defined in the computation scheme below, using the specification conventions as [5], clause 5.1:

- Mkey = cl-chipset-ID || E(ClCPK,LK)
- M_{initLk} =(Mkey || S(uSSK,Mkey))
- with || the bitwise concatenation function, cl-chipset-ID the chipset-ID of the client CPE, and with E() the asymmetrical encryption function and S() the asymmetrical signature function as defined in [5], clause 5.2.

On loading uSSK the **AS Slot** shall generate a new RK in accordance with clause A.3.

The computation of CW from LK and its inputs shall be identical to the **Key Ladder** Mechanism defined in [5], clause 5.2 with the same inputs as defined there and the same output (CW, CW-URI) but replacing the computation of LK1 by the RK random session key generated by the **Micro Server AS Slot**.

5.4 Interface to Content Processing System

The **Key Ladder** including the asymmetrical **Micro Server** mode extension can compute control words with complementary CW-URI. These shall be passed securely to a decryption or encryption resource in the content processing subsystem which may (temporarily) store the CW and CW-URI information in association with key synchronization information. For transport stream applications the key synchronization information consists of the current/next bit, thus specifying two storage locations for CW values. For file based applications a single CW location is available for the encryption and decryption resource.

Other information accompanying control words from the **AS Slot** to the content processing system is:

- Application of CW as encryption or decryption control word.

NOTE: Together with CW-URI this is a permission to apply CW.

- Export authentication information.
- Content properties.
- Odd/even property of the control word for TS mode descrambling.

5.5 **AS Key Ladder Block** input output definition

This clause defines the mapping of C-style variables and structures as a representation for the inputs to the K**ey Ladder** and **Authentication Mechanism** and the extensions thereof as defined in clause 5.2 and clause 5.3. The symbols for the input names are used in the rest of the present document to define the various applications.

The mapping of C-structures to an octet sequence is defined with the following (little endian based) rules:

- Bit-fields of structures are mapped first field first starting with the lowest bit (0) of the first octet.
- Structures of a length other than a multiple of 8-bit that are padded at the end to the next multiple of 8-bits with reserved bits that shall be set to zero to the next multiple of 8-bit.
- 16-bit, 32-bit and 64-bit entities are mapped in little endian order (least significant byte first).
- Arrays are mapped in increasing index order.

Octet sequences are mapped to bit strings using the OS2BSP function as defined in ETSI GS ECI 001-5-2 [5].

NOTE: The above rules ensure that the bit numbering order as used for integer values represented by c-variables is equal to that used in ETSI GS ECI 001-5-2 [5] for the corresponding inputs to the **Key Ladder Block**.

The naming convention for the variables is listed in Table 5.5-1.

Table 5.5-1: C-variable naming convention for Key Ladder interface

Key Ladder Block input or output	Bits	C-variable naming convention
CW-URI	64	ulong **cwUri**;
AD	256	uchar **ad**[32]; /* not used directly, see clause 5.2 */
SPK-URI	64	ulong **spkUri**;
SPK_i, i=1..16	2 048 × 16	typdef uchar PubKey[256]; PubKey **spk**[16]; /* (spk[i-1] == SPK_i) */
m		uchar **nSpk**;
input to V	64+ 2 048 + 2 048	typedef struct InputV{ ulong chipsetId; uchar elk1[256]; uchar signature[256]; } InputV; InputV **inputV**;
$E(LK_i,LK_{i+1})$, i=1..24; LK_{t+1}=r	256 × 24	typedef uchar SymKey[32]; Symkey **elk**[i]; /* (elk[i-1] == $E(LK_i,LK_{i+1})$) */ /* **C-input** == $E(LK_{t-1},LK_{t1})$, i.e. the one but last input ***/**
t		uchar **nElk**;
τ_b		value set to 0
Chipset-ID	64	ulong **chipsetId**;
Challenge	128	uchar **challenge**[16];
Response	128	uchar **response**[16];
Below the inputs and outputs defined in the present document are defined.		
ACF	128-8	uchar **acf**[15]; /* operation mode */
ARK	128	uchar **ark**[16];
P_i	2 048 × 32	PubKey **pk**[32]; /* first **m** values are applied */
C_i	sizeof(SessionConfig) × 32	SessionConfig **config**[?]; /* SessionConfig is defined in clause 6.2.2.6 */
XT	256	uchar **XT**[32]; /* value always set to 0 */
M_{initLk}	64+ 2 048 + 2 048	InputV **mInitLk**;

The following c-functions are defined using the above input variables to produce results.

```
SymKey blockV_blockC_KeyLadder(InputV inputV, SymKey spk)
```

Semantics:

This function computes the function of block V and block C in the **Key Ladder** to produce lk1.

In case of the asymmetrical server the following function computes the initialization message for the target **Micro Client** as defined in clause 5.3:

```
InputV asymInitLk1(SymKey lk1, PrivKey ussk, PubKey spk);
```

Semantics:

This function computes the initialization message initLk1 in accordance to clause 5.3.

The remaining functions of the **Key Ladder** are performed by:

```
keyLadder(SymKey lk1, ulong cwUri, uchar acf[15], uchar ark[16],
    PubKey popk[16], SSCnfg clCnf[16], uchar XT[32], ulong spkUri, uint nSpk, PubKey spk[16],
    uchar nElk, SymKey elk[32])
```

Semantics:

This function computes the remainder of the **Key Ladder** using lk1 as the result of block D in the **Key Ladder** to produce a CW result for the content processing system.

The functions of the **Authentication Mechanism** to compute an AK key are performed by:

```
SymKey AuthMech(InputV inputV, uchar acf[15], uchar ark[16],
    PubKey pk[16], SSCnfg clCnf[16], char XT[32], ulong spkUri, uint nSpk,
    uint spkIndx, PubKey spk[16])
```

Semantics:

This function computes **Authentication Mechanism** up to AK and delivers the result.

In order to use the computed AK key the following function is defined using the challenge-response interface and the function block d in the **Authentication Mechanism** of [5], clause 6.

```
uchar[16] AuthMechResponse(SymKey ak, uchar[16] challenge)
```

Semantics:

This function computes the response on a challenge input using AK as the authentication key as defined in the **Authentication Mechanism.**

5.6 ACF definition

The ACF input to the **Key Ladder Block** serves to define the main modes of operation mode. The value for the acf[0] parameter is defined in Table 5.6-1.

Table 5.6-1: ACF[0] for Key Ladder application

Acf[0]	Value	Description
AcfCw1Mode	0x11	**Key Ladder** operation as defined in the present document. acf [1]..acf[14] shall be equal to 0x00.
AcfAk1Mode	0x12	**Authentication Mechanism** operation as defined in the present document. The value of acf[1] is referred to as **AkModeField**. The applicable values are defined in Table 5.6-2. Acf[2]..acf[14] shall be equal to 0x00.
reserved	Other	Reserved for future use.

The complementary c-definition for this AcfCw1Mode for application as the **Key Ladder** parameter is:

```
const uchar acfCw1Mode= { AcfCw1Mode, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 };
```

The complementary c-definition for AcfAk1Mode mode for application as the **Key Ladder** parameter is:

```
const uchar acfAk1Mode= { AcfAk1Mode, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 };
```

Table 5.6-2: AkModeField definition for AcfAk1Mode

Register	Bit	Value		Description
AkUseFlag	8	0b0	**AkUseAS**	AK application for **Advanced Security System** only.
		0b1	**AkUseCl**	AK application for the **ECI Client**.
AkOnline	7	0b0	**AkOffline**	AK is established in a unidirectional "offline" mode. Challenge/responses can be pre-computed.
		0b1	**AkOnline**	AK is established using a random nonce AKRK which will require a challenge response to be computed "online".
AkAsAppl only if AkUseFlag= **AkUseAS** reserved otherwise	0..3	0x0	**AkConfigAuth**	Authenticate Configuration element of **AS Slot**.
		0x1	**AkLdUssk**	Use AK to decrypt and load a **Micro Server** uSSK key.
		0x2	**AkClImg**	Use AK to decrypt the key for decrypting the **ECI Client** image to be loaded.
		0x3.. 0xF	reserved	Reserved for future use.
RFU	other	0		Reserved for future use. Value shall be set to zero.

6 Advanced Security Slot

6.1 **Advanced Security Slot** introduction

The **Advanced Security System** contains state-information for each loaded **ECI Client**. The identification that binds an **ECI Client** to an **AS Slot** is the Platform Operation Public Key (POPK) of the **ECI Client**. The **ECI Host** loads the POPK of an **ECI Client** into an available **AS Slot**. From then on the state of an **AS Slot** is associated with this **ECI Client**. Any meaningful operation of the **AS Slot** will use POPK as an input, thus making the result specific for the bound **ECI Client** and meaningless for others.

ECI Host Robustness ensures an **ECI Client** can only access information of its designated slot: in case the **ECI Host** functions correctly it will ensure that only the designated **ECI Client** will have access to its own **AS Slot**. In case the **ECI Host** is somehow compromised the POPK "locking mechanism" ensures that only AS inputs to an **AS Slo**t that form a coherent set can produce meaningful results in the form of decryption keys (CW) and associated **Content Properties** or an Authentication Key (AK).

In case the **ECI Host** would decide to repurpose an **AS Slot** for another **ECI Client,** any built up state of the previous **ECI Client** (POPK) is erased.

6.2 **AS Slot** Definition

6.2.1 General

The **AS Slot** is defined in terms of ***state variables*** and ***input variables*** to ***state modification functions***. The representation of state, input and output values in this clause is chosen such that the operations performed on them are defined in terms of the here defined binary representations. This is specifically relevant with respect to their incorporation in cryptographic and **Key Ladder** computations. Actual implementations can select their own state representations but have to translate any custom representation to the representation specified here for input to any cryptographic operation.

All state variables of an **AS Slot** shall be robustly protected against any malicious modification. Some state variables hold information that has to remain secret: such registers shall be robustly protected from unauthorized access. Such variables are defined using "Secret" as part of the C-type definition. Any computation based on the value of a secret variable shall be kept secret except when such a result is explicitly shared. Any storage location of a secret value and/or computation with a secret value shall have the same **Robustness** as the one required for the **Key Ladder Block [**5**]**.

The **AS Slot** has sessions. Each session operates in accordance with the settings of its configuration which is part of the session state. The session configuration is set by the **ECI Client** and has to be authenticated before use by using the **Authentication Mechanism** or by using the implicit authentication properties of the **Key Ladder**.

All state variables and functions are defined in terms of the C-language [6]. The sequence order of the C-language is not strictly observed in the sense that items may be defined after their use. Arrays of fixed size are copied with a single assignment statement (rather than copying the pointer value) as if they were contained in a "struct" structure.

Regarding errors, the code has been kept more readable by defining implicit error checking. Assigning a reserved value to a state variable or field thereof (see definition of the field) shall be an error. If the right side of such an assignment expression is based on a single function parameter an error is returned for that parameter: value -i for parameter i.

All default values for fields and variables are defined as 0, unless explicitly specified otherwise.

6.2.2 **AS Slot** state definition

6.2.2.1 Slot and session state

The **AS Slot** state is defined as a common slot state and a session state per encryption or decryption session. The structure for the slot state is defined below. In Table 6.2-1 the fields are defined.

```
#define NSLOTS       /* (maximum) number of slots */
#define NSESSIONS    /* (maximum) number of sessions */
#define MaxSpkEncr 4 /* maximum number of encryption SPK values */

typedef SessionState {
    bool            active;
    uint            configAuthMode:4;
    uint            mh;
    SessionConfig   config;
    PubKey          spk;
    ulong           spkUri;
    uchar           spkIndx;
    int             coupledSessionId;
    uint            nEncr;
    PubKey          encrSpk[MaxSpkEncr];
    PubKey          encrPopk[MaxSpkEncr];
    ulong           encrCwUri;
    Secret SymKey   lk1;
    Secret PrivKey   ussk;
    RkState rkState;
    importExportState ies;
} SessionState ;

typedef struct SlotState {
    uint            version:4;
    uint            slotMode:4;
    uint            clientCheckFlag:1;
    uint            reserved:3;
    uint            POClRLVnr: 24;
    PubKey          popk;
    SymKey          slotRk;
    Secret SymKey   akClient;
    SessionState    se[NSESSIONS];
} FixedSlotState;

SlotState ss[NSLOTS]
```

Table 6.2-1: AS Slot state structure definition

Field	Description
active	True if session is active, false otherwise. Default state false.
configAuthMode	Mode by which the configuration of the slot has been authenticated. The permitted values are: **ConfigAuthModeNone:** 0x0, slot configuration has not been authenticated. **ConfigAuthModeAk1:** 0x1, slot configuration has been authenticated using the AK mechanism as defined in clause 6.2.4.8. All other values are reserved.
Mh	Media handle to which the **AS Slot** session is associated.
clientCheckFlag	A new **ECI Client** has been loaded. Verification of **POClRLVnr** shall proceed on session initialization. Default value is 0b1.
Reserved	Field is reserved; it shall be set to zero.
POClRLVnr	Version number of **Platform Operation** Client **Revocation List** that was used to verify the **ECI Client** before loading. Will be checked at each **ECI Client** session initialization against the minimum version that the **ECI Client** expects.
slotConfig	Slot configuration.
spk	Public key used to compute LK1 and AK.
spkUri	SPK vector usage rules information register used to compute LK1.
spkIndx	Index selecting SPK register location in SPK vector for LK1 calculation.
coupledSessionId	Only applicable if the **AS Slot** session is in decryption mode. Second (decryption) session coupled to this one. Decoded content streams are joined and **Content Properties** are compared. The default value is -1.
nEncr	Number of SPK/POPK input values used for encryption (excluding slot's spk).
encrSpk	SPK values for encrypting content with **Key Ladder**.
encrPopk	POPK values for encrypting content with **Key Ladder**.

Field	Description
encrCwUri	CwUri value for encrypting content with **Key Ladder**.
lk1	Top level link key for computing control words using the **Key Ladder**.
Ussk	**Micro Server** secret key (for **Micro Server** applications).
rkState	Random Key state of the session.
ies	Import/export state of the session.
version	Version of the slot state. The permitted values are: 0x1: version 1. All other values are reserved.
slotMode	Mode in which the slot operates. The permitted values are: **SlotModeDecr:** 0x1, slot operated in decryption mode. **SlotModeEncr:** 0x2, slot operates in encryption mode. All other values are reserved.
popk	Public Key of **ECI Client** using this slot.
slotRK	Random number used in online challenge response protocols, e.g. with a **Provisioning Server**. The value is set on slot initialization.
akClient	Authentication Key for Client processing purposes.
se	Session state (for all sessions in an **AS Slot**).
ss	**AS Slot** State (for all slots).

Unless stated otherwise the default value of each state element at initialization is zero.

6.2.2.2 Decryption configuration

The applicable configuration state for an **AS Slot** session in decryption mode is defined in the c-code below and its description is given in Table 6.2-2. It defines the details of the AS-slot session operation when operating in decryption mode. This data can be authenticated by applying suitable **Authentication Mechanism** or **Key Ladder** calculations.

```
typedef struct DecryptConfig {
    uint            configVersion:4;
    uint            reserved1:4;
    uint            klModeAuth:1;
    uint            akModeAuth:1;
    uint            rkKlMode:1;
    uint            spk0NoDecrypt:1;
    uint            reserved2:6;
    RKMode          rkDecrMode;
    EciRootState    minEciRootState;
    uint            minClientVersion:24;
} DecryptConfig;
```

Table 6.2-2: DecryptConfig structure definition

Field	Description
configVersion	Version of the decryption configuration. Defined value is 0x1: version 1. All other values are reserved. An **AS Slot** session shall refuse to execute any state transition function if this field does not have a permitted value.
reserved1	Reserved field; shall be set to 0.
klModeAuth	If this bit is set the **AS Slot** session shall apply the ClientConfig for authentication with all **Key Ladder** calculations. This bit itself is authenticated on all **Key Ladder** calculations.
akModeAuth	If this bit is set the **AS Slot** session shall verify that the configAuthMode is set to ConfigAuthModeAk1 before permitting any **Key Ladder** calculation. This bit itself is authenticated on all **Key Ladder** calculations.
rkKlMode	If this flag is set, **slotRK** shall be applied in all **Key Ladder** calculations for the **AS Slot**.
spk0NoDecrypt	When set it is not permitted to use spk[0] (spkIndx==0 as input to **Key Ladder** function) for authentication of the slot LK1 when in decryption mode.
reserved2	Reserved field; shall be set to 0.
rkDecrMode	Defines the application of a random session key for the **Key Ladder** calculations. See clause 6.2.2.5.
minEciRootState	Minimal value for **ECI** root version and root **Revocation List** version. If the CPS has applied an **ECI Root Key** or a root **Revocation List** for **ECI** authentication purposes less than the values in this structure no **Key Ladder** computation shall be permitted for the session.
minClientVersion	Version of the **ECI Client**. Used to verify **Revocation List** version numbers for POPK.

6.2.2.3 Encryption Configuration

The applicable configuration state for an **AS Slot** session in encryption mode is defined in the c-code below and its description is given in Table 6.2-3.

```
typedef struct EncryptConfig {
    uint            configVersion:4;
    uint            reserved1:4;
    uint            microServerVersion:24;
    uint            asymKlMode:1;
    unit            rkKlMode:1;
    uint            reserved2:22;
    RkMode          rkEncrMode;
    uchar           basicUriTrfr;
    uint            contPropControl;
    ContProp        defaultCP;
    EciRootState    minEciRootState;
} EncryptConfig;
```

Table 6.2-3: EncryptConfig structure definition

Field	Description
configVersion	Version of the encryption configuration. Defined value is 0x1: version 1. All other values are reserved. An **AS Slot** shall not execute any state transition function if this field does not have a permitted value.
Reserved1	Reserved field; shall be set to 0.
microServerVersion	Version number of the **Micro Server** configuration. Also used as minimum **Revocation List** version number for **Micro Client** authentication in asymmetrical **Micro Server** mode.
asymKlMode	If this flag is set the **Key Ladder** shall operate in accordance with the asymmetrical client **Authentication Mechanism** defined in clause 5.3.
rkKlMode	If this flag is set **slotRK** shall be applied in any **Key Ladder** calculation.
Reserved2	Reserved field; shall be set to 0.
rkEncrMode	Defines the application of a random key for the **Key Ladder** calculations. See clause 6.2.2.5.
basicUriTrfr	Defines state transformations of the basic URI from the import connection before applying the basic URI as content property of the encrypted content. See Table 6.2-5 for the values.
contPropControl	Defines how the **Content Properties** of the encrypted content are computed. See Table 6.2-4.
defaultCP	A default value for all content property fields. Application in the **Key Ladder** calculation is under control of the contPropControl field.
minEciRootState	Minimal value for **ECI** root version and root **Revocation List** version. If the CPS has applied an **ECI Root Key** or a root **Revocation List** for **ECI** authentication purposes less than the values in this structure, no **Key Ladder** computation shall be permitted.

The contPropControlFields field is an array of 16 2-bit fields. The 2-bit fields indicate how the Field1 **Content Properties** for the encrypted output is controlled. The decription is given in Table 6.2-4. CpControlFlag bit-2n and bit 2n+1 shall correspond to Field1-byte n.

Table 6.2-4: CpCtrl definition

Flag name	Value	Description
CpCtrlCopy	0b00	CP Field1 content property byte shall be copied from import connection
CpCtrlDef	0b01	CP Field1 content property byte shall be set to the value of the corresponding defaultCP byte
CpCtrlMS	0b10	CP Field1 content property byte is set by **Micro Server**
Reserved	0b11	Value is reserved

The basicUriTrfr field modifies the above behaviour of CPControlFlags for the BasicUri field when the state of its CPControlFlag equals **CPCopy**. Table 6.2-5 defines the alternate behaviour.

Table 6.2-5: BasicUriTrfr values and description

Flag name	Value	Description
JustCopy	0x00	Field1 content property byte is copied from import connection
NoMoreCopy	0x01	A basicURI state of **RedistributionProtected** shall be transformed into a **ViewOnly** state
Reserved	Other	Reserved for future use
NOTE: By setting the BasicUriTrfr state to NoMoreCopy the **Micro Client** system will only permit streaming for any protected content input to the **Micro Server**.		

6.2.2.4 Random session Key control

The RKMode structure as defined in the c-code below and Table 6.2-6 defines the mode in which the random session key has to be applied in the **Key Ladder**.

```
typedef struct RKMode {
    uint    mode:2;
    uint    limit:6;
} DecryptConfig;
```

Table 6.2-6: Random Key structure for decryption and encryption session

Field	Description
mode	Defines the mode of application of the random session key. The values are: • 0b00: **RKModeNone**, No Random session key inserted. • 0b10: **RKModeDataLimit** Random session key applied with data limit. • 0x11: **RKModeTimeLimit** Random session key applied with time limit. • 0b01: value reserved.
limit	The value defines the applicable limit in terms of real time seconds or kbytes of data that is decrypted or encrypted since the initialization of the random key. The function limitValue() defines the actual limit value that applies. Value 63 is reserved.

```
uint limitValue(uint limit) {
    uint val;

    if (limit==0) return 1;
    limit -=1;
    if (limit&0b1 == 0b0) val=2 else val=3;
    return val * (1<<(limit>>1));
}
```

6.2.2.5 Total session configuration

The complete configuration control information for an **AS Slot** session in encryption or decryption mode is defined in the SessionConfig structure defined below. In case of the **AS Slot** being in encryption mode it includes the configuration information for subsequent decryption.

```
typedef struct SessionConfig {
    EncryptConfig   encryptConfig; /* configuration for encryption */
    DecryptConfig   decryptConfig; /* configuration for decryption */
} SessionConfig;
```

The structure cpsEciRootState defining the **ECI** Root State for validating **ECI Certificate Chains** is defined in the c-code below and in Table 6.2-7.

```
typedef struct EciRootState {
    uchar       rootVersion;
    uint        rlVersion:24;
} EciRootState;

EciRootState cpsEciRootState; /* contains the minimum value from the CPS */
```

Table 6.2-7: EciRootState structure field description

Field	Description
rootVersion	Version of the **ECI Root Certificate**
rlVersion	Version of the **Revocation List** applied with the **Root Certificate**
NOTE: EciRootState is typically applied as an under bound (minimum value) that is permissible for the **ECI** root version and **Revocation List** version when loading **ECI Host** and **ECI Client** information.	

The following function is defined that checks if the cpsEciRootState is sufficient to proceed with computing a key:

```
bool cpsEciRootStateOk(uint slotId, uint sessionId) {
  if (ss[slotId].slotMode == SlotModeDecr)
    return
        (cpsEciRootState.rootVersion >=
         ss[slotId].se[sessionId].config.decryptConfig.minEciRootState.rootVersion)
     && (cpsEciRootState.rlVersion >=
         ss[slotId].se[sessionId].config.decryptConfig.minEciRootState.rlVersion);

  if (ss[slotId].slotMode == SlotModeEncr)
    return
        (cpsEciRootState.rootVersion >=
         ss[slotId].se[sessionId].config.encryptConfig.minEciRootState.rootVersion)
     && (cpsEciRootState.rlVersion >=
         ss[slotId].se[sessionId].config.encryptConfig.minEciRootState.rlVersion);

  /* following should not occur */
  return false;
}
```

Preconditions:

- **AS-slot** slotId is initialized.

6.2.2.6 Random Session Key state

For each decryption or encryption session associated with an **AS Slot,** the **AS Slot** stores random key state information as defined in the c-code below and described in Table 6.2-8.

```
typedef struct RkState {
    SymKey  rkCurrent;
    SymKey  rkNext;
    ulong   limitCounter;
} RkState;
```

Table 6.2-8: The RkState Random Key State field description

Field	Description
rkCurrent	Current random key used for insertion in **Key Ladder** to compute CW.
rkNext	Next value of the random key to be inserted in the **Key Ladder** to compute CW.
limitCounter	Counter indicating the usage state of the current key in units related to the limit value applying to the key. The value counts the remaining units that can still be encrypted or decrypted based on a CW computed with the rkCurrent.

The **limitCounter** field shall be incremented on application of the CW.

NOTE: Implementations may effectively implement the counter as part of the content processing system.

6.2.2.7 Import Export state

Each encryption session has one decryption session associated with it from which it imports the content to be re-encrypted. Import shall be possible simultaneously for (at least) two export groups of the exporting sessions, permitting seamless changeover.

Two decryption sessions can be coupled. This permits different sub-streams that require other control words (computed by the same **AS Slot**) to be merged into one compound stream with one set of **Content Properties** before forwarding to industry standard outputs or export. As part of merging the AS system checks that the **Content Properties** of the merged streams are equal.

NOTE: Comparing **Content Properties** can also involve the export group id, which ensures that export chain processing is required for both coupled sessions are equal.

In conjunction with the random key state this is the session bound state in the **AS Slot**. The session state 'se' is defined below in the c-code; the field descriptions are listed in Table 6.2-9.

```
#define MaxExpGroupIds 2

typedef struct ImportExportState {
    int     importSlotId;
    int     importSession;
    uchar   expGrpId[MaxExpGrpId];
    bool    importPermitted[MaxExpGrpId];
    RkState rkState;
} ImportExportState;

#define ImportNone -1
```

Table 6.2-9: ImportExportState structure definition

Field	Description
importSlotId	Only applicable if the **AS Slot** is in encryption mode. The value is the slot number from which content is imported ('import slot'). The default value is -1.
importSession	Only applicable if the **AS Slot** is in encryption mode. The value is the session number in the import slot from which content is imported. The default value is -1.
expGrpId[eid]	Only applicable if the **AS Slot** is in encryption mode. The export group id of the exporting **AS Slot** from which the content can be imported. Value 0x00 is reserved.
importPermitted[eid]	Only applicable if the **AS Slot** is in encryption mode. Set to true if the for **expGrpId[eid]** is permitted by the exporting **AS Slot**; false otherwise. The default value is false.
rkState	State of the random session key for this session.

The **AS System** shall remove an import session (set corresponding ImportPermitted field to false) if the corresponding decryption session is reset or re-initialized. The **AS System** shall reset all the sessions of an **AS Slot** on the reset or re-initialization of an **AS Slot**.

6.2.3 Content Property Authentication

ECI Clients that perform decryption functions provide the **ECI Host** with the Content Property values through the respective content property API. The **ECI Host** shall input these values to the **Advanced Security System** in combination with the data required to compute the control word for the applicable content. The **Advanced Security System** shall ensure the proper enforcement of the **Content Properties** and validate the **Content Properties** by using them to compute the C-input to the **AS Key Ladder Block**.

Micro Servers use the **Content Properties** passed on and/or processed by the **AS System** or the **ECI Client** using the same mechanism as above for computing in process of computing the C-input to the **Key Ladder**. **AS Slot**s used in encryption mode verify the **Content Properties** supplied by the **Micro Server** against those forwarded by the decryption resource in accordance with the **Micro Server** configuration settings. Encryption is halted on detection of a mismatch.

For the purpose of authentication and verification **Content Properties** are combined into a byte-sequence in two stages. The first stage combines smaller fixed length content property fields into *field1*. The fieldControl byte controls the presence of byte-size content property fields for authentication. In the second stage longer content property fields are composed into a byte sequence *field2*. *Field1* and *field2* are concatenated and are input to a hash function which condenses all fields into a 128-bit value for the C-input of the key ladder. Table 6.2-10 presents the field1 structure.

Table 6.2-10: field1 structure definition

Name	Type	Byte number	Description
fieldControl	FieldControl	0,1	This field defines a 16-bit value with the least significant bits in byte 0. See Table 6.2-11.
basicUri	byte	2	The value of this field shall correspond to BasicUri type specification, ETSI GS ECI 001-3 [3], table 9.8.2.5.1-1.
outputControl	byte	3	The value of this field shall correspond to Output Control Vector specification ETSI GS ECI 001-3 [3], table 9.8.2.6.1-1.
standardUri	byte [3]	4-6	The value of this field shall correspond to the Standard URI specification ETSI GS ECI 001-3 [3], table 9.8.2.3.1-1.
exportGroup	byte	7	Interpreted as unsigned integer representing the export group id that applies to the content. Value equal 0 shall be interpreted by the **ECI Host** as no export is permitted; values 0x80 - 0xFF are reserved.
parentalAuth	byte	8	Corresponds ParCond.basicCondition as defined in ETSI GS ECI 001-3 [3], clause 9.8.1.7.7-1, with bit [0..5] set to 0b000000.
Reserved	byte[7]	9-15	Bytes shall be set to 0x00 by **ECI Hosts** complying with the present document.

Table 6.2-11: FieldControl structure definition

Name	Bit(s)	Description
bit-<n>	2-16	This bit controls the validation of byte-<n> of field1. If the value is 0b1 it shall indicate that the value of bye-<n> shall be validated and be equal to the indicated field, if the value is 0b0 it shall indicate that byte-<n> shall not be validated and the value 0x00 shall be used instead for byte-<n> in field1. Bit-2 shall be set to 0b1 when used as input to compute a decryption control word. This ensures the basicUri is always authenticated against the value used at the time of encryption of the content.
Field2ctrl	0-1	Value 0b00 indicates field2 is not present. Value 0b01 indicates field present and uses the encoding as defined below. Values 0b10 and 0b11 are reserved and shall not be used.

Field2 definition uses a tag/length/value structure with an overall length field to ensure overall integrity. Field2 structure is defined below in this clause.

The function computeField1Decrypt computes the selection logic for the next step in the authentication:

```
    void computeField1Decrypt(uchar field1[16], uchar result1[16]) {
        int i;
        ushort fieldControl = field1[0] + field1[1]<<8;

        result1[0] = field1[0];
        result1[1] = field1[1];
        for (i=2; i<16; i++)
            if (fieldControl>>i & 0b1)
                result1[i] = field[i];
            else
                result1[i] = 0x00;
    }
```

An **AS Slot** in encryption mode shall compute the input to the content property authentication denoted as result1 and a cpMask field for comparing field1 bytes to field1 belonging to content of the importing session as:

```
    void computeField1Encrypt(
                uchar msField1[16],/* field1 for CP from Micro Server Client */
result1[16],            /* result CP for authentication in computing CW */
                ushort cpMask,              /* result mask for comparing msField1 to client's
                                               version of CP field1 */
                EncryptConfig ssEncrypt /* encryption configuration of the AS slot */
    ) {
        int i;
        uchar cp[16]; /* CP value to be computed */

        /* set control bytes of content properties */
        cp[0] = ssEncrypt.defaultCp[0];
        cp[1] = ssEncrypt.defaultCp[1];
        mask = 0x0000;
```

```
        /* process the contPropControl rules to compute cp */
        for (i=2; i<16; i++) {
            switch (ssEncrypt.contPropControl>>(2*i) && 0b11) {
                case CpCtrlCopy: /* shall be copied from import Client */
                    if (i==2) { /* basic URI byte */
                        /* process basicUriTrfr */
                        switch (ssEncrypt.basicUriTrfr) {
                        case BasicUriTrfrNoChange:
                            cp[i]= msField1[i];
                            break;
                        case BasicUriTrfrNoMoreCopy:
                            if ((clField1[2]&0b11) == RedistributionProtected)
                                cp[i]= (msField1[1] & 0xFC) + ViewOnly;
                            else
                                cp[i]= msField1[i];
                            break;
                    } else {   /* all other CP bytes */
                        cp[i]= msField1[i];
                    }
                    cpMask += 1<<i; /* msField1 byte i to be compared to imp client */
.                   break ;
                case CpCtrlDef: /* shall be set to default CP from configuration *?
                    cp[i] = ssEncrypt.defaultCP[i] ;
                    break ;
                case CpCtrlMS: /* shall be defined my software Micro Client */
                    cp[i] = msField1[i];
                    break;
            }
        }

        /* compute input to authentication function same was as for decryption */
        computeField1Decrypt(cp, result1);
    }
```

Field2 is a structured byte sequence as defined below:

```
typdef struct Field2 {
    uint length;        /* number of bytes in content, shall be a multiple of 4 */
    byte content[]; /* content defined below */
} Field2;
```

The content field of the Field2 structure shall contain a sequence of LargeProperty structures each with a unique tag. The LargeProperty is defined by the c-code below. :

```
typdef struct LargeProperty {
    uint  propertyTag;  /* see Table 6.2-12  */
    uint  length;       /* length of property field in bytes
    byte  property[];   /* contains the actual property value */
    byte  padding[];    /* additional bytes set to 0x00 to make LargeProperty a
                           multiple of 4 bytes large */
} LargeProperty;
```

The largeProperty propertyTag field values and corresponding property field definitions are defined in Table 6.2-12.

Table 6.2-12: largeProperty tag field values and meaning

propertyTag value	Property
0x00000000	Reserved
0x00000001	Corresponds to parameter **data** of the **setDcrMarkBasic** message as defined in ETSI GS ECI 001-3 [3], clause 9.8.1.7.5
0x00000002	Corresponds to parameter **data** of the **setDcrMarkExt** message as defined in ETSI GS ECI 001-3 [3], clause 9.8.1.7.6
0x00000003	Corresponds to the parameter **custURI** of **setDcrCustUri** message as defined in ETSI GS ECI 001-3 [3], clause 9.8.1.4.1
Other	Reserved for future use

The **AS System** may refuse any data exceeding its processing capacity for *field2*.

The **AS System** shall check the consistency of any Field2 data parameter using the following checks:

- Length of the constituent LargeProperty structures is equal to the length field of the Field2 structure.

- The padding bytes of all constituent LargeProperty structures are 0x00.

The associated data input C for the **Key Ladder** shall be computed from result1 and field2 according to the following c-code:

```
void computeInputC(uchar result1[16], unchar *field2, uchar input_C[16])
{
    uchar hash2[16], hashIn[32];
    uint i, length;

    if (result1[0] & 0b11 == 0x00) {
        /* no field2 to be included */
        for (i=0; i<16; i++) hashIn[i] = result1[i];
        asHash(hashIn, 16, 128, input_C);
    } else if result[0] & 0b11 = 0x01) {
        /* field2 to be included for input-C */
        length = (Field2 *)field2->length + 4;
        asHash(field2, length, 256, hash2);
        for (i=0; i<16; i++) hashIn[i] = result1[i];
        for (i=0; i<32; i++) hashIn[16+i] = hash2[i];
        asHash(hashIn, 48, 128, input_C);
    }
}
```

asHash is the hash function defined in clause A.1 of a byte sequence in the first parameter, the length of the byte sequence in the second parameter, a bit-length of the result in the third parameter and the result in the last parameter.

Robustness of the outer Hash computation (directly computing shall be at least as high as that of the Hash computation of the inner Hash. The measure of **Robustness** of a hash reflects the effort required for creating a discrepancy between any of the inputs of the hash function and their application of these inputs as content property as well as manipulating the hash function and/or its output.

An example of different levels of **Robustness** of the two hash computations is that the outer hash can be conducted by a robust hardware block whereas the inner hash can be conducted by a robust software implementation.

6.2.4 **AS Slot** functions

6.2.4.1 Overview

The **AS System** can perform various functions on behalf of **ECI Clients** by acting through the **ECI Host**. These functions form the basis for the Advanced Security API in [3]. An "event" reports an asynchronously occurring event back to the **ECI Client**. No response is possible. All other functions are designated to be either asynchronous or synchronous messages initiated by the **ECI Client**; their return values indicate the response status. The functions are listed in Table 6.2-13.

Table 6.2-13: Overview of Advanced Security Functions

Function name	Description	Clause
reqAsInitSlot	Initialize an **AS Slot**	6.2.4.2
reqAsAStartDecryptSession	Start a decryption session in an **AS Slot**	6.2.4.3
reqAsCoupleDecryptSession	Couple two decryption sessions into one	6.2.4.3
reqAsDecoupleDecryptSession	Decouple two coupled decryption sessions	6.2.4.3
reqAsStartEncryptSession	Start an encryption session	6.2.4.3
callAsNextKeySession	Change to the next random key	6.2.4.3
reqAsStopSession	Stop a session	6.2.4.3
reqAsExportConnSetup	Setup an **Export Connection** from decryption to encryption session	6.2.4.4
reqAsExportConnEnd	Terminate existing export session	6.2.4.4
reqAsLoadLk1	Load top level link key in **Key Ladder** for a session	6.2.4.5
reqAsComputeEncrCw	Compute encryption control word	6.2.4.6
reqAsComputeDecrCw	Compute decryption control word	6.2.4.7
reqAsComputeAkClient	Compute authentication key for use by the **ECI Client**	6.2.4.8
reqAsClientChalResp	Use authentication key on behalf of **ECI Client**	6.2.4.8
reqAsAuthDecrConfig	Authenticate the session configuration with **Authentication Mechanisms** (decryption mode)	6.2.4.9

Function name	Description	Clause
reqAsAuthEncrConfig	Authenticate the session configuration and encryption parameters with **Authentication Mechanisms** (encryption mode)	6.2.4.9
reqAsLdUssk	Load **Micro Server** secret key	6.2.4.10
reqAsMInikLk1	Compute asymmetrical **Micro Client** initialization message	6.2.4.11
reqAsClientImageDecrKey	Compute decryption key for **ECI Client** image	6.2.4.12
getAsSlotRk	Read slot random key	6.2.4.13
getAsSessionRk	Read session random key	6.2.4.13
getAsSessionLimitCounter	Read remaining units of session random key	6.2.4.13
setAsSessionLimitEvent	On reaching a limit value for remaining units send event	6.2.4.13
reqAsEventSessionLimit	Event message on reaching limit value	6.2.4.13
getAsClientRnd	Get a new random number for **ECI Client** applications	6.2.4.13
getAsSC	Get current Scrambling Control field status of content in a session	7.9
reqAsEventCpChange	Event message on content property change in imported content in an encryption session	7.9
setAsPermitCPChange	Enable/disable imported content property CP changes taking effect on control word selection for encryption in an encryption session	7.9
setAsSC	Set scrambling control field of encrypted content of an encryption session	7.9
reqAsEventSC	Event message on change of scrambling control field in session	7.9

The pseudo code in the sub-clauses of this clause contains error codes as return value of functions. The Error code values are defined in clause 6.2.4.15 including a verbal description.

6.2.4.2 **AS Slot** initialization

At loading time the **ECI Host** shall reserve an **AS Slot** in the **Advanced Security System** on behalf of each **ECI Client** to be loaded. The **ECI Host** will invoke the reqAsInitSlot function as defined below. All state information of the **AS Slot** shall be set to its default state; any **Export Connection** shall be reset. The **ECI Host** shall load the **ECI Client** using the loader core (see clause 9). The **AS Slot**'s *POClRLVnr* shall reflect the minimum version number of the POC **Revocation List** used to validate the client image. This value will be verified when the **ECI Client** initiates a session.

```
int  reqAsInitSlot(uint slotId, ECI_Certificate_Chain popkChain,
    uint slotVersion, slotMode)
```

Semantics:

All state information of **AS Slot slotId** shall be set to the default state; any **Export Connection** shall be reset.

Loading of POPK requires providing the chain for processing to the CPS system. The rules for processing of POPK chains are defined in clause 8.4. ECI_Certificate_Chain is defined in [3], clause 5.4.1. Once successfully validated the following c-code shall be executed:

```
/* initialise the slot state */
ss[slotId].popk = /* validated value of popk returned by CPS */;
ss[slotId].POClRLVnr = /* value used for client image verification */;
ss[slotId].version = slotVersion;
ss[slotId].slotMode = slotMode;
ss[slotId].configAuthMode = ConfigAuthModeNone;
ss[slotId].rkSlot = rnd128();
return ErrOk;
```

The function rnd128() returns a random 128 bit number as defined in clause A.3 as an array of 16 uchar's.

6.2.4.3 **AS Slot** session and random key control

An **AS Slot** supports different session states for different concurrent sessions. The following functions start and stop sessions for a slot:

```
int reqAsAStartDecryptSession(uint slotId, ushort mh, PubKey spk,
        SessionConfig config, uint *sessionId)
```

Semantics:

The following c-code shall be executed:

```
if (ss[slotId].slotMode != slotModeDecr) return ErrSlotMode;

/* check if a valid client revocation list was used */
if (config.decryptConfig.clientVersion >
    ss[slotId].clientPOClRLVnr) return ErrRevocEnforce;

/* locate any free sessionId; any algorithm is ok */
int i=0;
while (i<NSESSIONS && ss[slotId].se[i].active) i++;
if (i==NSESSIONS) return ErrNoMoreSessions;
/* i contains a non-active session administration block */
*sessionId = i;

/* initialise session state */
ss[slotId].se[i].active = true;
ss[slotId].se[i].mh = mh;
ss[slotId].se[i].coupledSessionId = -1;
ss[slotId].se[i].importPermitted = false;
ss[slotId].se[i].spk = spk;
ss[slotId].se[i].config = config;
ss[slotId].se[i].rkState.rkCurrent = rnd128();
ss[slotId].se[i].rkState.rkNext      = rnd128();
ss[slotId].se[i].rkState.limitCounter =
            limitValue(config.decryptConfig.rkDecrMode.limit);

if (!cpsEciRootStateOk(sdlotId,i)){
    ss[slotId].se[i].active = false;
    return ErrRevocEnforce;
}

return ErrOk;
```

Preconditions:

- **AS Slot** was successfully initialized.

NOTE: The mh (media handle) parameter permits the **ECI Host** to identify the AS decryption session associated to the content decryption session it started. It is not used by the **AS system** itself.

In order to couple two initialized sessions the coupleDecrypSessions function is provided. The second session is coupled to the first; the first becoming the main handle for the combined content.

```
int reqAsCoupleDecryptSession(uint slotId, uint sId1, uint sId2)
```

Semantics:

The following c-code shall be executed:

```
if (ss[slotId].slotMode != slotModeDecr) return ErrSlotMode;
if (!ss[slotId].se[sId1].active) return ErrParam2;
if (!ss[slotId].se[sId2].active) return ErrParam3;
if (ss[slotId].se[sId1].coupledSessionId != -1) return ErrSession1Coupled;
if (ss[slotId].se[sId2].coupledSessionId != -1) return ErrSession2Coupled;

se[slotId][sId1] = sId2;
/* the content processing system is informed on the session coupling &/

return ErrOk;
```

Preconditions:

- Both **AS Slot** sessions were successfully initialized.

The following function can be called to decouple a coupled session:

```
int reqAsDecoupleDecryptSession(uint slotId, uint sessionId)
```

Semantics:

The following c-code shall be executed:

```
if (ss[slotId].slotMode != slotModeDecr) return ErrSlotMode;
if (!ss[slotId].se[sessionId].active) return ErrParam2;
if (se[slotId][sessionId].coupledSessionId == -1)
    return ErrSessionNotCoupled;

ss[slotId].se[sessionId].ies.coupledSessionId = -1;
/* the content processing system is informed on the session decoupling */

return ErrOk;
```

Preconditions:

- The sessions were previously coupled.

The function to initiate an encryption session is:

```
int reqAsStartEncryptSession(uint slotId, ushort mh, uint importSlotId,
        int importSessionId, PubKey spk, SessionConfig config,
        uint nEncr, PubKey encrSpk[MaxSpkEncr],
        PubKey encrPopk[MaxSpkEncr], ulong encrCwUri, uint *sessionId)
```

Semantics:

The following c-code shall be executed:

```
If (ss[slotId].slotMode != slotModeEncr) return ErrSlotMode;
if (0 > nEncr || nEncr >= MaxEncr) return ErrParam4;

/* locate free sessionId; any algorithm is ok */
int i=0;
while (i<NSESSIONS && ss[slotId].se[i].active) i++;
if (i==NSESSIONS) return ErrNoMoreSessions;
/* i contains a non-active session administration block */

/* check if a valid client revocation list was used */
if (config.encryptConfig.microServerVersion >
    ss[slotId].clientPOClRLVnr) return ErrRevocEnforce;

*sessionId = i;

/* initialise session state information */
ss[slotId].se[i].active=true;
ss[slotId].se[i].mh = mh;
ss[slotId].se[i].spk = spk;
ss[slotId].se[i].config = config;
ss[slotId].se[i].encrCwUri = encrCwUri;

int j;
for (j=0; j<nEncr; j++) {
  ss[slotId].se[i].encrSpk[j] = encrSpk[j];
  ss[slotId].se[i].encrPopk[j] = encrPopk[j];
}

/* initialise random key state */
ss[slotId].se[i].rkState.rkCurrent = rnd128();
ss[slotId].se[i].rkState.rkNext    = rnd128();
ss[slotId].se[i].rkState.limitCounter =
            limitValue(config.encryptConfig.rkEncrMode.limit);

/* initialise import state */
ss[slotId].se[i].importSlotId = importSlotId;
ss[slotId].se[i].importSession = importSessionId;

if (!cpsEciRootStateOk(sdlotId,i)){
    ss[slotId].se[i].active = false;
    return ErrRevocEnforce;
}

return i;
```

Preconditions:

- **AS Slot** was successfully initialized.

The **ECI Host** can move forward the random key state (moving next to current) of a session using the following function:

```
int callAsNextKeySession(uint slotId, uint sessionId)
```

Semantics:

The following c-code shall be executed:

```
if (!ss[slotId].se[sessionId].active) return ErrNoSuchSession;

ss[slotId].se[sessionId].rkCurrent = ss[slotId].se[sessionId].rkNext;
ss[slotId].se[sessionId].rkNext = rnd128();
if (ss[slotId].slotMode == SlotModeEncr)
    se[slotId][sessionId].limitCounter =
       limitValue(
          ss[slotId].se[sessionId].config.encryptConfig.rkEncrMode.limit)
else if (ss[slotId].slotMode == SlotModeDecr)
    se[slotId][sessionId].limitCounter =
       limitValue(
          ss[slotId].se[sessionId].config.decryptConfig.rkDecrMode.limit);

return ErrOk;
```

Preconditions:

- **AS Slot** session was successfully initialized.

When operating in TS-mode the content processing system will signal the changeover of current/next control word to the associated **ECI Client** (see [3]). The **ECI Client** can use this message to trigger the computation of the next control word.

The **ECI Host** can stop the session and, as a consequence, terminate any pending **Export Connection**s from that session using the following function:

```
int reqAsStopSession(uint slotId, uint sessionId)
```

Semantics:

The following c-code shall be executed:

```
int i, j;

ss[slotId].se[sessionId].active = false;

/* decouple from any coupled decryption sessions */
for (j=0; j<NSESSESIONS; j++)
    if (ss[slotId].se[j].coupledSessionId == sessionId)
        ss[slotId].se[j].coupledSessionId = -1;
        /* the content processing system is informed of decoupling */

/* cancel all export sessions */
if (ss[slotId].slotMode == SlotModeDecr)
    for (i=0; i<NSLOTS; i++)
        for (j=0; j<NSESSIONS; j++)
            if (ss[i].se[j].importSlot == slotId &&
                ss[i].se[j].importSession == sessionId)
            {
                for (k=0; k<MaxExpGrpId; k++)
                    ss[i].se[j].importPermitted[k]= false;
                ss[i].se[j].importSlotId= -1;
                ss[i].se[j].importSession= -1;
            }

return ErrOk;
```

Preconditions:

- **AS Slot** session was successfully initialized.

6.2.4.4 **AS Slot** Export control

The export **Authentication Mechanism** permits the **ECI Host** to create an **Export Connection** from an **AS Slot** session in a decrypting **ECI Client** to an **AS Slot** session of a **Micro Server**, thus permitting the transfer of content from decrypting **ECI Client** to **Micro Server**. The **AS System** uses the **Certificate Processing Subsystem** to process the required export, import and export authentication chains using the exporting **AS Slot** sessions POPK and minClientVersion as a base for validating the export and subsequent import chain. The end result is that the **Export Connection** element for an export group ID is positively validated, or the connection refused. The actual connection is created from an (export) session to an import session.

```
int reqAsExportConnSetup(uint slotId, uint sessId, uint impSlotId,
        uint impSessId, uint grpIndx, CertSerialChain expCh,
        CertSerialChain impCh, CertSerialChain auth[])
```

Semantics:

expCh is the export chain from POPK to TPEGC or ESC **Certificate**. ImpCh is the import chain from TPEGC the ESC.

NOTE: impCh can be empty. auth[] is the sequence of export authentication chains required to co-authenticate sections of the import chain.

The CertSerialChain structure definition is defined in [3], clause 9.7.1.3.3.

The **AS Slot** first verifies the impCh using the auth[] export authentication chains and using the installed **ECI Root Key** and **Revocation List** version number.

The **AS Slot** then requests the **Certificate Processing Subsystem** to process the export plus import chain using the POPK and the AS State registers **POPK** and **ExportRlVersion** as root. The id of the first certificate in the export chain shall be stored in expGrpId.

On successful authentication, an export element is added to the **AS Slot** session state, containing the export group id and the slot id plus session id of the authenticated export **ECI Client**. The following c-code shall be executed to process to create the import connection. The authentication can be computed for two export group ids so as to permit a seamless changeover from one to the next export group in the content properties.

```
/* the CPS delivers the following variables on successful processing of the
   Export import chains */
PubKey impSpk;           /* the spk of the importing system */
uint   impConfigVersion; /* the config. Version nr of the export system */
uint   expGrpId;         /* the export group for which the export connection     is valid */


/* check if potential import slot is in decent state */
if (!( ss[impSlotId].slotMode == SlotModeEncr &&
       ss[impSlotId].se[impSessId].active &&
       ss[impSlotid].se[impSessId].spk == impSpk
       ss[impSlotId].se[impSessId].encryptConfig.microServerVersion >=
         impConfigVersion
   ) ) return ErrExportSlotBadState;
}

/* check if another import connection already exists */
if (ss[impSlotId].se[impSessId].ies.importSlotId != ImportNone)
  return ErrExportOngoing;
/* Set the import/export state of the import-session to reflect the export connection */
ss[impSlotId].se[impSessId].ies.importSlotId = slotId;
ss[impSlotId].se[impSessId].ies.importSession = sessId;
ss[impSlotId].se[impSessId].ies.expGrpId[grpIndx] = expGrpId;
ss[impSlotId].se[impSessId].ies.importPermitted[grpIndx] = true;
return ErrOk;
```

Preconditions:

- **AS Slot** session was successfully initialized.

After setting up an **Export Connection** it can also be terminated by the importing side (which will effectively halt the encryption session):

```
int reqAsExportConnEnd(uint slotId, uint sessionId)
```

Semantics:

The following c-code will be executed:

```
if (!(ss[slotId] != SlotModeEncr)) return ErrImportSlotBadState;
if (!(ss[slotId].se[sessionId].active)) return ErrParam2;
if (ss[slotId].se[sessionId].ies.slotId == -1) return ErrNoExport;

ss[slotId].se[sessionId].ies.importSlotId = -1;
ss[slotId].se[sessionId].ies.importSession = -1;
for (int i=0; i< MaxExpGrpId; i++)
  ss[slotId].se[sessionId].ies.importPermitted[i] = false;
return ErrOk;
```

Preconditions:

- **AS Slot** session was set up for import.

6.2.4.5 LK1 **Key Ladder** initialization

In order to perform **Key Ladder** mechanism operations in an **AS Slot,** the **ECI Host** can load the top level link key LK1 for subsequent **Key Ladder** output computations.

```
int reqAsLoadLk1(uint slotId, uint sessId, InputV inputV,
             ulong spkUri, uchar spkIndx)
```

Semantics:

The following c-code shall be executed:

```
if (ss[slotId].slotMode == SlotModeEncr) spkIndx = 0;
if (spkIndx >= 16) return ErrParam5;
/* check if spkUri in set_1 */
if ((spkUri>>spkIndx & 0b1) != 0b1) return ErrSpkUriViolation;
if (!ss[slotId].se[sessId].active) return ErrParam2;
if (spkIndx==0 && ss[slotId].slotMode==SlotModeDecr &&
    ss[slotId].se[sessId].config.decryptConfig.spk0NoDecrypt)
  return ErrSpk0NoDecrypt;

ss[slotId].se[sessId].spkUri = spkUri;
ss[slotId].se[sessId].spkIndx = spkIndx;

if (ss[slotId].slotMode == slotModeEncr &&
    ss[slotId].se[sessId].config.encryptConfig.asymKlMode
){
   ss[slotId].lk1= rnd128();
   return ErrOk;
}

ss[slotId].se[sessId].lk1 =
    blockV_blockC_keyladder(inputV,ss[slotId].se[sessId].spk);
return ErrOk;
```

Preconditions:

- **AS Slot** session was initialized.

6.2.4.6 Encryption Control Word calculation

Once **AS Slot** state field lk1 is set control words can be computed. cwIndx indicates the odd or even control word that is computed. The value can be 0 (even) or 1 (odd), and shall always be 0 for file based decryption.

```
int reqAsComputeEncrCw(uint slotId, uint sessId, ulong cwUri, uint nElk,
    SymKey elk[24], uchar XT[32], uint rkIndx, Field2 field2,
    uint cwIndx)
```

Semantics:

The following c-code shall be executed:

```
PubKey spk[MaxSpkEncr+1], popk[MaxSpkEncr+1]; /* temporary variables */
SessionConfig config[MaxSpkEncr+1]; /* temporary variable */

/* basic consistency checks */
if (!ss[slotId].se[sessId].active) return ErrParam2;
if (ss[slotId].slotMode != SlotModeEncr) return ErrSlotMode;
if (ss[slotId].se[sessId].config.encryptConfig.rkEncrMode.mode==0b00) {
    if (nElk<2) return ErrParam4;
} else {
    If (nElk<3) return ErrParam4;
}

/* verify if the slot configuration has been authenticated */
if (ss[slotId].se[sessId].configAuthMode != ConfigAuthModeAk1)
    return ErrNoConfigAuth;

/* verify if the CPS ECI Host Root state is sufficient to proceed */
if (!cpsEciRootStateOk(slotId,sessId)) return ErrRevocEnforce;


/* check if random slot-session key has to be applied */
SymKey rkAppl; /* random key that may have to be applied */
if (rkIndx == 0) {
        rkAppl = ss[slotId].se[sessId].rkState.rkCurrent;
} else if (rkIndx == 1) {
        rkAppl = ss[slotId].se[sessId].rkState.rkNext;
} else {
        return ErrParam7;
}

/* insert random slot key and random session key if required */
if (ss[slotId].se[sessId].config.encryptConfig.rkKlMode) {
        elk[0] = ss[slotId].slotRk;
}
if (ss[slotId].se[sessId].config.encryptConfig.rkEncrMode.mode != RKModeNone) {
        if (nSpk < 3) return ErrNoSlotRkInsert;
        elk[nSpk-1] = rkAppl;
}

/* compute input-C, insert in key ladder */
uchar result1[16], seField1[16];
ushort cpMask;

computeField1Encrypt(elk[nElk-2], result1, cpMask,
ss[slotId].se[sessId].config.encryptConfig);
computeInputC(result1, field2, elk[nElk-2]);

/* use ARK with value 0 */
uchar ark[16] = (uchar){0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0};

/* define spk, popk and config inputs to key ladder; using slot's spk/popk in position 0 and a
replication of the slot configuration */

spk[0]  = ss[slotId].se[sessId].spk;
popk[0] = ss[slotId].popk;
config[0] = ss[slotId]. se[sessId].config;
int i;
int nSpk = slot[slotId]. se[sessId].config.EncryptConfig.nEncr + 1;
for (i=0; i<nSpk-1; i++) {
  spk[i+1]  = ss[SlotId]. se[sessId].encrSpk[i];
  popk[i+1] = ss[SlotId]. se[sessId].decrSpk[i];
  config[i+1] = ss[slotId]. se[sessId].config;
}

/* define spkUri values */
ulong spkUri = (0x1<<(nSpk+1)) - 1; /* all SPKs can be used for decoding
 keys */
```

```
/* perform the key ladder calculation */
bool asym = ss[slotId].se[sessId].config.encryptConfig.asymKlMode;
Secret SymKey cw =
  KeyLadder(ss[slotId].se[sessId].lk1, ss[slotId].se[sessId].encrCwUri,
            AcfCw1Mode, ark, popk, config XT, ss[slotId].spkUri, nSpk, spk,
            nElk,  elk, asym);

/* cw is sent to the encryption resource along with cwUri, msField1, cpMask and cwIndx */

return ErrOk;
```

Preconditions:

- Session's LK1 was loaded.
- **AS Slot** session was authenticated if required.

6.2.4.7 Decryption Control Word calculation

Once **AS Slot** state field lk1 is set control words can be computed. cwIndx indicates the odd or even control word that is computed. The value can be 0 (even) or 1 (odd), and shall always be 0 for file based decryption.

```
int reqAsComputeDecrCw(uint slotId, sessionId, ulong cwUri, uint nSpk,
    uint nElk, SymKey elk[24], PubKey spk[16], PubKey popk[16], SSConflig config[16], uchar
    XT[32], uint rkIndx, Field2 field2, uint cwIndx)
```

Semantics:

The following c-code shall be executed:

```
/* basic consistency checks */
if (!ss[slotId].se[sessionId].active) return ErrParam2;
if (ss[slotId].slotMode != SlotModeDecr) return ErrSlotMode;
if (ss[slotId].se[sessionId].spkIndx >= nSpk) return ErrParam4;
if (ss[slotId].se[sessionId].config.decryptConfig.rkDecrMode.mode==0b00) {
    if (nElk<2) return ErrParam5;
} else {
    if (nElk<3) return ErrParam5;
}
uint si = ss[slotId].se[sessionId].spkIndx ;

/* verify if the slot configuration has been authenticated if so required */
if ( ss[slotId].se[sessionId].config.decryptConfig.akModeAuth &&
     ss[slotId].se[sessionId].configAuthMode != ConfigAuthModeAk1
   ) return ErrNoConfigAuth;

/* verify if the CPS ECI Host Root state is sufficient to proceed */
if (!cpsEciRootStateOk(slotId,sessionId)) return ErrRevocEnforce ;

/* ensure proper slot spk, popk and slotConfig are applied */
spk[si]= ss[slotId].se[sessionId].spk;
popk[si] = ss[slotId].se[sessionId].popk;

/* only authenticate the slot's decrypt configuration if required */
if ( ss[slotId].se[sessionId].config.decryptConfig.klModeAuth )
    ssConfig[si].decryptConfig = ss[slotId].ssConfig.decryptConfig;

/* in all cases authenticate the klModeAuth and akModeAuth fields */
config[si].decryptConfig.klModeAuth=
     ss[slotId].se[sessionId].config.decryptConfig.klModeAuth;
config[si].decryptConfig.akModeAuth =
     ss[slotId].se[sessionId].config.decryptConfig.akModeAuth;

/* check if random slot-session key may have to be applied */
SymKey rpAppl; /* random key that may have to be applied */
if (rkIndx == 0) {
        rpAppl = ss[slotId].se[sessionId].rkState.rkCurrent;
} else if (rkIndx == 1) {
        rpAppl = ss[slotId].se[sessionId].rkState.rkNext;
} else {
        return ErrParam11;
}

/* insert random slot key and random session key if required */
if (ss[slotId].se[sessionId].config.decryptConfig.rkKlMode) {
```

```
        elk[0] = ss[slotId].slotRk;
}
if (ss[slotId].se[sessionId].config.decryptConfig.rkDecrMode.mode != RKModeNone) {
        if (nSpk < 2) return ErrNoSlotRkInsert;
        elk[nSpk-2] = rpAppl;
}

/* compute input-C, i.e. elk[nElk-1] for content Property authentication */
/* veryfy basicUri control bit is set */
if (((elk[nElk-1][0]>>2)&0b1) != 0b1) return ErrBasicUriCtrl;
uchar result1[16];
computeField1Decrypt(elk[nElk-2],result1);
computeInputC(result1, field2, elk[nElk-2]);

/* use ARK with value 0 */
uchar ark[16] = (uchar){0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0};

/* perform the key ladder calculation */
Secret SymKey cw =
  KeyLadder(ss[slotId].se[sessionId].lk1, cwUri, AcfCw1Mode, ark,
            popk, ssConfig, XT, ss[slotId].se[sessionId].spkUri, nSpk,
            spk, nElk, elk, false);

/* cw is passed to the decryption resource session along with cwUri, result1 and cwIndx and the
sessions states media handle value */

return ErrOk;
```

Preconditions:

- Session's LK1 was loaded.
- **AS Slot** session was authenticated if required.

6.2.4.8 Computing akClient and its application

The **Key Ladder Block**'s **Authentication Mechanism** permits the secure calculation of secure keys for use by the **ECI Client** using the **Authentication Mechanism**:

```
int reqAsComputeAkClient(uint slotId, InputV inputV, uint nSpk,
    uchar spkIndx, PubKey spk[16], PubKey popk[16], SessionConfig akCnf[16],
    ulong spkUri,uchar XT[32], bool online)
```

Semantics:

The following c-code shall be executed:

```
/* basic consistency checks */
if (ss[slotId].slotMode==SlotModeEncr) spkIndx = 0;
if (spkIndx >= 16) return ErrParam4;
/* check if spkUri in set_1 */
if ((spkUri>>spkIndx & 0b1) != 0b1) return ErrSpkUriViolation;
if (ss[slotId].slotMode == SlotModeEncr) {
  if (akCnf[spkIndx].encryptConfig.configVersion != 0x1) return ErrParam7;
  if (akCnf.encryptConfig.microServerVersion >
      ss[slotId].clientPOClRLVnr) return ErrRevocEnforce;
  if ((cpsEciRootState.rootVersion <
      akCnf[spkIndx].encryptConfig.minEciRootState.rootVersion)
   || (cpsEciRootState.rlVersion <
      akCnf[spkIndx].encryptConfig.minEciRootState.rlVersion))
    return ErrRevocEnforce;
}
if (ss[slotId].slotMode == SlotModeDecr) {
  if (akCnf[spkIndx].decryptConfig.configVersion != 0x1) return ErrParam7;
  if (akCnf.decryptConfig.minClientVersion >
      ss[slotId].clientPOClRLVnr) return ErrRevocEnforce;
  if ((cpsEciRootState.rootVersion <
      akCnf[spkIndx].decryptConfig.minEciRootState.rootVersion)
   || (cpsEciRootState.rlVersion <
      akCnf[spkIndx].decryptConfig.minEciRootState.rlVersion))
    return ErrRevocEnforce;
}
/* ensure proper slot spk and popk are applied */
popk[spkIndx] = ss[slotId].popk;
```

```
/* ensure proper ACF and ARK are applied */
uchar ark[16] ;
uchar acf[15] = acfAk1Mode ;
acf[1] = AkUseCl;
if (online) {
acf[1] += AkOnline;
ark = ss[slotId].slotRk;
} else {
    acf[1] += AlOffline;
    ark = {0} ;
}

/* perform the authentication mechanism */
ss[slotId].akClient =
      AuthMech(inputV,acf,ark,popk,akCnf,XT,spkUri,nSpk,spkIndx,spk);
return ErrOk;
```

Preconditions:

- The slot has been be initialized.

In order to use the computed **ECI Client** AK key the following function is defined:

```
int reqAsClientChalResp(int slotId, uchar challenge[16],
    uchar *(response[16]));
```

Semantics:

The following c-code shall be executed:

```
*response = AuthMechResponse(ss[slotId].akClient, challenge);
return ErrOk;
```

Preconditions:

- The slot was initialized.
- The slot's AkClient has been successfully computed.

6.2.4.9 **AS Slot** Session Configuration Authentication

The **Key Ladder Block**'s **Authentication Mechanism** permits the authentication of the slot's session configuration by the **Provisioning Server**. The **Provisioning Server** can issue offline authentication information or require online authentication to take place by setting AkOnline in ACF. Two separate functions are available for authentication of a decryption and an encryption slot.

```
int reqAsAuthDecrConfig(uint slotId, uint sessId, InputV inputV,
  uint nSpk, uchar spkIndx, PubKey spk[16], PubKey popk[16], SSCnfg clCnf[16],
  ulong spkUri, uchar XT[32], bool online, uchar verifier[16])
```

Semantics:

The following c-code shall be executed:

```
/* basic consistency checks */
if (!ss[slotId].se[sessionId].active) return ErrParam2;
if (ss[slotId].slotMode!=SlotModeDecr) return ErrSlotMode;
if (spkIndx >= 16) return ErrParam5;
/* check if spkUri in set_1 */
if ((spkUri>>spkIndx & 0b1) != 0b1) return ErrSpkUriViolation;
if (spkIndx==0 && ss[slotId].slotMode==SlotModeDecr &&
    ss[slotId].se[sessId].config.decryptConfig.spk0NoDecrypt) return ErrSpk0NoDecrypt;

/* verify if the CPS ECI Host Root state is sufficient to proceed */
if (!cpsEciRootStateOk(slotId)) return ErrRevocEnforce;

/* ensure proper slot spk, popk and config are applied */
popk[spkIndx] = ss[slotId].popk;
spk[spkIndx] = ss[slotId].se[sessId].spk;
clCnf[spkIndx] = ss[slotId].se[sessId].config;

uchar ark[16];
uchar acf[15] = acfAk1Mode;
```

```
acf[1] = AkUseAS + AkConfigAuth;
if (online) {
        acf[1] = AkOnline;
        ark = ss[slotId].slotRk;
} else {
        acf[1] = AkOffline;
        ark = {0};
}

/* perform the authentication mechanism */
Secret SymKey ak =
      AuthMech(inputV,acf,ark,popk,clCnf,XT,spkUri,nSpk,spkIndx,spk);

uchar response[16] = AuthMechResponse(ak, verifier);

if (response == {0}) {
        ss[slotId].se[sessId].configAuthMode = ConfigAuthModeAk1;
        return ErrOk;
} else {
        ss[slotId].se[sessId].configAuthMode = ConfigAuthModeNone;
        return ErrSlotConfigAuthFail;
}
```

Preconditions:

- **AS Slot** session's LK1 was loaded.

The authentication for encryption includes verifying the encryption specific state.

```
int reqAsAuthEncrConfig(uint slotId, uint sessId, InputV inputV,
       uchar XT[32], bool online, uchar verifier[16])
```

Semantics:

The following c-code shall be executed:

```
PubKey spk[MaxSpkEncr+1], popk[MaxSpkEncr+1]; /* temporary variables */
SessionConfig config[MaxSpkEncr+1]; /* temporary variable */

/* basic consistency checks */
if ((ss[slotId.SlotMode != SlotModeEncr) return ErrSlotMode;

/* verify if the CPS ECI Host Root state is sufficient to proceed */
if (!cpsEciRootStateOk(slotId,sessId)) return ErrRevocEnforce;

/* define spk, popk and config inputs to key ladder; using slot's spk/popk in position 0 and a
replication of the slot configuration */

spk[0]  = ss[slotId].se[sessId].spk;
popk[0] = ss[slotId].popk;
config[0] = ss[slotId].se[sessId].config;
int i;
int nSpk = slot[slotId]. config.EncryptConfig.nEncr + 1;
for (i=0; i<nSpk-1; i++) {
  spk[i+1] = ss[SlotId].encrSpk[i];
  popk[i+1] = ss[SlotId].encrPopk [i];
  config[i+1] = ss[slotId].se[sessId].slotConfig;
}

/* define spkUri values */
ulong spkUri = (0x1<<(nSpk)) - 1;
                      /* all SPKs can be used for decoding content */

uchar ark[16];
uchar acf[15] = acfAk1Mode;
acf[1] = AkUseAS + AkConfigAuth;
if (online) {
        acf[1] = AkOnline;
        ark = ss[slotId].slotRk;
} else {
        acf[1] = AkOffline;
        ark = {0};
}
```

```
/* perform the authentication mechanism */
Secret SymKey ak =
      AuthMech(inputV,acf,ark,popk,clCnf,XT,spkUri,nSpk,spkIndx,spk);

uchar response[16] = AuthMechResponse(ak, verifier);

if (response == {0}) {
        ss[slotId].se[sessId].configAuthMode = ConfigAuthModeAk1;
        return ErrOk;
} else {
        ss[slotId].se[sessId].configAuthMode = ConfigAuthModeNone;
        return ErrSlotConfigAuthFail;
}
```

Preconditions:

- **AS Slot** session's LK1 was loaded.

6.2.4.10 Loading a **Micro Server** secret key

A **Micro Server** client can use the **AS Slot** operating in asymmetrical server mode and load a **Micro Server** Secret Key value ussk for subsequent establishment of a secure connection to a **Micro Client** chipset:

```
int reqAsLdUssk(uint slotId, uint sessId, InputV inputV,
                uchar XT[32], bool online, uchar mUssk[NUSSK])
```

Semantics:

The following c-code shall be executed:

```
PubKey spk[MaxSpkEncr], popk[MaxSpkEncr];
SessionConfig config[MaxSpkEncr];

/* basic consistency checks */
if (ss[slotId].slotMode!=SlotModeEncr) return ErrSlotMode;
if (!ss[slotId].se[sessId].config.encryptConfig.asymKlMode)
        return ErrSlotModeUndefined;

/* verify if the CPS ECI Host Root state is sufficient to proceed */
if (!cpsEciRootStateOk(slotId,sessId)) return ErrRevocEnforce;

spk[0]  = ss[slotId].se[sessId].spk;
popk[0] = ss[slotId].popk;
config[0] = ss[slotId]. se[sessId].config;
int i;
int nSpk = slot[slotId]. se[sessId].config.EncryptConfig.nEncr + 1;
for (i=0; i<nSpk-1; i++) {
  spk[i+1]  = ss[SlotId]. se[sessId].encrSpk[i];
  popk[i+1] = ss[SlotId]. se[sessId].decrSpk[i];
  config[i+1] = ss[slotId]. se[sessId].config;
}
/* define spkUri values */
ulong spkUri = (0x1<<(nSpk+1)) - 1; /* all SPKs can be used for decoding
 keys */

uchar ark[16];
uchar acf[15] = acfAk1Mode;
acf[1] = AkUseAS + AkLdUssk;
if (online) {
        acf[1] = AkOnline;
        ark = ss[slotId].slotRk;
} else {
        acf[1] = AkOffline;
        ark = {0};
}

/* perform the authentication mechanism */
Secret SymKey ak =
      AuthMech(inputV,acf,ark,popk,config,XT,spkUri,nSpk,0,spk);

/* perform AES ECB decoding of ussk */
int i,j;
uchar response[32];
for (i=0; i<NUSSK; i+=32){
response = AuthMechResponse(ak, &(mUssk[i]));
for (j=0; j<32; j++) ss[slotId].se[sessId].ussk[i+j] = response[j];
```

```
}
return ErrOk;
```

Preconditions:

- The session configuration was authenticated.

6.2.4.11 Generating MinitLk1 for **Micro Clients**

In asymmetrical **Micro Server** mode the **AS Slot** can generate **Key Ladder Block** initialization messages for **Micro Clients**:

```
InputV reqAsMInikLk1(uint slotId, uint sessId, ECI_Certificate_Chain ClCPK)
```

Semantics:

ECI_Certificate_Chain is defined in [3], clause 5.4.1 and contains the **Certificate Chain** for validating a **Micro Client**. This function first uses the CPS to validate ClCPK using slot[slotId] with POPK as father certificate and using ss[SlotId].se[sessId]. config.encryptConfig.microServerVersion as the minimum **Revocation List** version number for the first certificate in the chain. If this validation is successful the variable clcpk contains the client chipset public key, and the following c-code shall be executed:

```
return asymInitLk1(ss[slotId].lk1, slot[slotId].ussk, clcpk);
```

Preconditions:

- Ussk is initialized.
- Session is in asymmetrical encryption mode.

6.2.4.12 Computing **ECI Client** image decryption key

In order to perform loading of an encrypted image the **AS Slot** can provide an authentication key with which the key to decode the image can be decrypted. This function has to be executed before slot initialization:

```
int reqAsComputeImageKey(uint slotId, InputV inputV,
        symKey eKey , bool online, ECIRootState min_root_state)
```

Semantics:

The following c-code shall be executed:

```
/* a default slot configuration state is used */
SessionConfig config = {
  .decryptConfig = {
    .configVersion = 0x1,
    .reserved1 = 0x0,
    .klModeAuth = 0x0,
    .akModeAuth = 0x0,
    .rkKlMode = 0x0,
    .spk0NoDecrypt = false,
    .reserved2 = 0b000000,
    .rkDecrMode = { 0 },
    .minEciRootState = min_root_state,
    .expRlVersion = 0x0
  },
  .encryptConfig = { 0 }
} ;

if (!(cpsEciRootState.rootVersion >= min_root_state.rootVersion &&
     (cpsEciRootState.rlVersion >= min_root_state.rlVersion))
    return ErrRevocEnforce;

/* create straightforward popk/spk, XT, clCnf, */
PubKey popkArr[1]; /* also used for spk */
popkArr[0] = ss[slotId].popk;
SessionConfig cnf[1];
cnf[0] = config;
uchar XT[32] = {0};
ulong spkUri= 0x1;
```

```
uchar ark[16];
uchar acf[15] = acfAk1Mode;
acf[1] = AkUseAS + AkClImg;
if (online) {
        acf[1] = AkOnline;
        ark = ss[slotId].slotRk;
} else {
        acf[1] = AkOffline;
        ark = {0} ;
}

/* perform the authentication mechanism */
Secret SymKey ak =
      AuthMech(inputV,acf,ark,popkArr,cnf,XT,spkUri,1,0,popkArr);

Secret SymKey dImgKey = AuthMechResponse(ak, eImgKey);
/* dImgKey is subsequently used by the client loader to decrypt the client image using AES CBC mode
with IV=0 */

return ErrOk;
```

Preconditions:

- The slot is set to default state; slotRk is set to new random value.

NOTE: This function is not executed on request of the **ECI Client.**

6.2.4.13 Reading Advanced Security Information

The **AS System** provides the **ECI Client** access to data it generates and provides a general purpose random key function for the **ECI Client**.

NOTE 1: "get" and "set" functions defined in this clause do not generate automatic errors on undefined parameter values, but in case of get functions simply return an undefined value and in case of set functions do not have any effect.

The following function reads the **AS Slot**'s random key (typically used as a nonce for sessions):

```
SymKey getAsSlotRk(uint slotId)
```

Semantics:

The following c-code is executed:

```
return ss[slotId].slotRk;
```

In case the slot is not initialized a number is returned.

The following function reads the random key state of the session:

```
SymKey getAsSessionRk(uint slotId, uint sessionId, uint rkIndx)
```

Semantics:

The following c-code is executed:

```
if (rkIndx == 0)
  return se[slotId][sessionId].rkState.rkCurrent;
else
  return se[slotId][sessionId].rkState.rkNext;
```

In case the slot is not initialized a number is returned.

The limit counter of the session's random key can be read:

```
ulong getAsSessionLimitCounter (uint slotId, uint sessionId)
```

Semantics:

The following c-code is executed:

```
return se[slotId][sessionId].rkState.limitCounter;
```

In case the slot is not initialized a number is returned.

A limit counter value can be set on which an event is raised (e.g. to renew the random key sufficiently in time):

```
ulong setAsSessionLimitEvent(uint slotId, uint sessionId, ulong eventLimit)
```

Semantics:

An eventSessionLimitCounter event is raised once when the following condition is true after calling this function:

```
se[slotId][sessionId].rkState.limitCounter <= eventLimit;
```

NOTE 2: A second call overrides a previous call. Calling this function the second time with a very large value for eventLimit effectively cancels the event (except when the event was already raised).

The following event is raised on reaching an event limit for a session:

```
reqAsEventSessionLimit(uint slotId, uint sessionId)
```

NOTE 3: This event translated into an asynchronous message without corresponding response in ETSI GS ECI 001-3 [3].

6.2.4.14 Generating Client Random Numbers

The **ECI Client** can request a 128bit random number generated by the **AS System** by calling the following function:

```
SymKey getAsClientRnd()
```

Semantics:

The following c-code is executed:

```
return rnd128();
```

6.2.4.15 Error codes

The error codes values returned by the function defined in clause 6.2.4 are defined in Table 6.2-14.

These error codes follow the convention of error codes of messages between an ECI Host and an ECI Client as defined in ETSI GS ECI 001-3 [3], clause 9.

Table 6.2-14: Error return code definition

Error return code	Value	Description
ErrSlotMode	-256	**AS Slot** is not in proper mode for this operation
ErrNoMoreSessions	-257	No more sessions available
ErrSession1Coupled	-258	First session is already coupled
ErrSession2Coupled	-259	Second session is already coupled
ErrSessionNotCoupled	-260	Session is not coupled
ErrNoSuchSession	-261	Session does not exist
ErrExportNoSlot	-262	Export slot unknown
ErrExportSlotBadState	-263	Export slot in improper state
ErrExportOngoing	-264	Export session already has an **Export Connection**
ErrImportSlotBadState	-265	Importing slot not in encryption mode.
ErrNoExport	-266	No export ongoing to session
ErrSpkUriViolation	-267	SpkUri for slot SPK has improper value for slot mode.
ErrSlotModeUndefined	-268	Slot Mode has improper value for this operation
ErrRevocEnforce	-269	**ECI Revocation** does not permit slot to operate
ErrNoConfigAuth	-270	Slot configuration has not been properly authenticated
ErrNoSlotRkInsert	-271	ELK vector not sufficiently long to insert random key
ErrSpk0NoDecrypt	-272	spk[0] cannot be used to generate decryption control words
ErrBasicUriCtrl	-273	Basic URI field1 control bit not set
ErrOk	0	Successful call
ErrSlotConfigAuthFail	-274	Authentication of the slot's session configuration failed
ErrParam<N>	**-<N>**	Error in input parameter **N** (ErrParam1 has value -1 and signals an error in parameter 1)
	1..MaxInt	Successful call, value defined by message definitions

7 Scrambling/descrambling and Content Export

7.1 Basic Functionality

The content processing system can decrypt content. This content is accompanied by **Content Properties** and **Export Connection**s. Content can be forwarded to standard outputs if **Content Properties** permit this, and can be re-encrypted by a **Micro Server** in case of a matching **Export Connection**.

For the purpose of resource management **ECI** defines decryption and encryption resources. A resource is used to decrypt or encrypt content from a single media session encrypted or to be encrypted with a single CW at a time, and a decryption or encryption resource is connected to a single decryption or encryption **AS** Slot. For TS stream decryption the decryption resource has a dual buffer for an odd and an even CW. The odd or even CW is selected by the stream to be decoded. This can accommodate the need to change the control word on the fly in case the **Content Properties** of the content to be encrypted change. For file-based decryption and encryption the **ECI Host** provides the synchronization between CW and the content to be decrypted, which can be substantially faster than real-time. File-based decryption and re-encryption resources require only a single CW buffer.

NOTE: TS streams requiring two or more control words for descrambling of different elementary streams require multiple descrambling resources and thus multiple sessions.

ECI does not specify any specifics regarding buffering or (possibly extensive) intermediate processing like transcoding or watermarking that may be performed on the decrypted content passing from decryption to encryption resource. Such processes may cause significant delays. CPE manufacturers may select appropriate implementations causing a time-offset between decryption resource and a connected re-encryption resource. The re-encryption slot and **ECI Client** synchronize with the encryption of content.

7.2 Scrambler and Descrambler specifications

The descrambling function of an **ECI** CPE shall support the following descrambling algorithms in TS mode:

- CSA1/2, both PES and TS mode as defined in ETSI ETR 289 [9] and [i.1].

- CSA3, both PES and TS-mode ETSI TS 100 289 [10] and [i.2].
- DVB-CISSA PES and TS-mode ETSI TS 103 127 [11].

The descrambling function of an **ECI** CPE shall support the following descrambling algorithms in File mode:

- CENC AES128 CTR mode and AES128-CBC mode (both full sample and subsample encryption) as defined in ISO/IEC 23001-7 [12]. CENC and ISO/IEC 23009-4 [13] for MPEG-DASH.

The scrambling function of an **ECI** CPE shall support the following scrambling algorithms in TS mode:

- DVB-CISSA PES and TS-mode [11].

The scrambling function of an **ECI** CPE shall support the following scrambling algorithms in File mode:

- CENC AES128 CTR and CBC mode (both full sample and subsample encryption) as defined in [12] and [13]. The content processing system shall generate a unique initialization vector for content encrypted with a single CW for AES-CTR mode and follow rules for IV definition for AES-CBC mode as defined in [13]. Initialization vectors are accessible to the **ECI Host** for use to package content.

7.3 Export Control

Authenticated **Export Connections** are used by the decryption **AS Slot** sessions as tickets to authorize import and export by the decryption resource. A decryption resource shall permit export of decrypted content to an encryption resource if the **Export Connection** provided by the associated **AS Slot** session permits this for an export group ID and the **Content Properties** of the decrypted content signal the corresponding export group ID as defined in clause 6.2.4.4. A decryption resource shall not permit export of decrypted content to an encryption resource if the **Export Connection** of the export group selected by the export group ID's in the **Content Properties** is not a validated **Export Connection** provided by the associated **AS Slot**.

7.4 Output Control

Output control **Content Properties** are used to disable or to enable content export under protection of industry standard protection technologies on output connections of the CPE. A decryption resource shall permit export of decrypted content to an output if the output control information from the associated **AS Slot** session permits this. A decryption resource shall not permit export of decrypted content to an output if there is no permission in the output control information from the associated **AS Slot** session.

7.5 Content Property Comparison on Coupled Sessions

The content processing system shall verify that the **Content Properties** as defined in field1 of a session excluding the first two bytes are equal to the **Content Properties** of any coupled session. Export and Output of content of a coupled session with equal **Content Properties** shall be permitted. The combined streams shall be treated as one session from an **ECI** protection perspective from there on. Combining of a coupled stream shall be inhibited if the **Content Properties** of field1 excluding the first two bytes are not equal.

7.6 Content Property Propagation on Export

The decryption resource session shall propagate the field1 **Content Properties** set by the client and (partially) implicitly authenticated by the **Key Ladder** along with the content to the re-encryption session resources importing the decrypted content as defined in clause 6.2.4.6. The encryption sessions receiving the decrypted content check the designated field1 bytes against the value set for field1 for encrypting the content while applying a mask to select the fields that require propagation as defined by the function, thus ensuring that the designated decrypting client field1 bytes are propagated to the encrypted content.

The following c-code shall be executed by the encryption resource session on every change of the input values impField1, expField1 and cpMask:

```
uchar impField1[16];  /* field1 values for the imported content */
uchar expField1[16];  /* field1 values from the encryption CW computation */
```

```
ushort cpMask;        /* comparison mask */

bool propOk = true;  /* indicates if propagation of imported content is Ok */
int i;

for (i=2; i<16; i++)
    propOk &&= !(cpMask>>I & 0b1) || (impField1[i] == expField1[i]);

if (propOk) /* re-encrypt content */
else /* do not re-encrypt content */
```

7.7 Basic URI Enforcement on Export

The basic URI propagation from the decryption **AS Slot** session to the re-encryption **AS Slot** session is controlled through the following mechanisms, with slotId the ID of the encrypting slot and sessionId the ID of the encryption session therein:

1) The rights assigned to the content for the basic URI by the encryption **AS Slot** are not more liberal than those associated with the propagated content.

2) The **Micro Server** is authenticated: ss[slotId].se[sessionId].config.decryptConfig,akModeAuth is equal to 0b1.

3) If the basic URI does not permit replay of content (i.e. streaming mode) the following is checked on export:

   - ss[slotId].se[sessionId].config.decryptConfig.rkDecrMode.mode shall be unequal to RKModeNone (i.e. a random nonce is applied preventing replay of previously encoded content on a system restart); and

   - ss[slotId].se[sessionId].klModeAuth shall be set (value 0b1) ensuring the decryptConfig used by the server, including the random key insertion at the **Micro Client**, is authenticated and used by the **Micro Client** based on the **Key Ladder** computation.

7.8 Content Property Application on Industry Standard Outputs

A standard output, which typically is a physical output in combination with an industry standard protection system, shall use the **Content Properties** to select the appropriate output protection setting or to disable the output if no appropriate setting is possible. The precise rules thereto are defined in compliance rules.

The **Robustness** of the basic URI and Output Control **Content Properties** implementation shall be of a similar level as the Content Processing system.

The **Robustness** of the standard URI enforcement shall be at least as high as that of the **ECI Host** implementation, with due exception for functions with complex implementation requirements.

7.9 Control Word Synchronization

For processing TS streams the content processing system provides the following functions permitting control over control word changes (for encryption) and provides notifications on scrambling control field changes. The functions and events in this session adhere to the conventions defined in clause 6.2.4.

The **AS Slot** session can provide both an "odd" and an "even" control word to be applied for encryption or decryption of content.

In case of decryption the scrambling control field ETSI TS 100 289 [10] informs the decryption function which control word to use. No control word is used in case the content is signalled as unscrambled. The value of the result is equal to the scrambling control field, values as defined in [10], clause 5.1.

The following function reads the current status of the scrambling control field in the stream:

```
uint getAsSC(uint slotId, uint sessionId)
```

In case of encryption the applied control word can change on the basis of two events:

1) A change in the **Content Properties** of the imported content, which shall trigger a change in the control word applied for encryption. This change can be delayed by the **AS Slot** in order to complete an ongoing change in control word triggered by the following event.

2) A signal from the **AS Slot** session that the applied control word has to change.

In case the imported content is not scrambled, no scrambling shall be applied for encryption and the content scrambling control field shall be set to 0b00 at the first possible change location. Vice versa, in case the imported content is switching from not scrambled to scrambled, the content shall be scrambled with the next control word; the opposite key will be selected as compared to the content that was scrambled before the clear section of content.

The event signalling the imported content property change is defined as:

```
reqAsEventCpChange(uint slotId, uint sessionId)
```

Semantics:

The event signals a change in the **Content Properties** of the imported content if such content requires encryption.

The content processing system shall not permit a discrepancy between the encryption parameters and imported **Content Properties** for a longer period than the one specified in clause 4.5 of [i.3].

NOTE 1: On a change from encrypted to non-encrypted imported content this event will not be raised. **Content Properties** do not apply to non-encrypted content.

The content processing system permits the AS system to hold off any automatic change to an eventCpChange of the **Content Properties** on the following command:

```
setAsPermitCPChange(uint slotId, uint sessionId, bool permit)
```

Semantics:

This function sets the permission to allow an automatic change in control properties of the imported content to trigger a change in the control word on the encrypted content.

NOTE 2: This function should precede any next control word not computed on the basis of the forthcoming **Content Properties**, e.g. only reflecting a nonce or a random key change.

NOTE 3: If the change permission is disabled (`permit==false`) it should be restored within the permitted time for a discrepancy with the **Content Properties** of the imported content so as not to create a "blackout" in the re-encrypted stream.

The following function allows setting the scrambling control field of encryption to a certain status.

```
setAsSC(uint slotId, uint sessionId, uint scramblingControlField)
```

Semantics:

The value of the scrambling control field is set to the value of scramblingControlField on the first possible point of change in the stream. Only values 0b10 and 0b11 (scrambling with even and odd key respectively) are permitted for scramblingControlField.

The scrambling control field of the encrypted stream will be set to 0b00 (no scrambling) in case the imported content has a non-encrypted status.

The following event function is defined for decryption and encryption sessions:

```
reqAsEventSC(uint slotId, uint sessionId, uint scramblingControlField)
```

Semantics:

The event is raised on a change of the scrambling control field status.

8 Certificate Processing Subsystem

8.1 Basic processing rules for **Certificate Chains**

The **Certificate Processing Subsystem** can process certificate chains to authenticate items, based on an initial public key and a minimum **Revocation List** number. Most certificate chain processing is generic. This clause defines the generic processing rules for **Certificate Chains**. The following clauses define processing rules specific for various types of chains.

The **Certificate Chains** below are defined in [3], clause 5.4.

The CPS rule definition uses a stepwise approach for processing **Certificate Chains** starting at the start of the chain (the first **Revocation List**) using initial public key and the minimum **Revocation List** number. The first step is verification of the **Revocation List**. The second step verifies the next certificate in the chain. After once performing step 1 and 2, a new public key and **Revocation List** number are defined for processing the remainder of the chain. Step 1 and 2 are repeated until the whole chain is processed. In general, it is recommended that software functions offering chains pre-validate these chains so as to avoid that the CPS fails on processing a chain unexpectedly.

The generic processing steps for a **Certificate Chain** are:

1) The CPS shall perform the following verification on a **Revocation List**:

   a) The CPS shall check the **Revocation List format_version** field to match a version that it can interpret (see specific processing rules for chains) 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 **ECI Host** 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 last validated public key.

   d) The CPS shall verify whether the length of the **Revocation List** corresponds to its field values and that any variable length field has the appropriate length.

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

2) The CPS shall perform the following verification on a **Certificate**:

   a) The CPS shall verify if the ***next*** <type, entity_ id, version> of the **Certificate** in the chain is not revoked according to the last **Revocation List** and establish the minimum **Revocation List** version to accompany that **Certificate** according to the **base_rl_version** and **min_rl_version** fields of the last **Revocation List**.

   b) The CPS shall check the **Revocation List format_version** field to match a version that it is permitted to interpret.

   c) The CPS shall verify whether the length of the **Certificate** corresponds to its field values and that any variable length field has the appropriate length.

   d) The CPS shall verify the signature of the **Certificate** with the public key.

After processing step 1 and step 2 the public key and minimum **Revocation List** are updated. The public key will be equal to the public key field of the **Certificate** processed in step 2, the minimum **Revocation List** version that found in step 2a.

Not all **Certificates** require being accompanied by a revocation list. If the most significant bit of the type field of a certificate-id equals zero the **Certificate Processing Subsystem** shall require revocation list to accompany a **Certificate** for further chain processing. Any processing of revocation list and version number and revocation list version numbers in the above steps shall not apply in case no revocation list is required.

8.2 Specific rules for Host Image Chains

The CPS shall apply to the specific validation for Host Image chains:

1) First **Revocation List** is of type 0x1 (Manufacturer Revocation List).
2) First **Certificate** is of type 0x1 (Manufacturer Certificate).
3) Second **Revocation List** is of type 0x0 (Host Revocation List).
4) Second **Certificate** is of type 0x0 (Host Certificate).
5) A possible third **Certificate** is of type 0x98 (Host Image Series Certificate).

The public key of the last **Certificate** (either Host Certificate or Host Image Series Certificate) shall be used to validate the actual **ECI Host** image.

8.3 Specific rules for Client Image Chains

The CPS shall apply to the specific validation for Client Image chains:

1) First **Revocation List** is of type 0x2 (Vendor Revocation List).
2) First **Certificate** is of type 0x2 (Vendor Certificate).
3) Second **Revocation List** is of type 0x0 (Client Revocation List).
4) A possible second **Certificate** is of type 0x1 (Client Series Certificate).

The public key of the last **Certificate** (either Vendor Certificate or Client Series Certificate) shall be used to validate the actual **ECI Host** image, taking into account the last version number of the Client **Revocation List** for verifying the image version in case the last **Certificate** is the Vendor Certificate.

8.4 Specific rules for Platform Operation Certificates

The CPS shall apply to the specific validation for Platform Operation **Certificate Chains**:

1) First **Revocation List** is of type 0x3 (Operator Revocation List).
2) First **Certificate** is of type 0x3 (Operator Certificate).
3) Second **Revocation List** is of type 0x0 (Platform Operation Revocation List).
4) Second **Certificate** is of type 0x0 (Platform Operation Certificate).

8.5 Specific rules for Export/Import chains

8.5.1 Export Authorization chain processing

The export authentication chain and the corresponding section of the third party export chain shall be provided to the CPS.

The CPS shall commence with the minimum root version and **Revocation List** version as defined in ss[slotId].se[sessionId].config.decryptConfig.minEciRootState. It shall process the chain of EAOC and EAC **Certificates** and associated **Revocation Lists** verifying the following specific rules for this chain:

- The id of the root RL is 0x4 (Export Authorization Operator Revocation List).
- The id of the next **Certificate** (EAOC) is 0x4.
- The id of the next **Revocation List** (REAOC RL) is 0x0.

- The id of the following **Certificates** (EAC) in the chain is 0x0.
- The content of the extension field of the **Certificate** shall be equal to the corresponding export chain **Certificate** in the export chain.
- The id of the following **Revocation List** (EAC-RL) in the chain is 0x0.
- All **Certificates** of the export chain shall be sequentially validated by the Export Authorization chain.
- The first **Certificate** of a third party export chain section shall be a TPEGC (certificate id equal 0x5).
- The last **Certificate** of a third party export chain section shall be a TPEGC, ESC or ERC (certificate id equal 0x5, 0xE, 0xF respectively).
- The intermediate **Certificates** shall all be EGC (certificate id equal 0x4).
- If the last **Certificate** is a TPEGC, this shall be the start of a next export chain section. The above verification process shall be repeated for all subsequent sections of export authentication chains and third party export chain sections until a fully validated third party export chain results (ending in a ESC or an ERC).

8.5.2 Export Chain verification

The CPS shall commence with the POC public key, the export group index for which the export is to be established and the minimum **Revocation List** version number that should apply to the POC **Revocation List** as found in the **AS Slot** state ss[slotId].se[sessionId]. config.decryptConfig.minClientVersion field.

NOTE: Such validation relies on suitable authentication of the POPK and **Revocation List** version. This should be established using either AK mode authentication or implicit authentication using the **Key Ladder** (see clause 6.2.2.2, klModeAuth and akModeAuth fields).

The CPS shall process the POC-RL, EGC and EGC-RL and subsequent TPEGC or ESC as a regular **Certificate Chain**. The following additional rules shall be verified:

- The type of the EGC is 0x4.
- The export_group_id field of the EGC shall be equal to the export group index.
- The type of the EGC **Revocation List** is 0x4.
- The type of the EGC-RL is 0x4.
- The type of the TPEGC or ESC corresponds to the value in [3], table 5.2-2.

The processing of a TPEGC is specified in clause 8.5.3.
The processing of an ESC is specified in clause 8.5.4.

8.5.3 Third Party Export Chain verification

Processing of a third party export chain commences with the validation of the leading TPEGC and the minimum **Revocation List** version number for its **Revocation List**. Processing shall end with an ESC.

8.5.4 Export System Certificate processing

The ESC **Certificate** SPK (public key of the ESC) and minimum **Revocation List** version number of the father of the ESC are used to validate the **Export Connection**. The **Certificate** SPK shall match to the ss[slotId].spk field of the designated export slot. The minimum **Revocation List** version number shall be larger than the export ss[slotId].ssConfig.microServerVersion.

NOTE: The export slot SPK and microServerVersion have to be authenticated by the **AS Slot**'s AK **Authentication Mechanism** to ensure meaningful authentication.

8.5.5 Target Client Chain Processing Rules

Target Client Chain processing started with the POPK and the minimum **Revocation List** of a **Micro Server MSConfig** state. Target Client chains processing by the CPS shall follow the generic rules as defined in clause 8.1. In addition Target Client Chain processing by the CPS shall follow these specific rules:

1) First **Revocation List** is of type 0x0 (Target Revocation List).

2) First **Certificate** is of type 0x0 (Target Group Certificate) or 0x8 (**Micro Client** Certificate).

3) Step 1 and 2 are repeated in case the **Certificate** in step 2 is a Target Group **Certificate**.

The resulting **Micro Client** certificate public key is the Chipset Public Key that shall be used according to the mechanism described in clause 5.3.

8.6 CPS **ECI** Root Key initialization

At initialization time of the **AS System** the **ECI Host** loads the CPS with the latest information on the applicable root key and the **Revocation List** number.

```
function InitCPSEciRoot(uchar minRootKeyVersion, uint minRevListNr)
```

Semantics:

The following c-code shall be executed:

```
cpsEciRootState.rootVersion = minRootKeyVersion;
cpsEciRootState.rlVersion   = minRevListNr;
```

CPS will apply **rootKeyVersion** as the **ECI Root Key** version number and will apply **minRevListNr** to all chains provided to it for loading **ECI** credentials.

All other states of the **AS System** will be reset.

Note that the setting of both parameters by the **ECI Host** should ensure that all **ECI Clients** can be loaded and that the **ECI Host** is not revoked, yet none of the **ECI Clients** suffers **Revocation**.

9 Loader Core

9.1 Introduction

The **ECI** system uses a loader mechanism that permits **ECI Clients** to securely verify the version of the **ECI Host** and **ECI Client** credentials that are loaded so as to detect any known security issue. This permits the **ECI Host** and **ECI Clients** (both images and POPK) to be updated as a regular system operation function.

The loader for **ECI Host** and **ECI Client** images relies on certain **Robustness** principles defined as rules which are defined in the following clauses. The **Robustness** of implementation of these rules shall be defined by a suitable document outside the scope of the **ECI** Specification, but in general the rules are to have equal **Robustness** of implementation. Some rules are deemed to be implemented with a higher (prime) **Robustness** and are to be substantially more robust than the implementation of the **ECI Host**.

9.2 Host Loader Rules

The **ECI Host Loader** shall comply with the following rules:

1) The **ECI Host Loader** shall ensure the **ECI** Root version and **ECI** Root **Revocation List** version number used to validate the **ECI Host** Images is stored on power-on initialization and it shall not be possible to change this number from there on. This rule requires prime **Robustness**.

2) It shall not be possible to modify the **ECI Host Loader** itself. This rule requires prime **Robustness**.

3) It shall not be possible to modify or observe an **ECI Host** image once loaded in as far as this is required to prevent manipulation sensitive information or observation of secret information.

4) Any subsequent **ECI Host** Image verification (in case of a staged loader) performed by software stemming from a previous image shall use the same host certificate public key and **Revocation List** for verification.

It is recommended that staged loaders use a single secure mechanism for validating **ECI Host** images also used for validating the first loaded **ECI Host** image.

9.3 Client Loader Rules

The **ECI Client Loader** exists in the context of the **ECI Host**. The **ECI Host** sets the minimum **ECI** Root version and **ECI** Root **Revocation List** version it uses to validate **Certificate Chains** before loading any client related item. The **ECI Client Loader** shall comply with the following rules:

1) The **ECI Client** image shall be decrypted first if so required as defined in clause 9.5.

2) The **ECI Client** Image and POPK shall be validated using CPS processed chains as defined in clause 8. This rule requires prime **Robustness**.

3) The **ECI Client** Image or Client Series Image **Certificate** (as applicable) shall be co-verified with POPK and the Platform Operation Client **Revocation List**. The adequacy of the version number of this **Revocation List** is verified later on **AS Slot** session initialization by the **ECI Client**.

4) It shall not be possible to modify or observe an **ECI Client** image once loaded.

5) **ECI Clients** shall not be able to "break their sand-box" and observe or modify the **ECI Host** or **ECI Client** behaviour.

9.4 Revocation enforcement

ECI uses a robust enforcement mechanism for the verification of the **ECI Host** and **ECI Client** image credentials. This operates under the following rules:

1) The descrambler shall stop operating in case the **ECI** Root version and the minimum root **Revocation List** version number for verifying the **ECI Host Certificate Chain** were lower than the ones loaded by the **ECI Host** at initialization. This rule requires prime **Robustness**.

NOTE 1: This should be atypical since Host Root **Revocation Lists** should be updated regularly through channels of all operators, and the **ECI Host Loader** can use the latest **ECI Host** Root **Revocation List**.

2) The **AS System** will refuse to load any **ECI Client** whose **Certificate Chain** cannot be validated using the **ECI** Root version and minimum **ECI** Root **Revocation List** number set by the **ECI Host** on initialization as defined in clause 9.2. This rule requires prime **Robustness**.

3) The **AS Slot** will refuse to compute keys in case the minimum root version number and minimum root **Revocation List** version numbers required by the **ECI Client** are lower than the ones loaded by the **ECI Host** at initialization. This is defined in the computation rules for client image, encryption and decryption keys in clause 6.2.4. This rule requires prime **Robustness**.

NOTE 2: These rules ensure content security systems can require a minimal **ECI** Root state is be applied to the verification of all items loaded in an **ECI Host** before proceeding with any security sensitive operation.

9.5 Client Image decryption

For the purpose of decrypting an **ECI Client** image the **Advanced Security System** can decrypt the encrypted image decryption key provided by the **ECI Client's** operator and decrypting an **ECI Client** image as defined in ETSI GS ECI 001-3 [3], clause 7.8. **ECI Client** image decryption shall be performed before **ECI Client** image signature checking. The AS function used to compute the image decryption key is reqAsComputeImageKey as defined in clause 6.2.4.12. The **ECI Host** receives the required encrypted key information inputV (input message to the **Authentication Mechanism** from which to compute an authentication key), eKey (image decryption key encrypted with the authentication key) and "online" and "min_root_state" parameters as defined by ETSI GS ECI 001-3 [3], clause 7.8 from the operator and uses the **AS Slot** later provided to the Client to perform the decryption (see ETSI GS EC 001-3 [3], clause 7.8). Any nonce used for the image decryption key exchange session shall be fresh (value from the re-initialized **AS Slot**).

10 Timing requirements

10.1 Introduction

ECI Clients need to perform within certain timing constraints in order to meet the requirements of the security system they are a part of. For this, **ECI Clients** depend on certain performance characteristics of the functions the **AS System** offers (through the **ECI Host**). This clause defines the timing characterization of the **AS system** functions.

The **AS System** timing characterization divides the functions into four categories:

1) Functions requiring merely *administrative functions* in the **AS Slot**.

2) Functions requiring only *symmetrical cryptography operations*, like **Key Ladder** computations or decryptions with AK.

3) Functions requiring one to four *asymmetrical cryptography operations* in either the **Key Ladder Block** or CPS, like loading of LK1 and performing functions involving the computation of AK.

4) Functions requiring processing of potentially longer **Certificate Chains** like Import/Export chains and **Micro Client** authentication chains.

The **ECI Client** can invoke function of the last three categories through asynchronous messages. Functions in the first category can be either synchronous or asynchronous.

Asymmetrical cryptography operations take more time. Any ongoing asymmetrical cryptography operation shall not stall functions of first two categories. In case a function in category 1 or 2 requires a result from an operation in function 3 or 4 the **ECI Client** is responsible for synchronization of the result of the a function in category 3 or 4. I.e. it has to wait until the result of the asymmetrical operation is available (i.e. the result message is received) before invoking a function dependent on the result.

10.2 Administrative Functions

For the functions in category 1), the general criteria for symmetrical and asymmetrical messages shall apply.

10.3 Symmetrical Cryptography Functions

Functions invoking symmetrical cryptography operations shall be performed by the **AS System** under the conditions described in clause 4.5 of ETSI GR ECI 004 [i.3].

10.4 Asymmetrical Cryptography Functions

Functions invoking asymmetrical cryptography operations (e.g. involving the **Key Ladder** symmetrical key computations or using the result of the **Authentication Mechanism**) shall be performed by the **AS System** under the conditions described in clause 4.5 of ETSI GR ECI 004 [i.3].

Annex A (normative): Cryptography Function Definitions

A.1 Hash Function

The hash functions in the present document are all based on SHA256 as defined in NIST FIPS PUB 180-4 [7].

Function *hash* clause 5.2 is equal to SHA-256() as defined in NIST FIPS PUB 180-4 [7].

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 octetestring 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 bitstring (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 ETSI GS ECI 001-5-2 [5], clause 7.

A.2 Asymmetrical Cryptography

The asymmetrical encryption and decryption operations shall be defined by ETSI GS ECI 001-5-2 [5], clauses 8.2 and 8.3.

A.3 Random Number Generation

Random number generation as defined in the present document shall comply with NIST Special Publication 800-90A [8] and satisfy the following rules:

- At minimum at system start (reboot of a chip's **AS System**) a new secret unique random seed number shall be generated. The process depends on physics (noise) or other properties of the chip or its environment that are not replicable and cannot be manipulated. The entropy of the generated number shall be at least 128 bit.
- Any random numbers shall be generated with a deterministic pseudo random number generator based on the above random seed number in accordance with NIST Special Publication 800-90A [8]. The chip may reseed the generator regularly and/or increase the entropy as defined in [8], clause 8.7 using internal (noise) or external inputs that are hard to manipulate. At minimum the chip-id shall be used at a personalization string.

NOTE: In many AS applications the actual randomness of the random number generator is not critical, only the uniqueness over time is. These are typical nonce applications: e.g. random number for online authentication for replay prevention at decryption and insertion of a random number at encryption of content. Exception is the random key generated as LK1 in an encryption **AS Slot** in asymmetrical **Micro Server** mode.

Annex B (informative): Sample Micro DRM system application

B.1 Introduction

This annex provides a realistic example of an application of the **AS System** for the implementation of a Micro DRM system operating on a TS stream. The example operation of both encryption as well as decryption **ECI Clients** are presented. The focus of the presentation is on the concurrency of various actions and the sequence of control words and associated Micro DRM messages (from **Micro Server** to **Micro Client** and vice versa) that need to be generated. The Micro DRM system uses both random key generation at encryption as well as nonce generation at decryption (to prevent replay). It assumes both random keys have a limit.

B.2 Application scenario

The application scenario in Figure B.2-1 shows the state of the **Key Ladder** at the encryption side. LK_n is the third but lowest key in the key hierarchy. Below that are the nonce (N1 or N2) from the **Micro Client**, the **Content Properties** CP1 and CP2 (processed into input-C for the **Key Ladder** in stage n+2) and the random key seed R1 and R2 that input to **Key Ladder** stage n+3. From these **Key Ladder** inputs the control words CW1..CW4 are computed and applied to the content in conjunction with their associated **Content Properties**.

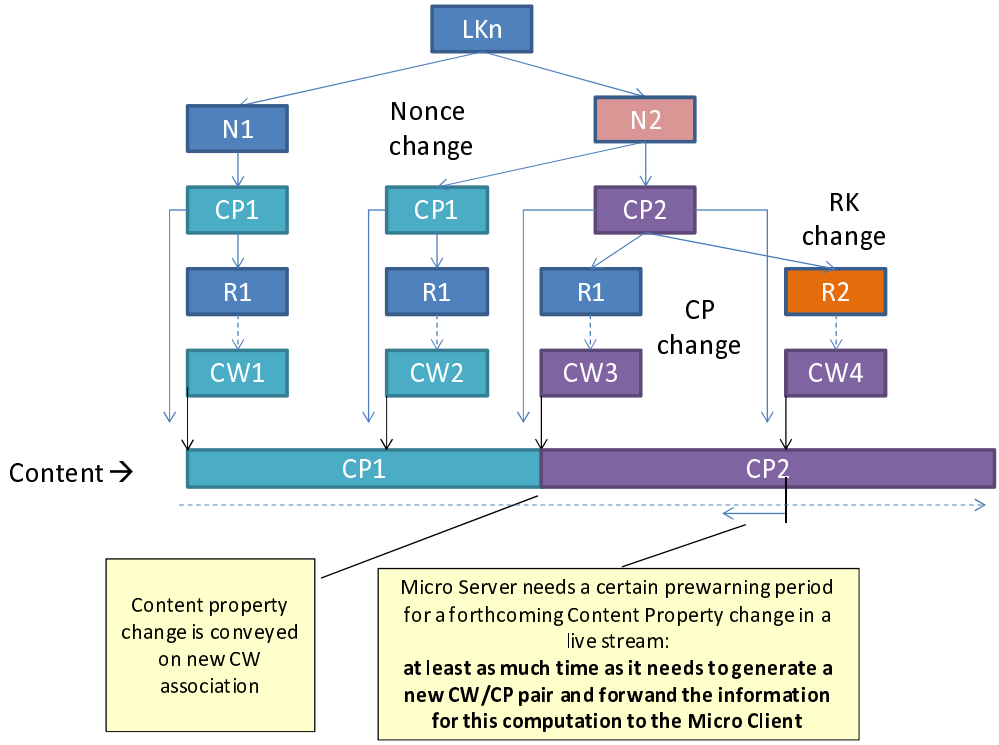

Figure B.2-1: Example of control word computation key hierarchy evolution

The starting state of the lowest three stages of the **Micro Server** key hierarchy is N1, CP1, and R1. From these CW1 is computed to encrypt the content. The initial state of the toggle bit is t1. In this example the **Micro Server** first receives a new nonce and decides it is time to apply it to future content in the form of CW2. It first sends an ECM-type message to the **Micro Client** with the new toggle bit t2 and the set of encrypted keys (t2, N2, CP1, R1), it waits for some time to make sure the **Micro Client** can receive and precompute the new control word CW2 and be ready for the forthcoming change. It then computes the new control word CW2 itself and commits to application, triggering a change in the toggle bit of the associated encrypted TS stream.

The next event in Figure B.2-1 shows a change in the **Content Properties** of the content to be encrypted. The **Micro Server** receives a message from the **ECI Host** that the **Content Properties** will change to CP2. The **Micro Server** sends an ECM-type message to its **Micro Client** with the new set of encrypted keys (t3,N2,CP2,R1) and precomputes CW3 from this new sequence of keys. At the moment the new **Content Properties** apply, the toggle bit on the encrypted content changes automatically and the new encryption control word CW3 and its associated **Content Properties** CP2 are effectively applied.

The last event is the **Micro Server** deciding to change the Random Key R1 to R2. The process is practically identical to that of changing a nonce. The **Micro Server** sends an ECM-type message with (t4,N2,CP2,R2) to the **Micro Client** permitting it to precompute CW4, and uses a delay to ensure the **Micro Client** has enough time to be ready. It then uses the **Key Ladder** to compute CW4, and applies it to the content, causing the state of the content toggle bit to become t4.

B.3 Assumptions and notation

The content is delivered by an exporting client plus associated decryption **AS Slot** to the import connection of the encryption **AS Slot**. The exporting client generates messages to the **ECI Host** signalling any changes in **Content Properties** ahead of the actual occurrence in the imported content. This uses the AS API in [3].

The following notation is used:

<event-name>(parameters) -> <pseudo-code statement> ; indicates that on event event-name (a message reception)with the following pseudo-code is executed.

The following events are defined:

- **e_cp(cp):** new content properties cp will be used on a forthcoming event (CW change) in the content to be encrypted. Preceds `e_cpe()`.
- **e_cpe():** content property change is eminent (due within a limited time).
- **e_cpch():** the content properties of the imported content just changed, in case the control word currently used is not reflecting that it will require an urgent change. On an automatic change of control word due to a control property change this events preceds `e_cw()`.
- **e_nn(nonce):** a new nonce message from the **Micro Client** has arrived at the **Micro Server** or is sent from the **Micro Client**.
- **e_cw():** the toggle bit changed on the re-encrypted content, and the (previously computed) new CW is applied to the content.
- **e_ecm(<parameters>):** reception of a message with new parameters of the next control word to be used.
- Events can be raised by timers.

cw(toggle_bit,random_key,nonce,content_properties) performs the generation of a control word for encrypting or decrypting content using the designated parameters. At the **Micro Server** first a message is generated with the same parameters which is forwarded to the **Micro Client**, there received there as `e_ecm(…).`

block_cpch() and unblock_cpch() use the message setAsPermitCPChange(…) to block or unblock automatic changes in the encryption control word due to changes in the content properties of the imported content.

changeCw(toggleBit) forces a changeover of the control word (toggle bit in the scrambling control field) at the encryption side using the message setAsSC() as defined in clause 7.9.

startTimer(timerHandle) starts a timer.

For variables and pseudo code a c-style notation is used.

B.4 **Micro Server** pseudo code

A key complication in the **Micro Server** means that it has to handle several concurrent and unsynchronized events that may trigger a change in control word:

- the arrival of content requiring new content properties (demarcated by a new control word application by the exporting client to content being decrypted);
- the forthcoming expiry of the nonce; and
- the forthcoming expiry of the random key.

The processing of a change in **Content Properties** needs to have priority since there is typically a limited time before the corresponding control word change in the decryption process triggers the new content property application. So the nonce and random key expirations should be set sufficiently conservatively since they may have to be postponed by the duration it takes to process a content property change (typically a few seconds). This assumes that the time between content property changes is always sufficient to permit the processing required for at least a nonce or random key change to a control word.

The processing of eminent changes in nonce or random key has two priorities. First a timer is set for a low priority. In case there is no pending change in **Content Properties** the change in nonce or random key is made, otherwise a timer is set for a high priority change. A high priority change in nonce or random key may overrule an eminent content property change. But the **Micro Server** may come to a wrong conclusion. In case this happens the content property change occurred before the nonce or random key change was applied to the content. In that case it has to recompute a new control word that also includes the new **Content Properties**. Also a content property change may occur almost immediately after a high priority nonce or random key change is applied. In that case the CW reflecting the new **Content Properties** and ECM that the **Micro Server** computes will be late.

If the TNONCEURGENT and TRKURGENT timer values can be set to a value of more than 10 seconds plus TECM and the maximum time between e_cpch() and e_cw(CPCHANGE) is less than 10 seconds such collisions can occur, since either any RK and nonce change can be scheduled before the period between e_cpch() and e_cw(CPCHANGE) of after such a period without the priority requiring to be raised.

Note that the manipulation of variables `rc` and `rn` as presented below cannot be done directly by the client but has to be performed using functions of the **AS System**.

```
/*
   four priority processing model with small shift of CP change time
   in case priority 4 is required (here & now non-anticipated change in CP):
    1) low priority nonce/rk change
    2) low priority CP change (cp eminent but e_cpch() did not occur)
       adopts any previous nonce or rk changes
    3) high priority nonce or rk change; reverts to old CP value
    4) high priority CP change; adopts pending nonce/rk changes and new cp;
       queues new changes

    Optimization may be possible to try to schedule pending nonce and rk changes
    immediately after a CP change; provides modest performance improvement

    State variable invariants/meanings:
    <x> = cp (content property), n (nonce) or r (random key)
    Invariant: p<x> = change in <x> in next CW (p = pending)
                      (not for low priority <n> or <r>)
               q<x> = queud change for <x>, not pending for next CW
               hpcp = high priority content property change (pcp || qcp)
   During a brief time between changeCw() and e_cw() all changes are queued.
   This temporary state is indicated with dhp==true;
 */

#define TECM 3000 /* delay between sending ecm message and changing CW */
#define TNONCEURGENT (2*TEMC + 1000)
#define TRKURGENT (2*TECM + 1000)
#define TNONCE /* some value; may be dynamically determined*/
#define TRK    /* some value; may be dynamically determined*/

toggle(bool t) { return !t }; /* toggles between true and false */

encryptionSession()
```

```
/* case rk & nonce change and cp change; unreliable warning cp change (priority with nonce/RK
change) */
/* first priority on nonce/rk lower than cp, but if urgent it is higher */
{
  SymKey  nc, nn;   /* current and next nonce */
  SymKey  rc, rn;   /* current and next Random Key */
  SymKey  cpc, cpn; /* current and next CP value */
  SymKey  nt, rt;   /* temporary value for nonce, random key */
  TimerHandle t_lpn, t_lpr;
               /* timers for low priority scheduling of nonce and rk change */
  TimerHandle t_n, t_r;
               /* timers for high priority scheduling of nonce and rk change */
  TimerHandle t_ecm_n1, t_ecm_r1;
               /* ecm timers for low priority (1) nonce and rk  ecm */
  Timerhandle t_ecm0, t_ecm1 t_ecm2, t_ecm3;
  TimerHandle t_ecm[4] = {t_ecm0, t_ecm1, t_ecm2, t_ecm3 };
               /* four level 2/3/4 priority level ecm timer pool */
  int t_ecm_cnt = 0;  /* counter for above timer pool allocation */
  bool pn, pr, pcp;   /* true if current CW reflects a change in nonce (nn),
                         random key (rn) or cp (cpn) value */
  bool qn, qr, qcp; /* true if a queued change in nonce, random-key or cp change */
  bool dhp;         /* delay (queue) any new events */
  book hpcp;        /* true if priority 4: high priority CP change */
  int  tCnt1, tCnt234;  /* tCnt<n> is the counter for number of timers
                        in priority <n> that are fired but not yet expired */
  bool t;           /* toggle bit */

/* some macro's are defined to permit reuse of code for processing events */

.* event for next random key */
#define next_r() { rc = rn; rn = rnd128(); startTimer(t_lpr,TRK); }


/* force changeCw on last cascaded higher priority timer unless it is a level 2
   priority cp change in which case the change of CW will be triggered by a CP
   change event */
#define process_emc2_timer(){\
  if (--tCnt234 == 0)\
    if (pn || pr || hpcp){\
      dhp = true; changeCw(toggle(t));\
    } else {\
      /* pcp == true, pn, pr, hpcp == false */\
      unblock_cpch();\
    };\
}

/* on cw-change update state with all processed changes */
#define end_pending() {\
  t = toggle(t);\
  if (pcp) { cpc = cpn; pcp = false };\
  if (pn)  { nc = nn; pn = false };\
  if (pr)  { next_r(); pr = false };\
}

/* move queued events to pending */
#define queued_to_pending() {\
  if (qcp && (!(qn || qr) || cphp)) {
    /* if priority 2 or 4 */\
    pcp = true; qcp = false\
  };\
  /* priority 3 events can be folded with priority 4 */
  if (qn)  { pn = true; qn = false };\
  if (qr)  { pr = true; qr = false };\
}

/* start cw/ecm for pending changes to cw */
#define start_pending() {\
  cnt = 0;\
  if (pcp) { cpt = cpn; cnt++ } else cpt = cpc;\
  if (pn)  { nt  = nn ; cnt++ } else nt  = nc;\
  if (pr)  { rt  = rn ; cnt++ } else rt  = rc;\
  if (cnt > 0) {\
    block_cpch();\
    cw(t,rt,nt,cpt);\
    tCnt234++;\
    startTimer(t_ecm[t_ecm_cnt++],TECM);\
    if (t_ecm_cnt >=4) t_ecm_cnt = 0 ;\
  }\
```

```
}


  /* only permit auto-changes of toggle bit when prepared */
  block_cpch();

  /* receive first cp and nonce values */
  for (int i=0; i<2;) {
    ->e_nn(&nc): i++;
    ->e_cp(&cpc): i++;
  }

  /* initialise state */
  pn = pr = pcp = hpcp = false;
  dhp = false;
  tCnt1 = tCnt2 = 0;
  rc = rnd128(); rn = rnd128() ;
  t = false;          /* should be initialised to first value in content */
  cw(t,rc,nc,cpc) ;   /* will start to be used automatically */

  while (!end_session) {
  ->e_nn(&nn) : startTimer(t_lpn,TNONCE) ;
                /* should occur before nonce limit runs out */
  ->e_cp(&cpn): /* e.g. compute new export licenses */ ;
  ->t_lpn()   : { /* low priority nonce change */
      if (pcp || pn || pr || cphp) {
         /* delay new nonce till urgent */
         startTimer(t_n,TNONCEURGENT);
      } else {
        nc = nn;
        cw(t,rc,nc,cpc);
        startTimer(t_ecm_n1,TECM) ;
        tCnt1++;
      }
    };
  ->t_lpr()   : { /* low priority rk change */
      if (pcp || pn || pr || cphp) {
        /* delay RK till urgent */
        startTimer(t_r,TRKURGENT);
      } else {
        next_r();
        cw(t,rc,nc,cpc);
        startTimer(t_emc_r1,TEMC);
        tCnt1++;
      }
    };
  ->t_emc_n1() :   /* low priority nonce ecm timer expiry */
  ->t_emc_r1() : { /* low priority rk ecm timer expiry */
      if (--tCnt1 == 0 && tCnt234 == 0) {
        changeCw();
        dhp = true;
      }
    };
  ->e_cpe()   : { /* cp change may occur from now on */
      if (dhp || (pn || qn)) qcp = true; /* assert(!hpcp) */
      else { pcp = true; start_pending() };
    };
  ->t_n()     : { /* urgent nonce change due */
      if (dhp || cphp) qn = true;
      else { pn = true; start_pending() };
    };
  ->t_r()     : { /* urgent random key change due */
      if (dhp || cphp) qr = true;
      else { pr = true; start_pending() };
    };
  ->e_cpch()  : { /* high priority change of CP needed */
      cphp = true;
      if (dhp) qcp = true;
      else {
        pcp = true;
        start_pending();
      }
    };
  ->t_ecm0()  :
  ->t_ecm1()  :
  ->t_ecm2()  :
  ->t_emc3()  : {
      process_timer();
```

```
    };
  ->e_cw()     : { /* assert( pcp && !pn && !pr && !cphp ) */
      end_pending();
      queued_to_pending();
      start_pending();
    };
  }
}
```

NOTE 1: In the **Micro Server** example presented above the **Micro Server** can generate and the **Micro Client** can receive multiple successive but different ECM messages with the same toggle bit. An extreme example is that first `e_n()` occurs, then within `TDELAY` `e_r()` occurs and then with `TDELAY` thereafter `e_cp(..)` occurs. In this case three successive ECM messages are sent, with the same toggle bit, with only the last once leading to a control word that is actually applied to the content.

NOTE 2: The above code assumes that a content property change `e_cp()` is always followed relatively quickly by an actual toggle bit change. It the new `cp` value is available much sooner this is of no benefit to the **Micro Server**. The key trigger point for it to generate a new CW is the event that a `cp`-change in the incoming content is eminent. That triggers that the old value of `cp` is replaced by the new value for all forthcoming CW calculations.

The minimum pre-warning time for triggering `e_n()` or `e_r()` in the above sample code is the worst case delay between an `e_cp()` event and the actual change of the subsequent control word `e_cw()`, plus 2 × `TDELAY` plus a minor amount of event delays and processing time.

B.5 **Micro Client** pseudo code

The **Micro Client** starts a session by generating two successive nonce messages (for current and next nonce). If it receives an ECM message, it simply computes the corresponding control word. It continues to generate a new nonce and send a new nonce message once it sees the last nonce it sent being applied in an ECM.

NOTE 1: Secure nonces cannot be generated directly by the **ECI Client** code but have to use the appropriate function of the **AS System**.

```
decryptionSession()
{
  Symkey nc, nn, ln;   /* current, next and last nonce */
  SymKey cp, cpp;  /* received and previous cp */
  SymKey r;        /* received random key */
  bool   t;        /* received toggle bit */
  SymKey n;        /* received nonce */
  bool   end_session; /* end of session reached */


  /* initialise and send nonces */
  nn = rnd128();
  e_nn(nn);
  cpp = Reserved; /* undefined value */
  ln  = Reserved;

  while (!end_session) {
    ->e_ecm(&t,&r&n&cp): {
        if (cp!=cpp) { /* new CP; send event to all export connections via host */
          e_cp(cp);
          cpp = cp;
        }
        cw(t,r,n,cp);
      };
    ->e_cw(): {         /* also triggered on first cw application */
        if (n != ln) {   /* new nonce actually used; move nonce forward */
          nc= nn; nn= rnd128();
          e_nn(nn);
          ln = n;
        }
      };
  } /* end while loop */
} /* end decryption session */
```

NOTE 2: It is not necessary to send the full nonce values back from **Micro Server** to **Micro Client**. An alternating bit can be used instead as an indirect reference. In addition it is not strictly necessary to send all parameters in all ECMs: only the changes need to be communicated to the **Micro Client**, with due note that in some cases all three **Key Ladder** inputs nonce, **Content Properties** and random key can change at once (see note 2 in clause B.4). Sending the toggle bit along is useful for synchronization and it avoids that any (deliberately or not deliberately) repeated ECM message is interpreted as a message for computing a next control word.

B.6 Micro DRM system cascading effect on ECM pre-delay

The **Micro DRM** server relies on a pre-warning period (pre-delay) of a forthcoming content property change from the **Micro Client** it imports content from to permit it to pre-calculate an ECM message and send it to its peer **Micro Client**. The time it takes to do the required processing (which can be relatively short: typically no significant computation is required) plus time to forward this ECM message to the **Micro Client** may be longer than the path used to convey the newly re-encrypted content to the **Micro Client**. That means that any pre-delay for the new ECM that the **Micro Client** experiences is correspondingly shorter than that experienced by the **ECI Client** that the content was originally imported from.

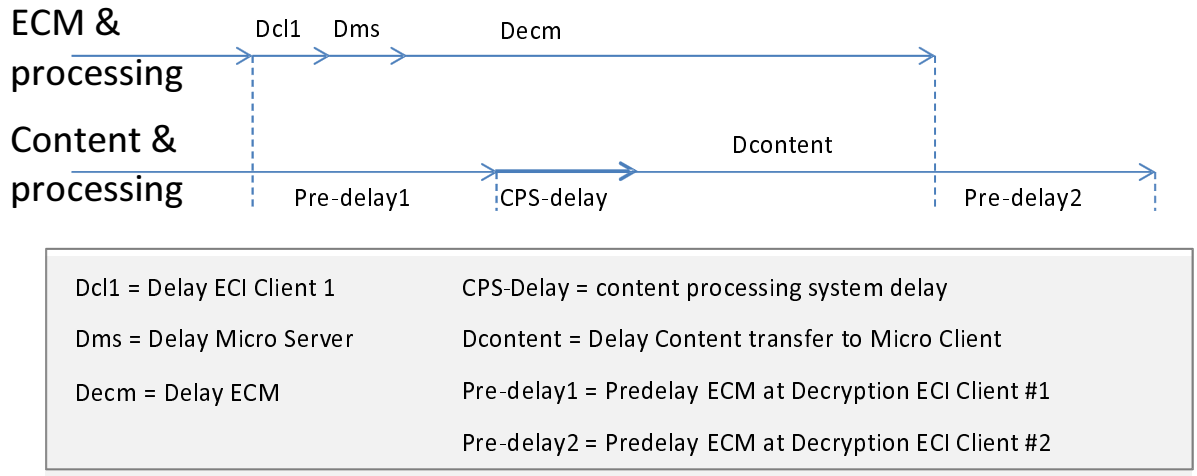

Figure B.6-1: Temporal relations for pre-delay and optional delay compensation

The content processing system can introduce a delay in the transfer of content to compensate for the delay in forwarding ECM messages as shown in Figure B.6-1. This delay can then be selected to be roughly equal to the delay difference. In case ECMs are inserted Decm and Dcontent are closely matched. But the processing delays in the decrypting **ECI Client** and those in the **Micro Server** to the point of actual insertion of the ECM in the TS stream should be compensated.

B.7 Content property change timing interface convention

As demonstrated in clause B.4 the **Micro Server** requires a pre-warning of a forthcoming content property change in its imported content. The convention for the minimum time period required to process the change and to send an ECM message to the **Micro Client** is referred to as TECM: in the following paragraphs of this annex an example is given. For this example the value of TECM is set to 3 s.

The convention for the minimum prewarning delay of a first decryption **ECI Client** in a chain of cascaded **ECI Clients** is TECM + TCASCADE. TCASCADE reflects the maximum cumulative delay of processing ECMs by **ECI Clients** in a cascaded chain of **Micro DRM Systems**. In this example TCASCADE is set to 2 s.

NOTE 1: In cases the content is also delayed this compensates for ECM processing delays. This is not desirable in streaming mode however.

The maximum for the **ECI Client** ECM processing delay (uncompensated Dcl1 + Dms as in clause B.6) is referred to as TDELAY, set in this example to 0,3 s. The values in this example permit 6 cascaded **Micro DRM Systems** to operate within the TCASCADE (2s), leaving TECM minimum pre-warning period for a **Micro Serve**r.

As demonstrated in clause B.4, in order to process a content property change in the incoming content without causing shift of the content property change **Micro Servers** require not only a pre-warning of a forthcoming content change but also require an upper limit to such a pre-warning period so that it can process other control word changes (e.g. nonce and random key changes) safely. In this example the upper limit pre-warning period TMAXWARN could be safely set to 10 s.

NOTE 2: In case these conventions are not followed, the effect may be a maximum TECM shift in the location for a content property change in re-encrypted content in one **Micro DRM System** and a maximum of TECM+6*TCASCADE shift in a cascade of 6 **Micro DRM Systems**.

It is highly recommended to design the nonce and random key low priority warnings (t_lpn and t_lpr in clause B.4) sufficiently early so as to permit one (or even a few) changes in the content properties to delay the processing of nonce and random key changes. If content property changes are sufficiently spaced in time (and TMAXWARN is observed) this should prevent any overruns in processing for nonce or random key changes.

The selection of timing parameters is important for the seamless handover of content between **ECI Clients**. Specific recommendations as to the values of the delay parameters TECM, TCASCADE, TDELAY and TMAXWARN are provided in [i.3].

Annex C (informative): Authors & contributors

The following people have contributed to the present document:

Rapporteur:
Dr. Jens Johann, Deutsche Telekom

Other contributors:
Dr. Dmitri Jarnikov, Irdeto

Peter Mann, BNetzA

Msc Marnix Vlot, UC-Connect (on behalf of Vodafone)

Annex D (informative): Change History

Date	Version	Information about changes
April 2017	0.0.1	As approved at ECI#15.

History

Document history		
V1.1.1	July 2017	Publication