Document Number 2009010

Content Protection Specification TC-2008-0001

November 10, 2009

Sponsored by:

1394 Trade Association

Accepted for publication by:

1394 Trade Association Board of Directors

Abstract:

Keywords:

Must, Must not, Required, Shall, Shall not, Should, Should not, Recommended, May, and Optional

1394 Trade Association Specification

1394 Trade Association Specifications are developed within Working Groups of the 1394 Trade Association, a non-profit industry association devoted to the promotion of and growth of the market for IEEE 1394-compliant products. Participants in Working Groups serve voluntarily and without compensation from the Trade Association. Most participants represent member organizations of the 1394 Trade Association. The specifications developed within the working groups represent a consensus of the expertise represented by the participants.

Use of a 1394 Trade Association Specification is wholly voluntary. The existence of a 1394 Trade Association Specification is not meant to imply that there are not other ways to produce, test, measure, purchase, market or provide other goods and services related to the scope of the 1394 Trade Association Specification. Furthermore, the viewpoint expressed at the time a specification is accepted and issued is subject to change brought about through developments in the state of the art and comments received from users of the specification. Users are cautioned to check to determine that they have the latest revision of any 1394 Trade Association Specification.

Comments for revision of 1394 Trade Association Specifications are welcome from any interested party, regardless of membership affiliation with the 1394 Trade Association. Suggestions for changes in documents should be in the form of a proposed change of text, together with appropriate supporting comments.

Interpretations: Occasionally, questions may arise about the meaning of specifications in relationship to specific applications. When the need for interpretations is brought to the attention of the 1394 Trade Association, the Association will initiate action to prepare appropriate responses.

Comments on specifications and requests for interpretations should be addressed to:

Editor, 1394 Trade Association
315 Lincoln, Suite E
Mukilteo, WA 98275
USA

Published by

1394 Trade Association
315 Lincoln, Suite E
Mukilteo, WA 98275 USA

Printed in the United States of America

Table of Contents

- This page left intentionally blank -

Part One: Introduction and Overview

1 Introduction

1.1 Purpose and Scope

This document details the content protection mechanisms and uses within the HANA system architecture. It is intended to provide adopters with the required elements and behavior for handling protected content.

The HANA content protection architecture is concerned with enabling licensed content to be protected from being accessed for unapproved use both within the HANA home network, and unapproved distribution beyond the HANA home network to which it is licensed.

1.2 Document Organization

This specification is organized into several parts.

Part One: Introduction and Overview

Part Two: Details the content cluster, and binding of protected content.

Part Three: Details supplemental information for use by implementers.

1.3 Notice

Use of this specification and the associated cryptographic materials and patents needed to implement it require one or more licenses be obtained by adopters. These include 5C for DTCP, HANA, and the associated Licensing Authority, as well as other rights holders.

1.4 Related Documents

This document is part of the HANA design specification which consists of this book, the HANA Design Guideline, and the HANA Content Services Book.

1.5 References

The following standards contain provisions that, through reference in this text, constitute normative provisions of this standard. At the time of publication, the editions indicated were valid. All standards are subject to revision, and parties to agreements based on this standard are encouraged to investigate the possibility of applying the most recent editions of the standards listed below.

[1] High-Definition Audio-Visual Network Alliance (HANA) Design Guideline 2.0

[2] HANA Content Services Book 2.1

[3] CEA-2027-A, A User Interface Specification for Home Networks Using Web-based Protocols, February 2005

[4] Errata for CEA-2027-A, April 2006

[5] CEA-2027-B, A User Interface Specification for Home Networks Using Web-based Protocols, February, 2007

[6] Advanced Access Content System: Introduction and Comment Elements 0.91

[7] 4C Content Protection System Architecture: http://www.4centity.com/data/tech/cpsa/cpsa081.pdf

[8] ANSI x9.31 FIPS Publication 186-2 + Change Notice

[9] NIST Special Publication 800-38A

[10] NIST Special Publication 800-38B

[11] NIST Special Publication 800-22

[12] FIPS Publication 197

[13] ATSC A/98

[14] RFC 2119

[15] Federal Information Processing Standards Publication 180-2

[16] IETF RFC 1737

[17] IETF RFC 2141

1.6 Keywords

This document follows the convention set out in RFC 2119:

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", “MAY", and "OPTIONAL" in this document are to be interpreted as described in RFC 2119.

Several keywords are used to differentiate levels of requirements as follows:

Shall or Must or Required

Indicate a mandatory requirement. Designers are required to implement all such mandatory requirements to assure interoperability with other products conforming to this standard. The addition of “Not” to either of these words indicates the prohibition of the specified requirement.

Should or Recommended

Denote flexibility of choice with a strongly preferred alternative, which may be ignored when the circumstances and implications of ignoring them are understood. The negation “Should not”, or “Not recommended” denote the flexibility of choice with a strongly preferred alternative which may be ignore under certain circumstances.

May or Optional

Indicate flexibility of choice with no implied preference.

1.7 Notation

1.7.1 Decimal Numbers

Decimal numbers are expressed as the value using the digits 0-9 (e.g. 170) without any additional notation.

1.7.2 Binary Numbers

Binary numbers are expressed using the digits 0 and 1 with the least significant digit on the right with a subscript of 2. e.g. 170 is expressed as 10101010_2

1.7.3 Hexadecimal Numbers

Hexadecimal numbers are expressed as the digits 0-9 and A-F with a subscript of 16. For example, 170 is expressed as AA_{16}.

1.7.4 Bit Ordering

Unless explicitly noted otherwise values expressed as an array of bits are numbered such that for an array of n bits the least significant bit is numbered 0 and the most significant bit is numbered n-1.

1.7.5 Byte Ordering

Unless explicitly noted otherwise multi-byte values must be stored in big-endian form meaning that the first byte stored by holds the most significant byte of the value. For example, $80FE_{16}$ is stored as 80_{16} followed by FE_{16}.

1.7.6 Operators

The following notation will be used for bitwise and arithmetic operations:

$[x]_{msb\_z}$ The most significant z bits of x. $[10010000_2]_{msb\_4} = 1001_2$

$[x]_{lsb\_z}$ The least significant z bits of x. $[10010000_2]_{lsb\_4} = 0000_2$

$[x]_{y:z}$ The inclusive range of bits between bit y and bit z in x.

$\sim x$ Bit-wise inversion of x.

$x \parallel y$ Ordered concatenation of x and y.

$x \oplus y$ Bit-wise Exclusive-OR (XOR) of two strings x and y.

$x + y$ Modular addition of two strings x and y.

$x \times y$ Multiplication of x and y.

$x - y$ Subtraction of y from x.

The following assignment and relational operators will be used:

= Assignment

== Equivalent (Equal to)

!= Not equal to

< Less than

> Greater than

<= Less than or equal to

>= Greater than or equal to

1.8 Nomenclature

1.8.1 Abbreviations

AACS	Advanced Access Content System
AES	Advanced Encryption Standard (FIPS Publication 197)
AIS	Authorized Input Domain
AOD	Authorized Output Domain
AT	Authorization Table
ASCCT	Advanced Secure Content Cluster Technology from IBM
CCI	Copy Control Information
CMI	Content Management Information
CTD	Content Tracking Data
DCL	Device Content List
DTCP	Digital Transmission Content Protection from 5C
FA	Fixed Area
GUID	Global Unique Identifier
HDMI	High-Definition Multimedia Interface
ISAN	International Standard Audiovisual Number
MAC	Message Authentication Code
MKB	Management Key Block
NIST	National Institute of Standards and Technology
PFA	Protected Fixed Area
SCSM	Serial Copy Management System
SRM	System Renewability Messages
TKB	Title Key Block
TS	Transport Stream
UCI	Universal Content Identifier
URN	Uniform Resource Name
UUID	Universally Unique Identifier
VLDA	Variable Length Data Area

1.8.2 Definitions

Term	Abbrev	Description

Term	Abbrev	Description
Authorization Table	AT	Table used by the HANA Content Cluster to hold devices in the cluster, and other state information about the cluster
Binding ID	ID_b	The cryptographic secret used to generate the Binding Key of an ASCCT Content Cluster
Binding Key	K_b	The cryptographic key used to bind data to an ASCCT Content Cluster
Device ID		A unique ID associated with the device; a 128 bit value assigned by the HANA License Authority
Cluster ID		An random value used to distinguish an ASCCT Content Cluster instance.
Content Cluster		A collection of one or more authorized HANA devices which are bound together by a common AT and Cluster ID.
Content Device (HANA device type)		A HANA Device which can unbind content bound to the HANA Content Cluster
Content Header		Data structure holding content information and content protection information for an item of protected content.
Device Keys		A set of secret cryptographic keys issued to a HANA device by the HANA Licensing Authority used for processing Management Key Blocks
HANA Device		A device which implements the required HANA design elements. A HANA legacy device is a device which implements the HANA 2.0 specification. All HANA devices implementing the HANA 2.1 specification also include a set of cryptographic elements which enable them to join HANA Content Clusters
HANA LA		HANA Licensing Authority. The entity which is authorized to manage issuance of the Management Key Block (MKB), Device Keys, Device IDs, and other elements to approved HANA devices.
Infrastructure Device (HANA device type_		A HANA Device which contains the ability to join a HCC, but which cannot unbind content bound to the HANA Content Cluster
Item ID		A unique ID used to identify and track a content item
Management Key	K_m	A cryptographic key extracted from a Management Key Block
Management Key Block	MKB	A data file issued by the HANA Licensing Authority which hold cryptographic values which can be accessed by HANA devices using Device Keys
Management Key Precursor	K_m^{-i}	A cryptographic key extracted from a Management Key Block. Application of a one-way function allows for computation of the next lower key in the sequence of precursor keys.
Replica ID		A unique identifier used to distinguish between instances of a given Item ID
Service (HANA device type)		An abstract service which can join a HANA Content Cluster and obtain the cluster Binding ID.
Title Key	K_t	key use to encrypt an item of protected content
Title Key Block	TKB	A data structure in the Content Header which holds bound Title Keys for

Term	Abbrev	Description
		an item of protected content

2 HANA Content Protection

2.1 Introduction

The HANA system architecture is focused on the network of devices in the home operating together to provide an easy to use experience for the user. The distinction is that HANA is not simply a network connecting devices, but instead is a unification of the devices in the home allowing users to consume content on any display from any source in the home.

Likewise, the content protection in HANA is designed to operate at the aggregated network level instead individual devices. In keeping with this approach content is licensed to the HANA Content Cluster as opposed to individual devices. Content in the home is free to move between and be copied to any HANA device in the cluster without any restriction. The content protection controls used enable movement of content within the cluster, while protecting the content from unauthorized redistribution of unprotected content to outside the HANA cluster.

HANA content protection is based on the following key technologies:

- DTCP from 5C provides an encrypted secure network connection between two licensed DTCP devices, preventing interception of the content stream flowing over it.
- HDCP protects an uncompressed digital video signal over a DVI or HDMI interface from a decoding (source) device to a rendering (presentation) display device.
- Advanced Secure Content Cluster Technology (ASCCT) from IBM which creates a Content Cluster to which content is cryptographically bound, rendering the content usable only in the cluster to which it is licensed.

This architecture standard addresses managing protected content in the following cases:

1. Content stored and decoded in the same HANA device
2. Stored content streamed to a Content Decoder over the Network
3. Copying of Stored Content
4. Ingestion of protected content from an outside source

Additionally, in the United States compliance with FCC mandates is required.

2.2 Use Cases

2.2.1 Content Stored and Decoded in the Same HANA Device

A device plays back protected content stored on it, using an internal decoder. HANA protects stored content using ASCCT. The device will unlock the ASCCT protected content and send it to an internal decoder, which will output it to either an internal display or to an external display connected via HMDI and protected using HDCP.

2.2.2 Stored Content Streamed to a Content Decoder over the Network

A device holding stored protected content unlocks the ASCCT protected content and streams it to one or more content decoders over the network. The content stream is protected using DTCP over the network.

2.2.3 Copying of Stored Content

Protected content is stored by HANA devices protected by ASCCT, Such ASCCT protected content can be freely copied between devices and storage technologies, even if the devices and storage devices/media are not HANA compliant. No counting of content copies is done since content is licensed to the HANA Content Cluster and not to individual devices.

Copying of content between devices can be done by movement of media or over the network. No additional protection to restrict access to the ASCCT protected content is required.

Only HANA devices in the Content Cluster to which the content is bound are able to unlock the content for decoding. HANA devices from other Content Clusters can view the content header information of ASCCT protected content, but cannot unlock the content for playback.

2.2.4 Ingestion of Protected Content from an Outside Source

Protected content entering the HANA network will arrive with one of three forms of protection: DTCP, ASCCT for HANA, or a third party protection system.

DTCP protected content can be directly streamed over the network to a device for decoding and playback.

DTCP protected content can also be bound to the HANA Content Cluster by transcribing it to ASCCT and storing the ASCCT protected content. The DCTP Copy Control Information is honored and preserved during the ASCCT transcription.

ASCCT protected content can be directly stored for later playback.

ASCCT protected content which is bound to the receiving device's Content Cluster can also be unlocked and sent to a decoder for immediate playback under the use cases described in 2.2.1 or 2.2.2.

Content protected by a third party protection system must either: 1) have the third party protection removed and replaced by either DTCP for streaming to another device or ASCCT for storage in the Content Cluster, or 2) have the content with third party content protection additionally wrapped with ASCCT for storage in the Content Cluster, with the third party content protection preserved.

Note: Choosing to preserve the third party content protection wrapped by ASCCT may limit consumption of the content by HANA devices as the only required content protection technologies for HANA devices are DTCP and ASCCT.

Details of transcription of a third party content protection system to either DTCP or ASCCT is out of the scope of this book.

2.3 FCC Requirements

There are two digital content security measures in the 2003 FCC Ruling on Unidirectional Digital Cable (a.k.a. Cable Plug-n-Play Ruling) that are pertinent to HANA Content Security:

- DTCP for ATSC Transport Stream over a network; and
- HDCP for uncompressed digital video signal over DVI or HDMI interface from a decoding (source) device to a rendering (presentation) display device.

HANA further adds to the regime an ATSC A/98 compliance requirement.

2.4 HDCP

Devices that support uncompressed digital input and/or output over DVI or HDMI interfaces must support HDCP.

Output of protected content by HANA devices is not permitted through analog ports.

Licensing note: Adopters whose devices have DVI/HDMI input and/or output are responsible for obtaining the needing HDCP license.

2.5 DTCP

HANA requires the use of DTCP for device to device transport of protected content when it is not cryptographically bound to the HANA cluster.

Contents that are encrypted using DTCP rely on a device-to-device secured authenticated channel to transport data a transmission link between a source to a sink device. The usage policy for DTCP-encrypted content is as defined by the appropriate licensing agreements from DTLA.

Licensing note: Adopters are responsible for obtaining the required licensed from DTLA for DTCP devices.

2.5.1 Copy Control Information

HANA devices that have both DTCP and HDCP support, such as a device that uses a DTCP-protected transport stream to input or output decoded, uncompressed digital video via its DVI or HDMI ports, is required to properly map the Copy Control Information (CCI) in accordance with their licenses from DTLA (www.DTCP.com) and Digital CP (www.Digital-CP.com)

Content marked as Copy Never may be streamed between HANA devices and must not be stored beyond what is allowed for trick-play use.

Content marked as Copy Once when imported and bound to the HANA Content Cluster is treated as having a single copy compliant with this requirement regardless of the number of copies made of the bound content. This is based on the principle that the HANA Content Cluster is the entity to which content is licensed, and not a single device.

CCI must be preserved when content is imported and bound to the HANA Content Cluster.

2.6 ASCCT / Content Cluster

The Advanced Secure Content Cluster Technology (ASCCT) from IBM is used in HANA to create the HANA Content Cluster. Content is cryptographically bound to the content cluster restricting it so that it is only playable by devices in the particular content cluster to which is has been bound.

Only devices that bind content to the cluster, or that unbind it for use in the cluster must support ASCCT. Support for ASCCT is optional for all other HANA devices.

Licensing note: Adopters are responsible for obtaining the required license and licensed elements from the HANA Licensing Authority.

2.6.1 Conditional Content Access

Conditional content access restrictions such as play count limit and time limit are supported.

2.6.2 Bound Content Storage

Content which has been bound to the HANA Content Cluster may be freely stored and copied on any storage media or device in its bound form.

2.7 Additional Controls

2.7.1 ATSC A/98

HANA further requires compliance to the recently established ATSC standard on carriage of System Renewability Messages (SRM), reference ATSC A/98. The standard stipulates that SRMs for content security systems such as DTCP and HDCP, may be transmitted using a standardized data structure in an ATSC-compliant Transport Stream. HANA devices must fully support ATSC Transport Streams both at the input and the output ports. Any ATSC TS that carries SRMs in manner compliant to A/98 will be similarly supported. HANA devices are required to propagate such messages to other devices in the HANA network.

When Digital CP specifies the proper response to its SRMs carried over an ATSC TS in accordance to A/98, HANA expects that all HDCP-compliant devices on a HANA network to meet their licensing obligations. Until such time, however, HANA devices are required to support the carriage of SRMs in ATSC TS within the HANA network.

- This page left intentionally blank -

Part Two: Content Cluster

3 Content Cluster

3.1 Introduction

HANA 2.1 content devices are organized into a domain of trust known as a Content Cluster. The Content Cluster is an implementation of ASCCT as specified in this document. HANA devices must have a set of licensed elements issued by the HANA Licensing Authority.

Not all HANA devices need to have the ability to join the Content Cluster. Only devices which need to unlock content bound to the Content Cluster require this functionality. Other HANA devices such as bridges and mass storage devices for instance do not require the ability to join the Content Cluster.

3.2 Bound Content

The content cluster forms a single logical device, which has a unique Cluster ID. Content is then cryptographically bound to the cluster such that only devices which are authorized members of the cluster, and have not been revoked by the HANA Licensing Authority can unlock the encrypted content for playback.

Devices must not playback content which is bound a cluster that the device does not belong to.

Devices must only belong to one cluster at a time. To join another content cluster, a device must abandon its membership in its prior cluster. Once a device abandons a cluster, it must not play content bound to the prior cluster.

Bound content can be freely moved without restriction or additional protection as the binding of the content to the cluster encrypts the content rendering it unplayable on devices that are not in the cluster. Bound content can be freely moved using file copying protocols over a network; it can also be copied to portable media such as flash drives or optical media, and moved manually between devices.

3.2.1 DTCP

When DTCP protected content is imported and bound to the Content Cluster the DTCP CCI bits (Copy-never, Copy-once, Copy-no more.) are preserved in the content header applied to the bound data.

When DTCP is used to stream bound content that was originally protected with DTCP, the preserved CCI settings are applied to the DTCP outbound connection.

3.3 Content Cluster Devices

Devices which implement the ASCCT system detailed in this document, and which possess the licensed device control elements needed from the HANA Licensing Authority are able to join the Content Cluster. Such devices are referred to as Content Cluster Devices.

3.3.1 Content Devices

Content devices are Content Cluster Devices which have the ability to unbind content for playback in the cluster, as opposed to devices which are Content Cluster Devices, but which do not implement the ability to unbind content.

3.3.2 Infrastructure Devices

Infrastructure devices are Content Cluster Devices which lack the ability to unbind content. They may join the Content Cluster, and may process the MKBs issued by the HANA Licensing Authority.

Infrastructure Devices DO NOT count against the device count limit for a content cluster.

3.3.3 Services

Services are a special device type made up of components which provide a "service"; for example a download service for purchasing movies for a cluster. They are commonly content providing services, they only join the cluster to perform special actions, and require a higher level of security or cluster integration than is possible as a purely "guest" system.

Services are able to be authorized to join the cluster and obtain the Binding ID via the *authorizeme* message. This would for example be used by a content delivery service to obtain the needed Binding ID to allow protected content to be bound to the content cluster.

Services device types DO NOT count against the device count limit for a content cluster.

3.3.4 Device Control Elements

Content Cluster devices must store and manage the following elements in compliance with the licensing rules for HANA Content Cluster devices:

1. Cluster ID, a public 128 bit random value..
2. Device ID, a public 128 bit unique identifier issued by the HANA Licensing Authority.
3. Binding ID, a SECRET 128 bit statistically unique identifier issued by the HANA Licensing Authority.
4. Device key set, a SECRET issued by the licensing authority, allowing the device to process Management Key Blocks.
5. Device Number, a SECRET issued by the licensing authority, allowing the device to process Management Key Blocks.
6. Content Cluster MKB, the Management Key Block currently in use by the content cluster the device belongs to, issued by the HANA Licensing Authority.
7. Content Cluster Authorization Table (AT), the Authorization Table currently in use by the content cluster the device is a member of.
8. Initial Management Key Block, an MKB supplied to the device at the time of manufacture issued by the HANA Licensing Authority. This must be retained for resetting the device back to the manufacturer's default state.
9. Initial Authorization Table, an AT supplied to the device at the time of manufacture. This must be retained for use for resetting the device back to the manufacturer's default state

3.3.5 HANA Logical Units

A Content Cluster Device is not the same as a HANA Logical Unit. For example, a HANA device may contain several HANA Logical Units such as a Video Display and a Digital Video Recorder. Only the components which need to unlock bound content need to be Content Cluster Devices.

3.4 HANA Legacy Devices

HANA devices prior to HANA 2.1 do not contain the needed broadcast encryption mechanisms to join the content cluster. For this reason legacy devices cannot unlock content bound to the domain.

Playback of bound content on a HANA legacy device therefore requires first having the content unbound by a HANA Content Device followed by outputting the unlocked content to the HANA legacy device using DTCP to protect the content stream.

4 Common Cryptographic Elements

4.1 Introduction

This chapter describes the common cryptographic elements used by the Content Cluster.

4.2 AES Symmetric Block Cipher Algorithm

Symmetric cryptographic functions are based on the Advanced Encryption Standard (AES) block cipher algorithm, as specified in FIPS Publication 197. Unless otherwise specified, the AES algorithm is used with data blocks of 128 bits and keys with lengths of 128 bits.

4.2.1 ECB Mode (AES_128E and AES_128D)

Electronic Codebook (ECB) mode, as specified in NIST Special Publication 800-38A, is used for management of encryption keys.

Encryption using the AES algorithm in ECB mode is represented by the following function

AES_128E(k, d)

where k is a 128 bit key, d is a 128 bit data block to be encrypted, and AES_128E returns the 128 bit result.

Decryption using the AES algorithm in ECB mode using either the “inverse cipher” or “equivalent inverse cipher” specified in FIPS Publication 197 is represented by the function

AES_128D(k, d)

where k is a 128 bit key, d is a 128 bit data block to be encrypted, , and AES_128D returns the 128 bit result.

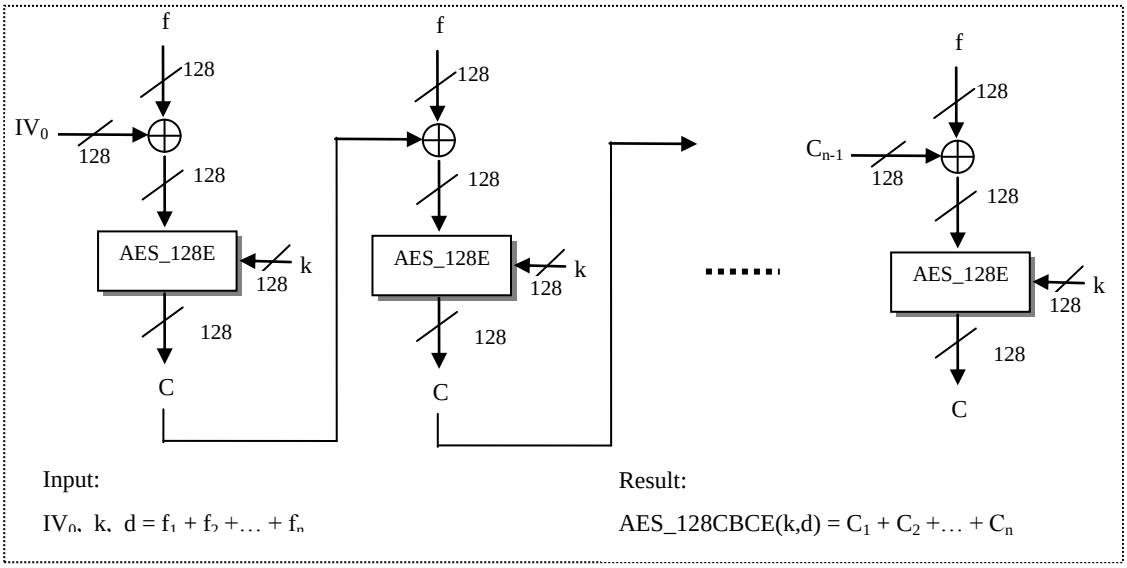
4.2.2 CBC Mode (AES_128CBCE and AES_128CBCD)

Cipher Block Chaining (CBC) mode of operation, as specified in NIST Special Publication 800-38A, is used for encryption of protected content.

Encryption using the AES algorithm in CBC mode is represented by the function which returns the encrypted frame as a result:

AES_128CBCE(*k, d*)

where *k* is a 128-bit key , *d* is the frame of data to be encrypted, and AES_128CBCE returns the encrypted frame.



Decryption using the AES algorithm in CBC using either the "inverse cipher" or "equivalent inverse cipher" specified in FIPS Publication 197 is represented by the function which returns the decrypted frame as a result:

AES_128CBCD(*k*, *d*)

Where *k* is a 128-bit key, *d* is the block of data to be decrypted, and AES_128CBCD returns the decrypted frame.

4.2.2.1 Initialization Vector

The Initialization Vector used at the beginning of a CBC encryption or decryption chain is the constant IV_0 provided by the HANA Licensing Authority.

4.2.2.2 Frame Size

The frame size used in HANA is 1024 bytes, which is the number of bytes to process before restarting the CBC chain.

4.2.3 AES-based One-way Function (AES_G)

This function is referred to as the AES-based One-way Function which returns a 128 bit result and is represented by:

AES_G(x_1, x_2)

Where x_1 and x_2 are 128-bit input values, and AES_G returns the 128 bit result.

The AES-based One-way Function result is calculated as

AES_G(x_1, x_2) = AES_128D(x_1, x_2) ⊕ x_2.b

4.2.4 AES Hashing Function (AES_H)

The AES hashing function defines a one way hash based on the AES algorithm and is represented as:

AES_H(x)

Where x is input data of arbitrary length and AES_H returns the 128 bit hashed value.

The AES algorithm processes the input data in 128 bit blocks. To accommodate this, the input data is padded so as to produce an input message which is a multiple of 128 bits.

Padding is performed by appending a "**1**" followed by **m** "**0**"s followed by a 64 bit integer which holds the length of the original message in bits. The result is a new padded message $\mathbf{x'}$ of n segments with total length **128 * n**.

The hash is calculated by applying $\mathbf{AES\_G(x'_n,h_i)}$ to each of the n segments of **x'**. With h_i calculated as

$$\mathbf{h_i = AES\_G(x'_i , h_{i-1})}$$

and the 128 bit initial value being the constant $\mathbf{h_0}$.

The final iteration **n** then produces the 128 bit result hash of the input message **x**,

$$\mathbf{h_n = AES\text{-}H(x)}$$

4.2.5 SHA Hashing Function

For the purpose of processing data to produce digital signatures, the Secure Hashing Algorithm (SHA), as defined in Federal Information Processing Standards Publication 180-2, is used.

4.3 Message Authentication Code (CMAC)

This specification uses the Cipher-based Message Authentication Code (CMAC) for message authentication. CMAC is defined in the National Institute of Standards and Technology (NIST) Special Publication 800-38B. AES_128 is used as the symmetric key block cipher.

The CMAC is represented by the following function which returns a 128 bit result.

CMAC(k, D) = AES_H(k || D)

Where k is the key to be used to create the MAC, D is the message to be authenticated.

It is thus a requirement of the context in which the CMAC is used to define the formation of the key k to be used in the CMAC calculation.

4.4 Random/Pseudorandom Number Generation

Unless specifically noted otherwise, the following generators shall be used, either singly, or together.

(1) Pseudorandom number generator based on a design described in ANSI X9.31

(2) Pseudorandom number generators defined in FIPS PUB 186-2 (+Change Notice)

(3) Random or pseudorandom number generator of equal or higher quality that passes the tests described in NIST Special Publication 800-22 when using the default parameters and other recommendations provided therein.

5 Management Key Block (MKB)

The content cluster and content binding used in HANA is built upon Broadcast Encryption technology. This system uses a pair of control elements, a Management Key Block (MKB) and device secrets, both issued by the HANA Licensing Authority to establish that a device is authorized by the licensing authority, and to enable the device to extract the control secrets stored in the MKB.

Chief among the secrets stored in the MKB is the Management Key (K_m). All valid devices will produce the same K_m from the same MKB. It should be understood however, that a different MKB, even of the same MKB version and issued by the same Licensing Authority, may produce a different K_m. This is why all devices in a Content Cluster use the exact same MKB.

The common K_m computed by all devices in the same cluster provides the basis used to build the Content Cluster. Devices are able to calculate the same key and authenticate with each other, proving that they are legitimate, without exchanging any secure or secret information (e.g. keys). This allows the messages between devices in a cluster to be exchanged without the need for a secure message transport over the network connecting the devices.

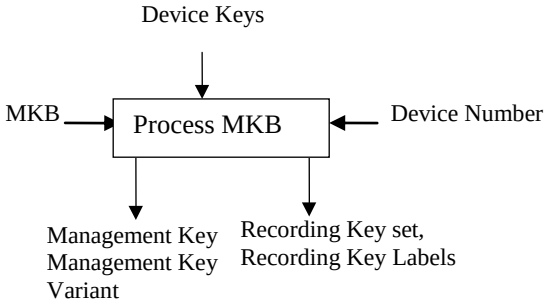
The MKB is not a protected secret. Its integrity is cryptographically guaranteed by the licensing authority to prevent tampering. The secrets it contains can only be successfully extracted by devices with valid Device Keys issued by the licensing authority which produced the MKB. Thus a new MKB can be easily distributed to devices by any number of channels, none of which need to take any protective actions to prevent disclosure of the MKB.

This system is renewable by issuing a new MKB version which has been updated to revoke devices. Revoked devices cannot process the MKB to successfully extract the secrets contained within it. They are then excluded from future interaction with valid devices.

As part of the renewability of the system, devices must be able to adopt a newer MKB as they encounter it. Once adopted by a device in a cluster, the clustering protocol distributes the new MKB to the other devices in the cluster. The cluster then uses the secrets in the new MKB as the basis for both device authentication and content binding.

The MKB used by HANA is built upon an NNL tree, widely described in cryptographic literature, which uses the subset-difference approach to enable devices in subsets within the larger tree of keys. It is used in the MKB because it provides an efficient method by which to manage revocation of devices, preventing the MKB from growing too quickly as device revocations are made within it. Devices can either be revoked individually, or in groups. The size of the MKB in a subset-difference tree system is of the same order as the size of a public key certificate revocation list, making the subset-difference tree equivalent in revocation power to a public key system, but unlike the revocation list, the MKB enforces revocations cryptographically.

5.1 MKB Processing Overview



The MKB is fundamentally a data structure holding a set of secrets generated by the Licensing Authority. These secrets are cryptographically protected and are accessed using the device's secrets issued by the same Licensing Authority to devices.

To process the MKB, devices are issued secrets by the Licensing Authority: a set of secret Device Keys (K_d), and a Device Number (D_n) which maps to a specific leaf node in the NNL tree.

It is important to note that the term *device* as used here can mean either a single physical device or a many physical devices which share the same set of Device Keys and Device Number. The allocation of device secrets to devices is covered in the licensing terms for an adopter, and is not discussed in this specification.

The MKB is processed to extract the secrets stored in it: Management Key K_m, Management Key Variant, Management Key Precursors for the device's Security Class and all lower classes, and Recording Keys for the device's Security Class and all lower classes.

Extracting the secrets from the MKB consists of processing the records held in it. The first phase of processing determines the specific index position in the records which are usable by the device keys available to the device. Once this is done, the records holding the Management Keys and Recording keys can be located within the MKB and processed using the Device Keys.

The following overview introduces the processing steps used for the MKB. The detailed calculations performed in these steps are discussed in a follow up section.

5.1.1 Subset-Difference Record

A device uses its Device Number (D_n) to locate the entry in the Subset-Difference record portion of the MKB that applies to the device. The index (*i*) of the entry is subsequently used to index the Management Key Variant Data and Variant Number records.

Additionally, the device determines the particular Device Key (K_d) to be used for this entry (subset), and uses it to calculate the Processing Key (K_p) for the subset.

Devices which are revoked in the MKB are not able to locate an applicable entry, and are not be able to proceed any further in processing the MKB.

5.1.2 Management Key Variant Data Record

The Management Key Variant (K_{mv}) is produced using the Management Key Data value in the i^{th} entry of the Management Key Variant Data Record. This value is combined with the Processing Key (K_p), and with an integrity hash calculated over the Type and Version Record and the Verify Media key Record, to calculate K_{mv}. The combination with the integrity hash makes the MKB self-authenticating.

5.1.3 Variant Number Record

The Variant Number (n), between 0 and 1023, is produced using the nonce located at the start of the Variant Number Record, and the Variant Number Data (D_{vn}) in the i^{th} entry in the Variant Number Record. These values combined with the Processing Key (K_p) are used to calculate *n*. This value *n* is used subsequently to index the records in the MKB holding the remaining keys to be extracted from the MKB.

5.1.4 Reverse Management Key Record & Verify Management Key Record

A candidate for the Management Key (K_m) is found by processing the n^{th} entry of the Reverse Management Key Record, which produces either the Management Key (K_m) for the MKB, or a Management Key Precursor (K^{-i}_m), depending on the Security Class of the device.

The candidate is evaluated using the data held in the Verify Management Key Record. If the candidate verifies, then K_m has been found. Otherwise, the value is a Management Key Precursor.

If a Management Key Precursor was found, the device uses this to calculate the Management Key (K_m) using a forward hash chain computation of *S* iterations starting from the precursor and repeating until the verified K_m has been found. That is the set of Management Key Precursors $\{K^{-S}_m, K^{-S+1}_{m,\ldots} K^{0}_m\}$ where $K_m^{\ 0}$ is also known as K_m. The value *S* is the Security Class of the device.

Only seven security classes are defined, so if the precursor computation cannot find a valid Management Key after seven attempts (the first candidate and six candidates from hash computations), then either the MKB is not valid (possibly altered) or the computation is in error.

The Management Key is used in various security calculations done for cluster management, and for calculating the Binding ID used to bind content to the cluster.

5.1.5 Recording Key Records

The recording keys for the device are produced from the MKB. For each Recording Key Record in the MKB, the n^{th} entry holds the data used to calculate any of the keys in the set. This data is combined with each Media Key Precursor the device has to calculate the Recording Key (K_r^{-S}) at the device's security class, S, and the Recording Keys for the lower security classes below S. For the first Recording Key Record, this produces the set of keys $\{ K^{-S}_{r,} K^{-S+1}{}_{r,..,} K^{0}_{r} \}_1$.

The MKB holds multiple Recording Key Records. The device shall in turn process each record to produce a Recording Key set for the device for that record: $\{ K^{-S}_{r,} K^{-S+1}{}_{r,..,} K^{0}_{r} \}_i.\}_j$.

When all the Recording Key Records have been processed, the device will have a set of sets of Recording Keys beginning at the device's Security Class *S* and all security classes below it. Thus, if there are *J* Recording Key Records in the MKB, the device would produce the set of sets:

$$\{ \{ K^{-S}_{r,} K^{-S+1}{}_{r,..,} K^{0}_{r} \}_1,\{ K^{-S}_{r,} K^{-S+1}{}_{r,..,} K^{0}_{r} \}_2,\ldots,\{ K^{-S}_{r,} K^{-S+1}{}_{r,..,} K^{0}_{r} \}_J \}$$

The recording key set is used by the device when it has to bind content to the cluster it belongs to. The recording key set is used to encrypt the Title Keys held in the Title Key Block for a bound content item.

Additionally, the Recording Key Data found in each record is used to produce a matching Recording Key Label, so the device will also produce the set of sets of Recording Key Labels:

$$\{ \{ L^{-S}_{r,} L^{-S+1}{}_{r,..,} L^{0}_{r} \}_1,\{ L^{-S}_{r,} L^{-S+1}{}_{r,..,} L^{0}_{r} \}_2,\ldots,\{ L^{-S}_{r,} L^{-S+1}{}_{r,..,} L^{0}_{r} \}_J \}$$

These labels are used to select which of the encrypted Title Keys in the Title Key Block of an item of content a device can decrypt.

5.1.6 Results

At the completion of processing the MKB, the device holds the following values:

1. Management Key K_m
2. Management Key Variant: K_{mv}
3. Management Key Precursors: $\{K^{-S}_{m}, K^{-S+1}{}_{m,\ldots} K^{-1}_{m}\}$
4. Security Class: *S*
5. Recording Key set: $\{ \{ K^{-S}_{r,} K^{-S+1}{}_{r,..,} K^{0}_{r} \}_1,\{ K^{-S}_{r,} K^{-S+1}{}_{r,..,} K^{0}_{r} \}_2,\ldots,\{ K^{-S}_{r,} K^{-S+1}{}_{r,..,} K^{0}_{r} \}_J \}$
6. Recording Key Label set: $\{ \{ L^{-S}_{r,} L^{-S+1}{}_{r,..,} L^{0}_{r} \}_1,\{ L^{-S}_{r,} L^{-S+1}{}_{r,..,} L^{0}_{r} \}_2,\ldots,\{ L^{-S}_{r,} L^{-S+1}{}_{r,..,} L^{0}_{r} \}_J \}$

5.2 Security Class

The HANA Content Protection architecture allows for devices to be assigned to one of 7 Security Classes. The security level of a device determines which Management Key Precursors and Recording Keys the device can calculate from a MKB.

Security levels are hierarchical so a device at a higher level can also calculate keys for the levels below it. Thus every device is able calculate the base security level 0 values which means that every device is able to calculate the Management Key (K_m) for the MKB.

The role of Security Classes is to enable restriction of bound content so that only devices at a particular level and above can access it.

The Security Class of a device is set by the Licensing Authority which produced the Device Keys and MKB.

A device determines what Security Class it is assigned to by calculating a candidate Management Key from the Reverse Management Key Record Data and using the data in the Verify Management Key Record from the MKB to test if the candidate validates as the Management Key. If it does not, then the device uses the candidate in a one way hash function to produce a new candidate. The new candidate is evaluated the same way. The number of times the one way hash function is performed to find the Management Key is the Security Class, 0 through 6, of the device.

If a device has not found a valid Management Key after six applications of the hash function, then either the MKB is not valid (possibly altered), or the computation is in error.

5.3 Licensed Device Secrets

The Licensing Authority (LA) provides a set of secrets to devices which allow a device to process an MKB.

These values must be protected as Highly Confidential as set out in the License Agreement between the manufacturer and the LA.

5.3.1 Device Key Set (K_d)

Each compliant device is given a set of secret 128 bit Device Keys each with an associated *Path Number*. The Device Keys and their associated Path Numbers are used to navigate the MKB and extract the secrets contained in it.

Device Keys may either be unique per device, or used commonly by multiple devices. The license agreement describes details and requirements associated with these two alternatives.

The set of Device Keys is not the complete set of keys used to encrypt the data in the MKB. The device can use its Device Keys and the data contained in the MKB to derive any of the remaining keys needed to process the MKB. This is detailed in a later section.

5.3.2 Device Number

Each device is licensed a 31-bit number called the Device Number *d*. The Device Number associates the device with a device node, a leaf in the NNL tree, and is used to determine which subset in the MKB applies to the device.

5.4 MKB Processing Detail

5.4.1 Subset-Difference Record

Each Device Key assigned to a device is associated with a subset of the devices (leaves) in the NNL tree. The subset is defined by a *subset-difference*, the leaves of one subtree in the NNL tree, designated by its root node, *u*, minus the leaves of another subtree, designated by its root node, *v*, where *v* is a descendant of *u*. The leaf in the NNL tree that the Device Number associates with the device is a descendant of the *u* node of the subset-difference associated with each of the Device Keys assigned to the device, but not a descendant of the *v* node. The device is enabled in each of these subset-differences.

From each of the Device Keys assigned to it, the device can calculate the keys associated with subset-differences that have the same *u* node as the subset-difference associated with the device key, and a *v* node descended from the *v* node of the subset-difference associated with the device key. The device cannot calculate keys associated with subsets in which the device is disabled, that is, its leaf node is a descendant of the subset-difference's *v* node, or its leaf node is not a descendant of the subset-difference's *u* node.

Each subset-difference is designated by a *path number*, "*u*" bit mask, m_u, and, "*v*" bit mask, m_v.

.The *path number* denotes the *v* node in the NNL tree associated with the subset-difference. This path number defines a sequence of branches from the root to that node in the tree as follows: starting with the

most significant bit, a '0' value indicates the path takes the 'left' branch of the tree and a '1' value indicates the path takes the 'right' side.

The masks are always a single sequence of 1-bits followed by a single sequence of 0-bits. The bit masks indicate "don't care" bits in the path number; if a bit is zero, that corresponding bit in the *uv* number is "don't care"; i.e., the path ends at this point. The farther from the root a node in the tree is, the shorter the sequence of 0-bits in the mask associated with that node. The m_u and m_v masks represent the depth of the respective nodes, *u* and *v*, from the root of the tree. The m_u mask always has more 0 bits than the m_v mask, because the node *u* is always closer to the root than the node *v*.

For conciseness, in the MKB, the path number and the "*v*" mask are encoded in a single 32-bit number, referred to as the *uv* number. The "*v*" mask is given by the lowest order 1-bit in the *uv* number. That bit, and all lower order 0-bits in the *uv* number, are zero bits in the "*v*" mask. All higher order bits in the "*v*" mask are ones. The following C code fragment illustrates one way to calculate the "*v*" mask from the *uv* value:

```
long v_mask = 0xFFFFFFFF;
while ((uv & ~v_mask) == 0) v_mask <<= 1;
```

The "*u*" mask is given separately as the number of low order zeros in the mask. The following C code fragment illustrates one way to calculate the "*u*" mask from the encoded value:

```
long u_mask = 0xFFFFFFFF;
u_mask <<= uMaskEncoded;
```

There is a distinction between *device numbers* and *device node numbers;* the latter is used in the key order format from the Licensing Agency. Device node numbers are device numbers which include the encoded mask. Since device numbers always correspond to leaves in the tree, the device *node* numbers always have their low-order bit on, and their mask is always $FFFFFFFE_{16}$. In other words, the device node number is the device number shifted left by 1, with the low-order bit set to 1

5.4.2 Calculation of Subsidiary Device Keys and Processing Keys

For the purpose of processing an MKB to calculate K_m, Device Keys are used to calculate subsidiary Device Keys and Processing Keys using the AES-G3 function depicted in **Error! Reference source not found.**.

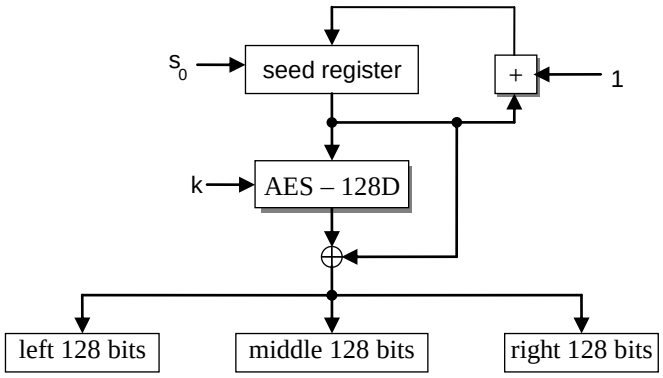


Figure 5-1 – Triple AES Generator (AES-G3), generating device keys and subsidiary keys

A 128-bit input Device Key (which may be a subsidiary Device Key) is denoted 'k' in this diagram. This loop is executed three times to produce 384 output bits, incrementing the seed register by one each time.

The output of AES_128D is XORed with the seed register's output at each step. For each AES-G3 calculation, the seed register is initialized by the 128-bit value S_0

The 384 output bits are interpreted as follows:

1. The first 128 bits is the subsidiary Device Key for the left child of the current node, or it is ignored if the Device Key 'k' is a leaf Device Key, AES_128D(k, s_0) ⊕ s_0.
2. The second 128 bits is the Processing Key, AES_128D(k, s_0+1) ⊕ (s_0+1).
3. The third 128 bits is the subsidiary Device Key for the right child of the current node, or it is ignored if the Device Key 'k' is a leaf Device Key, AES_128D(k, s_0+2) ⊕ (s_0+2).
4. By using the AES-G3 function in this way, the device calculates all subsidiary Device Keys that it needs from the few Device Keys that it stores at manufacturing time.

5.4.3 Calculation of Management Key Variant

For purposes of calculating the Management Key Variant, the Management Key Block includes two major parts: the subset-difference identification part, and the key data part. For each subset-difference included in the identification part, there are 16 bytes of key data in the key data part. The key data is one-for-one with the identified subset-differences. For example, the 23rd subset-difference identified in the Explicit Subset-Difference Record is associated with the 23rd section of the Management Key Variant Data field; that is, it begins at offset (23-1)*16 from the start of the Management Key Variant Data field of the Management Key Variant Data Record.

Subset-differences are encoded as *uv* numbers and two masks, a "*u*" mask denoted m_u and a "*v*" mask denoted m_v. A subset-difference applies to a device if the *u* node is on a path from the device's node to the root of the tree, but the *v* node is not. This is simple to calculate using the *uv* number, the appropriate mask, and the device node number *d*. By definition, a device "*d"* is on a path to a "*uv"* number with mask "*m"* if and only if:

(*d* & *m*) == (*uv* & *m*)

Thus, a subset-difference applies if and only if:

((*d* & m_u) == (*uv* & m_u)) and ((*d* & m_v) != (*uv* & m_v))

The first part of the "and" statement tests that the device's node is in the subset, i.e. the device's node is in the sub-tree rooted in *u*. The second part of the 'and' statement tests that the device's node is not in the sub-tree rooted in *v*. Hence, the full statement tests if the device's node is in the subset difference "*u* minus *v".* This subset difference *uv* contains the nodes in the sub-tree rooted at *u* that *do not* belong to the sub-tree rooted at *v*.

The device searches through the Explicit Subset-Difference Record fields, looking at the identified subset-differences, until it finds the one that applies to it. At that point the device either has the Device Key, or is able to derive the subsidiary Device Key, associated with that subset-difference. It finds the appropriate stored Device Key as follows: assuming the Explicit Subset-Difference Record value is *uv*, m_u, and m_v, and the stored Device Key has *uv*', m'_u, and m'_v, the appropriate Device Key is the one that meets the following condition:

(m_u == m'_u) and (((*uv* & m'_v) == (*uv*' & m'_v))

If m'_v equals m_v, the starting Device Key is the final Device Key, and is used directly to derive the Processing Key, as described above. Usually, however, the starting Device Key's node is further up in the tree, and the actual Device Key will have to be derived. The device does that as follows:

1. *Initialization*. *m* = the stored *v* mask m'_v. D = the starting Device Key.

Use AES-G3 on D, as described above, to determine a left subsidiary Device Key, a Processing Key, and a right subsidiary Device Key.

2. Look at the most significant zero bit in *m*. If the corresponding bit in the incoming *uv* number is 0, D = left subsidiary Device Key from step 2. Otherwise, D = right subsidiary Device Key from step 2.

3. *Iteration*. Arithmetic shift *m* right one bit. If it does not equal the incoming *v* mask m_v, repeat starting at step 2.

Once the device has the correct Device Key D, it calculates a Processing Key K using AES-G3 as described above. Using that Processing Key K and the appropriate 16 bytes of encrypted key data C, the device calculates the 128-bit Management Key Variant K_{mv} as follows:

$$\mathbf{K_{mv} = AES\_128D(K, C) \oplus (00000000000000000000000000_{16} \,||\, \mathit{uv}) \oplus integrity\text{-}hash}$$

The appropriate encrypted key data C is found in the Management Key Variant Data Record in the Management Key Block. The *integrity hash* is the AES_H hash of *Type and Version* Record and the *Verify Management Key* Record of the MKB.

A device may discover, while processing the Management Key Block, that none of the subset-differences identified in the block apply to it. In that case, the device shall conclude that it is revoked. Device behavior in this situation is implementation defined. As an example, a device could exhibit a special diagnostic code, as information to a service technician.

5.4.4 Management Key Block Format

A Management Key Block is formatted as a sequence of contiguous records. Each record begins with a one-byte Record Type field, followed by a three-byte Record Length field. The Record Type field value indicates the type of the record, and the Record Length field value indicates the number of bytes in the Record, including the Record Type and the Record Length fields themselves. Record lengths are always multiples of 4 bytes.

Using its Device Keys, a device processes the records of the MKB one-by-one, in order, from first to last. The device shall not make any assumptions about the length of records, and shall instead use the Record Length field value to go from one record to the next. If a device encounters a record with a Record Type field value it does not recognize, that is not an error; it shall ignore that Record and skip to the next. Likewise, if a Record Length indicates a record is longer than the device expects, that is also not an error; it shall ignore the additional record data.

The following subsections describe the currently defined record types, and how a device processes each. All multi-byte integers, including the length field, are "Big Endian"; in other words, the most significant byte comes first in the record.

If an MKB contains duplicate or unexpectedly missing records, or the MKB is otherwise improperly formatted, the device behavior shall be manufacturer specific.

5.4.4.1 Example MKB Record Ordering

Figure 3-3 shows an example MKB. It must be noted that other record orders are possible. Notice that for every *"uv"* related field there is a Management Key Data field and for the record types shown in the left, there is only *one* record.

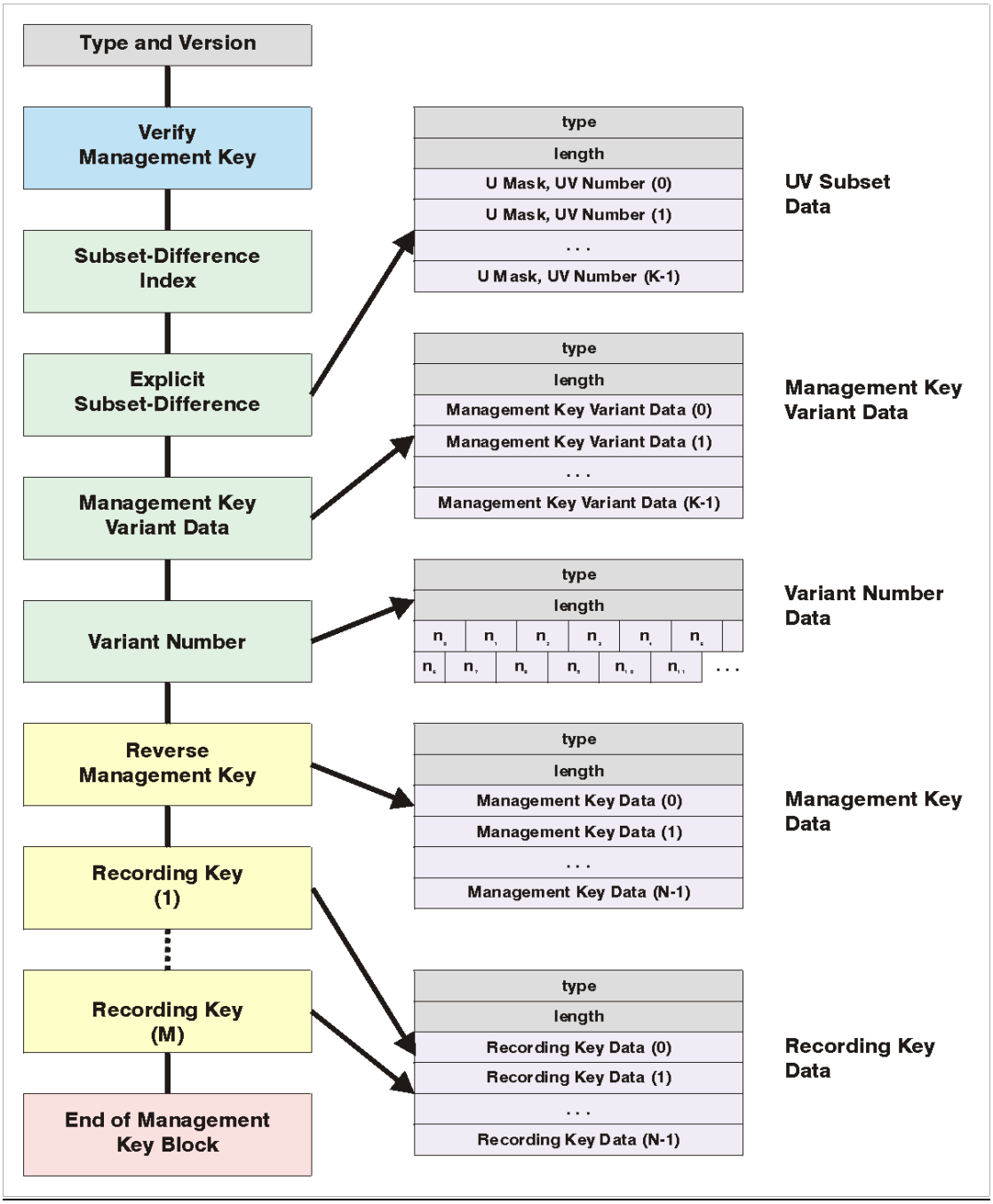


Figure 5-2 Example of a Management Key Block Showing a Valid Order of Records

5.4.4.2 Type and Version Record

Table 5-1 – *Type and Version* Record Format

Byte \ Bit	7	6	5	4	3	2	1	0
0	Record Type: 10_{16}							
1	Record Length: $00000C_{16}$							
2								
3								
4	MKBType: 00201003_{16}							
5								
6								
7								
8	Version Number							
9								
10								
11								

A properly formatted Management Key Block shall have exactly one *Type and Version* Record as its first record. Recording devices shall use the Version Number in this record to determine if a new Management Key Block is, in fact, more recent than the Management Key Block that is currently in use. The Version Number is a 32-bit unsigned integer. Each time the License Authority changes the revocation, it increments the version number and inserts the new value in subsequent Management Key Blocks. Thus, larger values indicate more recent Management Key Blocks. The Version Numbers begin at 1; 0 is a special value used for test Management Key Blocks.

5.4.4.3 Verify Management Key Record

Table 5-2 – *Verify Management Key* Record Format

Byte \ Bit	7	6	5	4	3	2	1	0
0	Record Type: 81_{16}							
1	Record Length: 000014_{16}							
2								
3								
4	Verification Data (D_v)							
…								
19								

A properly formatted MKB shall have exactly one *Verify Management Key* Record. It shall precede the *Explicit Subset-Difference* Record, the *Subset-Difference Index* Record, the *Management Key Variant Data* Record, the *Variant Number* Record, and the *Reverse Management Key Data* Record, although it may not immediately precede them. Bytes 4 through 19 of the Record contain the ciphertext value

D_v = AES_128E (K_m, $0123456789ABCDEF_{16}$ || $XXXXXXXXXXXXXXXX_{16}$)

Where:

- $XXXXXXXXXXXXXXXX_{16}$ is an arbitrary 8-byte value
- K_m is the correct final Management Key value.

The presence of the *Verify Management Key* Record in an MKB is mandatory. The device may use the *Verify Management Key* Record to verify the correctness of a given MKB, or of its processing of it. The device shall verify the correctness of the MKB by observing the following condition:

$$[AES\_128D(K_m, D_v)]_{msb\_64} == 0123456789ABCDEF_{16}$$

Where:

- K_m is the Management Key value.

A device in a higher security class than the base security class will calculate a *Management Key Precursor* K^{-S}_m instead of a Management Key. The 'S' in exponent indicates the particular security class. The device calculates lower security class Management Key precursors using the following one-way function:

$$K^{-(S-1)}_m = AES\_G(K^{-S}_m, KCD)$$

$$\text{i.e. } K^{-5}_m = AES\_G(K^{-6}, KCD)$$

Where:

- KCD is a confidential constant provided by the HANA Licensing Authority
- K^0_m is the management key, K_m

5.4.4.4 Explicit Subset-Difference Record

Table 5-3 – *Explicit Subset-Difference* Record Format

Bit / Byte	7	6	5	4	3	2	1	0
0	Record Type: 04_{16}							
1	Record Length							
2								
3								
4	U Mask (0)							
5	UV Number (0)							
…								
8								
9	U Mask (1)							
10	UV Number (1)							
...								
13								
14	. . .							
Length-1								

In this record, each subset-difference is encoded with 5 bytes. The mask for *u* is given by the first byte. That byte is treated as a number, the number of low-order 0-bits in the mask. For example, the value 01_{16} denotes a mask of $FFFFFFFE_{16}$; value $0A_{16}$ denotes a mask of $FFFFFC00_{16}$.

The last 4 bytes are the *uv* number, most significant byte first.

If when searching the list of subset-differences a device encounters a *u* mask value whose high-order two bits are non-zero, without first finding an applicable subset, it may conclude it is revoked. In other words, if the *u* mask is not of the form $00xxxxxx_2$, this marks the end of the list. The device's action in this case is manufacturer-specific.

The length of this record is always a multiple of 4 bytes. Thus, there may be unused bytes at the end of the record.

5.4.4.5 Subset-Difference Index Record

Table 5-4 – *Subset-Difference Index* Record Format

Bit / Byte	7	6	5	4	3	2	1	0
0	Record Type: 07_{16}							
1	Record Length							
2								
3								
4	Span (number of devices)							
…								
7								
8	Offset 0							
9								
10								
11	Offset 1							
12								
13								
14	Offsets 2 – Offset N							
…								
Length-1								

This is a speed-up record which may be ignored by devices not wishing to take advantage of it. It is a lookup table which allows devices to quickly find their subset-difference in the *Explicit Subset-Difference* Record, without processing the entire record. This *Subset-Difference Index* Record is always present, and always precedes the *Explicit Subset-Difference* Record in the MKB, although it does not necessarily immediately precede it. Furthermore, the *Subset-Difference Index* Record is guaranteed to be within the first one megabyte of the Management Key Block. For the purpose of designing for performance, a one megabyte buffer is sufficient to process the MKB; however, it is the manufacturer's choice how large a buffer is devoted to that purpose. (Informatively, it is always possible to treat the Management Key Block as a stream using a relatively small buffer.) Nonetheless, devices shall always be capable of processing Management Key Blocks exceeding one megabyte in size.

This record contains a "span", the number of devices per index offset, and a number of 3-byte offsets. These offsets refer to the byte offset within the following *Explicit Subset-Difference* Record, with 0 being the start of the record. Devices whose device number is between 0 and span-1 shall begin processing the *Explicit Subset-Difference* Record at Offset 0. Devices whose number is between span and 2*span-1, shall begin processing the *Explicit Subset-Difference* Record at Offset 1, and so on. Equivalently, if a device's number is *d*, it finds its offset within the *Explicit Subset-Difference* Record at offset 3**d*/span + 8 in this record.

Note that a device's number *d* is its node number shifted right by 1, because the low-order bit of the node number is always 1 to denote a leaf node.

The length of this record is always a multiple of 4 bytes. Thus, there may be unused bytes at the end of the record.

5.4.4.6 Management Key Variant Data Record

Table 5-5 – *Management Key Variant Data* Record Format

Bit / Byte	7	6	5	4	3	2	1	0
0	Record Type: $0C_{16}$							
1	Record Length							
2								
3								
4	Management Key Variant Data (0)							
…								
19								
20	Management Key Variant Data (1)							
...								
35								
36	. . .							
.								
.								
.								
Length-1								

This record gives the associated encrypted Management Key Variant Data for the subset-differences identified in the *Explicit Subset-Difference* Record. Each subset-difference has its associated 16 bytes in this record, in the same order it is encountered in the *Explicit Subset-Difference* Record. This 16-byte values the cipher text value C in the Management Key Variant calculation in Section 5.4.2.

The *Explicit Subset-Difference* Record always precedes this record, although it may not immediately precede it.

The length of this record is always a multiple of 4 bytes.

Notice that adding a new cover sub-tree to the MKB or new encryption requires 21 bytes: 5 bytes for its "*uv*" data and 16 bytes for the Management Key Variant Data. On average there are 1.28 encryptions per revocation.

5.4.4.7 Variant Number Record

Table 5-6 – *Variant Number* Record Format

Bit / Byte	7	6	5	4	3	2	1	0
0	Record Type: $0D_{16}$							
1	Record Length							
2								

3	
4	Nonce
…	
19	
20	Variant Number Data
...	
Length-1	

This record gives the associated encrypted variant number data for the subset-differences identified in the *Explicit Subset-Difference* Record. Each subset-difference has its associated 10 bits in this record, in the same order it is encountered in the *Explicit Subset-Difference* Record. These 10-bit fields are tightly packed in the "Variant Number Data" field.

Using the "Nonce" field in this record and the Processing Key (K_p) associated with the subset-difference, the Class II Device calculates a 10-bit key value as follows:

$$\mathbf{K_{vn} = [AES\_G(K_p, Nonce)]_{msb10}}$$

It then XORs the K_{vn} value with the appropriate 10 bits in the "Variant Number Data" field to obtain the variant number, from 0 to 1023. The Variant Number is used to process the *Reverse Management Key* Record and the *Recording Key* Records, to identify the appropriate content variations as described in the format-specific books of this specification.

The *Explicit Subset-Difference* Record always precedes this record, although it may not immediately precede it.

The length of this record is always a multiple of 4 bytes. If necessary, it is padded with 0 bits.

5.4.4.8 Reverse Management Key Record

Table 5-7 – *Reverse Management Key* Record Format

Bit / Byte	7	6	5	4	3	2	1	0
0	Record Type: $0E_{16}$							
1	Record Length							
2								
3								
4	Management Key Data (0)							
…								
19								
20	Management Key Data (1)							
...								
35								
…	…							
16*N+4	Management Key Data (N)							
…								

16*N+19	

This record allows a device to decrypt the Management Key using a Management Key Variant. A device will calculate a Management Key Variant from the *Management Key Variant Data* Record and will determine which variant number (0:1023) it is using the *Variant Number* Record. It uses its Variant Number to index into this record; if its Variant Number is *n* (0 to 1023), its index is 4 + 16n bytes from the start of the record.

Denoting the 16 bytes at that offset as C, the device calculates the Management Key as follows:

$$K_m = AES\_128D(K_{mv}, C) \oplus (00000000000000000000000000000000_{16} \,||\, 000000_2 \,||\, n)$$

Where:

- K_{mv} is the Management Key Variant it has previously calculated
- *n* is the Variant Number.

For some devices this calculation may produce a Management Key Precursor instead of a Management Key.

The *Management Key Variant Data* Record and the *Variant Number* Record always precede this record, although they may not immediately precede it.

The length of this record depends on the number (N) of Management Key Variants used in the MKB, and is always a multiple of 4 bytes.

5.4.4.9 Recording Key Record

Table 5-8 – Recording Key Record Format

Byte \ Bit	7	6	5	4	3	2	1	0
0	Record Type: $B0_{16}$							
1	Record Length							
2								
3								
4	Nonce							
…								
19								
20	Recording Key Data (0)							
...								
28								
…	…							
9*N+20	Recording Key Data (N)							
…								
9*N+28								

This record allows a device to decrypt the Recording Key from a Management Key Variant. A device will calculate a Management Key Variant from the *Management Key Variant Data* Record and will determine which variant number (0:1023) it is using the *Variant Number* Record (section 5.4.4.7). It uses its Variant

Number to index into this record; if its Variant Number is *n* (0 to 1023), its index is 20 + 9n bytes from the start of the record.

First, the device calculates its nonced Management Key Variant as follows:

$$\mathbf{K_{mvn} = AES\_G(K_{mv}, nonce \oplus (00000000000000000000000000000000_{16} \,||\, 000000_2 \,||\, n)}$$

Where:

- K_{mv} is the device's Management Key Variant it has previously calculated
- *n* is the Variant Number.

Denoting the 9 bytes at that offset as C, the device calculates the 9-byte recording key data (D) as follows:

$$\mathbf{D = [K_{mv}]_{msb72} \oplus C}$$

Finally, the device calculates a Recording Key as follows:

$$\mathbf{K_r = AES\_G(K_m, D \,||\, r_0)}$$

Where

- r_0 is a confidential constant provided by the HANA Licensing Authority
- In addition, the high-order 13 bits of D become the *label* of the Recording Key. This label is used in the Title Key Block of each piece of content.

Note that Recording Keys also come in precursor versions for higher security classes. For example, the Recording Key for class *i* would be calculated as follows:

$$\mathbf{K^{-i}_r = AES\_G(K^{-i}_m, D \,||\, r_0)}$$

There can be any number of *Recording Key* Records in an MKB. Each *Recording Key* Record allows a compliant device to calculate one Recording Key, so a single MKB will give a device multiple Recording Keys. The *Management Key Variant Data* Record and the *Variant Number* Record always precede these records, although they may not immediately precede them.

The length of this record depends on the number (N) of Management Key Variants used in the MKB, and is always padded to make it a multiple of 4 bytes.

5.4.4.10 End of Management Key Block Record

Table 5-9 – *End of Management Key Block* Record Format

Bit / Byte	7	6	5	4	3	2	1	0
0	Record Type: 02_{16}							
1	Record Length							
2								
3								
4	Signature Data							
…								
Length-1								

A properly formatted MKB shall contain an *End of Management Key Block* Record. When a device encounters this Record it stops processing the MKB, using whatever K_m value it has calculated up to that point as the final K_m for that MKB (pending possible checks for correctness of the key, as described previously).

The length of this record is always a multiple of 4 bytes.

6 Content Cluster Devices

Building upon the values used in the previous section to process the MKB and the keys extracted from the MKB using them, the following section details the elements used in devices to build the content cluster, bind content to it, and to manage its state.

6.1 Cluster ID (C_{id})

The Cluster ID a non-secret (public) 128 bit identifier which is used to identify the Content Cluster. It is initially set to a random value when the device's Authorization Table is initialized at time of manufacture.

When a device resigns from a cluster it generates a new random value using the random number generator defined in section 4.4, and updates the cluster ID in the initial AT retained by the device for use during a device reset operation.

6.2 Binding ID (ID_b)

The Binding Identifier is a secret 128-bit identifier that is shared among devices in the Content Cluster. This value MUST be protected in accordance with compliance and robustness rules associated with the license.

This value is randomly generated by a device using the random number generator defined in section 4.4 of this document.

6.3 Device ID (ID_p)

The Device ID is a non-secret (public) 128 bit identifier issued by the HANA Licensing Authority to each Content Cluster device.

6.4 MKB

The MKB is a non-secret file issued by the HANA Licensing Authority.

The current MKB is the MKB shared by all devices in the Content Cluster.

Additionally, devices must retain the initial MKB issued by the Licensing Authority to it at the time of its manufacture. This is used in performing a reset of the device back to the manufacturers default state.

6.5 Authorization Table (AT)

The list of authorized members in the Content Cluster, both current and historically, is held in the Authorization Table (AT). Each device in the Content Cluster has a copy of the AT which is used to hold the current state of the Content Cluster.

Devices additionally must retain the initial AT which they were issued when manufactured. This is used when the device is reset back to its manufacturer's default state.

6.6 AT Hash (AT_{hash})

The AT Hash is a 128 bit value calculated by hashing all the lines of the AT except the final line which contains the AT Hash. The specific processing of the AT to produce the AT Hash is detailed in the chapter on the Authorization Table.

6.7 Binding Key (K_b)

The binding key for the cluster is calculated using the Binding ID (ID_b) of the cluster, the Management Key (K_m), and the hash of the Authorization Table (AT_{hash}), as follows:

$$K_b = AES\_G\ (K_m, ID_b \oplus AT_{hash})$$

6.8 Device Peer Key (K_p)

The device peer key (K_p) is calculated using the device's public (peer) ID (ID_p) and Management Key (K_m) as follows:

$K_p = AES\_G(K_m, ID_p)$

where K_m is the base management key of the receiving device's MKB.

This paragraph is informative. A device requesting authorization to enter the cluster will need to request the receiving device's current MKB in order to calculate K_m. The device peer key is used to calculate the Message Authentication Code (MAC) on all required cluster protocol messages.

6.9 ASCCT Nonce ($ASCCT_{nonce}$)

The ASCCT Nonce is a random or pseudo-random number created using the random number gernation in section 4.4..

6.10 MKB and AT Files

Each cluster is defined by two files: the MKB and the authorization table (AT). Because of the built-in authentication in each file, they can be normal files with no special security, however devices must be able to validate that each file has not been tampered with.

6.11 Device State

The behavior of a device's interaction with other clusters is controlled via its device states. The following is a high-level description of the expected behavior of the devices in each of the possible states.

6.11.1 Join Enabled

The Join state controls the behavior of the device when it discovers a cluster to which it is not a member, and how it responds to requests from devices to join its cluster.

1. A device in the join state shall attempt to get authorized to join any new cluster which it encounters on a network.

   This means that if a device in **join** state receives an *imhere* message from a cluster it is not part of, it should send a corresponding *authorizeme* message in response. The exchange will then follow the Cluster Merge sequence.

2. A device is the join state the device shall process *authorizeme* messages from devices, and shall approve such cluster membership requests if appropriate

6.11.2 Join Disabled

This mode should be selected when a device is connected to a network with other clusters that it SHOULD not merge with– such as public wireless network in an airport lounge.

1. A device must not attempt to join any new cluster nor authorize any other devices into the cluster it belongs to.
2. A device shall attempt to join any new cluster that is advertised by another device which is already part of their cluster's AT.

   In other words, devices should ignore *imhere* requests from other clusters unless the sending device is in their current AT.

3. A device will accept and propagate new AT and MKB files for the clusterit belongs to.
4. A device will ignore *authorizeme* requests of type other than "external".

6.11.3 Selecting state

Devices shall support having their state selectable via the HANA UI. In addition to this, it is permissible for a device to allow device state selection via other means such as switches, buttons, or telnet, as determined by the manufacturer.

Devices may also change their states automatically to manage device behavior in special situations, such as mobile devices being connected in the home, and not when they are mobile. For example, a mobile device could be in the Join Enabled state when physically docked or connected by wireless to the home network, but switch to Join Disabled when undocked and out of wireless range of the home.

6.12 On-Device Storage

In addition to the device key set and licensed secret constants issued by the Licensing Authority, and device configuration, a device must store the MKB and AT files in use by the cluster, and the secret Binding ID for the cluster.,. Additionally, devices may hold content bound to the cluster.

6.12.1 Bound Content Storage

This specification does not place any specific requirements on the layout or media used to store content on a device. Bound content is sufficiently strongly protected that it may be freely accessed and moved by devices and users. This means that bound content may be stored onto any media which physically can hold it. There are no special requirements for storage media such as media serial numbers, secrets, or protected access modes.

6.12.2 MKB and AT Storage

Devices must store the cluster MKB and AT so that tampering with them is either prevented or they can be validated to detect tampering. The MKB and AT files can be validated by securely storing the AES_H hash of each file, and checking this against the files when used by the device.

There is no limit to the size of the MKB and AT files.

The MKB and AT files do not require additional protection from reading by users or devices. But they are protected against tampering. An acceptable solution if secure storage is not available isto store the AES hash of the current MKB and current AT in a secure location and use these values to verify the MKB and AT files used by the device..

The MKB and AT files may be held on either internal storage in the device or may be stored on network connected storage devices.

A device detecting that its copies of the MKB and/or AT have been corrupted, shall not proceed until correct versions are obtained. In other words, the device is not useable until the situation is corrected. How the correct versions are obtained is manufacturer-specific; for example, it may be possible to obtain intact versions from other devices in the cluster. Also, devices may keep any number of backup copies of the MKB and AT for this purpose.

6.12.3 Binding ID Storage

A device must retain the secret Binding ID for the cluster to which it belongs in secure storage. This value must be protected as a secret value.

7 Authorization Table (AT)

The Authorization Table is primarily used to maintain the list of devices authorized to share content on a cluster. It is maintained independently by all devices in the cluster and **SHALL** be updated and synchronized as part of the protocol as devices join or leave the cluster.

Devices only belong to one active cluster at a time. For purposes of performing the join cluster operation, devices must be capable of belonging to a second cluster; this is necessary so that clusters can merge

Since each device has its own copy of the MKB and AT, and because devices power-on and off at various times and portable devices are disconnected and reconnected periodically, it is possible for the cluster to momentarily to be out of sync in these files. In that case, the cluster management protocol works quickly to resolve the discrepancy. In fact, at its heart, the cluster management protocol is a protocol designed to securely eliminate discrepancies in these two files.

Although compliant devices quickly work to resolve discrepancies, discrepancies also may exist if the cluster is under attack. For example, if a circumvention device has joined the cluster, that device may be unable or unwilling to allow the MKB to be updated. Therefore, except in the case of backup/restore, devices shall not process content unless it is protected with the binding key from the device's own version of the MKB and AT.

The AT is comprises tags that follow standard well-formed XML syntax whose attribute data **SHALL** consists of UTF-8 characters only.

7.1 Format

7.1.1 Hex Encoding

If a value is said to be "hex encoded" then each binary octet is encoded as two hexadecimal digits ([0-9a-fA-F]) representing the octet code. For example, 16-bit integer 6704_{10} (whose binary representation is 1101000110000) would be hex encoded as "1A30".

Any schema element value or attribute value defined to be "hex encoded" **SHALL** be of the XML Schema built-in data type "hexBinary".

7.1.2 Base 64 Encoding

If a value is said to be "base64 encoded" then the entire binary stream is encoded using the Base64 Content-Transfer-Encoding defined IETF RFC 2045.

Any schema element value or attribute value defined to be "base 64 encoded" **SHALL** be of the XML Schema built-in data type "base64Binary".

7.1.3 Lexicographical Order

The term lexicographical order in the context of this document and its adaptations **SHALL** mean the ordering of named elements in dictionary order by letter sequence. The process of combining two or more sets of named elements using lexicographical ordering will, by definition, create a single well-ordered set.

This paragraph is informative. For the purpose of ordering XML element names and attributes (which **SHALL** never have spaces), lexicographical ordering would essentially mean sorting the names alphabetically to establish a specific ordering normative to all implementations.

7.1.4 MAC Calculation Syntax Constraints

To simplify the processing requirements to reach a wider range of consumer electronic and portable devices, the normal XML syntax is further constrained:

The normalized Authorization Table **SHALL** be valid against the AT schema.

All elements **SHALL** be self closing.

All element names **SHALL** be lowercase.

All attribute names **SHALL** be lowercase.

All element names and attribute names **SHALL** be formed from lowercase human readable ASCII, i.e., ASCII character ranges '0'-'9', 'a'-'z'.

All attribute-value pairs **SHALL** be in sequence order as declared in the official Authorization Table schema.

There **SHALL** be no white space characters in the Authorization Table other than:

- If the XML element has an attribute-value pair, a single space character **SHALL** separate the element tag name from the first attribute-value pair.
- If the XML element has multiple attribute-value pairs, a single space character **SHALL** separate each attribute-value pair within the element.

Note that there is no space character preceding the element name or following the last attribute-value pair or following the element closing characters '/>'.

The line feed characters '0D16', and '$0A_{16}$', (in that order) **SHALL** separate each XML element.

All binary data (e.g. IDs and MAC data) assigned to attribute values **SHALL** be encoded in lowercase readable hexadecimal notation. See section 7.1.1 Hex Encoding for details.

Syntax Example

The following is an example that shows a multi-line Authorization Table that follows the above outlined syntax rules:

```
<cluster id="hex value" …/>/u0D/u0A
<tag1 attribute1=value1 attribute2=value2 …/>/u0D/u0A
<tag2 attribute1=value1 attribute2=value2 …/>/u0D/u0A

…
<tagN attribute1=value1 attribute2=value2 …/>/u0D/u0A
<mac value="hex value"/>
```

where '/u0D' and '/u0A' represent the Unicode-8 characters for Carriage Return (CR) and Line Feed (LF) respectively.

7.1.5 Normalization Rules

The following rules **SHALL** also be applied when normalizing the AT:

- There **SHALL** be only one <cluster> element.
- There **SHALL** be only one <mac> element.
- The <cluster> **SHALL** be the first element to appear in the AT.
- The <mac> element **SHALL** be the last element to appear in the AT.
- All elements that may appear (occur) more than once in the Authorization Table **SHALL** have an 'id' attribute as the first attribute to appear after the element tag name. Note: this is for improved processing performance.
- All elements that appear (occur) with same tag name **SHALL** be lexically sorted by the hex encoded value of their "id" attribute" in increasing order.
- All <device> elements **SHALL** immediately follow the <cluster> element.
- All <service> elements **SHALL** immediately follow any <device> elements.
- All <mac> elements **SHALL** immediately follow any <service> elements.
- All attribute value pairs that follow the 'id' attribute value pair **SHALL** appear in lexicographical order by attribute name.

- All elements that follow the last <content> element **SHALL** appear in lexicographical order by element tag name.

A valid AT **SHALL** always have at least one <device> element or one <service> element

Devices **SHALL** preserve, but **MAY** ignore (not process) elements and attributes that are not part of this (common) specification (schema) which may have been placed in the AT by other ASCCT versions.

7.2 Device Element

Each device in the cluster has a device element in the AT. A device element has the following attributes: id, role, and state.

7.2.1 Attribute: id

This is the Device ID of the device as issued by the licensing authority.

7.2.2 Attribute: role

This is device role in the cluster.

7.2.2.1 role = "peer"

The device is a Content Device. Peer devices count against the cluster limit, depending on their *state* attribute.

7.2.2.2 role = "infrastructure"

The device is an Infrastructure Device. Infrastructure devices DO NOT count against the cluster limit.

7.2.3 Attribute: state

The following are valid states for devices in the authorization table

State	Description
authorized	The device is authorized as a compliant device on the current active cluster.
pending	A device has been marked for deletion in the AT but has not yet acknowledged its deletion by changing its state to delete in the AT.
delete	The device has been deleted from the cluster and **SHALL NOT** be permitted to rejoin the cluster automatically. Explicit user intervention is required to rejoin.

7.2.3.1 "authorized" state

The purpose of this state is to indicate the device is an authorized device on the cluster and is therefore trusted and authorized to share ASCCT protected content with other authorized devices on the same cluster.

Peer type Devices in the authorized state count SHALL toward the cluster limit.

7.2.3.2 "pending" state

The purpose of this device state is to account for devices that the cluster owner has deleted from the cluster, but the deleted device itself has not acknowledged the deletion by completing a successful *imhere* update with another device in the cluster that is still in the "authorized" state.

Devices in the AT with state "pending" **SHALL** count towards the cluster "limit". This treatment mitigates the number of times a person can create and sell such devices.

Devices **SHALL** never remove a device entry in the AT with the "pending" state.

Devices **SHALL** only be permitted to change an AT device entry with state "pending" to state "delete" after the device in "pending" state acknowledges it has been deleted by completing a successful *imhere* update.

7.2.3.3 "delete" state

The purpose of this device state is to indicate devices that have resigned from the cluster.

Devices **SHALL** never remove a device entry in the AT with the "delete" state.

Devices **SHALL** no longer share content or send or receive cluster protocol messages for devices with state "delete" in their authorization table.

Devices in the AT with state "delete" **SHALL NOT** count towards the cluster "limit".

7.2.3.4 State Transitions

The following state transitions are permitted:

A device can change its state in the AT in the following transitions:

1. Add itself to the AT with the state "authorized" as the result of a successful *authorizeme* message.
2. Change from "authorized" to "delete"
3. Change from "pending' to "delete"
4. Change itself from "delete" to "authorized" as the result of a successful *authorizeme* message used to rejoin the cluster.

A device may change the state of another device in the following transition:

1. Change from "authorized" to "pending"

7.3 Authorization Table Hash (AT_{hash})

The Authorization Table Hash is an element that is used to calculate the Binding Key (K_b).

AT_{hash} = AES_H (AT)

The hash of the AT is AES Hash of all the lines of the AT up to, but not including the final <mac> line.

Devices **SHALL** verify they are still an authorized device in the actual AT (and not deleted) before using the hash in any cryptographic operations such as using the binding key. If they are not authorized, they **SHALL** abort the operation as well as any higher level operation or procedure that would use the resultant AT_{hash}.

7.4 Sample Authorization Table

A simple normalized Authorization Table might appear as follows:

```
<cluster id="6237d46210c94990937506155b067540"/>
<device id="19901e191de14151a5d11506c115d1f2" role="peer" state="authorized"/>
<device id="289fa678be5113cf31889dc3ed45c1e9" role="peer" state="authorized"/>
<device id="1f49bc8e9f673ee7f890078fe14bc889" role="peer" state="delete"/>
<mac value="8b4bf1c643e5e337cdfa6ebb61dde2b8"/>
```

This is trivial example for illustration purposes. Real AT's MUST follow the schema in Section 13.

8 Cluster Management

This section contains the details of how clusters of devices are assembled and maintained. A device may belong to more than one cluster – however it will only play content which is bound the cluster(s) to which it is a member.

A cluster has a maximum limit of Content Devices which may belong to it. A cluster which exceeds this limit must either have Media devices removed from it via device resignation, or obtain authorization to grow from the HANA LA.

8.1.1 Device Discovery

Device discovery is detailed in the HANA Design Guideline document.

Upon discovery of a device the detecting device shall send an *imhere* message to the discovered device.

It should be noted that devices joining a network will also discover other devices that are already there. The means that the sending of *imhere* messages is actually done by both the devices that were already on the network, and the device which has just joined.

At this point the devices on the network are aware of the detected device, and the detected device is aware of the other cluster devices on the network. Following the exchange of the *imhere* messages, the newest MKB in the network will be adopted by all devices regardless of their cluster membership and join status.

8.1.2 Guest Devices

Devices are permitted to "visit" other HANA clusters. For example a mobile device taken to a hotel room which has a HANA network. Such a device should change its join state to DISABLED so that it does not join the local cluster. While visiting such a device can playback the bound content from its home cluster to rendering devices in the cluster it is visiting. This can be done by performing the HANA binding decoding and sending it via DTCP to a rendering device.

8.2 Cluster Management Use Cases

The use cases for cluster management detail how the control elements of the HANA secure cluster, the domain's Management Key Block (MKB), and Authorization Table (AT), are used. These control elements are identical on each device in a domain. In addition to the MKB and AT, each device in a HANA domain possesses a set of device keys, which are issued to it during manufacture. These keys remain on the device, and must be protected from disclosure or access.

The domain management scenarios focus on how the MKB, and AT are updated and shared.

The MKB used in a cluster is updated when a newer version of MKB is introduced to the cluster by a device with an updated MKB joining or reconnecting to the cluster, or when a device in the cluster imports an updated MKB from, for example, new media, a firmware update, or a download/push via a set top box.

The Authorization Table (AT) is changed whenever a device is added to or removed from the cluster. The AT essentially contains the list of all the devices which are, or have ever been, part of the cluster.

8.2.1 Device Reconnected to Network

When a device is reconnected, or powered up on a HANA network it will connect with the other devices in the cluster. The device will examine the MKB in use in the cluster and update itself if its version is older than the version in use. The cluster Authorization Table hash will be checked, and an updated AT obtained if needed. Finally, if the device contains content bound to the cluster, and the cluster ID or binding key has changed, the device will rebind all such content.

8.2.2 Management Key Block Update

When an updated MKB is introduced into the HANA Content Cluster, the first device to receive the updated replaces the MKB used by the domain with the new one. It the uses “imhere” messages to notify every other device in the domain that the MKB has been updated..

8.2.3 Get Authorization Table

When a device detects that another device in the cluster has a different authorization table hash, it shall obtain a copy of the other device’s AT and attempts to merge it with its own.

8.2.4 Periodic Device Heart Beats

Devices shall issue period heart beat messages to the devices in the AT of the cluster they belong to. The frequency of these messages shall be random, but at least once every two minutes to each powered-on device in the device’s cluster.

The device shall use *imhere* messages for delivery of this heart beat pulse.

8.2.5 Cluster Binding Key Updated

A device which has an updated Cluster Binding Key shall notify every other device in the cluster of the updated key via *imhere* messages.

8.2.6 Revoked Device

When a device is revoked by an updated MKB used in the domain, it is no longer capable of being a member of the Content Cluster. It will no longer be able to use the binding key to unbind content, or to validate cluster messages.

8.2.7 Cluster Reset

Devices shall have a cluster reset option which, when selected, shall revert the device to its original factory state.

8.3 Device limits

Only Content Devices count against the device limit for the cluster.

When a cluster reaches the device count limit, no additional content devices may be added until it needs to obtainreceives authorization in order from the Licensing Authorityadd.

The device limit is set by the Licensing Authority.

9 Cluster Messaging

The following are the messages used by devices for cluster management. Additional messages for content and content directory functions are defined in those sections of this document.

9.1 Cluster Messages

The Content Cluster is established and managed via a small set of messages which are exchanged between devices: *imhere, authorizeme, getAT, getMKB,* and *getDeviceProfile*.

9.1.1 imhere

Purpose:

There are actually three purposes to this message:

1. **Initiate Authorization**: A compliant device **SHALL** send this message to Cluster devices that are "discovered" on their logical network.
2. **Immediate Binding Update**: A compliant device **SHALL** send this message unsolicited to every device in its authorization table whenever its binding key has changed. This case most often occurs when devices alter the contents of their AT or they adopt a new MKB.
3. **Interval Binding Update**: A device **SHALL** send this message unsolicited to every device in its authorization table at irregular intervals (of approximately 2 minutes) and during certain device operations. This acts as a heart beat message.

Message Processing:

The receiving device **SHALL** perform these additional processing steps:

- If the MKB Hash in the message does not match the hash of its current MKB, the receiving device **SHALL** proceed to update its MKB according to the Update MKB operation. This update operation **SHALL** occur regardless of the receiving device's "join" state.
- If the sending device is an authorized member of the receiving device's cluster in its current AT, and the AT Hash in the message does not match the hash of its current AT, the receiving device **SHALL** proceed to update its AT according to the Update AT operation.
- If the sending device is not an authorized member of the receiving device's cluster and the receiving device has a device state of join "enabled", the receiving device **SHALL** request to join the cluster using an "authorizeme" message.
- A device **SHALL** perform a merge test as defined by the "should join" operation and **SHALL** only proceed to become authorized if all merge tests pass.

Input Parameters:

Element	Required	Description
`clusterid`	yes	The public ID of the sender's cluster.
`deviceid`	yes	The public ID of the sending device (ID_p).
`mkbhash`	yes	The cryptographic hash of the device's current MKB.
`athash`	yes	The cryptographic hash of the device's current AT.
`location`	yes	The current URI of the device for the purpose of sending messages to the device.

Success Response Data:

Element	Description

`accepted`	The message was received and processed. The receiving device will determine if it will send subsequent messages following returning this response.

Failure Response Data:

Element	Description
rejected: invalid	The message had missing or invalid elements
rejected: revoked	The receiving device is in a revoked state and cannot receive the message
rejected: refused	The receiving device is a state where it is not accepting messages

9.1.2 authorizeme

Purpose:

This message is sent by a device to a target device in order to request authorization to "join" the target device's cluster.

If the receiver of this message determines that the sending device is an authorized member of its current cluster (already in its AT), and the AT of the sending device is equivalent to the receivers AT (i.e. would result in the same AT_{hash} calculation), the receiver **SHALL** always respond with the "authorized" response Message.

Message Processing:

The receiving device **SHALL** perform these additional processing steps:

- If the receiving device is not in the 'authorized' state in its own AT table it **SHALL** respond 'rejected' reason code 'refused'.
- The receiving device **SHALL** validate that the 'mac' value on the 'authorizeme' message matches the MAC that the receiving device calculates against the entire received 'authorizeme' message. If the 'mac' value on the message does not validate, the receiving device **SHALL** respond rejected with reason code 'unverified'.
- The receiving device **SHALL** determine whether or not the sending device has an entry in its current AT.
- If the sending device has an entry in the receiving device's AT the receiving device **SHALL** respond according to the sending device's state attribute value as follows:
    - If the sending device entry has state 'authorized' the receiving device **SHALL** respond with the 'authorized' response type.
    - If the sending device entry has a state other than 'authorized' the receiving device **SHALL** respond 'rejected' with reason code 'refused'.
- If the sending device does not have an entry in the receiving device's AT, the receiving device **SHALL** continue with the following processing steps:
    - If the receiving device's join state is "disabled" the receiving device **SHALL** respond rejected with the 'refused' reason code.
    - If the receiving device's join state is "enabled" the receiving device addsSHALL add an entry for the sending device into its own AT with state "authorized" and with an initial "count" value. The device **SHALL** update its binding information with the updated AT and respond to the sending device with the "authorized" response message.
- If the receiving device encounters an error it **SHALL** respond rejected with the 'refused' reason code.

If the receiving device authorizes the sending device by adding an entry to its AT and sending its Binding ID inin an authorized response, its AT now is different from the sender's AT. Subsequent "imhere" processing should result in a merged AT.

Input Parameters:

Element	Required	Description
`clusterid`	yes	The public ID of the sender's cluster.
`deviceid`	yes	The public ID of the sending device (ID_p).
`location`	yes	The current URI (as defined by ASCCT schema) of the device for the purpose of sending messages to the device.
`mac`	yes	The Message Authentication Code (CMAC) for the data contained in this message. CMAC(K_p, D) Where K_p is the device peer key (K_p) and D is the concatenation of all element values in schema defined order.

Success Response Data:

Element	Description
`authorized:` AES-128E(K_p, ID_b)	This response is given when the device is authorized to join the cluster. The data following the "authorized:" string is the encryption of theBinding ID, (ID_b) with the device's peer key K_p in readable hexBinary format.

Failure Response Data:

Element	Description
Rejected: invalid	This response is given if there are missing parameters or a syntax error in the original message.
Rejected: stopped	The authorizing device is in the stop state.
Rejected: unverified	This response is given if the MAC in the message does not verify.
Rejected: limit	The cluster is already at the allowed number of devices

9.1.3 getMKB

Purpose:

This message is used by devices to request the MKB from another device (in binary base64 encoded form)**Input Parameters**:

Element	Required	Description
`clusterid`	yes	The public ID of the sender's cluster.
`deviceid`	yes	The public ID of the sending device (ID_p).

Success Response Data:

Element	Description
`MKB`	The base64 encoded representation of the MKB used by the receiving device

Failure Response Data:

Element	Description
rejected: invalid	The message had missing or invalid elements
rejected: refused	The receiving device is a state where it is not accepting messages or it has refused the request

9.1.4 getAT

Purpose:

This message is used by a device to request the normalized Authorization Table (AT) from another device.

Input Parameters:

Element	Required	Description
`clusterid`	yes	The public ID of the sender's cluster.
`deviceid`	yes	The public ID of the sending device (ID_p).

Success Response Data:

Element	Description
`AT`	The normalized XML representation of the receiving device's Authorization Table (AT).

Failure Response Data:

Element	Description
rejected: invalid	The message had missing or invalid elements
rejected: refused	The receiving device is a state where it is not accepting messages or it has refused the request

9.1.5 getDeviceProfile

Purpose:

This message is used by a device to request the device profile of another device. This can be used as a fast means to verify a device's class, capabilities and other information. The actual device profile data is intended to be extended by adaptations and manufacturers, by means of XML schema extension.

Input Parameters:

Element	Required	Description
`clusterid`	yes	The public ID of the sender's cluster.
`deviceid`	yes	The public ID of the sending device (ID_p).

Success Response Data:

Element	Description

`profile`	The XML representation of the device's profile as defined by ASCCT schema.

Failures Response Data:

Element	Description
rejected: invalid	The message had missing or invalid elements
rejected: refused	The receiving device is a state where it is not accepting messages or it has refused the request

9.2 Message Delivery

Devices exchange messages as XML encoded messages over the IP network, as defined in the HANA Design Guideline document.

Messages can be delivered in any order. The cluster will always converge on a common state amongst all devices on the network.

9.3 General Message Receive Processing

This section describes the general message processing requirements compliant ASCCT devices **SHALL** perform on all received cluster protocol messages defined in this book. Specific cluster protocol messages or additional messages defined in adaptation books may define additional processing requirements.

The receiving device **SHALL** perform the following processing steps when receiving any cluster protocol message in order:

1. If the message contains invalid XML, the XML does not adhere to schema, or the elements, attributes or values do not adhere to this specification, the receiving device **SHALL** respond with "rejected" with reason "invalid".
2. If the message contains valid XML, but the message type is not known or supported, the receiving device **SHALL** respond with "rejected" with reason "unsupported".
3. If any of the required parameters are missing, the receiving device **SHALL** respond with "rejected" with reason "invalid".
4. If the receiving device is currently in "stopped" state (i.e. not processing required cluster protocol messages due to a normal operation), the receiving device **SHALL** respond with "rejected" with reason "stopped".
5. If the receiving device is unable to process the message due to network or processor limitations, the receiving device **SHALL** respond with "rejected" with reason "busy".
6. If the sending device's entry in the Authorization Table (AT) of the receiving device has state "revoked", the receiving device **SHALL** respond with "rejected" with reason "revoked".
7. If the message is not a getMKB message and the sending device's entry in the Authorization Table (AT) of the receiving device has state "deleted", the receiving device **SHALL** respond with "rejected" with reason "refused".
8. Ifs where the receiving device encounters any other error that prevents it from responding to the message, the receiving device **SHALL** respond "rejected" with reason "refused".
9. If the received message has a "MAC" element, the receiving device **SHALL** verify it against the message data. If the MAC value passed with the message is incorrect, the receiving device **SHALL** respond with "rejected" with reason "unverified".

   This paragraph is informative. In order to verify the MAC on a message the receiving device must use the MKB of the device that sent the message. The cluster devices will over time be triggered to retrieve and converge on the most recent MKB available in the cluster, and hence will be able to

eventually calculate the correct MAC. During the convergence period messages between devices using different MKB files will fail until both devices are using the same MKB.

9.4 Cluster Device Operations

,the domain's Management Key Block (MKB), and Authorization Table (AT),. in accordance with compliance and robustness rules associated with the license. This section defines operations that devices use to carry out various cluster management operations defined in section 8.2 Cluster Management Use Cases.

9.4.1 Should Join Test

A device executing the "Should Join" test compares this operation candidate normalized AT from a device in another cluster against its own normalized AT and performs a series of tests to determine if the candidate AT is the dominant AT and if the resultant merge of the two ATs would result in a valid cluster.

The device **SHALL** pass all the following tests in order to return a success response to this operation. if any test fails it **SHALL** indicate a failure response:

1. **Limit Test:** The device **SHALL** determine if the merge of the two ATs would result in a cluster whose device limit was exceeded. If it determines that the merge would result in a cluster whose device limit is exceeded it **SHALL** fail the Join test.
    - A device **SHALL** test using the larger of the cluster device 'limit' attribute values from the two ATs.
2. **Dominance Test:** The device **SHALL** determine if the candidate AT represents a cluster which is dominant to its own cluster. If its own cluster is dominant it **SHALL** fail this join test.
    - If the device's current AT has a device limit greater than that of the candidate AT it **SHALL** fail the test (i.e. it is the dominant cluster and the other device should join its cluster).
    - If the number of device elements in the device's current AT (except for those devices in the 'delete' state) is greater than that of the candidate AT it **SHALL** fail this join test (i.e. it is the dominant cluster and the other device should join its cluster).
    - Otherwise, if both ATs contain the same number of device elements the device **SHALL** compare its public cluster ID to that of the candidate AT. If its ID value is less when lexicographically compared it should fail this join test (i.e. it is the dominant cluster by ASCCT definition).
3. **Element Merge**: The device **SHALL** test "merge" all AT elements in both ATs to determine if there are any errors. If any element merge errors exist it **SHALL** fail this join test.

9.4.2 Merge AT

This operation is typically initiated as a result of a device receiving an "imhere" message and/or the device determining one of the following conditions apply:

- A new MKB is being adopted by the cluster and archived binding information must be updated
- A device's membership in the cluster has changed such as:
    - The device is leaving its current cluster to join another cluster
    - Another device is joining the device's current cluster
- A new service has been identified
- If the device determines that the AT submitted for merge is not equal to its current AT, the merge process **SHALL** proceed as follows:

1. **Limit Test:** The device **SHALL** first determine if merging ATs would result in a cluster whose device limit would be exceeded. All device elements **SHALL** be counted except for those in the 'delete' state. If the device limit would be exceeded the merge operation **SHALL NOT** proceed. If this merge operation is initiated as part of a cluster protocol message that message **SHALL**

return a response of 'rejected' with reason 'limit'. If the ATs have different values for the cluster 'limit' attribute the device **SHALL** calculate the limit against the largest of the two 'limit' values.

2. **Dominance Test:** The device **SHALL** determine which cluster is the dominant cluster (i.e. whose binding ID should be adopted) as follows:
    - The device **SHALL** adopt the AT with the largest device limit
    - The AT with the largest set of device elements **SHALL** be the dominant cluster
    - Otherwise, if both ATs contain the same number of <device> elements the dominant cluster **SHALL** be the one whose public cluster ID has the least value when lexigraphically compared.
3. **Element Merge**: Once the cluster has determined which cluster is the dominant cluster the merge process **SHALL** follow these steps:
    - Merge all <cluster> elements
        - Archive its previous binding ID_b in order to restore archived content. This is done by adding a <cluster> element to its AT with the 'id' attribute equal to the public cluster ID_b of the cluster which it is leaving, a 'state' attribute whose value **SHALL** be 'archive' and an attribute 'archiveid' (i.e. ID archive or ID_a) whose value **SHALL** be an encryption of its former binding ID.
        - The calculation of the 'archiveid' value is as follows:

          $$AES\_128CBCE\ (K_m, ID_b)$$

          where ID_b is the Binding ID of its former cluster.
        - The device **SHALL** rebind all cluster elements whose state is 'archive'.
        - Note: Duplicate <cluster> elements with the same public cluster IDs **SHALL** be eliminated in the AT merge operation.
    - Merge all <device> elements
        - When the two device elements have the same "id" the conflict SHALL be resolved by:
            1. The device element with the highest 'count' attribute **SHALL** be adopted.
            2. If both device elements have the same "count" attribute, then the winning device element SHALL be chosen as the one having the highest precedent device state from the following list, which is ordered highest to lowest: DELETE, JOINED, PENDING

               The count attribute on the chosen element SHALL then be increment by one in the merged AT.
    - Merge all <service> elements
        - If a service element in one AT has the same ID as the service element in the other AT and the element has a counter attribute. The device **SHALL** adopt the service element with the largest counter value.
        - If the two ATs have a service element with the same "id" attribute the most recent service element **SHALL** be adopted. This can be determined by the element which has the highest 'count' attribute value.
    - Merge all <content> elements
        - If a service element in one AT has the same ID as the service element in the other AT and the element has a counter attribute. The device **SHALL** adopt the service element with the largest counter value.
        - If the two ATs have a content element with the same "id" attribute the most recent content element **SHALL** be adopted. This can be determined by the element which has the highest 'count' attribute value.

9.4.3 Join Cluster

This operation **SHALL** be used by a device to determine if it should become authorized on the sending device's cluster. A device **SHALL** first determine whether or not it is currently a member of the sender's cluster. If not, the behavior depends on the current Join state of the device.

This operation **SHALL** proceed if the following criteria are met:

- If the device's join state is "enabled".
- If the device's join state is "disabled" but the sending device is an "authorized" device in its current AT and the sending device has an updated AT or newer MKB.

  Note, this rule assures that changes to a cluster's AT are always propagated; in particular, it assures that changes to the states of devices are propagated (e.g. a device was removed from the cluster and now has an entry with state 'delete').
- The device **SHALL** determine if the join is valid by comparing the candidate AT to its own AT.

If the previous criteria are met, the device **SHALL** proceed as follows:

- The receiving device seeks to become authorized on the other's cluster with an *authorizeme* message.
- If the response to the *authorizme* message is "authorized" the following steps **SHALL** be followed:
  - The authorizing device updates the AT adding a <device> element for the requesting device. It then notifies the other cluster devices of the updated AT with *imhere* messages.
  - The requesting device decrypts and stores the binding key from the message response.
  - The requesting device, upon notification of the updated AT by the authorization device, retrieves the updated AT and merges it with the AT on the requesting device. It then notifies the other cluster devices of the updated AT with *imhere* messages.

9.4.4 Leave Cluster

A departing device **SHALL** perform the following steps when leaving the cluster:

1. The departing device **SHALL** change its state in the AT from the 'authorized" to "delete".
2. The departing device **SHALL** immediately propagate its updated AT by sending *imhere* messages to all the other devices in the cluster.
3. The departing device **MUST** succeed in propagating it's updated AT to at least one other authorized device in its current AT.

   Note that the departing device **SHALL NOT** return its AT to the manufactured state until at least one other authorized device in the cluster has performed a successful "getAT" operation against the device leaving the cluster.

It is also possible for another compliant device, with a richer user interface, to initiate a delete sequence on behalf of the departing device.

In this case, the following steps **SHALL** be performed by the initiating or departing device:

1. The initiating device **SHALL** change the departing device's state from "joined" to "pending" in the AT.
2. The initiating device **SHALL** send *imhere* messages to all authorized devices in its cluster (i.e. devices in its AT), including the departing device.
3. The departing device **SHALL**, upon observing it has been sent an AT with its device state equal to "pending", change its entry in its copy of the AT from "pending" to "delete".
4. The departing device **MUST** succeed in propagating it's updated AT to at least one other authorized device in its current AT.

Note that the departing device **SHALL NOT** return its AT to the manufactured state until at least one other authorized device in the cluster has performed a successful "getAT" operation against the device leaving the cluster.

All devices **SHALL** provide a method by which they can be deleted from any clusters to which they belong. It is the manufacturer's option whether a device provides an interface to allow it to initiate the deletion of another device. Since the consequences of leaving a cluster are serious, it **SHOULD** be difficult for a user to invoke a deletion event by accident.

Unless prompted by user action, devices **SHALL NOT** attempt to send *authorizeme* messages to clusters from which they have been deleted (i.e. the device has an entry in the AT of the cluster they are attempting to join, with the state of "delete").

Devices **SHALL NOT** attempt to send *authorizeme* messages to devices which have entries in the sender's AT with state "delete".

9.4.5 Joining a Device to the Cluster

HANA Content and some HANA Infrastructure are manufactured with device keys, and a Management Key Block (MKB). Devices are initialized at manufacture to be in a cluster containing only itself. The device thus has an Authorization Table (AT), a cluster ID, and cluster binding key.

Upon connection to a HANA network the device will go through the device discovery flow described above. This will result in both the device, and the other devices on the network being aware of the device's presence.

A device SHOULD require that the user employ the HANA UI to join the device to the HANA cluster. (an OPTION to a the manufacturer to configure devices to automatically join HANA clusters the device finds on the network.). This sets the device into Join mode. When the user takes this action the following will take place:

1. The new device will issue an *authorizeme* message to any one of the devices in the cluster.
2. The receiving device SHALL determine if the cluster is below the authorized device limit,
    a. If the cluster is already at the device limit the UI SHALL display an error message with instructions on how the cluster can be authorized to go beyond the current device limit. No further action is required.
    b. If it is below the authorized limit, it will add the new device to the AT and will respond with an Authorized Message and the following steps will take place.
3. The authorizing device SHALL notify all the other devices in the cluster of the updated AT.
4. The new device SHALL "abandon" the current cluster it is a member of and replace its active cluster with the new one.
5. The new device SHALL obtain the updated AT from the authorizing device and merge it with its current AT.
6. If the MKB on the authorizing device, the new device SHALL obtain the newer MKB and replace the MKB it currently has.
7. The new device SHALL contact every other device in the cluster with an *imhere* message. This has the affect of notifying each device of the updated AT and MKB that the device has to offer.

9.4.6 Removing a Device from the Cluster

The act of a device joining a cluster is a persistent event. If a device is later powered off or disconnected, it is still part of the cluster. However, if the user later sells the device, it is to both the current user's and the content owners' interests to have the device excluded from the old cluster. From the user's point of view, he does not want this absent device to count against the Cluster Device limit. From the content owner's

point of view, he does not want copies of cluster content that happen to be stored in the sold device to be playable in the new home.

A device must delete itself from the cluster AT. It cannot be removed by another device. When a device resigns from a cluster the following occurs:

1. The device changes its status in the AT from “joined” to “delete”. (Its entry is not removed from the AT only its state is updated) The device will no longer count against the cluster member limit.
2. The device notifies every other device in the cluster (that is online) via *imhere* messages of the updated AT.
3. The device changes its cluster ID and cluster binding ID to new random values using the random number generators in section 4.4.
4. The device updates the cluster id used in its archived initial Authorization Table.

It is also possible for another device to initiate a delete sequence on behalf of the leaving device. In this case, the steps are as follows:

1. The initiating device changes the leaving device’s state from “joined” to “pending” in the AT and sends *imhere* messages to all devices in the cluster, including the leaving device.
2. The leaving device, upon observing it is in “pending” state, changes its entry in its copy of the AT from “pending” to “delete”
3. The leaving device advertises the new AT by sending *imhere* messages to all the active devices in the cluster.

All devices shall provide a method by which they can be deleted from the cluster they belong to. It is the manufacturer’s option whether a device provides an interface to allow it to initiate the deletion of another device. Since the consequences of leaving a cluster are serious, it SHOULD be difficult for a user to invoke a deletion event by accident.

Devices SHALL not attempt to rejoin a cluster they have been deleted from, unless prompted by user action. Conversely, devices SHOULD NOT attempt to get authorized in a new cluster by a device listed as in the “delete” state in the AT of the cluster the joining device is already part of.

10 Bound Content

Content is bound to the HANA Content Cluster by encrypting the content with one or more Title Keys. This section describes how content is bound to the Content Cluster, how the bound Title Keys are stored, and how the associated metadata needed to manage the bound content is structured.

10.1 Encrypted Title Keys

Protected Content is encrypted by one or more Title Keys. This is done either when it is recorded by a device in the cluster, or by a content delivery channel prior to its arrival in the cluster. To bind the content to the Content Cluster, these Title Keys are themselves encrypted using one or more keys derived from the MKB, the AT, and the Cluster ID of the Content Cluster. These encrypted Title Keys are Bound Title Keys and can be decrypted by the cluster device which bound them, as well as the other devices in the cluster.

The Bound Title Keys are stored in the Title Key Block section of the Content Header of the item of protected content.

10.2 Storage of Bound Content

Bound content is sufficiently strongly protected that it may be freely accessed and moved by devices and users. This means that bound content may be stored onto any media which physically can hold it. There is no special requirement for storage media such as media serial numbers, secrets, or protected access modes

To unbind content for playback, devices must know which specific versions of the AT and MKB were used to calculate the Binding Key that binds the content to the cluster. Devices must store bound content so that the correct associated AT and MKB can be accessed for unbinding operations.

For storage used exclusively by the device, it is up to the device manufacturer to decide how to store bound content and maintain the mapping to the associated AT and MKB.

10.3 Storage Partitions for Interoperability

When bound content is stored and may be accessed by different devices a common storage format is need to ensure interoperability. The following format shall be used when bound content is stored on media which may be read by more than one device.

Bound content is organized into partitions. Each partition contains the AT and MKB used by all bound content in the partition. All content in the partition must use the same AT and MKB.

A partition can hold one or more bound content items.

More than one partition maybe held on a given media at once. Partitions do not need to be from the same cluster.

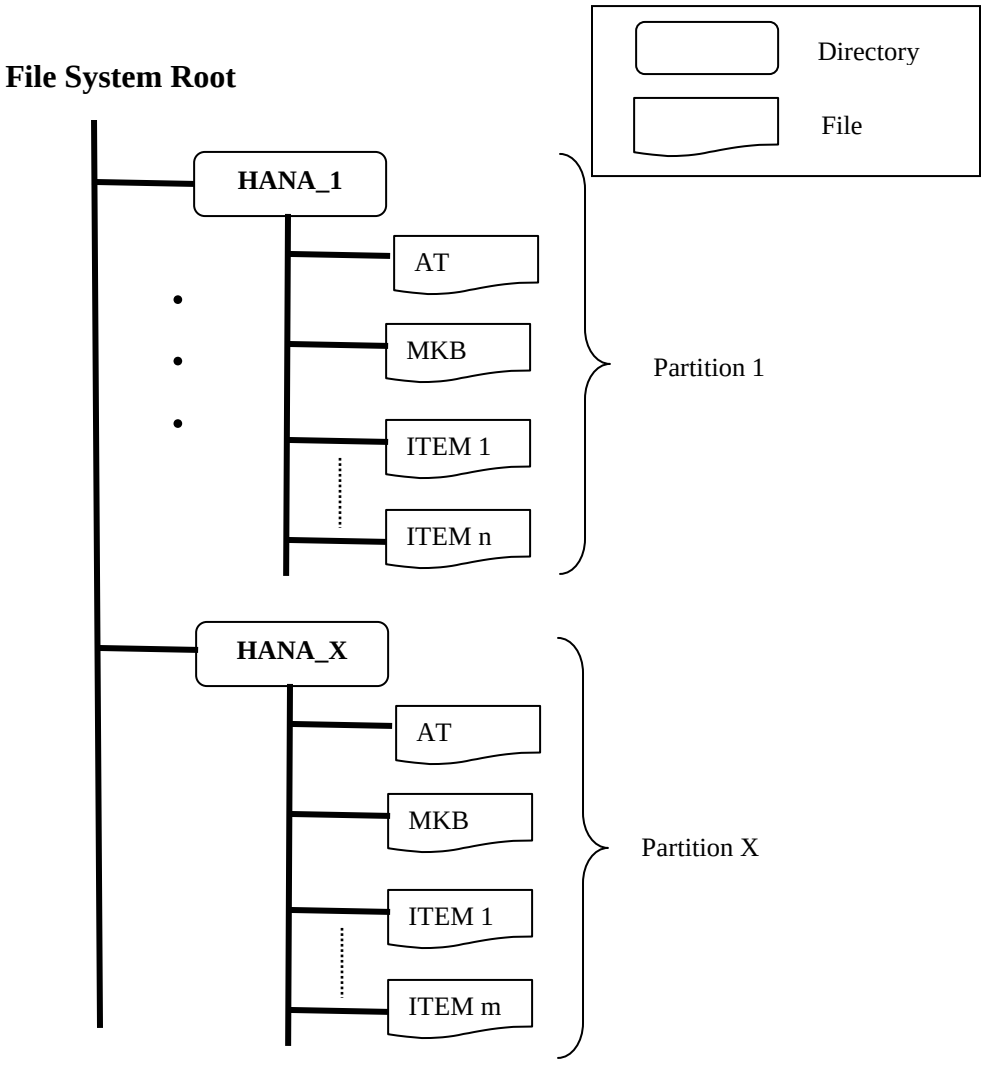


Figure 10-1: Storage Partition Layout

10.4 Content Header

The ASCCT content header is comprised of following data areas in order:

- Fixed Area (FA) – Mandatory
- Protected Fixed Area (PFA) – Mandatory
- Title Key Block (TKB) – Mandatory
- Variable Length Data Area (VLDA) – Optional repeating data structures

Immediately following these data areas is the actual encrypted (content) data.

The length of the fixed portion of the header is 2048 bytes and it **SHALL** be maintained for each unique piece of ASCCT protected content.

The variable length data portion is optional, but when it exists, it is comprised of well defined repeating structures of known sizes(s) described below.

The actual encrypted content data immediately follows the ASCCT content header as either a content stream or the most significant part of prepackaged (stored) content (i.e. the first thing logically read from the file system).

10.4.1.1 General Structure Overview

Content Section	Area	Offset (bytes)	Field Size (bytes)	Area Description / Purpose
ASCCT Header	Fixed Area (FA)	0-1023	1024	Contains fixed data that can be quickly used to verify that the data that follows is ASCCT compliant. Contains flags that can be quickly processed to determine handling of the data that follows. Provides critical cryptographic verifiers to receivers to validate that they are authorized and are able to decrypt the data that follows the header.
	Protected Fixed Area (PFA)	1024-2047	1024	Contains signed fixed data that identifies the content uniquely and information that can be used by the device to manage the content for playback, Input/Output operations or for Copy and Move operations. Describes additional variable length data that can be used to carry additional information about the content (see Variable length section below for details).
	Title Key Block	2048-x	Even multiples of 1024	Contains title keys encrypted with recording keys. Length **SHALL** be an even multiple of 1024 (1K) byte The Maximum size **SHALL** be 21504 (21K) bytes
	Variable Length Data	x+1 – p	1..n "Variable Length Data Structures"	Describes optional variable length data that can be used to carry additional information about the content including, but not limited to: Additional Protection systems (on top of ASCCT) Watermarking information Content metadata Transactional Information / Licensing information Related Services (e.g. AACS Managed Copy, or Prepared Video) Etc.
Encrypted Content Data	N/A	p+1 – End of Stream (EOS)	Size of Encrypted Content Data	The actual encrypted data that follows the header. The CPS URN in the Fixed Area (FA) identifies the Content Protection System as defined by the Licensing Authority (LA). The Cipher Type, Strength, and Mode are defined by the Crypto Flags in the Fixed Area .

10.4.2 Fixed Area (FA)

The Fixed Area contains information that allows processors of ASCCT content to perform fast calculations to verify that they are able to continue processing, such as:

- Confirm that the data is indeed ASCCT protected content data
- Confirm that the ASCCT header version is one that the processor understands
- Identify the specific ASCCT adaptation using CPS URN
- Verify that key cryptographic elements are current (i.e. MKB, AT)
- Confirm that content belongs to this cluster (i.e. Cluster ID)
- Verify that the cipher, key strength and block mode is one that the processor can use

10.4.2.1 Fixed Area Field Overview

Fixed Area (FA)	0-4	5	**Data Block Identifier**	Identify header as ASCCT Content The value **SHALL** be the characters 'ASCCT'
	5-12	8	**MKB Type and Version**	MKB Type and Version. A 4-byte type followed by a 4-byte version as contained within the MKB type and version record
	13-16	4	**Header Version**	Initial version number. A 2-byte major version followed by a 2-byte minor version. Current version **SHALL** be '0x0100'
	17	1	**Cipher Hint Flags**	Flag bits that identify the Cryptographic Cipher, Cipher Strength, and (Block) Mode
	18-49	32	**MKB Hash (required)**	To verify that the MKB matches (AES_H(MKB))
	50-81	32	**AT Hash (required)**	To verify that the AT matches (AES_H(AT))
	82	1	**PFA Security Level**	To determine the device security class (0-6)
	83-114	32	**PFA Signature**	The secure signature of all data within PFA (including all trailing zero bytes). This field guarantees integrity of the data in the PFA which includes CCI and CMI.
	115-146	32	**Cluster ID (optional)**	Public cluster ID
	147-402	256	**CPS URN**	URN that identifies the Content Protection System (CPS) used to protect the content. The default **SHALL** be: "urn:cps:ascct:hana:2008:0002-0001"
	403-1023	621	**Reserved**	Reserved. **SHALL** be filled with zeroes.

The verification data of the MKB hash, the AT hash, and the hash of the Binding ID may be ignored by the device. However, those devices that are capable of validating verification data **SHALL** check the verification data for currency.

The verification data can be used by the device to check the currency of its MKB and AT against that of the sender of the content (e.g. live stream) or, in the case of stored content (e.g. partitioned), the device can use the verifiers to determine if the content needs rebinding (and perhaps all content in the partition in which it was stored) to the most current MKB and/or AT.

By default, the content itself follows the end of the header (both the fixed portion and any variable length data.

10.4.2.2 Cipher Hint Flag (Byte)

Area	Offset (bits)	Field Size (bits)	Field Name	Field Description / Allowed Values
Cipher Hint Byte (1 byte)	0-1	2	**Cipher Identifier Bits**	Hint flag bits to help processors quickly identify the cipher used to encrypt the data (packets). Cipher ID bits: 00b : AES 01b : C2 10b : TDES Others: Reserved.
	2-4	3	**Cipher Strength Bits**	Hint flag bits to help processors quickly identify the cipher strength used to encrypt the data (packets). Cipher Strength bits: 000b : 64 001b : 128 010b : 256 Others: Reserved.
	5-7	3	**Cipher Encoding Mode Bits**	Hint flag bits to help processors quickly identify the cipher (block) encoding mode used to encrypt the data (packets). Encoding (Block) Mode bits: 000b : CBC Mode 001b : CTR Mode 010b : EBC Mode Others: Reserved.

10.4.2.3 PFA Security Level

This value represents the device security class with which the signature in the Protected Fixed Area, (PFA Signature) is calculated (protected). See section 5.2 Security Class for a discussion of device security class.

10.4.2.4 PFA Signature

The PFA Signature is calculated with the Management Key Precursor (K_m^{-i}), where the device security class value "i" is provided in the PFA Security Level field. This means that devices that are trusted to change any data within the PFA data area (including CCI and CMI data) must be able to calculate the Management Key Precursor that matches the PFA Security Level value.

The PFA Signature **SHALL** be calculated as follows:

PFA Signature = MAC(K_m^0, D)

Where K_m^0 is the Management Key (K_m) and 'D' is the PFA data (the entire 1024 bytes).

10.4.3 Protected Fixed Area (PFA)

The protected fixed area is used to convey data that devices can verify has not been tampered with. It contains content identifiers, copy control information and content management data. Every ASCCT device of the required security class is able to calculate and verify the PFA signature hash for the data described in the PFA. If the PFA verification fails then devices **SHALL** conclude that the content header has been tampered with and the ASCCT device **SHALL** stop processing the ASCCT content.

10.4.3.1 Protected Fixed Area Field Overview

<table>
<tr><th>Area</th><th>Offset (bytes)</th><th>Field Size (bytes)</th><th>Field Name</th><th>Field Description</th></tr>
<tr><td rowspan="2"></td><td>1024-1119</td><td>96</td><td>Content ID</td><td>Item ID || Instance ID [|| Replica ID]

Where the Item ID, Instance ID and Replica ID are defined as follows:
<table>
<tr><th>Offset (bytes)</th><th>Size (bytes)</th><th>Name</th><th>Description</th></tr>
<tr><td>1056-1087</td><td>32</td><td>Item ID</td><td>AES_H(Content URN)</td></tr>
<tr><td>1088-1119</td><td>32</td><td>Instance ID</td><td>AES_H(Binding Nonce)</td></tr>
<tr><td>1120-1151</td><td>32</td><td>Replica ID</td><td>AES_H(Item ID || Device ID || Time)
Note: The Time value used here does not require a secure clock.</td></tr>
</table></td></tr>
<tr><td>1120-1183</td><td>64</td><td>Binding Nonce</td><td>The value that uniquely binds the content to a given cluster. This value SHALL be calculated as follows:
Cluster ID || ASCCT_Nonce</td></tr>
</table>

	1184-1119	16	Binding Device ID	The Device ID of the device that performed the initial binding of the content into the cluster. After initial content binding any subsequent copies or replicas SHALL preserve the Binding Device ID for the life of the content.
	1200-1231	32	Transaction ID	Transaction ID (optional)
	1232-1487	256	Content URN (mandatory)	A statistically unique Universally Unique Identifier (UUID) for the unique published work or content. This can be assigned by a registering agency, created dynamically by an ASCCT device or created by the provider of the content. If this value is generated it **SHALL** be done so such that any 2 devices who create this when joined will not likely experience collisions on device join. 1 x URN of length 256 2 x URN of length 128 Concatenated See section 12.2 Content URN for URN format and generation
	1488-1503	16	Embedded CCI	Embedded Copy Control Information (CCI).
	1504-1631	128	Content Mgmt Information (CMI)	Embedded Content Management Information (CMI) (i.e. usage rules).
	1632-1759	128	Content Tracking Data	Embedded Content Tracking Data (CTD) (i.e. content usage auditing).
	1760-1763	4	AID Data	Authorized Input Domain (AID) data. A series of flags values that indicate the input domain that provided the source (origin) of the content.
	1764-1767	4	AOD Data	Authorized Output Domain (AOD) data. A series of flags and values that indicate if the associated content can be copied to well known output protection systems and media (domains).
	1768-1769	2	Variable Data Flags	Operational Flags that may affect processing of variable length data blocks. Reserved for future use **SHALL** be filled with zeroes.
	1770-1785	16	Variable Data Length	Actual total length of the variable length data area (for skipping ahead).

	1786-2047	262	Reserved	Reserved. **SHALL** be filled with zeroes.

10.4.3.2 Embedded CCI

The following values are valid for embedded CCI:

- 0x00: **Copying Not Restricted**.
- 0x01: **No Further Copying Permitted**. .
- 0x02: **One Generation Copy Permitted**.
- 0x04: **Copying Prohibited**.

10.4.3.3 Content Management Information (CMI)

Area	Offset (bytes)	Field Size (bytes)	Field Name	Field Description
Content Mgmt Data 128-bytes	0	1	Enforcement Bits	Flag bits to indicate whether specific elements of Content Management Data should be utilized or not. Any enforcement bits that are set SHALL cause the device to enforce the corresponding conditions.
	1	1	ASCCT Copy Control Information	Flags to indicate how content copies and moves are to be treated at a cluster level (TBD).
	2-17	16	Activation Date-Time	Date/Time after which playback of the corresponding content is permitted. <left-padding><date>T<time> where <date> is YYYYMMDD and <time> is 'hhmmss' Where < left-padding > SHALL be a series of space characters (i.e. 0x20) which are added so that the date-time representation occupies the entire 16 byte field.
	18-33	16	Expiration Date-Time	<left-padding><Date/Time after which playback of the corresponding content is not permitted. <date>T<time> where <date> is YYYYMMDD and <time> is 'hhmmss' Where < left-padding > SHALL be a series of space characters (i.e. 0x20) which are added so that the date-time representation occupies the entire 16 byte field.
	34-53	20	Max. Access Duration	Duration which represents the total allowed access/play time (note: play time of < 1 second is not meaningful). <left-padding><P[nnY][nnM][nnD]T[nnH][nnM][nnS]> Where < left-padding > SHALL be a series of space characters (i.e. 0x20) which are added so that the date-time representation occupies the entire 20 byte field.

	54-57	4	Max. Access Count (base 16)	Number of times content can be accessed/played back within the cluster. 0x0000 - 0x00FF: actual number of playbacks allowed. 0xFFFF: unlimited playback allowed. Other values : Reserved See Min. Access Duration for the granularity of what qualifies a single playback.
	58-67	10	Min. Access Duration	Defines granularity against which a single Access/Playback Count is registered. This value SHALL follow the ASCCT Time Format using only Hours, Minutes and Seconds duration values. <left-padding><T[nnH][nnM][nnS]>: actual playback duration granularity Where < left-padding > **SHALL** be a series of space characters (i.e. 0x20) which are added so that the date-time representation occupies the entire 10 byte field. Note: For this field no 'P' (Period) character separator **SHALL** be used at the beginning.
	68-127	60	Reserved	SHALL be filled with zeroes.

10.4.3.4 Content Tracking Data

Area	Offset (bytes)	Field Size (bytes)	Field Name	Field Description
Content Tracking Data 128-bytes	0	1	Reserved	Reserved. **SHALL** be filled with zeroes.
	1	1	ASCCT Copy Control Status	Current status bits of Move / Copy Control Information within the cluster. Initially **SHALL** be filled with zeroes.
	2-17	16	First Access Date-Time	Date/Time the content was first accessed/played. Initially **SHALL** be filled with zeroes. <date>T<time> where <date> is YYYYMMDD and <time> is hhmmss
	18-33	16	Last Access Date-Time	Date/Time the content was last accessed/played. Initially **SHALL** be filled with zeroes. <date>T<time> where <date> is YYYYMMDD and <time> is hhmmss

	34-53	20	Total Access Duration	Duration which represents the total play time (note play time of < 1 second is not meaningful). Initially **SHALL** be filled with zeroes. <PnnYnnMnnDTnnHnnMnnS>
	54-57	4	Access Count (base 16)	Number of times the content can be played back on a single device 0x0000 - 0x00FF : actual number of accesses Other values : reserved
	58-127	70	Reserved	**SHALL** be filled with zeroes.

10.4.4 Title Key Block (TKB)

In general, a piece of content may have many Title Keys. The Title Key Block (TKB) associated with the content allows compliant devices to decrypt the Title Keys, which in turn allows them to decrypt the content. Note that a single device might not be able to decrypt all of the Title Keys in the TKB. Which Title Keys edare needed to decode a particular part of the content is application-specific. However, the TKB format is common across all applications..

The HANA devices SHALL use Recording Keys (and their security class variants) to bind content they record to the cluster. Bound Reocrding Keys are derived as follows: using the following:

$$K^{-i}_{rb} = AES\_G(K^{-i}_{r}, ID_b \oplus AES\_hash(AT))$$

These Bound Recording Keys are used to encrypt the Title Keys. The Encrypted Bound Title Keys are stored in the Title Key Block.

The Title Key Block **SHALL** be filled with 0_{16} data to the next 1024 byte boundary in the content header.

Table 5-1 – *Title Key Block* Format

Bit / Byte	7	6	5	4	3	2	1	0
0	Title Key Block Type: $B0_{16}$							
1	Length (bytes)							
2								
3								
4	Title Key Block Entry 0							
…								
23								
…	…							
Length – 20	Title Key Block Entry N							
...								
Length-1								
	1024 - Length mod 1024 0_{16} fill to 1024 byte boundary							

Where a Title Key Block is entry is as follows:

Table 10-2 – Title Key Block Entry Format

Byte \ Bit	7	6	5	4	3	2	1	0
0	Security Class			Binding Key Label Bits (12-8)				
1	Binding Key Label Bits (7-0)							
2	Encrypted Title Key Label							
3								
4	Encrypted Title Key							
…								
19								

- The first two bytes denote the binding key used to encrypt the Title Key.
    - The high order 3 bits denote the Security Class of the key from 0:6. The encoding 111_2 specifies that the binding key was calculated from one of the Management Key Variants, which have no security class.
    - The remaining 13 bits denote the label of the key:
        - If the Security Class field is 0:6, 0000000000000_2 specifies that the binding key was calculated from the Management Key. Other values specify a binding key calculated from one of the Recording Keys.
        - If the Security Class field is 111_2, the low order 10 bits specify the Management Key Variant (0:1023).

  The Encrypted Title Key Label field. This field gives the label of the Title Key, calculated as follows:

  $$L = D \oplus [AES\_G(K_t, (bytes\ 0:1) \parallel t_0)]_{msb16}$$

  Where D is the data in this two-byte field, K_t is the Title Key, and t_0 is a confidential constant provided by the HANA Licensing Authority.
- The Encrypted Title Key field. This field contains the encrypted title key, encrypted with AES_128E using the key denoted by bytes 0:1.

The keys encrypting the Title Keys are the "bound" versions of the denoted key. In other words, if K is the key, the bound version is:

$$K' = AES\_G(K, ID_b \oplus AES\_hash(AT))$$

Devices may encounter entries in the TKB which are protected using Management Key Variants. This mode is indicated in the Title Key Label Field. The binding information for Management Key Variants is slightly different. When a piece of prepared content is authored using Management Key Variants, the MKB must accompany the content. In that case, the TKB will contain a special Title Key denoted with the label $FFFF_{16}$. This Title Key is used as the sole binding key data in the calculation above (omitting ID_b and the AT hash). This "Title Key" (actually it is data) will be the first Title Key entry in the block and will be encrypted with K_b (the cluster-bound version of K_m)[1].

A recording device shall use its Recording Keys when recording protected content for the cluster. The recording device picks one or more Title Keys (as required by the application), and then produces a TKB in which each Title Key is encrypted with *every one* of the Recording Keys that the recording device knows.

[1] Of course, if access to the content is restricted by security class, relevant keys are K^{-i}_m and K^{-i}_b, for example.

When the MKB or AT hash change, the Bound Recording Keys are updated. Content must be rebound using the new set of bound keys. To rebind the content, the device processes the TKB, and for each encrypted Title Key it can decrypt, it decrypts the Title Key using the appropriate old bound key and then encrypts the Title Key using the updated Bound Recording Keys. This produces a new TKB which replaces the old one. Since a device may not have the ability to decode every entry in the old TBK, it ignores those entries which it cannot process and does not include them in the new TKB.

The Recording Keys are designed so that every device in the cluster shares at least one Recording Key with every other device in the cluster. When playing back a recording, the playing device looks through the TKB, checking the Binding Key Labels of each entry. It ignores entries for which it does not have the associated key. From the other entries, however, it will be able to decrypt Title Keys. It is guaranteed to have enough Title Keys to decrypt the parts of the content it is entitled to play.

10.4.5 Variable Length Data

The variable length data portion of the ASCCT header is containsa structured data that conveys additional information about the content such as content metadata, content management, transformation rules for content when being copied to authorized outputs, and/or transformation rules preserved from authorized input sources.

10.4.5.1 Variable Length Data Structure

Area	Byte Offset	Size (Bytes)	Field Name	Field Description
VLD Header	0-127	128	VLD Type URN	URN for the data contained in the data section.
	128-129	2	VLD Version	Variable Length Data (VLD)Version. **SHALL** be ‘0x0100’ for version 1.0
	130-145	16	Data Length	Specified as the number of blocks of block size ‘L’ ‘L’ **SHALL** be 16 bytes The actual length in bytes of VLD Data is Data Length × ‘L’.
	146-255	110	Reserved	Reserved. **SHALL** be filled with zeroes.
VLD Data	256 – End of Header	Data Length * L	Data	URN specific data. Note: this data could be binary data, structured data such as XML and perhaps further encoded by other systems. The URN would imply treatment (processing) of data in this area. Note: The encoding of the actual data is completely specifiable by the data owner. It can be binary, structured XML, Base 64 Binary, ASN-1, etc.

11 Content Management

11.1 Use Cases

Content is bound to the HANA cluster by encrypting it with Title Key (K_t) which is in turn protected by encrypting it and storing in the Title Key Block. When the cluster definition is updated by a device being added or removed, the Cluster Binding Key is also updated, requiring all the Title Keys in the cluster to be rebound.

A cluster maintains a record of all content which has been bound to it. It is the possession of a Title Key, and not the bytes of the bound content which is essential to track for a bound item of contentThe file which holds the bound content consists of two logical pieces: the content header, and the content data. The content header, as detailed elsewhere in this document, contains information about the content item, the cluster it belongs to, as well as usage rules associated with the content.

Any usage rules for the bound content are interpreted by specific digital rights management systems above and beyond the HANA cluster binding scheme. That is to say this document details the task of accessing the content bound to a HANA cluster in a way that will maintain the integrity of the binding of the licensed content to the cluster. Once accessed, the actual decoding of the content data and adherence toed usage rules are up to the device decoding the data and the device interpreting the rules. Hence, the interpretation and execution of such rules are not specified here. However, the reader should always remain aware that bound content may carry usage rules which will be applied above and beyond the HANA rules for bound content detailed here.

The bound content use cases deal with the actions applied to bound content in the HANA cluster.

Two essential rules prevail throughout these use cases:

1. Bound content can be freely used and copied throughout the cluster. The principal used here is that a cluster is the equivalent of single logical device.
2. Transferring bound content out of a cluster is the act of transferring the license to the content to the destination. The principal followed is to apply the usage rules that would apply if the content and its license were granted by possession of the physical media holding the content (i.e., a DVD). Just as a person must posses the physical media with the one distinct copy of the content held on it, to transfer the license to use the content from the media, a HANA device must possess the sole, distinct instance of the bound Title Key in the cluster from which the content is to be exported to transfer the bound content out of the cluster. When this condition is satisfied, the content may be transferred out of the cluster, so long as such a transfer does not violate the CCI or extend CCI rules associated with the content item.

11.1.1 Import of Bound Content

Content is bound to the cluster by encrypting it with a Title Key and in turn encrypting the Title Key and storing it in the Title Key. The two steps of this process need not be done at the same time. Typically content is prepared by a distribution system for delivery to a HANA home network by encrypting it with its Title Key. This key need not be unique to either the media title nor to a HANA Content Cluster ID. The second and final step in the delivery is for the distribution system to obtain the Binding Key for the cluster to which the content is being bound, and to re-encrypt the Title Key Block with recording keys bound to the cluster with the Cluster Binding Key.

11.1.2 User wants to create a copy of bound content

Bound content can be freely copied within the cluster. Any device may make a copy of any bound content freely. The content directory on the device receiving the content **SHALL** be updated to reflect the new instance of the content item. The device will then notify the other cluster devices of the change to its content directory. This may be used to create additional copies onto portable devices, . or to create backup copies onto either on-line or off-line media.

When a copy of bound data is made, the Replica ID in the Content Header is recalculated to establish a new distinct value. This in turn will require a recalculation of the hash of the Fixed Protected Area which holds the Replica ID field.

The device also adds the updated Replica ID the list of replicas for the content item in its Device Content List. This in turn causes the device to notify other devices in the cluster that its Device Content List has changed.

11.1.3 User wants to import protected content into the cluster

Adding content to a cluster involves adding a Content Header in front of the content data, encrypting the content data with the Title Key, and encrypting the Title Key with the Cluster Binding Key. This can be done either by a device in the cluster, or it can be done externally to the cluster. The end result is two files: the data file, and the title key file.

Any usage rules, such as CCI bits associated with the content, can be encoded in the Content Header. at the time of binding If the import is received as streamed content using DTCP, the DTCP usage rules (copy-once, copy-never, etc) are preserved in the Content Header applied to the bound content.

11.1.4 User wants to play bound content

If the content is bound to the cluster which the device is in, it will decrypted the Title Key and access the content.

If the content is bound to a different cluster from the one the device is a member of, it shall refuse to play it. The device may optionally present the user with a means to obtain a proper license to the content. Once the license had been obtained, the device would import the content and play it.

11.1.5 User wants to play stored content item to more than one sink

This use is permitted within the cluster without restriction from a HANA content protection view.

11.2 Rebinding Title Keys

Whenever the MKB or the AT is updated, a new Cluster Binding Key is generated. As a result all content bound with the previous binding key must have its Title Key Block entries re-encrypted with the new Bound Recording Keys due to the new Cluster Binding Key.

A Content Device is responsible for re-encrypting Title Key Block data for the content it manages in the associated cluster's bound data partition(s) with the new Cluster Binding Key that. Specifically, the device is responsible for any content to which is has byte level access, whether as internal storage, external direct attach, or external network attach.

In each partition there must be a separate copy of the cluster MKB (keymgmt.blk file) and AT (auth.tab file) in the root of the partition so devices have the necessary information to update the encrypted title keys. A device accessing a file in a partition in bulk storage shall perform the following steps, in order:

1. If either the partition MKB file or the partition AT file does not exist, it skips to step 5.
2. Reads the MKB and AT in the partition and compares each with the one the device has. If they are identical, the device exits and proceeds to access the content file.
3. If either the MKB or the AT in the partition is more recent than the one the device has, the device adopts it, and sends out the necessary "imhere" messages to other devices to alert them that the binding key has changed. Itthen goes back to step 2.
4. If the AT in the partition is disjoint to the device's AT, the device attempts to merge the two ATs. If the merge fails (i.e., the merged AT would exceed the cluster device limitand the device is not capable of connecting to the external authorization center), the device exits but shall not access the content file. Otherwise, the device accepts the merged AT and goes back to step 2.

5. If either the MKB or AT in the partition is an older version or non-existent, the device obtains a write lock on the MKB and then the AT in that order. If the device is unable to obtain both locks, it releases any lock it did obtain and tries again later.
6. Once the device obtains the locks, it checks for the existence of the backup files keymgmt.bak and auth.bak in the partition. If they both do not exist, it skips to step 12. The existence of the backup files is evidence that a previous device has crashed while trying to update the encrypted title keys in the partition. The current device must first correct this before proceeding.
7. If either the MKB file or the AT file do not exist, the device renames the backup file to the missing file or files, erases any remaining backup files, releases the locks, and goes back to step 1.
8. If there is only one backup file, the device erases it and skips to step 12.
9. The device calculates the *backup binding key* from the backup files. It also calculates the *partition binding key* from the partition's current MKB and AT.
10. The device loops through all the encrypted title keys.
    a. For each encrypted title key, the device decrypts it with both the partition binding key and the backup binding key.
    b. If neither binding key result in the correct verification string, the content has been somehow corrupted and the device action is manufacturer-specific. The device shall continue to loop, however.
    c. If the verification string indicates that the title key was correctly decrypted by the backup binding key, the device shall re-encrypt it with the partition binding key.
11. The device erases the backup files.
12. If either the MKB or AT is missing in the partition, the device writes both its MKB and AT to the partition, releases the locks, and exits to access the content file. This is a normal case for an empty partition where the device is writing the first file. However, if there are existing content files in the partition, this is an error condition. The existing content files have been lost, and the device action is manufacturer-specific.
13. The device checks that either the MKB or the AT or both are still down-level. If neither is, it releases the locks and exits to access the content file.
14. The device renames the partition MKB and AT to keymgmt.bak and auth.bak.
15. The device writes its MKB and AT as the new partition MKB and AT (keymgmt.blk and auth.tab).
16. The device calculates the old binding key based on the now backed-up MKB and AT in the partition.
17. The device computes its bound recording keys from the MKB
18. The device loops through all the encrypted title keys in the Title Key Block and re-encrypts them with the new bound recording keys for the device. The security class of each bound Title Key in the Title Key Block MUST be preserved during the rebinding operation.
    a. For each encrypted title key, the device decrypts it with the old binding key.
    b. If the decryption does not result in the correct verification string, the content has been somehow corrupted and the device action is manufacturer-specific. The device shall continue to process each entry in the Title Key Block.
    c. The device encrypts the title key with the new bound Recording Keys the device has. and saves it back into the TKB.
19. The device erases the backup MKB and AT files.
20. The device releases the locks and exits to access the content file.

The foregoing algorithm assumes that the network storage device has a robust lock mechanism that can recover from network failures. If a network storage device whose lock mechanisms are absent or unreliable is used, the following principles apply:

1. Each recording device (i.e., each device that might create a file on the network storage) must have its own separate partition.
2. The recording device is alone responsible for updating the encrypted title keys. It shall follow the algorithm above, without the lock and unlock steps.
3. Other reading devices accessing the content in the partition shall treat the media as if it were removable backup storage. They SHALL follow the restore algorithm described in the following section.

11.3 Backup and Restore

Backup of bound data can be performed by without any special cryptographic operations, as long as the backup operation is careful to save the MKB and the AT. A restore operation typically will involve a Content Device since the content will have to be rebound, unless cluster's MKB or AT has not changed between the backup and the restore. The Media Device performing the restore must check the MACs in the saved AT; this will implicitly verify that the Cluster ID in the saved AT is the one the restore device uses.

In general, the device doing the restore treats the MKB and AT on the backup media as if it has just received an imhere message from a device advertising that MKB and AT. If the backup media has a more recent version (this should be rare), the device replaces its own MKB and/or AT. If the AT in the partition is not equal to the device's AT, the device attempts to merge the two ATs. If the merge fails (i.e., the merged AT would exceed the cluster device limit and the device is not capable of connecting to the external authorization center), then the device shall not restore the content.

If the device's own MKB or AT are more recent that the ones on the backup media, the device may choose if allowed by the media to update the MKB, AT, and encrypted title keys on backup media. In that case, the device shall follow the algorithm for encrypting title keys described in the previous section.

It is possible that the binding ID used in the user's backup data is out of date because two clusters merged and the ID on the backup media was the one abandoned. In that case the action is manufacturer-specific. Devices are allowed securely maintain lists of old merged, abandoned clusters' binding IDs for the purpose of doing a restore but are not required to do so. Devices providing this function shall verify that the AT in the abandoned cluster is a subset (not necessarily a proper subset) of the AT of the cluster to which the user has requested that the content be restored.

11.4 Content Activation and Expiration

The ASCCT content header provides a means for transmitting both activation and expiration information. Specific treatment of time-based information in the content header is left to specific adaptation books or to sections that describe transfer to/from HANA authorized input/output protection technologies.

12 Content Header Element Formats

12.1 Universal Resource Name

A Uniform Resource Name (URN) is a Uniform Resource Identifier (URI) that adheres to requirements as described within IETF RFC 1737. IETF RFC 2141 section "URN Syntax" indicates:

Uniform Resource Names (URNs) are intended to serve as persistent, location-independent resource identifiers and are designed to make it easy to map other namespaces (that share the properties of URNs) into URN-space. Therefore, the URN syntax provides a means to encode character data in a form that can be sent in existing protocols.

URN Syntax

The format should follow the accepted URN format/syntax (RFC 2141):

```
<URN> ::= "urn:" <NID> ":" <NSS>
```

where <NID> is the Namespace Identifier, and <NSS> is the Namespace Specific String.

Uniqueness

In most cases, ASCCT relies on URNs to be a structured, statistically unique Universally Unique Identifier (UUID) that can be used to uniquely identify entities (by name).

If a URN is used that is not statistically unique, such as an International Standard Audiovisual Number (ISAN), an additional UID (e.g. ":<UID>") must be appended to the end of the base (non-unique) URN.

For example, ISANs in general are not statistically unique since multiple formats and variations on the same "work" would be assigned the same URN. When using ISAN as part of the ASCCT Content URN an additional value that would make the URN a statistically unique UUID MUST be appended to the ISAN. For example, to make the "Spider Man" ISAN unique, an additional value would be appended using the URN separator ":" (colon) as follows:

urn:isan:0000-0000-9E59-0000-O-0000-0000-2:urn:ASCCT:0909127800731203412988710

The where the underlined portion of the string is the device-unique URN appended to the general URN.

12.2 Content URN

This section defines a Content URN as a special URN used by ASCCT which must be 256 bytes in length or less in order that it may be included in the content header and also be made unique as described above.

Examples of content URNs that could be used within ASCCT content headers.

- **International Standard Audiovisual Number (ISAN)**
  The movie "Spider Man" **:** *urn:isan:0000-0000-9E59-0000-O-0000-0000-2 : <UID>*
- **Universal Content Identifier (UCI) : IETF RFC 4179**
  The movie "Snow Flower" : urn:uci:I001+SBSi-B10000083052
- **ASCCT Generated: ASCCT:uid:<NSS> where NSS is a generated UUID**
  urn:cps:ascct:2007:0001-0000:<UUID>

12.2.1 Content Protection System (CPS) URN

This section contains examples of CPS URNs that could be used in ASCCT content headers.

The default ASCCT URN is:

```
"urn:cps:ascct:2007:0001-0000"
```

HANA defines the following CPS URN:

"urn:cps:ascct:hana:2008:0002-0001"

12.2.2 Variable Length Data (VLD) Type URN

Adaptations of this book, as well as other specifications that describe treatment of authorized input and output data formats, may define VLD Types for conveying additional data. The VLD Type must be recognized and registered with the HANA licensing authority. Using this method, an adaptation or other specification can define an approved VLD Type URN to include in the VLD header.

12.3 Date-Time Format (ISO 8601 basis)

Date-Time values are represented in the following format:

<date>T<time>Z

where the string length is always 16 bytes:

length = 8 bytes for <date> + 1 byte for 'T' character + 6 bytes for <time> + 1 byte for Z = 16 bytes

Time is stored in the UTC (Cordinated Universal Time) time zone.

Combining *date* and *time* representations is quite simple. It is in the format of <date>T<time>. The *date* and *time* sections are any proper representation of the date and time created by following the standard.

Where:

- <date> is of the form 'YYYYMMDD' and <time> is of the form 'hhmmss'
- 'YYYY' is the 4-digit year value
- 'MM' is the 2-digit month value
- 'DD' is the 2-digit day value
- 'hh' is the 24-hour, 2-digit hour value (range: 00 to 24)
- 'mm' is the 2-digit minute value (range: 00 and 59)
- 'ss' is the 2-digit second value (range: 00 and 59 or 60 in the exceptional case of an added leap second)

Note: midnight is a special case and can be referred to as both "00:00" and "24:00". The notation "00:00" is used at the beginning of the day, and is the most frequently used one. At the end of a day use "24:00".

12.4 Duration Format (ISO 8601 basis)

Durations are represented by the format**:**

`PnnYnnMnnDTnnHnnMnnS` (length = 20 bytes)

Where:

- 'P' indicates a Period (duration) follows
- 'nnY' indicates the number of Years, where 'nn' can be a 1 or 2-digit number
- 'nnM' indicates the number of Months, where 'nn' can be a 1 or 2-digit number
- 'nnD' indicates the number of Days, where 'nn' can be a 1 or 2-digit number
- 'T' is a separator to indicate the Time portion of the value follows (if needed, see truncation rules below).
- 'nnH' indicates the number of Hours, where 'nn' can be a 1 or 2-digit number
- 'nnM' indicates the number of Minutes, where 'nn' can be a 1 or 2-digit number

- ‘nnS’ indicates the number of Seconds, where ‘nn’ can be a 1 or 2-digit number

Example:

- The Duration value “P1Y4M14DT8H30M00S” would be interpreted as a Period of one Year, four Months, fourteen Days, eight Hours, thirty Minutes and no (zero) seconds.

Notes:

- Truncated representations are permitted only at the leading or trailing portions of the duration value. This allows for efficiently representing durations which do not require very large or very small granularities. However, the final duration string can not omit intermediate values.
- If the digit ‘0’ (zero) precedes a non-zero digit (e.g. padding) it will be preserved and treated as a placeholder by processors.

Part Three: Supplemental Information

13 Authorization Table Schema

The following schema defines the XML format of the Content Cluster Authorization Table.

```
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema targetNamespace="http://www.ascct.org/10/authtable"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns:types="http://www.hanalliance.org/hana/21/types"
  elementFormDefault="qualified">

  <xsd:import namespace="http://www.hanalliance.org/hana/21/types"
      schemaLocation="hanatypes.xsd">
   </xsd:import>


   <xsd:simpleType name="IdentifierAttr">
     <xsd:restriction base="types:IdentifierType">
     </xsd:restriction>
   </xsd:simpleType>

   <xsd:simpleType name="deviceRoleType">
     <xsd:restriction base="types:String32Type">
     </xsd:restriction>
   </xsd:simpleType>

   <xsd:simpleType name="serviceRoleType">
     <xsd:restriction base="types:String32Type">
     </xsd:restriction>
   </xsd:simpleType>

   <xsd:complexType name="clusterType">
        <xsd:attribute name="id" type="authtable:IdentifierAttr" use="required"/>
        <xsd:attribute name="limit" type="xsd:unsignedInt" use="required"/>
        <xsd:attribute name="count" type="xsd:int" use="required"/>
   </xsd:complexType>

   <xsd:complexType name="macType">
      <xsd:attribute name="value" type="types:HashType" use="required"/>
      <xsd:attribute name="deviceclass" type="xsd:int" use="required"/>
      <xsd:attribute name="method" type="authtable:hashFunctionType" use="required"/>
   </xsd:complexType>

   <xsd:simpleType name="deviceStateType">
      <xsd:restriction base="types:String32Type">
          <xsd:enumeration value="authorized"></xsd:enumeration>
          <xsd:enumeration value="delete"></xsd:enumeration>
          <xsd:enumeration value="pending"></xsd:enumeration>
      </xsd:restriction>
   </xsd:simpleType>

   <xsd:simpleType name="hashFunctionType">
      <xsd:restriction base="types:String32Type">
          <xsd:enumeration value="cmac"></xsd:enumeration>
          <xsd:enumeration value="hmac"></xsd:enumeration>
      </xsd:restriction>
   </xsd:simpleType>

    <xsd:complexType name="deviceType">
      <xsd:attribute name="id" type="authtable:IdentifierAttr" use="required"/>
      <xsd:attribute name="state" type="authtable:deviceStateType" use="required"/>
      <xsd:attribute name="role" type="authtable:deviceRoleType" use="required"/>
      <xsd:attribute name="count" type="xsd:int" use="required"/>
   </xsd:complexType>


   <xsd:complexType name="serviceType">
      <xsd:attribute name="id" type="authtable:IdentifierAttr" use="required"/>
```

```
      <xsd:attribute name="role" type="authtable:serviceRoleType" use="required"/>
      <xsd:attribute name="urn" type="types:URNType" use="required"/>
      <xsd:attribute name="uri" type="types:locationType" use="required"/>
      <xsd:attribute name="count" type="xsd:int" use="required"/>
   </xsd:complexType>

    <xsd:complexType name="authorizationTableType">
      <xsd:sequence>
        <xsd:element name="cluster" type="authtable:clusterType" minOccurs="1"
maxOccurs="1"/>
        <xsd:element name="device" type="authtable:deviceType" minOccurs="1"
maxOccurs="unbounded"/>
        <xsd:element name="service" type="authtable:serviceType" minOccurs="1"
maxOccurs="unbounded"/>
        <xsd:element name="mac" type="authtable:macType" minOccurs="1"
maxOccurs="unbounded"/>
      </xsd:sequence>
   </xsd:complexType>

    <xsd:element name="AT" type="authtable:authorizationTableType"/>
</xsd:schema>
```

14 Message Schema

The following are the message schema use for the messages exchanged by HANA devices for performing Cluster Management and Content Directory transactions.

14.1 Common Message Types

This defines the XML types used in the message schema.

```
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema targetNamespace="http://www.hanalliance.org/hana/21/types"
   xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:ts="http://www.hanalliance.org/hana/21/types">

   <xsd:simpleType name="AEShashType">
      <xsd:restriction base="xsd:string">
         <xsd:length value="32" fixed="true"></xsd:length>
      </xsd:restriction>
   </xsd:simpleType>

   <xsd:simpleType name="locationType">
      <xsd:restriction base="xsd:anyURI">
         <xsd:minLength value="1"></xsd:minLength>
      </xsd:restriction>
   </xsd:simpleType>

   <xsd:simpleType name="typeType">
      <xsd:restriction base="xsd:string">
         <xsd:enumeration value="peer"></xsd:enumeration>
         <xsd:enumeration value="external"></xsd:enumeration>
         <xsd:minLength value="1"></xsd:minLength>
      </xsd:restriction>
   </xsd:simpleType>

   <xsd:simpleType name="macType">
      <xsd:restriction base="ts:AEShashType"></xsd:restriction>
   </xsd:simpleType>

   <xsd:simpleType name="MKBType">
      <xsd:restriction base="ts:AEShashType"></xsd:restriction>
   </xsd:simpleType>

   <xsd:simpleType name="ATType">
      <xsd:restriction base="ts:AEShashType"></xsd:restriction>
   </xsd:simpleType>

   <xsd:simpleType name="contentIDType">
      <xsd:restriction base="ts:AEShashType"></xsd:restriction>
   </xsd:simpleType>

   <xsd:simpleType name="deviceIDType">
      <xsd:restriction base="xsd:hexBinary">
         <xsd:length value="8" fixed="true"></xsd:length>
      </xsd:restriction>
   </xsd:simpleType>

   <xsd:simpleType name="clusterIDType">
      <xsd:restriction base="ts:AEShashType"></xsd:restriction>
   </xsd:simpleType>
</xsd:schema>
```

14.2 Cluster Management Messages

The following defines the XML for Cluster Management messages defined in section 9.1.

```
<?xml version="1.0" encoding="UTF-8"?>
```

```
<xsd:schema
   targetNamespace="http://www.hanaalliance.org/hana/21/cluster"
   xmlns:xsd="http://www.w3.org/2001/XMLSchema"
   xmlns:ts="http://www.hanalliance.org/hana/21/types"
xmlns:cs="http://www.hanaalliance.org/hana/21/cluster">
   <xsd:import namespace="http://www.hanalliance.org/hana/21/types"
      schemaLocation="hanatypes.xsd">
   </xsd:import>

   <xsd:complexType name="clusterMessageType">
      <xsd:choice minOccurs="1">
         <xsd:element name="authorizeme"
            type="cs:authorizemeMsgType">
         </xsd:element>
         <xsd:element name="imhere" type="cs:imhereMsgType"></xsd:element>
         <xsd:element name="getMKB" type="cs:getMKBMsgType"></xsd:element>
         <xsd:element name="getAT" type="cs:getATMsgType"></xsd:element>
      </xsd:choice>
   </xsd:complexType>

   <xsd:complexType name="clusterMessageResponseType">
      <xsd:choice minOccurs="1">
         <xsd:element name="imhere"
            type="cs:imhereResponseMsgType">
         </xsd:element>
         <xsd:element name="authorizeme"
            type="cs:authorizemeResponseType">
         </xsd:element>
         <xsd:element name="getMKB"
            type="cs:getMKBMessageResponseType">
         </xsd:element>
         <xsd:element name="getAT"
            type="cs:getATMessageResponseType">
         </xsd:element>
      </xsd:choice>

   </xsd:complexType>

   <xsd:complexType name="authorizemeMsgType">
      <xsd:sequence>
         <xsd:element name="clusterid" type="ts:clusterIDType"></xsd:element>
         <xsd:element name="deviceid" type="ts:deviceIDType"></xsd:element>
         <xsd:element name="location" type="ts:locationType"></xsd:element>
         <xsd:element name="type" type="ts:typeType"></xsd:element>
         <xsd:element name="mac" type="ts:macType"></xsd:element>
      </xsd:sequence>
   </xsd:complexType>

   <xsd:complexType name="imhereMsgType">
      <xsd:sequence>
         <xsd:element name="clusterid" type="ts:clusterIDType"></xsd:element>
         <xsd:element name="deviceid" type="ts:deviceIDType"></xsd:element>
         <xsd:element name="location" type="ts:locationType"></xsd:element>
         <xsd:element name="mkb" type="ts:MKBType"></xsd:element>
         <xsd:element name="at" type="ts:ATType"></xsd:element>
      </xsd:sequence>
   </xsd:complexType>

   <xsd:complexType name="getMKBMsgType">
      <xsd:sequence>
         <xsd:element name="clusterid" type="ts:clusterIDType"></xsd:element>
      </xsd:sequence>
   </xsd:complexType>

   <xsd:complexType name="getATMsgType">
      <xsd:sequence>
         <xsd:element name="clusterid" type="ts:clusterIDType"></xsd:element>
      </xsd:sequence>
   </xsd:complexType>
```

```
<xsd:element name="message" type="cs:clusterMessageType"></xsd:element>

<xsd:complexType name="authorizemeResponseType">
   <xsd:choice>
      <xsd:element name="authorized"
         type="cs:authorizedType">
      </xsd:element>
      <xsd:element name="rejected" type="cs:authorizeRejectType">
      </xsd:element>
   </xsd:choice>
</xsd:complexType>


<xsd:complexType name="authorizedType">
   <xsd:attribute name="bindingid" type="xsd:hexBinary">
      <xsd:annotation>
         <xsd:documentation>
            This is the Cluster Binding ID encrypted with the
            devices key Kp
         </xsd:documentation>
      </xsd:annotation>
   </xsd:attribute>
</xsd:complexType>

<xsd:complexType name="authorizeRejectType">
   <xsd:attribute name="reason"
      type="cs:authorizeRejectReasonType">
   </xsd:attribute>
</xsd:complexType>

<xsd:simpleType name="authorizeRejectReasonType">
   <xsd:restriction base="xsd:string">
      <xsd:enumeration value="busy"></xsd:enumeration>
      <xsd:enumeration value="invalid"></xsd:enumeration>
      <xsd:enumeration value="refused"></xsd:enumeration>
      <xsd:enumeration value="revoked"></xsd:enumeration>
      <xsd:enumeration value="unverified"></xsd:enumeration>
      <xsd:enumeration value="exists"></xsd:enumeration>
      <xsd:enumeration value="limit"></xsd:enumeration>
      <xsd:enumeration value="stopped"></xsd:enumeration>
   </xsd:restriction>
</xsd:simpleType>

<xsd:simpleType name="rejectReasonType">
   <xsd:restriction base="xsd:string">
      <xsd:annotation>
         invalid is the msg parameters were not correct
      </xsd:annotation>
      <xsd:enumeration value="invalid"></xsd:enumeration>
      <xsd:enumeration value="revoked">
         <xsd:annotation>
            revoked indicates the responding device is revoked
         </xsd:annotation>
      </xsd:enumeration>
      <xsd:enumeration value="refused"></xsd:enumeration>
   </xsd:restriction>
</xsd:simpleType>

<xsd:complexType name="imhereResponseMsgType">
   <xsd:choice>
      <xsd:element name="accepted"></xsd:element>
      <xsd:element name="rejected"
         type="cs:rejectReasonType">
      </xsd:element>
   </xsd:choice>
</xsd:complexType>


<xsd:complexType name="getMKBMessageResponseType">
   <xsd:choice>
```

```
         <xsd:element name="MKB" type="xsd:hexBinary">
            <xsd:annotation>the MKB</xsd:annotation>
         </xsd:element>
         <xsd:element name="rejected"
            type="cs:rejectReasonType">
         </xsd:element>
      </xsd:choice>
   </xsd:complexType>

   <xsd:complexType name="getATMessageResponseType">
      <xsd:choice>
         <xsd:element name="AT" type="xsd:hexBinary"></xsd:element>
         <xsd:element name="rejected"
            type="cs:rejectReasonType">
         </xsd:element>
      </xsd:choice>
   </xsd:complexType>


   <xsd:element name="response"
      type="cs:clusterMessageResponseType">
   </xsd:element>
</xsd:schema>
```