# Advanced Access Content System (AACS)

## *Introduction and Common Cryptographic Elements*

*Intel Corporation*
*International Business Machines Corporation*
*Matsushita Electric Industrial Co., Ltd.*
*Microsoft Corporation*
*Sony Corporation*
*Toshiba Corporation*
*The Walt Disney Company*
*Warner Bros.*

*Revision 0.91*
*February 17, 2006*

This page is intentionally left blank.

# Preface

## Notice



## Intellectual Property

Implementation of this specification requires a license from AACS LA LLC.

## Contact Information

Please address inquiries, feedback, and licensing requests to AACS LA LLC:

- Licensing inquiries and requests should be addressed to licensing@aacsla.com.
- Feedback on this specification should be addressed to comment@aacsla.com.

The URL for the AACS LA LLC web site is http://www.aacsla.com.

This page is intentionally left blank.

# Table of Contents

This page is intentionally left blank.

# List of Figures

This page is intentionally left blank.

# List of Tables

This page is intentionally left blank.

# Chapter 1
# Introduction

## 1.1 Purpose and Scope

The Advanced Access Content System (AACS) specification defines an advanced, robust and renewable method for protecting audiovisual entertainment content, including high-definition content. The specification is organized into several "books". This document, the *Introduction and Common Cryptographic Elements* book, describes the overall goals of AACS, and defines cryptographic procedures that are common among its various defined uses. Other books provide additional details specific to particular optical media types and formats. Those available at or around the time of this publication are:

- Pre-recorded Video Book (format-independent)
- Recordable Video Book (format-independent)
- Blu-ray Disc Pre-recorded Book (format-specific)
- Blu-ray Disc Recordable Book (format-specific)
- HD DVD and DVD Pre-recorded Book (format-specific)
- HD-DVD and DVD Pre-recorded Book Confidential Part for CE System (format-specific)
- HD-DVD and DVD Pre-recorded Book Confidential Part for PC-based System (format-specific)
- HD DVD Recordable Book (format-specific)

The use of this specification and access to the intellectual property and cryptographic materials required to implement it will be the subject of a license. A license authority referred to as AACS LA LLC (hereafter referred to as AACS LA) is responsible for establishing and administering the content protection system based in part on this specification.

## 1.2 Objectives and Design Criteria

AACS is designed to meet the following general criteria:

- Meet the content owners' requirements for robustness and system renewability
    - Content encryption based on a published cryptographic algorithm.
    - Limit access to protected content to only licensed compliant implementations.
    - Support revocation of individual compromised devices' keys.
    - Limit output and recording of protected content to a list of approved methods.
- Suitable for implementation on both general-purpose computer and fixed-function consumer electronics platforms.
- Applicable to both audio and video content, including high-definition video.
- Applicable to various optical media formats.
- Transparent to authorized use by consumers.

To meet these general objectives, AACS is based in part on the following technical elements:

- Robust encryption of protected content using the AES cipher.
- Key management and revocation using advanced Media Key Block technology.

Additional specific criteria and technical elements apply to particular uses of AACS, as described in other books of this specification.

## 1.3 Organization of this Document

This document is organized as follows:

- Chapter 1 provides an introduction.
- Chapter 2 describes core cryptographic functions, based on the AES cipher algorithm.
- Chapter 3 describes a cryptographic key management procedure using the Media Key Block.
- Chapter 4 describes the drive authentication scheme in common use for AACS media.
- Chapter 5 describes the use of AACS media in combination with an on-line connection.

## 1.4 References

This specification shall be used in conjunction with the following publications. When the publications are superseded by an approved revision, the revision shall apply.

AACS LA, *License agreement (not available at the time of this release)*

ANSI X9.31-1998, *Digital Signatures Using Reversible Public Key Cryptography for the Financial Services Industry (rDSA)*, ANSI, September 9, 1998. (Informative)

National Institute of Standards and Technology (NIST), DIGITAL SIGNATURE STANDARD (DSS), FIPS Publication 186-2 (+Change Notice), January 27, 2000.

National Institute of Standards and Technology (NIST), *A Statistical Test Suite for Random and Pseudorandom Number Generators for Cryptographic Applications*, NIST Special Publication 800-22, with revisions dated May 15, 2001.

National Institute of Standards and Technology (NIST), *Recommendation for Block Cipher Modes of Operation – Methods and Techniques*, NIST Special Publication 800-38A, 2001 Edition

National Institute of Standards and Technology (NIST), *Secure Hash Standard*, FIPS Publication 180-2, August 1, 2002.

National Institute of Standards and Technology (NIST), *Advanced Encryption Standard (AES)*, FIPS Publication 197, November 26, 2001.

National Institute of Standards and Technology (NIST), *Cipher-based Message Authentication Code (CMAC)*, NIST Special Publication 800-38B, May, 2005.

RSA Laboratories, *PKCS #1 (v2.1): RSA Cryptography Standard*, June 14, 2002.

Mt. Fuji Commands for Multimedia Devices Version 6 Revision 0.7 or later

Internet Engineering Task Force (http://www.ietf.org), The TLS Protocol. Version 1.0 (RFC 2246)

D. Naor, M. Naor, and J. Lotspiech. “Revocation and Tracing Schemes for Stateless Receivers”. In Advances in Cryptology - CRYPTO 2001.Springer-Verlag Inc. LNCS 2139, 2001, 41-62.

## 1.5 Document History

This document version 0.91 supersedes version 0.90 dated April 13, 2005. It contains editorial improvements since the 0.90 version, plus the following changes:

- Two-way drive/host authentication replaces the previous one-way authentication method.
- The number of permissible random number generators has been reduced.
- An AES-based hash has been defined for use in some calculations involving keys.
- A Media Key Block type field has been defined.
- All devices are now required to check the signature on the Media Key Block.
- A new Host Revocation List record has been added to the Media Key Block.
- Added clarifying language for the purpose of KCD.
- A definition of Partial MKB has been added.
- Moved Managed Copy to Pre-recorded Video book
- Added ISAN as a requirement for the Content ID defined in chapter 5

## 1.6 Future Directions

With its advanced, robust cryptography, key management and renewal mechanisms, it is expected that this technology will develop and expand, through additions to this specification, to address content protection for additional storage types, application formats and usage models, as authorized by AACS LA.

## 1.7 Notation

### 1.7.1 Numerical Values

This specification uses three different representations for numerical values. Decimal numbers are represented without any special notation. Binary numbers are represented as a string of binary (0, 1) digits followed by a subscript 2 (e.g., $1010_2$). Hexadecimal numbers are represented as a string of hexadecimal (0..9, A..F) digits followed by a subscript 16 (e.g., $A3C2_{16}$).

### 1.7.2 Bit and Byte Ordering

Certain data values or parts of data values are interpreted as an array of bits. Unless explicitly noted otherwise, bit positions within an n-bit data value are numbered such that the least significant bit is numbered 0 and the most significant bit is numbered n-1.

Unless explicitly noted otherwise, big-endian ordering is used for multiple-byte values, meaning that byte 0 is the most significant byte.

### 1.7.3 Operations

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

| | |
|---|---|
| $[x]_{msb\_z}$ | The most significant z bits of x. |
| $[x]_{lsb\_z}$ | The least significant z bits of x. |
| $[x]_{y:z}$ | The inclusive range of bits between bit y and bit z in x. |
| ~x | Bit-wise inversion of x. |
| x \|\| y | Ordered concatenation of x and y. |
| x ⊕ y | Bit-wise Exclusive-OR (XOR) of two strings x and y. |
| x + y | Modular addition of two strings x and y. |
| x × y | Multiplication of x and y. |
| x – y | Subtraction of y from x. |

The following assignment and relational operators will be used:

| | |
|---|---|
| = | Assignment |
| == | Equal to |
| != | Not equal to |
| < | Less than |
| > | Greater than |
| <= | Less than or equal to |
| >= | Greater than or equal to |

## 1.8 Terminology

This section contains definitions of terms used in this document for which the definitions could not easily be obtained from other sources. Standard cryptographic or multi-media terms are not defined. In addition to terms that are unique to AACS, some common terms such as “Title” or “Format” are defined because they are often used in different contexts and may have multiple or mis-leading definitions. A definition of these “common terms” has been provided in order to convey the precise meaning as used in this specification.

Application

This term is used as a reference to a particular type of content being protected by AACS. Examples include Commercial Movies produced by a movie studio or Commercial Audio content produced by a record label.

Binding Nonce

A value used to control access to data items stored in the Protected Area of a Recordable media and to prevent “Rollback” Attacks. It also serves to cryptographically bind content to that instance of Recordable media.

Format

This term is used as a reference to the underlying physical format which will store the content being protected by AACS. Examples include HD DVD and Blu-ray.

Media Identifier

An identifier that is unique to each Recordable media. It is used to cryptographically bind content to that instance of Recordable media and to prevent indiscriminate copying to other media.

Media Key

A key that is used to unlock the Title Keys stored on a media that contains Titles protected by AACS. The Media Key can be computed by successfully processing a MKB.

Media Key Block (MKB)

A critical component of the subset difference tree key management system. The MKB is a data block that provides access to a common key (Media Key) that can be accessed by any device that contains the necessary secret keys and has not been revoked. Refer to Chapter 3 for additional details.

Pre-recorded Media Serial Number

Each piece of media may have a unique identifier. The Serial Numbers do not have to be unique across different Content IDs, although they may. The Serial Numbers also may have a self-checking property such that they cannot be easily forged. If the media is an optical disc, it might be recorded in the Burst Cutting Area to enable licensed replicators to record unique values for each disc.

Protected Area

A section of data storage on Recordable media that is both readable and writable but only by the drive. On systems that require Drive Authentication, the host can only read data values from the Protected Area by using the Drive Authentication process and the host can use Drive Authentication process to request the drive to write data to the Protected Area but can not specify the actual data values to be written to the Protected Area.

Subset Difference Tree or NNL-Tree

A tree-based key management system based on broadcast encryption. The only broadcast encryption system with equivalent revocation power as a public key system.

Title Key

The key used to encrypt a Title.

Volume Identifier

An identifier that is unique to each individual item of content. This identifier is stored onto Pre-recorded media in a way that can not be duplicated on Recordable media. This prevents content stored on Pre-recorded media from being indiscriminately copied to Recordable media. This identifier will not be unique to each individual instance of a media.

## 1.9 Abbreviations and Acronyms

The following are alphabetical lists of abbreviations and acronyms used in this and other books that make up the complete AACS technical specifications:

Industry Standard Acronyms:

| | |
|---|---|
| AES | Advanced Encryption Standard |
| ANSI | American National Standards Institute |

| | |
|---|---|
| BCA | Burst cutting area |
| BD | Blu-ray Disc |
| CBC | Cipher Block Chaining |
| CE | Consumer Electronics |
| CMAC | Cipher based Message Authentication Code |
| CSS | Content Scramble System |
| DHCP | Dynamic Host Configuration Protocol |
| DSA | Digital Signature Algorithm |
| DVD | Digital Versatile Disc |
| ECB | Electronic Codebook |
| ECDSA | Elliptic Curve based Digital Signature Algorithm |
| FIPS | Federal Information Processing Standards |
| HD DVD | High-Definition Digital Versatile Disc |
| HTTP | Hypertext Transfer Protocol |
| HTTPS | Secure Hypertext Transfer Protocol |
| MAC | Message Authentication Code |
| MPEG | Moving Picture Experts Group |
| NIST | National Institute of Standards and Technology |
| PC | Personal Computer |
| SHA | Secure Hashing Algorithm |
| URL | Uniform Resource Locator |
| XML | eXtensible Markup Language |

AACS Acronyms:

| | |
|---|---|
| AACS | Advanced Access Content System |
| AACS LA | AACS Licensing Administrator, LLC |
| CHT | Content Hash Table |
| CRL | Content Revocation List |
| DRL | Drive Revocation List |
| HRL | Host Revocation List |
| KCD | Key Conversion Data |
| MKB | Media Key Block |
| TUF | Title Usage File. |

This page is intentionally left blank.

# Chapter 2
# AACS Common Cryptographic Functions

## 2. Introduction

This chapter describes common cryptographic functions upon which protection mechanisms described in this specification are based. The functions are described here in isolation; their specific uses as part of encryption, key management and renewability mechanisms are described elsewhere in this document, as well as in other books of this specification.

### 2.1 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 (see reference in Section 1.4). Unless otherwise specified, the AES algorithm is used with data blocks of 128 bits and keys with lengths of 128 bits.

#### 2.1.1 ECB Mode (AES-128E and AES-128D)

For purposes such as management of cryptographic keys, the AES cipher is used with the Electronic Codebook (ECB) mode of operation specified in NIST Special Publication 800-38A (see reference in Section 1.4).

Hereafter, encryption using the AES algorithm in ECB mode as just described is represented by the function

AES-128E(k, d)

where d is a 128-bit data value to be encrypted, k is a 128-bit key, and AES-128E returns the 128-bit encryption result.

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

AES-128D(k, d)

where d is a 128-bit data value to be decrypted, k is a 128-bit key, and AES-128D returns the 128-bit decryption result.

#### 2.1.2 CBC Mode (AES-128CBCE and AES-128CBCD)

For purposes such as encryption and decryption of protected content, the AES cipher is used with the Cipher Block Chaining (CBC) mode of operation specified in NIST Special Publication 800-38A (see reference in Section 1.4).

Hereafter, encryption using the AES algorithm in CBC mode as just described is represented by the function

AES-128CBCE(k, d)

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

Decryption using the AES algorithm in CBC mode as described above (using either the “inverse cipher” or “equivalent inverse cipher” specified in FIPS Publication 197) is represented by the function

AES-128CBCD(k, d)

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

The size of the frame of data to be encrypted or decrypted (i.e. how often a new CBC chain is started) depends on the particular application, and is defined for each in the corresponding format specific books of this specification. Unless otherwise specified, the Initialization Vector used at the beginning of a CBC encryption or decryption chain is a constant, $iv_{0,}$ which is:

$0BA0F8DDFEA61FB3D8DF9F566A050F78_{16}$

### 2.1.3 AES-based One-way Function (AES-G)

AACS uses a cryptographic one-way function based on the AES algorithm. This function is referred to as the AES-based One-way Function, and is represented by

$AES\text{-}G(x_1, x_2)$

where $x_1$ and $x_2$ are 128-bit input values, and $AES\text{-}G(x_1, x_2)$ returns the 128-bit result.

Figure 2-1 depicts the AES-based One-way Function.

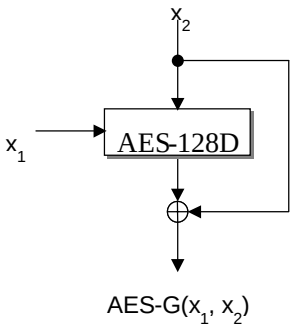



**Figure 2-1 – AES-based One-way Function**

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

$AES\text{-}G(x_1, x_2) = AES\text{-}128D(x_1, x_2) \oplus x_2$.

### 2.1.4 AES Hashing Function (AES_H)

For the purpose of processing data to produce a condensed representation in certain calculations involving keys, a hashing procedure based on the AES algorithm is used. This procedure, referred to as the AES-based Hashing Function, is represented by

AES-H(x)

where x is input data of arbitrary length, and AES-H(x) returns the corresponding 128-bit hash value.

Prior to hashing, the data to be hashed (x) is padded using the standard SHA-1 method, to wit: The message or data file should be considered to be a bit string. The length of the message is the number of bits in the message (the empty message has length 0). The purpose of message padding is to make the total length of a padded message a multiple of 128 bits. The AES hash sequentially processes blocks of 128 bits when computing the message digest. The following specifies how this padding shall be performed. As a summary, a "1" followed by m "0"s followed by a 64-bit integer are appended to the end of the message to produce a padded message of length 128 * n. The 64-bit integer is the length of the original message in bits. By way of example, a 56-bit message would be padded with 72 bits as follows: $80000000000000038_{16}$. A 64-bit message would be padded with 192 bits as follows: $80\ldots040_{16}$. A 128-bit message would be padded with 128 bits as follows: $80\ldots080_{16}$.

The padded data x' is divided into n 128-bit blocks, represented as $x'_1, x'_2, \ldots, x'_n$, which are used in calculating the hash as shown in Figure 2-2.

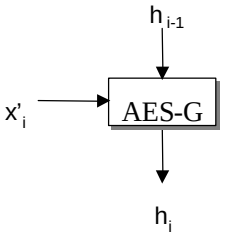



**Figure 2-2 – AES-based Hashing Function**

A 128-bit initial value $h_0$ is given by the following constant:

$2DC2DF39420321D0CEF1FE2374029D95_{16}$

For i from 1 to n, the 128-bit value $h_i$ is calculated as

$h_i = \text{AES-G}(x'_i , h_{i-1})$.

The value $h_n$ is the result of the hashing function, i.e., $\text{AES-H}(x) = h_n$.

### 2.1.5 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.

### 2.1.6 Message Authentication Code (CMAC)

For the purpose of generating a Message Authentication Code to protect the integrity of information, the Cipher-based Message Authentication Code (CMAC), as defined in the National Institute of Standards and Technology Special Publication 800-38B, is used. This standard requires the use of a symmetric key block cipher for which the Advanced Encryption Standard (AES), as defined in Federal Information Processing Standards Publication 197, is used.

Hereafter, using the CMAC to create a message authentication code is represented by the following function:

M = CMAC(k, D)

where k is the key to be used to create the MAC, D is the data to be authenticated and M is the resulting MAC. The 128-bit length of MAC is used in this specification.

## 2.2 Random/Pseudorandom Number Generation

This section describes random/pseudorandom number generators for use in generating values such as cryptographic keys. Unless specifically noted otherwise, one or plural of the following random/pseudorandom number generators shall be used. (1) Pseudorandom number generator based on a design described in ANSI X9.31 (see reference in Section 1.4) that is illustrated in Figure 2-3 and described below (2) Pseudorandom number generators defined in FIPS PUB 186-2 (+Change Notice) (see reference in Section 1.4) (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 (see reference in Section 1.4)

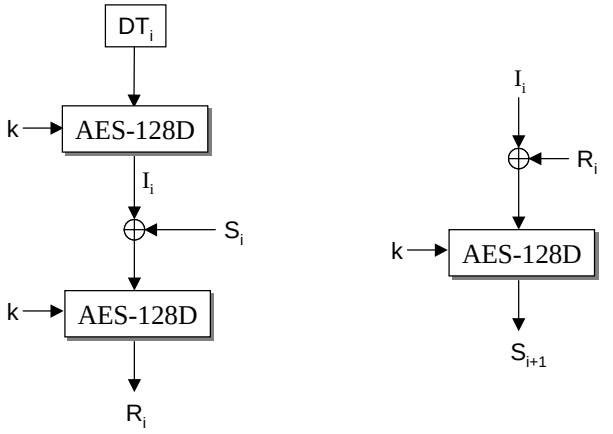



**Figure 2-3 – Random/pseudorandom Number Generator Example**

In the figure, k is a 128-bit constant value, DT is a 128-bit value that is updated on each iteration (such as a date/time vector or monotonic counter) and S is a seed value.

The combination of fixed value k and initial seed value $S_0$ shall be unpredictable and unique per licensed product, and S shall be maintained in a non-volatile register, in which case a source of entropy is not required and DT must be ensured to be non-repeating only until the next time the licensed product is re-started.

128-bit random values $R_i$ (i=0,1,…) are generated via the following calculations:

$I_i$ = AES-128D(k, $DT_i$),

$R_i$ = AES-128D(k, $I_i \oplus S_i$), and

$S_{i+1}$ = AES-128D(k, $I_i \oplus R_i$).

Unless explicitly noted otherwise, the values k and S shall be treated as highly confidential as described in the license agreement.

## 2.3 Digital Signature (AACS_Sign and AACS_Verify)

All digital signatures in AACS utilize the ECDSA digital signature scheme defined in[American National Standard for Financial Services publication X9.62.] FIPS PUB 186-2 (+Change Notice) (see reference in Section 1.4).

The following table shows parameter values of the curve that shall be used in conjunction with ECDSA

**Table 2-1 – ECC Parameters**

| ECC Parameter | Value |
|---|---|
| p | 900812823637587646514106462588455890498729007071 |
| a | -3 |
| b | 366394034647231750324370400222002566844354703832 |
| Base Point (G) | 264865613959729647018113670854605162895977008838 |
| | 518410759548831625104133927451689362961878086 97 |
| Order of Base Point (r) | 900812823637587646514106555566573588779770753047 |

All operations in the elliptic curve domain are calculated on an elliptic curve defined over GF(p). The public key $K_{pub}$ (a point on the elliptic curve) and private key $K_{priv}$ (a scalar value satisfying $0 < K_{priv} < r$) shall satisfy the equation:

$K_{pub} = K_{priv} G$

Hereafter, using ECDSA to create a digital signature is represented by the following function:

S = AACS_Sign($K_{priv}$, D)

where $K_{priv}$ is the private key to be used for the signature, D is the data to be signed, and S is the resulting signature.

Hereafter, using ECDSA to verify a digital signature is represented by the following function:

AACS_Verify($K_{pub}$, S, D)

where $K_{pub}$ is the public key to be used to verify the signature, S is the signature to be verified, and D is the data for which the signature is being verified.

All devices are required to store the AACS LA's root public key, denoted $AACS\_LA_{pub}$, in a way that resists malicious modification. For certain applications, such as content signing, the AACS LA defines additional public keys. Devices supporting those applications must, in addition, store those keys, as defined in other books in this specification.

# Chapter 3
# AACS Common Cryptographic Key Management

## 3. Introduction

This chapter describes an advanced cryptographic key management procedure, depicted in Figure 3-1, which uses a Media Key Block, based on the subset-difference tree method, to provide system renewability in the form of device revocation. The procedure is described here in isolation; its use as part of the overall protection system is described elsewhere in this specification.

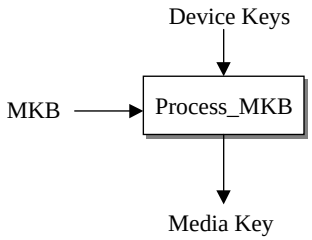



**Figure 3-1 – Common AACS Cryptographic Key Management Procedure**

Device Keys ($K_{d\_0}$,$K_{d\_1}$,…,$K_{d\_n-1}$) are used to decrypt one or more elements of a Media Key Block (MKB), in order to extract a secret Media Key ($K_m$). Table 3-1 lists the elements involved in this process, along with their sizes.

**Table 3-1 – Common Cryptographic Key Management Elements**

| Key or Variable | Size |
|---|---|
| Device Keys ($K_{d\_0}$,$K_{d\_1}$,…,$K_{d\_n-1}$) | 128 bits each |
| Media Key Block (MKB) | Variable, multiple of 4 bytes |
| Media Key ($K_m$) | 128 bits |

The remainder of this section describes this cryptographic key management procedure in detail.

## 3.1 Device Keys

Each compliant device is given a set of secret Device Keys when manufactured. The actual number of keys may be different in different media types. These Device Keys, referred to as $K_{d\_i}$ (i=0,1,…,n-1), are provided by AACS LA, and are used by the device to process the MKB to calculate $K_m$. The set of 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. A device shall treat its Device Keys as highly confidential, as defined in the license agreement.

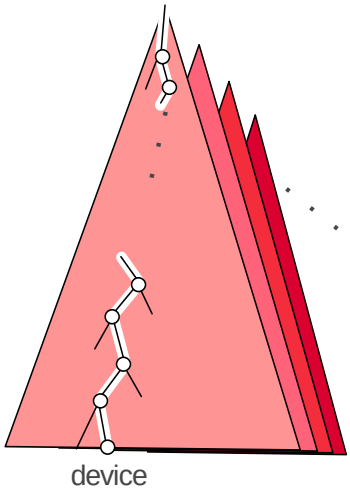
## 3.2 Media Key Block (MKB)

The Media Key Block (MKB) enables system renewability. The MKB is generated by AACS LA, and allows all compliant devices, each using their set of secret Device Keys, to calculate the same $K_m$. If a set of Device Keys is compromised in a way that threatens the integrity of the system, an updated MKB can be released that causes a device with the compromised set of Device Keys to be unable to calculate the correct $K_m$. In this way, the compromised Device Keys are "revoked" by the new MKB. Compliant devices shall be able to locate the MKB on media as defined in the format specific books of this specification.

### 3.2.1 Subset-Difference Tree Overview (Informative)

The technology used for the MKB is referred to as the subset-difference approach. 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.

The system is based on a large master tree of keys, where each device is uniquely associated with a leaf node of the tree. Each device receives a set of Device Keys. Each set of Device Keys contains exactly one unique "Leaf Key". A given set of Device Keys enables derivation of every key in the master tree *except* the keys between its leaf and the root (lower level keys are picked to be robust one-way functions of higher level keys, such that it is possible to calculate a key lower in the tree given a key higher in the tree). So, a device with a given set of Device Keys cannot derive its own Leaf Key, but any device with any other Device Key set can. Further, corresponding to every *sub-tree* in the master tree is another system of keys. For example, one level down from the root of the master tree there are two sub-trees, each with its own system (tree) of keys, in addition to the system of keys in the master tree. Likewise, in the next level down in the master tree there are four sub-trees, each with its own system of keys. By the time you reach the bottom of the tree, each pair of devices belongs to their own sub-tree of height 1. For each sub-tree corresponding to a node in the master tree between a given device's leaf and the root, that set of Device Keys enables derivation of every key in that sub-tree *except* the keys between its leaf and the root within that sub-tree. The figure on the right illustrates the system of keys from the point of view of a single device, with sub-trees shadowing off to the right. The nodes in white correspond to keys that *cannot* be derived using that set of Device Keys.




This scheme enables efficient revocation of any combination of Leaf Keys. To revoke only a single Leaf Key (i.e., to enable calculation of the Media Key by any set of Device Keys *except* the set containing that Leaf Key), the Media Key Block will include the Media Key encrypted only by the master tree's Leaf Key that is being revoked. If the Media Key is encrypted using a key located higher in the tree, the effect will be revocation of a contiguous range of Leaf Keys which are commonly rooted at that node. Efficient revocation of noncontiguous Leaf Keys can be accomplished through use of the sub-tree key systems; a key associated with a single contiguous group of revoked Leaf Keys in a single sub-tree is used to encrypt the Media Key. This encryption entry in the MKB allows devices that do not have that key to successfully compute the Media Key while simultaneously preventing all devices containing the revoked Leaf Keys from successfully computing the Media Key because they do contain that key in their Device Key set. A similar encryption is repeated with other sub-trees, until all devices containing the revoked Leaf Keys are unable to compute the Media Key and all other

devices can. It is this mechanism that gives this method its name. Sub-trees identify subsets, and each subset has a set of contiguous revoked nodes, the "difference".

Thus, the subset-difference tree has to store at least encryption for each revoked Leaf Key, and occasionally additional encryptions to pick up non-revoked sets not covered by the smaller sub-trees. On average, there are 1.28 encryptions per revocation. Various optimizations are possible that influence MKB size and/or device requirements, which are not described here. The following sections describe the details of the subset-difference tree MKB as used by AACS.

### 3.2.2 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 Figure 3-2.

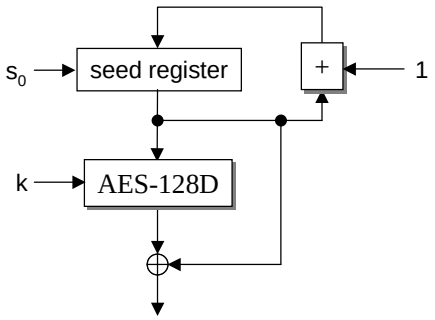



**Figure 3-2 – Triple AES Generator (AES-G3)**

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 "$s_0$", which is given by the following constant:

$7B103C5DCB08C4E51A27B01799053BD9_{16}$

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, $s_0$) $\oplus$ $s_0$.
2. The second 128 bits is the Processing Key, AES-128D(k, $s_0$+1) $\oplus$ ($s_0$+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) $\oplus$ ($s_0$+2).

By using the AES-G3 function in this way, the device can calculate all subsidiary Device Keys that it needs from the few Device Keys that it stores at manufacturing time.

### 3.2.3 Storing Device Keys

Each device is given its Device Keys and a 31-bit number *d* called the *device number*. For each Device Key, there is an associated number denoted the *path* number, and the "*u*" bit mask, $m_u$, and the "*v*" bit mask, $m_v$. The *path* number denotes the position in the tree associated with the Device Key. This *path* number defines a path 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. These 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 device number, *path* number, and masks denote nodes within a binary tree, where *u* is an ancestor of *v*. These masks represent the depth of the respective nodes, *u* and *v*, from the root of the tree. The deeper the position of a node in the tree, the shorter the sequence of 0-bits in the mask associated to that node.

As a result, the $m_u$ mask always has more 0 bits than the $m_v$ mask. The subset-difference is the subtree rooted at node *u* minus the sub-tree rooted at node *v*.

For conciseness, the *path* number and the "*v*" mask are encoded in a single 32-bit number, referred to as the *uv* number. The mask for *v* is given by the first lower-order 1-bit in the *uv* number. That bit, and all lower-order 0-bits, are zero bits in the "*v*" mask. 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;
```

By the same token, AACS distinguishes between *device numbers* and *device node numbers*, and uses the latter in the key order format. 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.

### 3.2.4 Calculation of Media Key

The Media 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 is associated with the 23rd section of the Media Key Data field; that is, it begins at offset (23-1)*16 from the start of the Media Key Data field of the Media Key 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:

$$((d \,\&\, m_u) == (uv \,\&\, m_u)) \text{ and } ((d \,\&\, m_v) != (uv \,\&\, m_v))$$

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 will either have the Device Key, or will be 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:

$$(m_u == m'_u) \text{ and } ((uv \,\&\, m'_v) == (uv' \,\&\, m'_v))$$

If $m'_v$ equals $m_v$, the starting Device Key is the final Device Key, and can be 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.

2. Use AES-G3 on D, as described above, to determine a left subsidiary Device Key, a Processing Key, and a right subsidiary Device Key.
3. 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.
4. *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 Media Key $K_m$ as follows:

$$K_m = \text{AES-128D}(K, C) \oplus (000000000000000000000000_{16} \,||\, uv)$$

The appropriate encrypted key data C is found in the Media Key Data Record in the Media Key Block.

A device may discover, while processing the Media 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.

### 3.2.5 Media Key Block Format

A Media 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 calculates $K_m$ by processing Records of the MKB one-by-one, in order, from first to last. The device must not make any assumptions about the length of Records, and must 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.

A properly formatted MKB shall have exactly one *Verify Media Key* Record, one *Type and Version* Record, one *Explicit Subset-Difference* Record, one *Subset-Difference Index* Record, one *Media Key Data* Record, one *Drive Revocation List* Record, one *Host Revocation List* Record, and one *End of Media Key Block* Record. If an MKB contains duplicate records of any of these record types, or the MKB is otherwise improperly formatted, the device behavior shall be manufacturer specific. If an MKB is missing any of these record types, the device shall not process the MKB.

#### 3.2.5.1 Type and Version Record

**Table 3-2 – *Type and Version* Record Format**

| Bit / Byte | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| 0 | Record Type: $10_{16}$ | | | | | | | |
| 1 | Record Length: $00000C_{16}$ | | | | | | | |
| 2 | | | | | | | | |
| 3 | | | | | | | | |
| 4 | MKBType: $000x1003_{16}$ | | | | | | | |
| 5 | | | | | | | | |

| | |
|---|---|
| 6 | |
| 7 | |
| 8 | Version Number |
| 9 | |
| 10 | |
| 11 | |

A properly formatted Media 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 Media Key Block is, in fact, more recent than the Media Key Block that is currently on the media. The Version Number is a 32-bit unsigned integer. Each time the AACS LA changes the revocation, it increments the version number and inserts the new value in subsequent Media Key Blocks. Thus, larger values indicate more recent Media Key Blocks. The Version Numbers begin at 1; 0 is a special value used for test Media Key Blocks.

For AACS applications, the MKBType field is one of two values:

**$00031003_{16}$** (Type 3). This is a normal Media Key Block suitable for being recorded on a recordable media.

**$00041003_{16}$** (Type 4). This is a Media Key Block that has been designed to use Key Conversion Data (KCD). Thus, it is suitable only for pre-recorded media from which the KCD can be derived.

It is not an error for a Type 3 Media Key Block to be used for controlling access to content on pre-recorded media. In this case, the device shall not use the KCD.

### 3.2.5.2 Host Revocation List Record

**Table 3-3 - *Host Revocation List* Record**

| Byte \ Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| 0 | Record Type: $21_{16}$ | | | | | | | |
| 1 | Record Length | | | | | | | |
| 2 | | | | | | | | |
| 3 | | | | | | | | |
| 4 | Total Number of Entries | | | | | | | |
| ... | | | | | | | | |
| 7 | | | | | | | | |
| 8 | Number of Entries in this Signature Block ($N_1$) | | | | | | | |
| ... | | | | | | | | |
| 11 | | | | | | | | |
| 12 | Host Revocation List Entry (0) | | | | | | | |
| … | | | | | | | | |
| 19 | | | | | | | | |
| 20 | Host Revocation List Entry (1)<br>...<br>Host Revocation List Entry ($N_1$ - 2) | | | | | | | |
| ... | | | | | | | | |
| (12+ ($N_1$-1)*8)-1 | | | | | | | | |
| (12+ ($N_1$-1)*8) | Host Revocation List Entry ($N_1$ - 1) | | | | | | | |
| … | | | | | | | | |

| | |
|---|---|
| (12+ $N_1$*8)-1 | |
| (12+ $N_1$*8) | Signature for Block 1 |
| | |
| (52+ $N_1$*8)-1 | |
| (52+ $N_1$*8) | Number of Entries in this Signature Block ($N_2$) |
| (53+ $N_1$*8) | |
| (54+ $N_1$*8) | |
| (55+ $N_1$*8) | |
| (56+ $N_1$*8) | Host Revocation List Entries… |
| … | |
| (56+($N_1$+$N_2$)*8)-1 | |
| (56+($N_1$+$N_2$)*8) | Signature for Block 1 and 2 |
| … | |
| (96+($N_1$+$N_2$)*8)-1 | |
| (96+($N_1$+$N_2$)*8) | More Signature Blocks… |
| … | |
| Length – 1 | |

A properly formatted Media Key Block shall have exactly one *Host Revocation List* Record as its second record. This record provides a list of hosts that have been revoked by the AACS LA. The AACS specification is applicable to PC-based system where a drive and PC host act together as the Recording Device and/or Playback Device for AACS protected content. AACS uses a drive-host authentication protocol for the host to verify the integrity of the data received from the drive, and for the drive to check the validity of the host application. The *Type and Version* Record and the *Host Revocation List* Record are guaranteed to be the first two records of a Media Key Block, to make it easier for drives to extract this data from an arbitrary Media Key Block.

The drive shall keep the highest-version-number *Host Revocation List* Record it has seen. (The version number is found in the previous *Type and Version* Record.) During authentication, the drive shall check that the Host ID in the Host Certificate is not in its host revocation list. These first two records combined together, i.e. the Type and Version Record followed by the Host Revocation List Record, are also referred to as a Partial Media Key Block or Partial MKB for the purpose of storing the HRL in the drive.

A Host Revocation List Record consists of:

- A 1-byte Record Type value, where $21_{16}$ shall be used to indicate a *Host Revocation List* Record.
- A 3-byte Record Length field value that indicates the number of bytes in the Record, including the Record Type and the Record Length fields themselves.
- The total number of Host Revocation List Entry fields that follow.
- The number of Host Revocation List Entry fields in the first signature block.
- A list of 8-byte Host Revocation List Entry fields, the length of this list being equal to the number in the first signature block.
- A signature verifiable using the $AACS\_LA_{pub}$ on the fields. This signature covers the entire *Type and Version* Record, and also the data in the *Host Revocation List* Record beginning with the type byte and ending with the byte immediately preceding the signature. For the first signature block, the data being signed together with the signature itself shall not exceed 32KB. Drives must verify the signature using the following procedure.

  AACS_Verify(AACS_$LA_{pub}$, Signature Data, *Type and Version* and *Host Revocation List*)

- If there are more Host Revocation List Entry fields than fit within the 32KB limit, there will be one or more subsequent signature blocks. Each block begins with a four-byte “number of entries in this

signature block" field, a list of that number of Host Revocation List Entry fields, and a signature. The signature covers the entire list, not just that signature block. In other words, the subsequent signatures cover the entire *Type and Version* Record, and the data in this HRL record up to, but not including, the signature itself. Drives only need to verify the signature on the block(s) they intend to store, and then only if the version number is more recent than the one that they have previously stored.

Drives are required to store only the data being signed for the first signature block, but not required to store the signature itself. Furthermore Drives are not required to store subsequent signature blocks. However, they may do so if they so choose.

**Table 3-4 – Host Revocation List Entry**

| Bit / Byte | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| 0 | Range | | | | | | | |
| 1 | | | | | | | | |
| 2 | Host ID | | | | | | | |
| 3 | | | | | | | | |
| 4 | | | | | | | | |
| 5 | | | | | | | | |
| 6 | | | | | | | | |
| 7 | | | | | | | | |

A Host Revocation List Entry includes:

- A 2-byte Range value indicates the range of revoked ID's starting from the ID contained in the record. A value of zero in the Range field indicates that only one ID is being revoked.
- A 6-byte Host ID value identifying the host being revoked.

The length of the *Host Revocation List* Record is always a multiple of four bytes.

### 3.2.5.3 Drive Revocation List Record

**Table 3-5 - *Drive Revocation List* Record**

| Byte \ Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| 0 | Record Type: $20_{16}$ | | | | | | | |
| 1 | Record Length | | | | | | | |
| 2 | | | | | | | | |
| 3 | | | | | | | | |
| 4 | Total Number of Entries | | | | | | | |
| ... | | | | | | | | |
| 7 | | | | | | | | |
| 8 | Number of Entries in this Signature Block ($N_1$) | | | | | | | |
| ... | | | | | | | | |
| 11 | | | | | | | | |
| 12 | Drive Revocation List Entry (0) | | | | | | | |
| … | | | | | | | | |
| 19 | | | | | | | | |
| 20 | Drive Revocation List Entry (1)<br>...<br>Drive Revocation List Entry ($N_1$ - 2) | | | | | | | |
| ... | | | | | | | | |
| (12+ ($N_1$-1)*8)-1 | | | | | | | | |
| (12+ ($N_1$-1)*8) | Drive Revocation List Entry ($N_1$ - 1) | | | | | | | |
| … | | | | | | | | |
| (12+ $N_1$*8)-1 | | | | | | | | |
| (12+ $N_1$*8) | Signature for Block 1 | | | | | | | |
| | | | | | | | | |
| (52+ $N_1$*8)-1 | | | | | | | | |
| (52+ $N_1$*8) | Number of Entries in this Signature Block ($N_2$) | | | | | | | |
| (53+ $N_1$*8) | | | | | | | | |
| (54+ $N_1$*8) | | | | | | | | |
| (55+ $N_1$*8) | | | | | | | | |
| (56+ $N_1$*8) | Drive Revocation List Entries… | | | | | | | |
| … | | | | | | | | |
| (56+($N_1$+$N_2$)*8)-1 | | | | | | | | |
| (56+($N_1$+$N_2$)*8) | Signature for Block 1 and 2 | | | | | | | |
| … | | | | | | | | |
| (96+($N_1$+$N_2$)*8)-1 | | | | | | | | |
| (96+($N_1$+$N_2$)*8) | More Signature Blocks… | | | | | | | |
| … | | | | | | | | |
| Length – 1 | | | | | | | | |

A properly formatted Media Key Block contains exactly one *Drive Revocation List* Record. It follows the *Host Revocation List* Record, although it may not immediately follow it.

The *Drive Revocation List* Record is identical to the *Host Revocation List* Record, except it has type $20_{16}$, and it contains Drive Revocation List Entries, not Host Revocation List Entries. The Drive Revocation List Entries refer to Drive IDs in the Drive Certificates.

The signature for each signature block covers the entire *Type and Version* Record, and also the data in the *Drive Revocation List* Record beginning with the Record Type byte and ending with the byte immediately preceding the signature. For the first signature block, the data being signed together with the signature itself shall not exceed 32KB. Hosts must verify the signature using the following procedure.

AACS_Verify(AACS_LA$_{pub}$, Signature Data, *Type and Version* and *Drive Revocation List*)

Hosts are required to store only the data being signed for the first signature block, but not required to store the signature itself, Furthermore, hosts are not required to store subsequent signature blocks. However, they may do so if they so choose.

The host shall keep the highest-version-number *Drive Revocation List* Record it has seen. The host shall check that the Drive ID in the Drive Certificate is not in the *Drive Revocation List* Record during the authentication process.

A Drive Revocation List Entry includes:

- A 2-byte Range value indicates the range of revoked ID's starting from the ID contained in the record. A value of zero in the Range field indicates that only one ID is being revoked.
- A 6-byte Drive ID value identifying the drive being revoked.

The length of the *Drive Revocation List* Record is always a multiple of four bytes.

### 3.2.5.4 Verify Media Key Record

**Table 3-6 – *Verify Media 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 Media Key* Record. It shall precede the *Explicit Subset Difference* Record, the *Subset Difference Index* Record, and the *Media 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, and $K_m$ is the correct final Media Key value.

The presence of the *Verify Media Key* Record in an MKB is mandatory. The device may use the *Verify Media Key* Record to verify the correctness of a given MKB, or of its processing of it. If everything is correct, the device should observe the condition:

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

where $K_m$ is the Media Key value.

Note that for a device that is required to use *Key Conversion Data* (KCD), processing of the Media Key Block will result in a *Media Key Precursor* $K_{mp}$ instead of a Media Key. The rules applicable to devices that use KCD are in the AACS Compliance Rules. The use of $K_{mp}$ in devices prevents the direct use of compromised Device Keys from those devices in a different type of device that does not use $K_{mp}$. To obtain the actual Media Key, the Media Key Precursor must be processed with KCD using the following process:

$K_m$ = AES-G($K_{mp}$, KCD)

The device obtains the Key Conversion Data by mechanisms that are independent of the Media Key Block. These mechanisms are defined in the format specific books of this specification. A device that is useing KCD, generally must apply the KCD before verifying the correctness of its processing of the Media Key Block. However, it is possible that it may be processing an old Media Key Block to which the KCD data has not been incorporated in its part of the tree. Therefore, such a device shall use the *Verify Media Key* Record to determine if it needs to apply the KCD data or not. In other words, if the purported Media Key Precursor actually verifies as the Media Key, then it shall not apply the KCD data, even if it is a type of device that normally would use KCD data.

Devices that do not use the KCD are not required to verify the Media Key, although they may.

### 3.2.5.5 Explicit Subset-Difference Record

**Table 3-7 – *Explicit Subset-Difference* Record Format**

| Byte \ Bit | 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. (See section 3.2.3 for a definition of the *uv* number.)

If a device encounters a *u* mask value whose high-order two bits are non-zero, without 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. However, it is common for proactively-renewed devices to find themselves revoked if they are at a down-level version. In this case, the update to the new version should be as seamless as possible for the consumer.

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

### 3.2.5.6 Subset-Difference Index Record

**Table 3-8 – *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 can 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 Media 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 Media Key Block as a stream using a relatively small buffer.) Nonetheless, devices must always be capable of processing Media 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 should begin processing the *Explicit Subset-Difference* Record at Offset 0. Devices whose number is between span and 2*span-1, should begin processing the *Explicit Subset-Difference* Record at Offset 1, and so on. Equivalently, if a device’s number is *d*, it can find 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 will always be a multiple of 4 bytes. Thus, there may be unused bytes at the end of the record.

### 3.2.5.7 Media Key Data Record

**Table 3-9 – *Media Key Data* Record Format**

| Byte \ Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| 0 | Record Type: $05_{16}$ | | | | | | | |
| 1 | Record Length | | | | | | | |
| 2 | | | | | | | | |
| 3 | | | | | | | | |
| 4 | Media Key Data (0) | | | | | | | |
| … | | | | | | | | |
| 19 | | | | | | | | |
| 20 | Media Key Data (1) | | | | | | | |
| ... | | | | | | | | |
| 35 | | | | | | | | |
| 36 | . . . | | | | | | | |
| . | | | | | | | | |
| . | | | | | | | | |
| . | | | | | | | | |
| Length-1 | | | | | | | | |

This record gives the associated encrypted Media Key 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 is the ciphertext value C in the Media Key calculation in Section 3.2.4

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 Media Key Data. On average there are 1.28 encryptions per revocation.

### 3.2.5.8 End of Media Key Block Record

**Table 3-10 – *End of Media Key Block* Record Format**

| Byte \ Bit | 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 Media 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 *End of Media Key Block* Record contains the AACS LA's signature on the data in the Media Key Block up to, but not including, this record. Devices must verify the signature, using the following procedure.

AACS_Verify(AACS_$LA_{pub}$, Signature Data, MKB)

If any device determines that the signature does not verify or is omitted, it must refuse to use the Media Key. Devices that are accessing the Media Key Block solely to use the *Host Revocation List* Record or the *Drive Revocation List* Record, which have their own embedded signatures, are not required to check the signature on the entire Media Key Block. Of course, they are required to check the signatures on the records they actually use, as specified in sections 3.2.5.2 and 3.2.5.3.

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

Figure 3-3 shows an example MKB with an example record ordering. However, it is possible to construct many other valid sequences. Notice that for every *"uv"* related field there is a Media Key Data field and for the record types shown in the left, there is only *one* record.

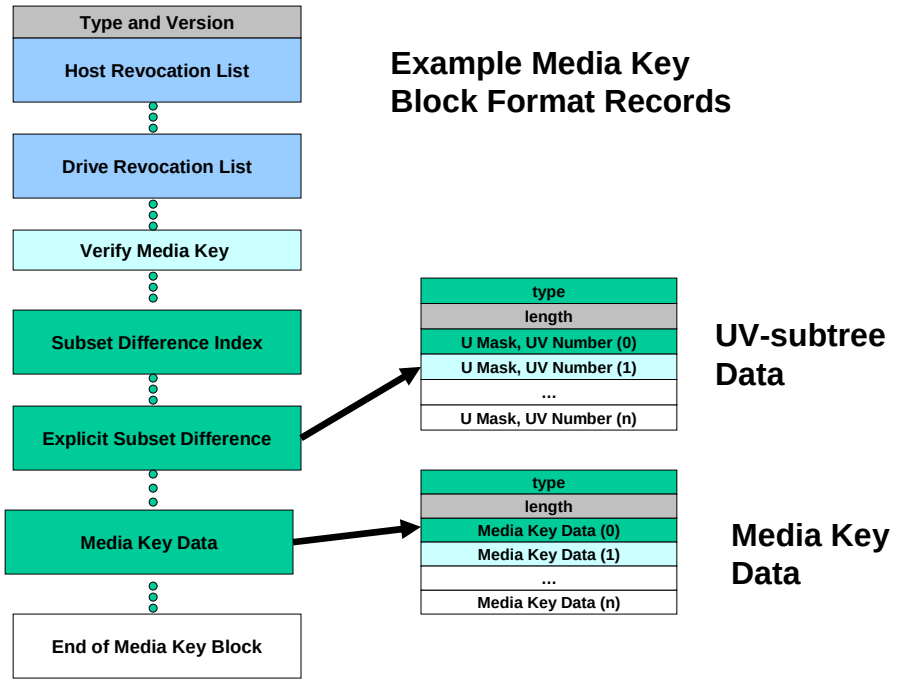



**Figure 3-3 Example of Media Key Block Showing a Valid Order of Records**

### 3.2.6 Read/Write Media Key Blocks

Media Key Blocks can be updated to later versions on Recordable media. Additional details on this process can be found in the Recordable book portion of this specification.

# Chapter 4
# Additional Procedures for Drive-Host Configurations

## 4. Introduction

AACS specification is applicable to a PC-based system. In such a system, a drive and PC host act together as the Recording Device and/or Playback Device for AACS protected content. Note that a new, robust and renewable form of drive authentication is introduced; recording or playback of AACS protected content is *not* permitted using drives that only support authentication associated with the Content Scramble System (CSS) for DVD-Video. The procedure for recording or playback of the content is the same as described in relevant document of AACS specification, except for additional steps that are required for the host to read and verify the integrity of the Volume Identifier, Pre-recorded Media Serial Number, Media Identifier and Protected Area Data values it receives from the drive, and to ensure the Protected Area Data is securely written to the media. The following figures (Figure 4-1 through Figure 4-5) illustrate these procedures.

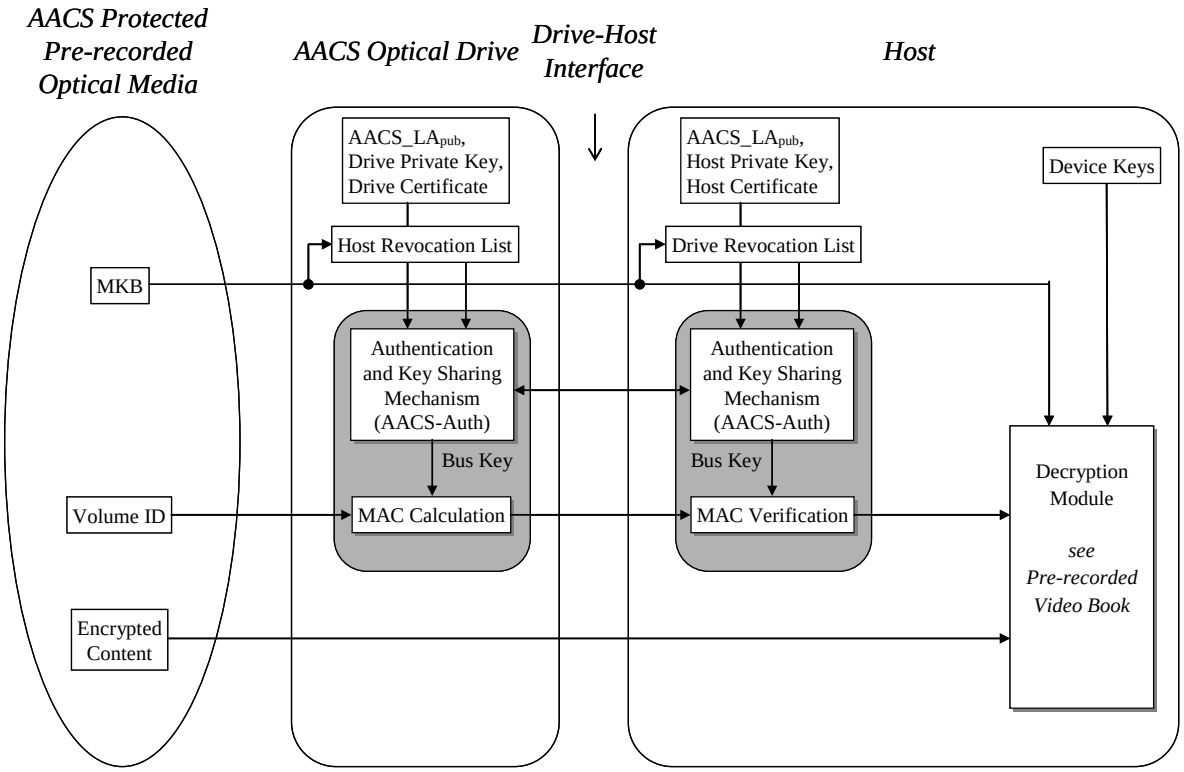



**Figure 4-1 – Reading of Volume Identifier in a PC–based System**

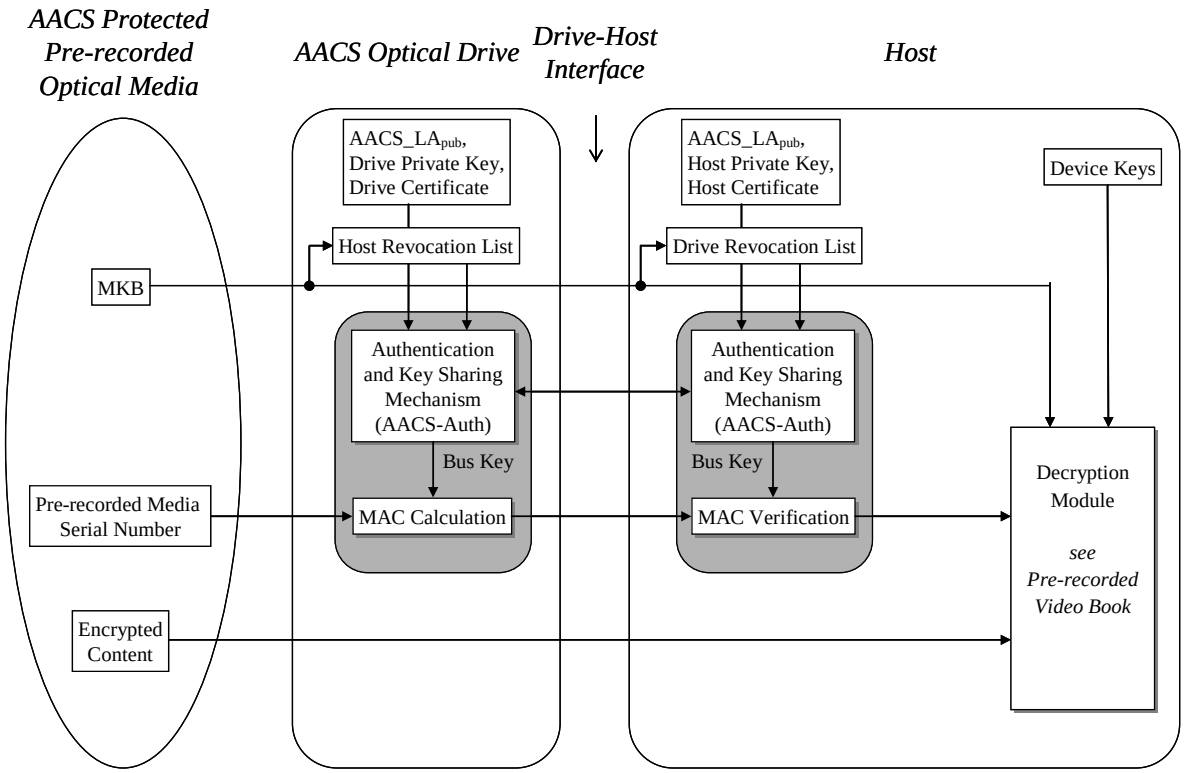



**Figure 4-2 – Reading of Pre-recorded Media Serial Number in a PC-based System**

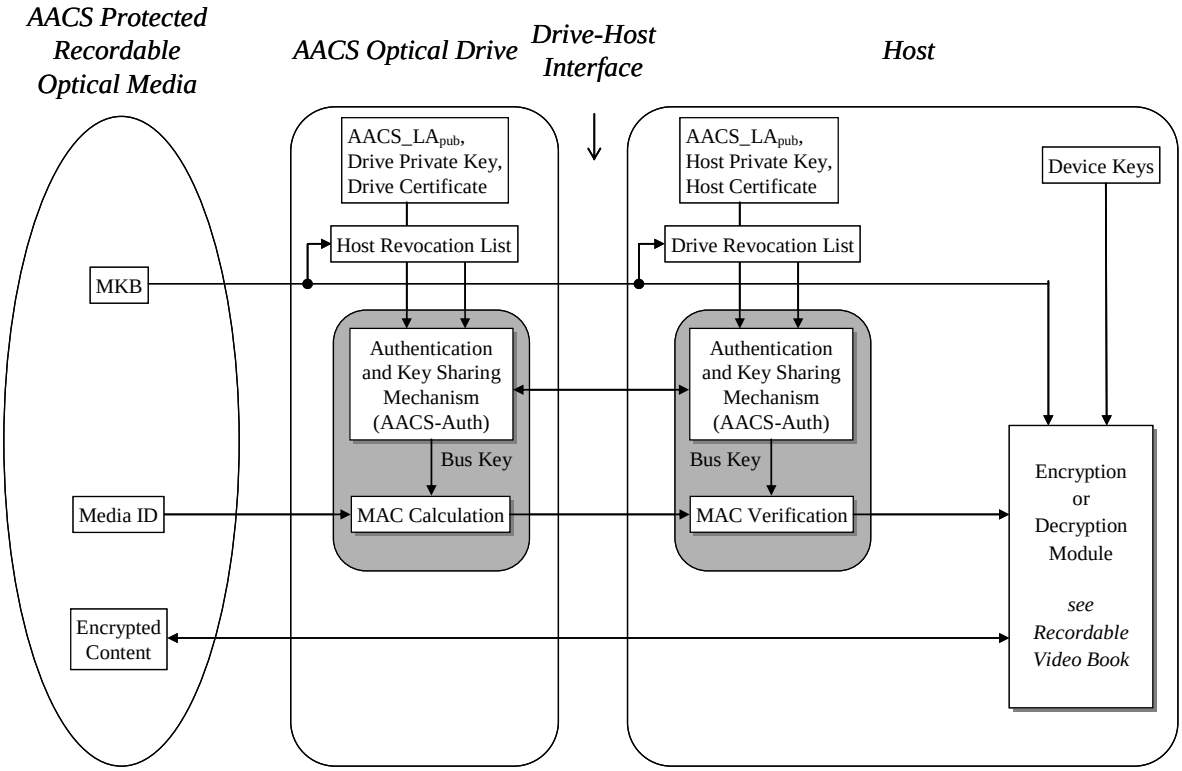



**Figure 4-3 – Reading of Media Identifier in a PC-based System**

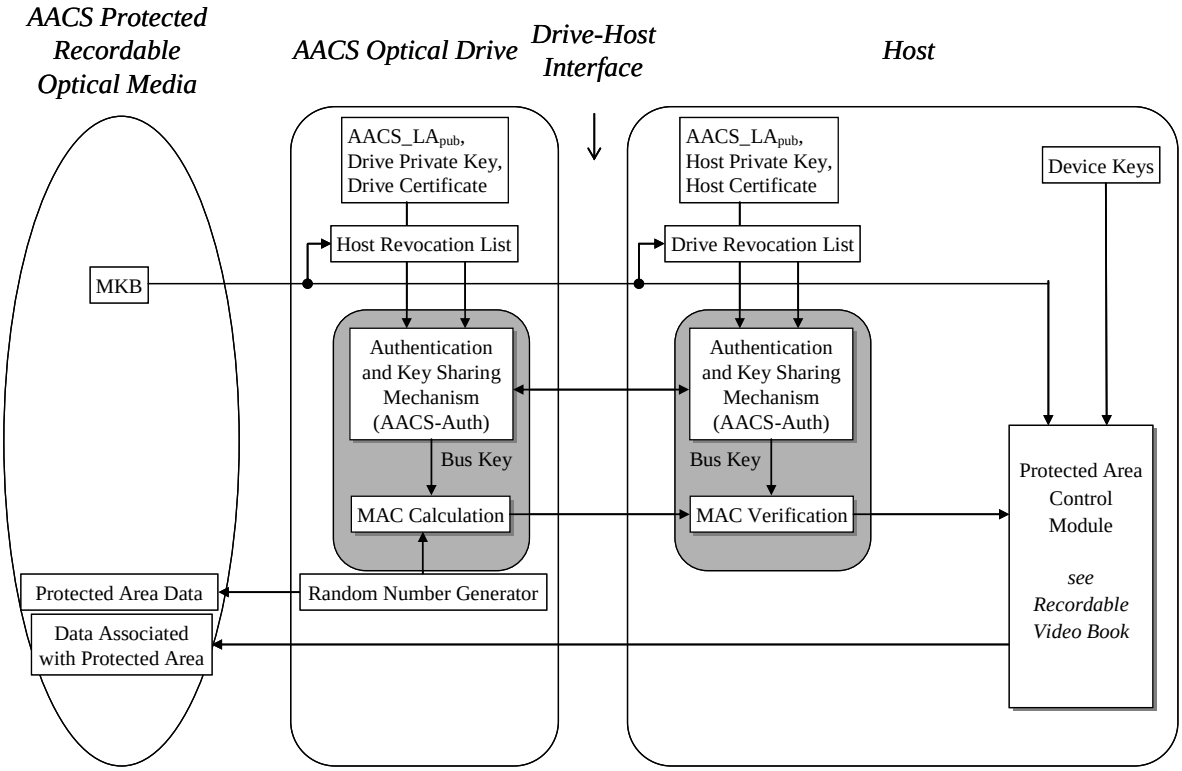



**Figure 4-4 – Generating, Transferring, and Writing of Protected Area Data in a PC-based System**

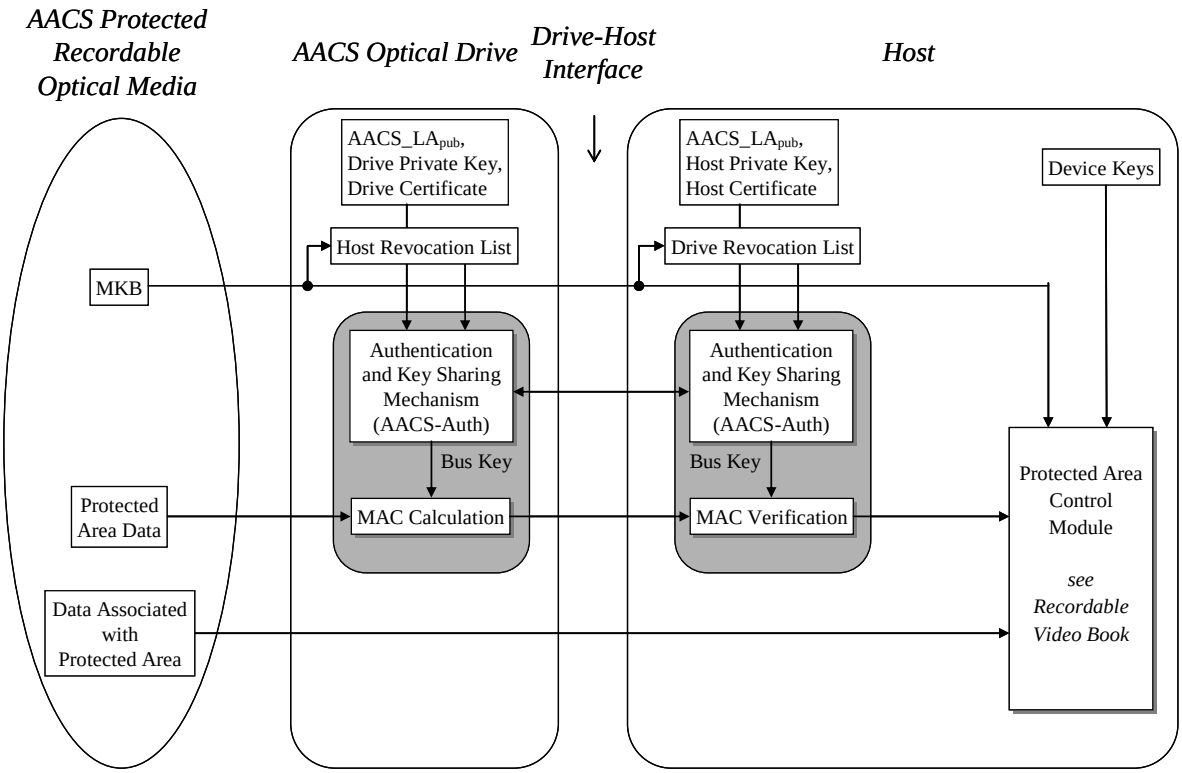



**Figure 4-5 – Reading of Protected Area Data in a PC-based System**

## 4.1 Drive Certificate

AACS licensed drive must have the $AACS\_LA_{pub}$, a Drive Certificate and a Drive Private Key in order to perform the required drive authentication. Table 4-1 shows the format of the Drive Certificate.

**Table 4-1 – Drive Certificate**

| Byte \ Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| 0 | Certificate Type: $01_{16}$ | | | | | | | |
| 1 | (reserved) | | | | | | | BEC |
| 2<br>3 | Length: $005C_{16}$ | | | | | | | |
| 4<br>:<br>9 | Drive ID | | | | | | | |
| 10<br>11 | (reserved) | | | | | | | |
| 12<br>:<br>51 | Drive Public Key | | | | | | | |
| 52<br>:<br>91 | Signature Data ($Drive\_Cert_{sig}$) | | | | | | | |

Each Drive Certificate includes:

- An 8-bit Certificate Type value, where $01_{16}$ shall be used to indicate a first-generation AACS drive
- A Bus Encryption Capable (BEC) bit, where $1_2$ shall be used to indicate that the drive is capable of performing the Bus Encryption that is to be later specified. If the drive is not capable of performing the Bus Encryption, the BEC bit shall be set to $0_2$. When both the drive and PC host are capable of performing the Bus Encryption, the Bus Encryption shall be performed. The details of Bus Encryption will be specified at a future date. Until such time, the BEC bit will be set to $0_2$.
- A 2-byte Length of the certificate data including signature.
- A 6-byte unique Drive ID.
- A 40-byte Drive Public Key.
- A 40-byte Signature Data ($Drive\_Cert_{sig}$), that is to be verified by using the $AACS\_LA_{pub}$.

  AACS_Verify($AACS\_LA_{pub}$, $Drive\_Cert_{sig}$, Drive_Cert)

  where $Drive\_Cert_{sig}$ is Byte 52 through Byte 91 of the Drive Certificate and Drive_Cert is Byte 0 through Byte 51 of the Drive Certificate.

## 4.2 Host Certificate

AACS licensed PC host must have the $AACS\_LA_{pub}$, a Host Certificate and a Host Private Key in order to perform the required drive authentication. Table 4-2 shows the format of the Host Certificate.

**Table 4-2 – Host Certificate**

| Byte \ Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|

| | | |
|---|---|---|
| 0 | Certificate Type: $02_{16}$ | |
| 1 | (reserved) | BEC |
| 2<br>3 | Length: $005C_{16}$ | |
| 4<br>:<br>9 | Host ID | |
| 10<br>11 | (reserved) | |
| 12<br>:<br>51 | Host Public Key | |
| 52<br>:<br>91 | Signature Data ($Host\_Cert_{sig}$) | |

Each Host Certificate includes:

- An 8-bit Certificate Type value, where $02_{16}$ shall be used to indicate a first-generation AACS PC host
- A 1-bit Bus Encryption Capable (BEC) bit, where $1_2$ shall be used to indicate that the PC host is capable of performing the Bus Encryption that is to be later specified. If the PC host is not capable of performing the Bus Encryption, the BEC bit shall be set to $0_2$. When both the drive and PC host are capable of performing the Bus Encryption, the Bus Encryption shall be performed. The details of Bus Encryption will be specified at a future date. Until such time, the BEC bit will be set to $0_2$.
- A 2-byte Length of the certificate data including signature.
- A 6-byte unique Host ID.
- A 40-byte Host Public Key.
- A 40-byte Signature Data ($Host\_Cert_{sig}$), that is to be verified by using the $AACS\_LA_{pub}$.

  $AACS\_Verify(AACS\_LA_{pub}, Host\_Cert_{sig}, Host\_Cert)$

  where $Host\_Cert_{sig}$ is Byte 52 through Byte 91 of the Host Certificate and Host_Cert is Byte 0 through Byte 51 of the Host Certificate.

## 4.3 Drive Authentication Algorithm for AACS (AACS-Auth)

By the drive authentication, the drive and the PC host verify each counterpart is an AACS compliant device that has valid certificate signed by the AACS LA and can sign and verify digital signatures specified in this document. In addition, the drive and the PC host verify each counterpart is not revoked by checking the Host Revocation List (HRL) and the Drive Revocation List (DRL), respectively. To do this, the drive shall store the most recent HRL it has encountered and the PC host shall store the most recent DRL it has encountered. When the drive authentication is successful, the drive and the PC host have a shared Bus Key (BK), and can proceed to the further steps. Figure 4-6 shows the protocol flow for drive authentication and key sharing used in AACS.

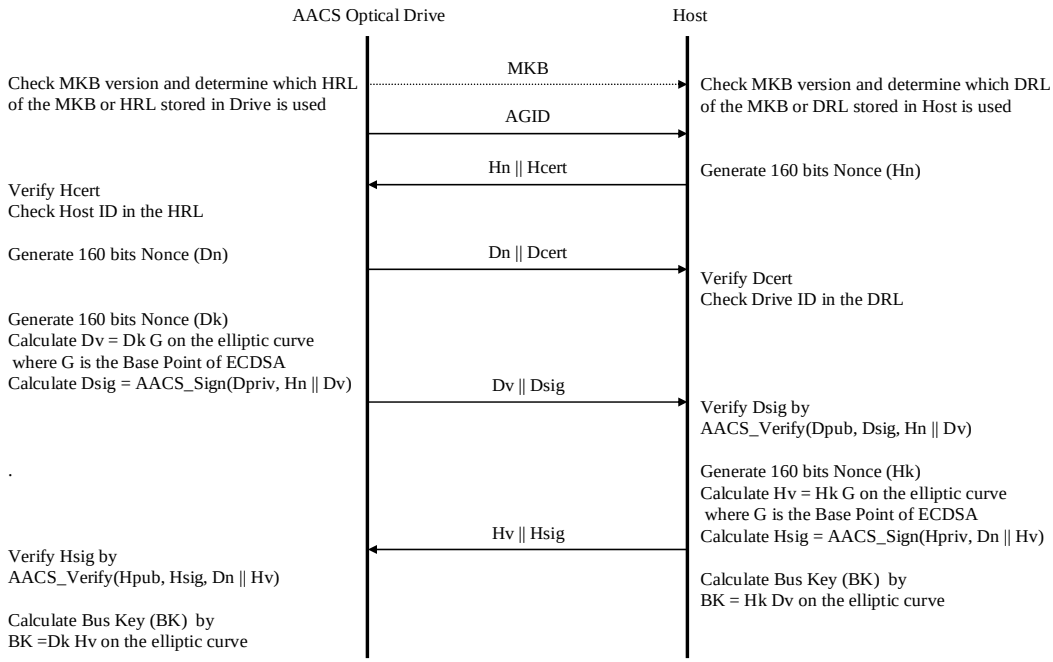



**Figure 4-6 – Drive Authentication Algorithm for AACS**

The drive authentication is performed in the following procedure.

1. For the first authentication to the media inserted, the host reads an MKB recorded on the media to check if its version is higher than the version of DRL that it has stored in its non-volatile memory. If not, the host determines to use the DRL in its non-volatile memory for the subsequent drive authentication procedure and go to step 3.
2. If the version of MKB recorded on the media is higher than the version of DRL that the host has stored in its non-volatile memory, the host verifies the signature in the *Drive Revocation List* Record of MKB. If the signature is correctly verified, the host determines to use the DRL in the MKB for the subsequent drive authentication procedure; otherwise, the host shall abort the drive authentication procedure.
3. For the first authentication to the media inserted, the drive reads an MKB recorded on the media to check if its version is higher than the version of HRL that it has stored in its non-volatile memory. If not, the drive determines to use the HRL in its non-volatile memory for the subsequent drive authentication procedure and go to step 5. The location of the MKB is described in the relevant Format-specific book in this specification. This step and the next step could be deferred after step 5 but must be performed before step 9.
4. If the version of MKB recorded on the media is higher than the version of HRL that the drive has stored in its non-volatile memory, the drive verifies the signature in the *Host Revocation List* Record of MKB. If the signature is correctly verified, the drive determines to use the HRL in the MKB for the subsequent drive authentication procedure; otherwise, the drive shall abort the drive authentication procedure.
5. The host acquires an Authentication Grant Identifier (AGID) from the drive. The AGID is used to manage multiple threads of drive authentications up to 4, if the drive supports the multiple threads.
6. The host generates 160 bits random number as nonce $H_n$
7. The host sends the nonce $H_n$ generated in step 6 and the Host Certificate to the drive.

8. The drive checks if the value of the Certificate Type is $02_{16}$ and the value of the Length is $005C_{16}$ in the Host Certificate. If these checks fail, the drive shall determine the host is not compliant and shall abort the drive authentication procedure.
9. The drive verifies the signature of the Host Certificate using the AACS LA Public Key.

   $AACS\_Verify(AACS\_LA_{pub}, Host\_Cert_{sig}, Host\_Cert)$

   where $Host\_Cert_{sig}$ is Byte 52 through Byte 91 of the Host Certificate and Host_Cert is Byte 0 through Byte 51 of the Host Certificate.

   If the verification fails, the drive shall determine the host is not compliant and shall abort the drive authentication procedure.
10. The drive checks the Host Revocation List determined in either step 3 or step 4 to ensure that the Host ID of the Host Certificate has not been revoked. If the Host ID is found revoked, the drive shall abort the drive authentication procedure.
11. The host sends a request to the drive to return a nonce $D_n$ and the Drive Certificate.
12. The drive generates 160 bits random number as nonce $D_n$
13. The drive sends the nonce $D_n$ generated in step 11 and the Drive Certificate to the host.
14. The host checks if the value of the Certificate Type is $01_{16}$ and the value of the Length is $005C_{16}$ in the Drive Certificate. If these checks fail, the host shall determine the drive is not compliant and shall abort the drive authentication procedure.
15. The host verifies the signature of the Drive Certificate using the AACS LA Public Key.

    $AACS\_Verify(AACS\_LA_{pub}, Drive\_Cert_{sig}, Drive\_Cert)$

    where $Drive\_Cert_{sig}$ is Byte 52 through Byte 91 of the Drive Certificate and Drive_Cert is Byte 0 through Byte 51 of the Drive Certificate.

    If the verification fails, the host shall determine the drive is not compliant and shall abort the drive authentication procedure.
16. The host checks the Drive Revocation List determined in either step 1 or step 2 to ensure that the Drive ID of the Drive Certificate has not been revoked. If the Drive ID is found revoked, the host shall abort the drive authentication procedure.
17. The host sends a request to the drive to return a point on the elliptic curve $D_v$ and its associated signature.
18. The drive generates 160 bits random number as $D_k$
19. The drive calculates a point on the elliptic curve $D_v$.

    $D_v = D_k G$

    where G is the base point of the elliptic curve
20. The drive creates a digital signature of the concatenation of the nonce $H_n$ received in step 7 and the point on the elliptic curve $D_v$ calculated in step 17.

    $D_{sig} = AACS\_Sign(AACS\_Drive_{priv}, H_n \| D_v)$
21. The drive sends the point on the elliptic curve $D_v$ calculated in step 17 and the digital signature $D_{sig}$ created in step 18 to the host.
22. The host verifies the signature of the concatenation of the nonce $H_n$ and the point on the elliptic curve $D_v$.

    $AACS\_Verify(AACS\_Drive_{pub}, D_{sig}, H_n \| D_v)$

    If the verification fails, the host shall determine the drive is not compliant and shall abort the drive authentication procedure.
23. The host generates 160 bits random number as $H_k$
24. The host calculates a point on the elliptic curve $H_v$.

    $H_v = H_k G$

    where G is the base point of the elliptic curve
25. The host creates a digital signature of the concatenation of the nonce $D_n$ received in step 12 and the point on the elliptic curve $H_v$ calculated in step 22.

    $H_{sig} = AACS\_Sign(AACS\_Host_{priv}, D_n \| H_v)$

26. The host sends the point on the elliptic curve $H_v$ calculated in step 22 and the digital signature $H_{sig}$ created in step 23 to the drive.
27. The drive verifies the signature of the concatenation of the nonce $D_n$ and the point on the elliptic curve $H_v$.

    AACS_Verify(AACS_Host$_{pub}$, $H_{sig}$ , $D_n$ || $H_v$)

    If the verification fails, the drive shall determine the host is not compliant and shall abort the drive authentication procedure.

Then, the drive and the host share a Bus Key as follows:

28. The drive calculates the Bus Key from a point of the elliptic curve as follows

    BK = [x-coordinate of $D_k$ $H_v$]$_{lsb\_128}$

    where BK is the least significant 128-bit of x-coordinate.
29. The host calculates the Bus Key from a point of the elliptic curve as follows

    BK = [x-coordinate of $H_k$ $D_v$]$_{lsb\_128}$

    where BK is the least significant 128-bit of x-coordinate.

## 4.4 Protocol for Transferring Volume Identifier

The Volume Identifier is securely transferred between the drive and the host using the following procedure: Figure 4-7 shows the protocol flow of transferring the Volume Identifier (Volume ID).

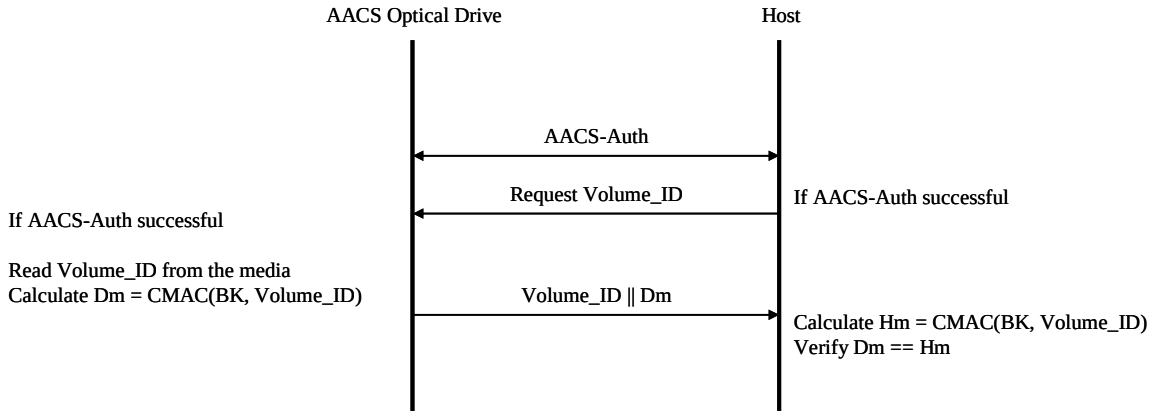



**Figure 4-7 – Protocol Flow of transferring Volume Identifier**

1. The drive and the host carry out the drive authentication and key sharing (AACS-Auth), as described in Section 4.3. If the AACS-Auth procedure is successful, the drive and the host proceed with the remaining steps.
2. The host sends a request to the drive to return Volume ID from the drive
3. The drive reads Volume ID (Volume_ID) from the media and calculates a message authentication code ($D_m$) from the Volume ID and the Bus Key (BK) calculated in step 26 in Section 4.3.

   $D_m$ = CMAC(BK, Volume_ID)
4. The drive sends Volume_ID read in step 3 and the message authentication code $D_m$ calculated in step 3 to the host.
5. The host calculates a message authentication code ($H_m$) from the Volume_ID received in step 4 and the Bus Key (BK) calculated in step 27 in Section 4.3.

   $H_m$ = CMAC(BK, Volume_ID)
6. The host verifies if the $D_m$ received in step 4 matches the $H_m$ calculated in step 5. If the verification succeeds, then the host may trust the Volume ID; otherwise, the host should stop processing the media.

## 4.5 Protocol for Transferring Pre-recorded Media Serial Number

The Pre-recorded Media Serial Number is securely transferred between the drive and the host using the following procedure: Figure 4-8 shows the protocol flow of transferring the Pre-recorded Media Serial Number (PMSN).

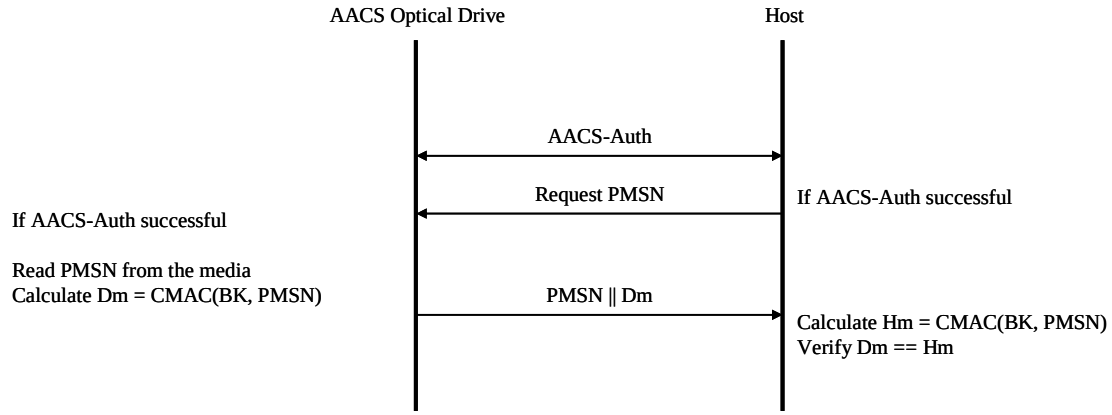



**Figure 4-8 – Protocol Flow of transferring Pre-recorded Media Serial Number**

1. The drive and the host carry out the drive authentication and key sharing (AACS-Auth), as described in Section 4.3. If the AACS-Auth procedure is successful, the drive and the host proceed with the remaining steps.
2. The host sends a request to the drive to return Pre-recorded Media Serial Number from the drive
3. The drive reads Pre-recorded Media Serial Number (PMSN) from the media and calculates a message authentication code ($D_m$) from the PMSN and the Bus Key (BK) calculated in step 26 in Section 4.3.

   $D_m$ = CMAC(BK, PMSN)
4. The drive sends PMSN read in step 3 and the message authentication code $D_m$ calculated in step 3 to the host.
5. The host calculates a message authentication code ($H_m$) from the PMSN received in step 4 and the Bus Key (BK) calculated in step 27 in Section 4.3.

   $H_m$ = CMAC(BK, PMSN)
6. The host verifies if the $D_m$ received in step 4 matches the $H_m$ calculated in step 5. If the verification succeeds, then the host may trust the Pre-recorded Media Serial Number; otherwise, the host should stop processing the media.

## 4.6 Protocol for Transferring Media Identifier

The Media Identifier is securely transferred between the drive and the host using the following procedure: Figure 4-9 shows the protocol flow of transferring the Media Identifier (Media ID).

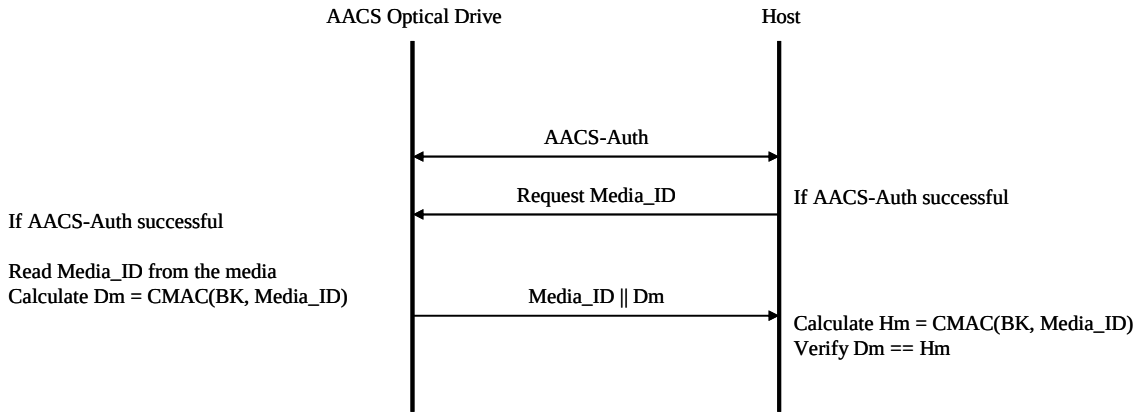



**Figure 4-9 – Protocol Flow of transferring Media Identifier**

1. The drive and the host carry out the drive authentication and key sharing (AACS-Auth), as described in Section 4.3. If the AACS-Auth procedure is successful, the drive and the host proceed with the remaining steps.
2. The host sends a request to the drive to return Media ID from the drive
3. The drive reads Media ID (Media_ID) from the media and calculates a message authentication code ($D_m$) from the Media ID and the Bus Key (BK) calculated in step 26 in Section 4.3.

   $D_m$ = CMAC(BK, Media_ID)
4. The drive sends Media_ID read in step 3 and the message authentication code $D_m$ calculated in step 3 to the host.
5. The host calculates a message authentication code ($H_m$) from the Media_ID received in step 4 and the Bus Key (BK) calculated in step 27 in Section 4.3.

   $H_m$ = CMAC(BK, Media_ID)
6. The host verifies if the $D_m$ received in step 4 matches the $H_m$ calculated in step 5. If the verification succeeds, then the host may trust the Media ID; otherwise, the host should stop processing the media.

## 4.7 Protocol for Updating the Protected Area and Associated Data

The Protected Area is used to store and restore the Binding Nonce (Protected Area Data) created by the drive as described in the Recordable Video Book of this specification. Because the data flows for the Protected Area is bi-directional, there are two protocols for transferring the Protected Area Data.

- Protocol for writing Protected Area Data
- Protocol for reading Protected Area Data

When requesting the drive to update the Binding Nonce, the host shall use the following sequence:

1. Read the current value of the Binding Nonce as specified in Section 4.7.2 and cache that current value to be used in Step 3.
2. Request the drive to commit a new Binding Nonce to the Protected Area as specified in Section 4.7.1.
3. Read the new value of the Binding Nonce as specified in Section 4.7.2 and verify that the new value is not the same value that was retrieved and cached in Step 1. If the Binding Nonce has not been correctly updated, then the host shall abort the procedure that necessitated the update of the Binding Nonce.

### 4.7.1 Protocol for Writing Protected Area Data

The Protected Area Data is written using following procedure. Figure 4-10 shows the protocol flow of writing the Protected Area Data.

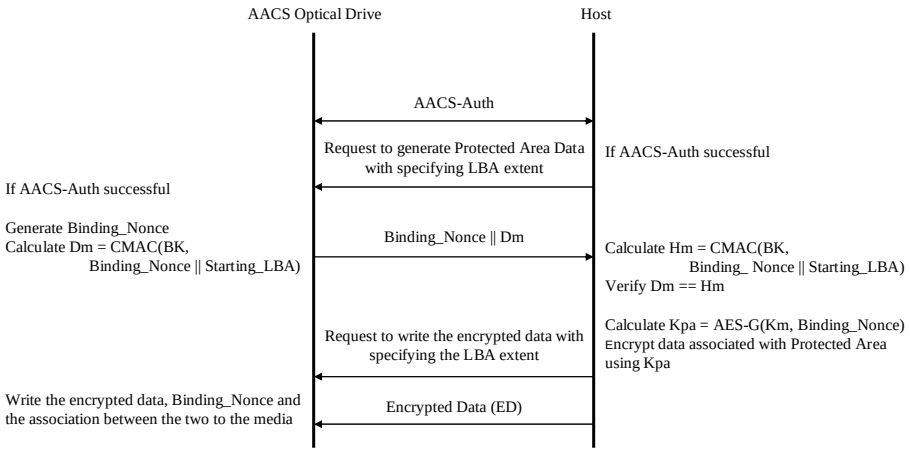



**Figure 4-10 – Protocol Flow of writing Protected Area Data**

1. The drive and the host carry out the drive authentication and key sharing (AACS-Auth), as described in Section 4.3. If the AACS-Auth procedure is successful, the drive and the host proceed with the remaining steps.
2. The host sends a request to the drive to generate a new 128-bit Binding Nonce that will be written into the Protected Area. The command data includes the LBA extent with which to associate the new Binding Nonce.
3. The drive generates the Binding Nonce and caches it along with the LBA extent from step 2.
4. The drive calculates a message authentication code ($D_m$) from the concatenation of the Binding Nonce and the starting address of the LBA extent (Starting_LBA) and the Bus Key (BK) calculated in step 26 in Section 4.3.

   $D_m$ = CMAC(BK, Binding_Nonce || Starting_LBA)
5. The drive sends the newly created Binding Nonce and the message authentication code $D_m$ calculated in step 4 to the host.
6. The host calculates a message authentication code ($H_m$) from the concatenation of the Binding Nonce and the starting address of the LBA extent (Starting_LBA) and the Bus Key (BK) calculated in step 27 in Section 4.3.

   $H_m$ = CMAC(BK, Binding_Nonce || Starting_LBA)
7. The host verifies if the $D_m$ received in step 5 matches the $H_m$ calculated in step 6. If the verification fails, the host shall abort the current write operation.
8. The host calculates the Protected Area Key ($K_{pa}$) as follows:

   $K_{pa}$ = AES-G($K_m$, Binding_Nonce)
9. The host encrypts the Data associated with Protected Area (D) using $K_{pa}$ as specified in the Recordable Video book of this specification.
10. The host requests to write the encrypted Data associated with the Protected Area (ED) as a user data file whose LBA extent was specified in step 2.

11. Upon receiving the write request the drive writes the encrypted Data associated with the Protected Area (ED), the Binding Nonce, and the association between the two to the media.

### 4.7.2 Protocol for Reading Protected Area Data

The Protected Area Data is read by the host using the following procedure. Figure 4-11 shows the protocol flow of reading the Protected Area Data.

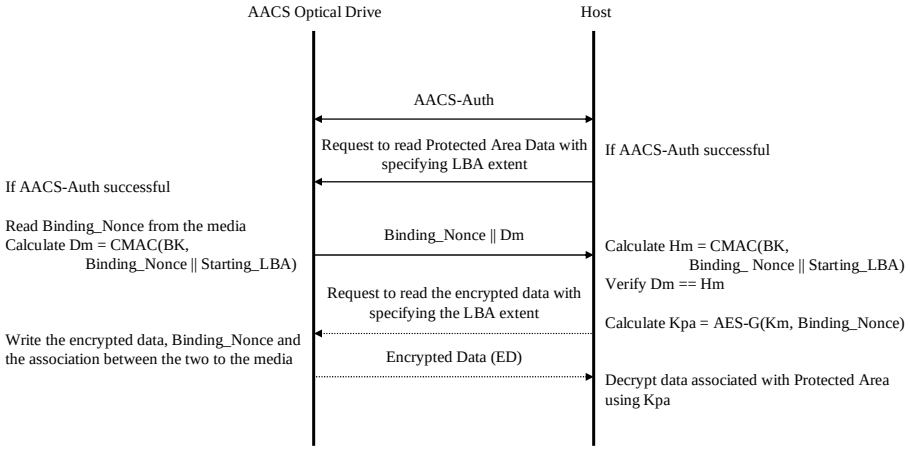



**Figure 4-11 – Protocol Flow of reading Protected Area Data**

1. The drive and the host carry out the drive authentication and key sharing (AACS-Auth), as described in Section 4.3. If the AACS-Auth procedure is successful, the drive and the host proceed with the remaining steps.
2. The host sends a request to the drive to read the Binding Nonce from the Protected Area. The command data contains the LBA extent for the user data file that is associated with the Protected Area Data.
3. Upon request from the host, the drive reads the Binding Nonce from the designated Protected Area specified in step 2.
4. The drive calculates a message authentication code ($D_m$) from the concatenation of the Binding Nonce and the starting address of the LBA extent (Starting_LBA) and the Bus Key (BK) calculated in step 26 in Section 4.3.

   $D_m$ = CMAC(BK, Binding_Nonce || Starting_LBA)
5. The drive sends the Binding Nonce and the message authentication code $D_m$ calculated in step 4 to the host.
6. The host calculates a message authentication code ($H_m$) from the concatenation of the Binding Nonce and the starting address of the LBA extent (Starting_LBA) and the Bus Key (BK) calculated in step 27 in Section 4.3.

   $H_m$ = CMAC(BK, Binding_Nonce || Starting_LBA)
7. The host verifies if the $D_m$ received in step 5 matches the $H_m$ calculated in step 6. If the verification fails, the host shall abort the current read operation.

The following steps are not directly related to Reading Protected Area Data and may be skipped.

8. The host calculates the Protected Area Key ($K_{pa}$) as follows:

   $K_{pa}$ = AES-G($K_m$, Binding_Nonce)

9. The host reads the encrypted Data associated with the Protected Area (ED) as a user data file whose LBA extent was specified in step 2.
10. The host decrypts the encrypted Data associated with the Protected Area (ED) using Kpa as specified in the Recordable Video book of this specification.

## 4.8 Updating Host Revocation List in Non-volatile Memory of Drive

An AACS licensed drive shall retain in non-volatile storage, the most recent Host Revocation List (HRL) data which it encounters and has verified. To do this, for the first AACS drive authentication to the media inserted, the drive shall read an MKB recorded on the media to check if its version is higher than the version of HRL that it has stored in its non-volatile memory. The location of the MKB is described in the relevant Format-specific book in this specification. The version of HRL is represented by the Version Number recorded in the Type and Version Record of the MKB. If the version of MKB recorded on the media is higher than the version of HRL that the drive has stored in its non volatile memory, the drive verifies the signature in the *Host Revocation List* Record of MKB as specified in section 3.2.5.2. If the signature is successfully verified, the drive shall replace the previously stored HRL data, if any, with the newly read HRL data. Note that the replacement process need not be performed during the drive authentication procedure but shall be performed before the media is ejected.

When persistently storing HRL data, the drive shall have at least 32K bytes of non-volatile memory for that purpose. This size is sufficient to store the first signature block of the *Host Revocation List* Record.

## 4.9 Updating Drive Revocation List in Non-volatile Memory of Host

An AACS licensed PC host shall retain in non-volatile storage, the most recent Drive Revocation List (DRL) data which it encounters and has verified. To do this, for the first AACS drive authentication to the media inserted, the host shall read an MKB recorded on the media to check if its version is higher than the version of DRL that it has stored in its non-volatile memory. The version of DRL is represented by the Version Number recorded in the Type and Version Record of the MKB. If the version of MKB recorded on the media is higher than the version of DRL that the host has stored in its non volatile memory, the host verifies the signature in the *Drive Revocation List* Record of MKB as specified in section 3.2.5.3. If the signature is successfully verified, the host shall replace the previously stored DRL data, if any, with the newly read DRL data. Note that the replacement process need not be performed during the drive authentication procedure but shall be performed before the media is ejected.

When persistently storing DRL data, the host shall have at least 32K bytes of non-volatile memory for that purpose. This size is sufficient to store the first signature block of the *Drive Revocation List* Record.

## 4.10 Mt. Fuji Command Extensions for AACS

The Mt. Fuji specification defines commands and related structures used to control the drive (logical unit). This section describes extensions to that specification for logical unit that support AACS functionality. Some additional information that is not found in the Mt. Fuji specification is also given, including the precise format of AACS data values returned by the logical unit.

### 4.10.1 AACS Feature

The Mt. Fuji specification defines a number of Features, which are sets of commands, mode pages, and behaviors or operations supported by a logical unit. Features implemented by a logical unit are reported to the host via the GET CONFIGURATION command. This command can be used to identify all possible Features, as well those Features that are current (i.e. currently available, which may depend on factors such as the type of media currently loaded). The AACS Feature is defined as shown in Table 4-3.

**Table 4-3 – AACS Feature**

| Feature Code | Feature Name | Description | Mandatory Commands |
|---|---|---|---|
| $010D_{16}$ | AACS | Ability to perform AACS authentication | REPORT KEY (Key Class $02_{16}$)<br>(Support of KEY Format $100000_2$ is conditional)<br>SEND KEY (Key Class $02_{16}$)<br>READ DISC STRUCTURE<br>(Format Codes $80_{16}$ $81_{16}$, $82_{16}$ and $83_{16}$) |

The AACS Feature Descriptor, obtained via the GET CONFIGURATION command, is shown in Table 4-4.

**Table 4-4 – AACS Feature Descriptor**

| Bit / Byte | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| 0 | (msb) Feature Code = $010D_{16}$ | | | | | | | |
| 1 | | | | | | | | (lsb) |
| 2 | Reserved | | | Version | | | Persistent | Current |
| 3 | Additional Length = $04_{16}$ | | | | | | | |
| 4 | Reserved | | | | | | | BNG |
| 5 | Block Count for Binding Nonce | | | | | | | |
| 6 | Reserved | | | | Number of AGIDs | | | |
| 7 | AACS version | | | | | | | |

The **Current** bit, when set to zero, indicates that this Feature is not currently active. When set to one, the Feature is active. The AACS Feature shall be active if and only if an AACS compliant media is loaded. A method to determine whether the loaded media is AACS compliant is format specific.

If Binding Nonce generation is supported, the BNG bit shall be set to $1_2$ and KEY Format $100000_2$ shall be supported. Otherwise it shall be set to zero.

The Block Count for Binding Nonce shall specify how many blocks are required to store the Binding Nonce for the media

The Number of AGIDs field indicates the maximum number of AGIDs that the logical unit supports concurrently.

The AACS version field shall be set to $01_{16}$.

The other fields of the AACS Feature Descriptor shall be set as described in the Mt. Fuji specification.

## 4.10.2 REPORT KEY Command Extensions

The REPORT KEY command with Key Class $02_{16}$ is used for AACS. The REPORT KEY command with Key Class $02_{16}$ requests the start of the AACS authentication process, requests data necessary for authentication and for generating a Bus Key, generates and returns or just returns the Binding Nonce and ends the AACS authentication process.

**Table 4-5 – REPORT KEY Command**

| Byte \ Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| 0 | Operation code ($A4_{16}$) | | | | | | | |
| 1 | LUN (Obsolete) | | | Reserved | | | | |
| 2 | (msb) | | | | | | | |
| 3 | | | | | | | | |
| 4 | | | | Reserved/Address | | | | |
| 5 | | | | | | | | (lsb) |
| 6 | Reserved/Block Count | | | | | | | |
| 7 | Key Class | | | | | | | |
| 8 | (msb) | | | | | | | |
| 9 | | | | Allocation Length | | | | (lsb) |
| 10 | AGID | | | Key Format | | | | |
| 11 | Vendor-Specific | | | Reserved | | NACA | Flag | Link |

The **Key Format** field indicates the type of information that is requested to be sent to the host. Key Format values defined for AACS are shown in Table 4-6.

**Table 4-6 – Key Format Code Definition for REPORT KEY command (Key Class = $02_{16}$)**

| Key Format | Returned Data | Description | AGID Use |
|---|---|---|---|
| $000000_2$ | AGID for AACS | Returns an AUTHENTICATION GRANT ID for Authentication for AACS | Reserved & N/A |
| $000001_2$ | Drive Certificate Challenge | Returns a Drive Certificate Challenge | Valid AGID required |
| $000010_2$ | Drive Key | Returns a Drive Key | Valid AGID required |
| $100000_2$ | Binding Nonce | Generates and stores a Binding Nonce and returns it | Valid AGID required |
| $100001_2$ | Binding Nonce | Returns a Binding Nonce | Valid AGID required |
| $111111_2$ | None | Invalidate specified AGID for AACS | Valid AGID required |

The Reserved/Address field contains a value which depends on the value in the Key Format field.

- For Key Format field = $100000_2$ (Generate Binding Nonce), the Reserved/Address field contains the starting address of the LBA Extent to which the Binding Nonce is to be recorded.
- For Key Format field = $100001_2$ (Read Binding Nonce), the Reserved/Address field contains the starting address of the LBA Extent from which the Binding Nonce is to be read.
- For other Key Format values - The Reserved/Address field shall be reserved.

The Reserved/Block Count field specifies a value which depends on the value in the Key Format field.

- For Key Format field = $100000_2$ (Generate Binding Nonce), the Block Count field contains the length of the LBA Extent to which the Binding Nonce is to be recorded. The length of the LBA Extent shall be no less than the value in the Block Count for Binding Nonce field in the AACS Feature Descriptor shown in Table 4-4

- For Key Format field = $100001_2$ (Read Binding Nonce), the Block Count field contains the length of the LBA Extent from which the Binding Nonce is to be read. The range of LBAs shall be no less than the value in the Block Count for Binding Nonce field in the AACS Feature Descriptor shown in Table 4-4
- For other Key Format values - The Reserved/Block Count field shall be reserved.

The AGID field identifies the Authentication Grant ID that was used for the authentication process.

The other fields of the REPORT KEY Command Descriptor Block shall be set as described in the Mt. Fuji specification.

### 4.10.2.1 Getting Authentication Grant ID for AACS

Table 4-7 shows the format of the data returned by the REPORT KEY command when Key Class of $02_{16}$ and Key Format of $000000_2$ are used.

**Table 4-7 – REPORT KEY Data Format (with Key Format = $000000_2$, Key Class = $02_{16}$)**

| Bit / Byte | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| 0 | (msb) | | | REPORT KEY Data Length ($0006_{16}$) | | | | |
| 1 | | | | | | | | (lsb) |
| 2 | Reserved | | | | | | | |
| 3 | Reserved | | | | | | | |
| 4 | Reserved | | | | | | | |
| 5 | Reserved | | | | | | | |
| 6 | Reserved | | | | | | | |
| 7 | AGID | | Reserved | | | | | |

This Key Format requests the logical unit to return an Authentication Grant ID for AACS. After an AACS Authentication Grant ID is obtained, the AACS Authentication procedure defined in this Book can be carried out, using the REPORT KEY and SEND KEY commands as described in the Mt. Fuji specification.

### 4.10.2.2 Returning Drive Certificate Challenge

Table 4-8 shows the format of the data returned by the REPORT KEY command when Key Class of $02_{16}$ and Key Format of $000001_2$ are used.

**Table 4-8 – REPORT KEY Data Format (with Key Format = $000001_2$, Key Class = $02_{16}$)**

| Bit / Byte | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| 0 | (msb) | | | REPORT KEY Data Length ($0072_{16}$) | | | | |
| 1 | | | | | | | | (lsb) |
| 2 | Reserved | | | | | | | |
| 3 | Reserved | | | | | | | |
| 4 | (msb) | | | Nonce ($D_n$) | | | | |
| : | | | | | | | | |
| 23 | | | | | | | | (lsb) |
| 24 | (msb) | | | Drive Certificate | | | | |
| : | | | | | | | | |
| 115 | | | | | | | | (lsb) |

This Key Format is used to carry out steps 10 through 12 of the protocol shown in Section 4.3.

The REPORT KEY Data Length field specifies the length in bytes of the following REPORT KEY data that is available to be transferred to the host. The REPORT KEY Data Length value does not include the REPORT KEY Data Length field itself. For the Key Format of $000001_2$, the value of this field is $0072_{16}$.

Bytes 4 through 23 return the 160-bit nonce $D_n$ value.

Bytes 24 through 115 return the 92-byte Drive Certificate specified in Section 4.1.

When the loaded disc is not AACS compliant media, this command with Key Format = $000001_2$ shall be terminated with CHECK CONDITION Status, 5/6F/01 COPY PROTECTION KEY EXCHANGE FAILURE – KEY NOT PRESENT.

### 4.10.2.3 Returning Drive Key

Table 4-9 shows the format of the data returned by the REPORT KEY command when Key Class of $02_{16}$ and Key Format of $000010_2$ are used.

**Table 4-9 – REPORT KEY Data Format (with Key Format = $000010_2$, Key Class = $02_{16}$)**

| Bit / Byte | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| 0 | (msb) | | | REPORT KEY Data Length ($0052_{16}$) | | | | |
| 1 | | | | | | | | (lsb) |
| 2 | Reserved | | | | | | | |
| 3 | Reserved | | | | | | | |
| 4 | (msb) | | | Elliptic Curve Point ($D_v$) | | | | |
| : | | | | | | | | |
| 43 | | | | | | | | (lsb) |
| 44 | (msb) | | | Signature ($D_{sig}$) | | | | |
| : | | | | | | | | |
| 83 | | | | | | | | (lsb) |

This Key Format is used to carry out steps 15 through 19 of the protocol shown in Section 4.3.

The REPORT KEY Data Length field specifies the length in bytes of the following REPORT KEY data that is available to be transferred to the host. The REPORT KEY Data Length value does not include the REPORT KEY Data Length field itself. For the Key Format of $000010_2$, the value of this field is $0052_{16}$.

Bytes 4 through 43 return the 320-bit elliptic curve point $D_v$ value specified in step 17 in Section 4.3.

Bytes 44 through 83 return the 320-bit signature $D_{sig}$ value specified in step 18 in Section 4.3.

### 4.10.2.4 Generating and Reporting Binding Nonce

Table 4-10 shows the format of the data returned by the REPORT KEY command when Key Class of $02_{16}$ and Key Format of $100000_2$ are used.

**Table 4-10 – REPORT KEY Data Format (with Key Format = $100000_2$, Key Class = $02_{16}$)**

| Bit / Byte | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| 0 | (msb) | | | REPORT KEY Data Length ($0022_{16}$) | | | | |
| 1 | | | | | | | | (lsb) |
| 2 | Reserved | | | | | | | |

| | |
|---|---|
| 3 | Reserved |
| 4 | (msb) |
| : | Binding Nonce |
| 19 | (lsb) |
| 20 | (msb) |
| : | Message Authentication Code |
| 35 | (lsb) |

This Key Format is used to carry out steps 2 through 5 of the protocol shown in Section 4.7.1.

The REPORT KEY Data Length field specifies the length in bytes of the following REPORT KEY data that is available to be transferred to the host. The REPORT KEY Data Length value does not include the REPORT KEY Data Length field itself. For the Key Format of $100000_2$, the value of this field is $0022_{16}$.

Bytes 4 through 19 return the 128-bit Binding Nonce value.

Bytes 20 through 35 return the 128-bit message authentication code value specified in step 4 in Section 4.7.1.

When the logical unit has not established a Bus Key for the authentication process, this command with Key Format = $100000_2$ shall be terminated with CHECK CONDITION Status, 5/6F/02 COPY PROTECTION KEY EXCHANGE FAILURE – KEY NOT ESTABLISHED.

### 4.10.2.5 Reading Binding Nonce

Table 4-11 shows the format of the data returned by the REPORT KEY command when Key Class of $02_{16}$ and Key Format of $100001_2$ are used.

**Table 4-11 – REPORT KEY Data Format (with Key Format = $100001_2$, Key Class = $02_{16}$)**

| Bit / Byte | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| 0 | (msb) | | | | | | | |
| 1 | REPORT KEY Data Length ($0022_{16}$) | | | | | | | (lsb) |
| 2 | Reserved | | | | | | | |
| 3 | Reserved | | | | | | | |
| 4 | (msb) | | | | | | | |
| : | Binding Nonce | | | | | | | |
| 19 | (lsb) | | | | | | | |
| 20 | (msb) | | | | | | | |
| : | Message Authentication Code | | | | | | | |
| 35 | (lsb) | | | | | | | |

This Key Format is used to carry out steps 2 through 5 of the protocol shown in Section 4.7.2.

The REPORT KEY Data Length field specifies the length in bytes of the following REPORT KEY data that is available to be transferred to the host. The REPORT KEY Data Length value does not include the REPORT KEY Data Length field itself. For the Key Format of $100001_2$, the value of this field is $0022_{16}$.

Bytes 4 through 19 return the 128-bit Binding Nonce value.

Bytes 20 through 35 return the 128-bit message authentication code value specified in step 4 in Section 4.7.2.

When the logical unit has not established a Bus Key for the authentication process, this command with Key Format = $100001_2$ shall be terminated with CHECK CONDITION Status, 5/6F/02 COPY PROTECTION KEY EXCHANGE FAILURE – KEY NOT ESTABLISHED.

### 4.10.3 READ DISC STRUCTURE Command Extensions

Logical units that implement the AACS Feature support extensions to the READ DISC STRUCTURE command. The READ DISC STRUCTURE command, shown in Table 4-12, requests that the logical unit transfer data from areas on the specified media to the host.

**Table 4-12 – READ DISC STRUCTURE Command**

| Bit / Byte | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| 0 | Operation code ($AD_{16}$) | | | | | | | |
| 1 | LUN (Obsolete) | | | Reserved | Sub-command | | | |
| 2 | (msb) Address | | | | | | | |
| 3 | | | | | | | | |
| 4 | | | | | | | | |
| 5 | | | | | | | | (lsb) |
| 6 | Layer Number | | | | | | | |
| 7 | Format Code | | | | | | | |
| 8 | (msb) Allocation Length | | | | | | | |
| 9 | | | | | | | | (lsb) |
| 10 | AGID | | Reserved | | | | | |
| 11 | Vendor-Specific | | Reserved | | | NACA | Flag | Link |

The Sub-command field indicates the type of command definition to expand this command for other media type than DVD. This value shall be set to $0000_2$ for DVD or HD DVD media and shall be set to $0001_2$ for BD media.

The Format Code field indicates the type of information that is requested by the host. New Format Code values defined for AACS are shown in Table 4-13, along with corresponding usage of the Layer Number and Address fields.

**Table 4-13 – AACS Format Code definitions for READ DISC STRUCTURE command**

| Format Code | Returned Data | Layer Number Field Usage | Address Field Usage | Description |
|---|---|---|---|---|
| $80_{16}$ | Volume Identifier of AACS | Reserved | Reserved | Returns the Volume Identifier specified by AACS |
| $81_{16}$ | Pre-recorded Media Serial Number of AACS | Reserved | Reserved | Returns the Pre-recorded Media Serial Number specified by AACS |
| $82_{16}$ | Media Identifier of AACS | Reserved | Reserved | Returns the Media Identifier specified by AACS |
| $83_{16}$ | Media Key Block of AACS | Layer Number | Pack number | Returns the Media Key Block in Lead-in specified by AACS |

The AGID field identifies the Authentication Grant ID that was used for the authentication process.

For Format Code $83_{16}$, the Address field is used to specify which MKB Pack is to be read. This field enables the host to read a Media Key Block Frame contained in multiple Packs.

The other fields of the READ DISC STRUCTURE Command Descriptor Block shall be set as described in the Mt. Fuji specification.

### 4.10.3.1 Volume Identifier (Format Code $80_{16}$)

**Table 4-14 – READ DISC STRUCTURE Data Format (With Format Code = $80_{16}$)**

| Bit / Byte | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| 0 | (msb) | | | DISC STRUCTURE Data Length ($0022_{16}$) | | | | |
| 1 | | | | | | | | (lsb) |
| 2 | Reserved | | | | | | | |
| 3 | Reserved | | | | | | | |
| 4 | (msb) | | | Volume Identifier | | | | |
| : | | | | | | | | |
| 19 | | | | | | | | (lsb) |
| 20 | (msb) | | | Message Authentication Code | | | | |
| : | | | | | | | | |
| 35 | | | | | | | | (lsb) |

This command with this Format Code is used to carry out steps 2 through 4 of the protocol shown in Section 4.4.

The DISC STRUCTURE Data Length field specifies the length in bytes of the following DISC STRUCTURE data that is available to be transferred to the host. The DISC STRUCTURE Data Length value does not include the DISC STRUCTURE Data Length field itself. For the Format Code of $80_{16}$, the value of this field is $0022_{16}$.

Bytes 4 through 19 return the 128-bit Volume Identifier value.

Bytes 20 through 35 return the 128-bit message authentication code value specified in step 3 in Section 4.4.

When the logical unit has not established a Bus Key for the authentication process, this command with Format Code = $80_{16}$ shall be terminated with CHECK CONDITION Status, 5/6F/02 COPY PROTECTION KEY EXCHANGE FAILURE – KEY NOT ESTABLISHED.

### 4.10.3.2 Pre-recorded Media Serial Number (Format Code $81_{16}$)

**Table 4-15 – READ DISC STRUCTURE Data Format (With Format Code = $81_{16}$)**

| Bit / Byte | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| 0 | (msb) | | | | | | | |
| 1 | DISC STRUCTURE Data Length ($0022_{16}$) | | | | | | | (lsb) |
| 2 | Reserved | | | | | | | |
| 3 | Reserved | | | | | | | |
| 4 | (msb) | | | | | | | |
| : | Pre-recorded Media Serial Number | | | | | | | |
| 19 | | | | | | | | (lsb) |
| 20 | (msb) | | | | | | | |
| : | Message Authentication Code | | | | | | | |
| 35 | | | | | | | | (lsb) |

This Command with this Format Code is used to carry out steps 2 through 4 of the protocol shown in Section 4.5.

The DISC STRUCTURE Data Length field specifies the length in bytes of the following DISC STRUCTURE data that is available to be transferred to the host. The DISC STRUCTURE Data Length value does not include the DISC STRUCTURE Data Length field itself. For the Format Code of $81_{16}$, the value of this field is $0022_{16}$.

Bytes 4 through 19 return the 128-bit Pre-recorded Media Serial Number value.

Bytes 20 through 35 return the 128-bit message authentication code value specified in step 3 in Section 4.5.

When the logical unit has not established a Bus Key for the authentication process, this command with Format Code = $81_{16}$ shall be terminated with CHECK CONDITION Status, 5/6F/02 COPY PROTECTION KEY EXCHANGE FAILURE – KEY NOT ESTABLISHED.

### 4.10.3.3 Media Identifier (Format Code $82_{16}$)

**Table 4-16 – READ DISC STRUCTURE Data Format (With Format Code = $82_{16}$)**

| Bit / Byte | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| 0 | (msb) | | | | | | | |
| 1 | DISC STRUCTURE Data Length ($0022_{16}$) | | | | | | | (lsb) |
| 2 | Reserved | | | | | | | |
| 3 | Reserved | | | | | | | |
| 4 | (msb) | | | | | | | |
| : | Media Identifier | | | | | | | |
| 19 | | | | | | | | (lsb) |
| 20 | (msb) | | | | | | | |
| : | Message Authentication Code | | | | | | | |
| 35 | | | | | | | | (lsb) |

This Command with this Format Code is used to carry out steps 2 through 4 of the protocol shown in Section 4.6.

The DISC STRUCTURE Data Length field specifies the length in bytes of the following DISC STRUCTURE data that is available to be transferred to the host. The DISC STRUCTURE Data Length value does not include the DISC STRUCTURE Data Length field itself. For the Format Code of $82_{16}$, the value of this field is $0022_{16}$.

Bytes 4 through 19 return the 128-bit Media Identifier value.

Bytes 20 through 35 return the 128-bit message authentication code value specified in step 3 in Section 4.6.

When the logical unit has not established a Bus Key for the authentication process, this command with Format Code = $82_{16}$ shall be terminated with CHECK CONDITION Status, 5/6F/02 COPY PROTECTION KEY EXCHANGE FAILURE – KEY NOT ESTABLISHED.

### 4.10.3.4 MEDIA KEY BLOCK (Format $83_{16}$)

**Table 4-17 – READ DISC STRUCTURE Data Format (With Format Field = $83_{16}$)**

| Bit / Byte | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| 0 | (msb) | | | DISC STRUCTURE Data Length | | | | |
| 1 | | | | | | | | (lsb) |
| 2 | Reserved | | | | | | | |
| 3 | Total Packs | | | | | | | |
| 4 | (msb) | | | MEDIA KEY BLOCK Pack Data | | | | |
| : | | | | | | | | |
| 32,771 | | | | | | | | (lsb) |

This Command with this Format Code is used to transfer the Media Key Block (MKB) recorded in the Lead-in Area of a pre-recorded and recordable media that exist in format specific manner. Please note that the Media Key Block is transferred without using the AACS authentication process.

The DISC STRUCTURE Data Length field specifies the length in bytes of the following DISC STRUCTURE data that is available to be transferred to the host. The DISC STRUCTURE Data Length value does not include the DISC STRUCTURE Data Length field itself.

The Total Packs field reports the total number of MKB Packs that are available for transfer to the host, which is calculated by dividing total MKB data length by 32,768 with counting fractions as one.

The Address field in the command specifies which of the available MKB Packs shall be read. The first MKB Pack shall be addressed as MKB Pack number $00000000_{16}$ and the nth MKB Pack shall be addressed as MKB Pack number n-1. The last MKB Pack may end with unused bytes, which shall be zero-filled.

The MEDIA KEY BLOCK Pack Data field returns the requested MKB Pack. The MEDIA KEY BLOCK Pack Data length is 32,768 and the DISC STRUCTURE Data Length is $8002_{16}$.

## 4.10.4 SEND KEY Command Extensions

The SEND KEY command with Key Class $02_{16}$ is used for AACS. The SEND KEY command with Key Class $02_{16}$ provides data necessary for authentication and for generating a Bus Key and ends the authentication process.

**Table 4-18 – SEND KEY Command**

| Bit / Byte | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| 0 | Operation code ($A3_{16}$) | | | | | | | |
| 1 | Reserved | | | | | | | |
| 2 | Reserved | | | | | | | |
| 3 | Reserved | | | | | | | |
| 4 | Reserved | | | | | | | |
| 5 | Reserved | | | | | | | |
| 6 | Reserved | | | | | | | |
| 7 | Key Class | | | | | | | |
| 8 | (msb) Parameter List Length | | | | | | | |
| 9 | | | | | | | | (lsb) |
| 10 | AGID | | Key Format | | | | | |
| 11 | Vendor-Specific | | Reserved | | | NACA | Flag | Link |

The **Key Format** field indicates the type of information that is sent to the logical unit. Key Format values defined for AACS are shown in Table 4-19.

**Table 4-19 – Key Format Code Definition for SEND KEY command (Key Class = $02_{16}$)**

| Key Format | Returned Data | Description | AGID Use |
|---|---|---|---|
| $000001_2$ | Host Certificate Challenge | Send a Host Certificate Challenge to logical unit | Valid AGID required |
| $000010_2$ | Host Key | Send a Host Key to logical unit | Valid AGID required |
| $111111_2$ | None | Invalidate specified AGID for AACS | Valid AGID required |

The AGID field identifies the Authentication Grant ID that was used for the authentication process.

The other fields of the SEND KEY Command Descriptor Block shall be set as described in the Mt. Fuji specification.

### 4.10.4.1 Sending Host Certificate Challenge

Table 4-20 shows the format of the data sent by the SEND KEY command when Key Class of $02_{16}$ and Key Format of $000001_2$ are used.

**Table 4-20 – SEND KEY Parameter List (with Key Format = $000001_2$, Key Class = $02_{16}$)**

| Bit / Byte | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| 0 | (msb) | | | SEND KEY Parameter List Length ($0072_{16}$) | | | | |
| 1 | | | | | | | | (lsb) |
| 2 | Reserved | | | | | | | |
| 3 | Reserved | | | | | | | |
| 4 | (msb) | | | Nonce ($H_n$) | | | | |
| : | | | | | | | | |
| 23 | | | | | | | | (lsb) |
| 24 | (msb) | | | Host Certificate | | | | |
| : | | | | | | | | |
| 115 | | | | | | | | (lsb) |

This Key Format is used to carry out steps 7 through 9 of the protocol shown in Section 4.3.

The SEND KEY Parameter List Length field specifies the length in bytes of the following SEND KEY Parameter List that is transferred from the host. The SEND KEY Parameter List Length value does not include the SEND KEY Parameter List Length field itself. For the Key Format of $000001_2$, the value of this field is $0072_{16}$.

Bytes 4 through 23 specify the 160-bit nonce $H_n$ value.

Bytes 24 through 115 specify the 92-byte Host Certificate specified in Section 4.2.

When the loaded disc is not AACS compliant media, this command with Key Format = $000001_2$ shall be terminated with CHECK CONDITION Status, 5/6F/01 COPY PROTECTION KEY EXCHANGE FAILURE – KEY NOT PRESENT.

When the Host Certificate verification fails or the Host ID of the Host Certificate is found revoked, the command with Key Format = $000001_2$ shall be terminated with CHECK CONDITION status, 5/6F/00 COPY PROTECTION KEY EXCHANGE FAILURE – AUTHENTICATION FAILURE.

### 4.10.4.2 Sending Host Key

Table 4-21 shows the format of the data sent by the SEND KEY command when Key Class of $02_{16}$ and Key Format of $000010_2$ are used.

**Table 4-21 – SEND KEY Parameter List (with Key Format = $000010_2$, Key Class = $02_{16}$)**

| Bit / Byte | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| 0 | (msb) | | | SEND KEY Data Length ($0052_{16}$) | | | | |
| 1 | | | | | | | | (lsb) |
| 2 | Reserved | | | | | | | |
| 3 | Reserved | | | | | | | |
| 4 | (msb) | | | Elliptic Curve Point ($H_v$) | | | | |
| : | | | | | | | | |
| 43 | | | | | | | | (lsb) |
| 44 | (msb) | | | Signature ($H_{sig}$) | | | | |
| : | | | | | | | | |
| 83 | | | | | | | | (lsb) |

This Key Format is used to carry out steps 24 through 25 of the protocol shown in Section 4.3.

The SEND KEY Parameter List Length field specifies the length in bytes of the following SEND KEY Parameter List that is transferred from the Host. The SEND KEY Parameter List Length value does not include the SEND KEY Parameter List Length field itself. For the Key Format of $000010_2$, the value of this field is $0052_{16}$.

Bytes 4 through 43 specify the 320-bit elliptic curve point $H_v$ value specified in step 22 in Section 4.3.

Bytes 44 through 83 specify the 320-bit signature $H_{sig}$ value specified in step 23 in Section 4.3.

When the signature verification fails, the command with Key Format = $000010_2$ shall be terminated with CHECK CONDITION status, 5/6F/00 COPY PROTECTION KEY EXCHANGE FAILURE – AUTHENTICATION FAILURE.

This page is intentionally left blank.

# Chapter 5
# Uses of On-line Connections

## 5. Introduction

This chapter specifies details regarding the use of an on-line connection to enable certain enhanced uses of AACS-protected content. From the AACS point of view, a device is not required to have an on-line connection capability or support enhanced uses. However, individual format licenses may require it. AACS defines several different enhanced modes for using AACS content with on-line connections:

- *AACS Network Download Content*. This on-line content is intended to be recorded on AACS-protected media. An on-line transaction serves to bind the content to a particular piece of media.
- *AACS On-line Enabled Content*. This content is pre-recorded on pre-recorded media, or part of the initial download in AACS Network Download content, but only made playable by an on-line transaction.
- *AACS Streamed Content*. This is stream content logically associated with pre-recorded or AACS Network Download Content, but delivered on demand across the Internet.
- *AACS Managed Copy*. Content protected by AACS and contained on Pre-recorded Media includes an offer to allow at least one copy of that title onto alternative media such as a Home Media Server. The device performing the Managed Copy will need to obtain authorization from a Remote Server as a part of making this copy. The requirements to support AACS Managed Copy are defined in the *Pre-recorded Video* book.

In addition, AACS content can be used in other, unspecified ways as part of an on-line transaction. As a rule, if the transaction requires the use of AACS-defined keys such as device keys or media keys, the AACS specification applies, and the specification defines a minimum support level for that transaction in every AACS device. If, on the other hand, the transaction is accomplished using non-AACS generated keys such as Title Keys, the transaction is outside of the scope of the AACS specification, and it is not mandatory for AACS devices to support it. In all cases, however, the AACS compliance rules apply.

The word “title” is often overloaded. For example, a title can refer to a full-feature movie, a TV program, a music album, etc. Thus, the word is also commonly used to refer to entire DVD-Video disc. However, for the purpose of the primary AACS specification books, we follow existing DVD-Video usage and define *Title* to be a distinct path through a piece of content. That is, a Title is a logical grouping of content material to be presented in a specific order in time. The correspondence between movie titles (i.e. titles as we see them listed in the theaters) and AACS Titles is not necessarily a one-to-one mapping. If so designed by the content author, a movie might have several AACS Titles associated with it. Figure 5-1 shows such an example, where each line represents a Title, or path through the content. In this example, Title1 might correspond to the movie as is shown in the theaters and Title2 might correspond to the full-featured version of the movie with some extra scenes. Notice that both titles share content ‘elements’. Playback of Title1 does not require an on-line connection, whereas playback of Title2 does. Figure 5-1 also shows that one or more Titles can be encrypted by the same Title Key. Following existing DVD-Video usage, AACS defines a *Title Set* to be a group of Titles that are encrypted with the same Title Key. Figure 5-1 shows two Title Sets. In this terminology, a Title Key is actually a Title *Set* Key, although we use the terms interchangeably. The equivalence mapping to the corresponding terminology for each Format is provided in the format specific books of this specification.

The following are examples of enhanced uses:

1. Access to Titles on Pre-recorded media where the Title Keys for that content is already on the media but an on-line connection is needed to grant access to the title. (Type B in Figure 5-1)
2. Access to Titles on Pre-recorded media where the Title Keys for that content was not included on the media (Type C in Figure 5-1)
3. Download of new Title Sets to be recorded on Recordable media

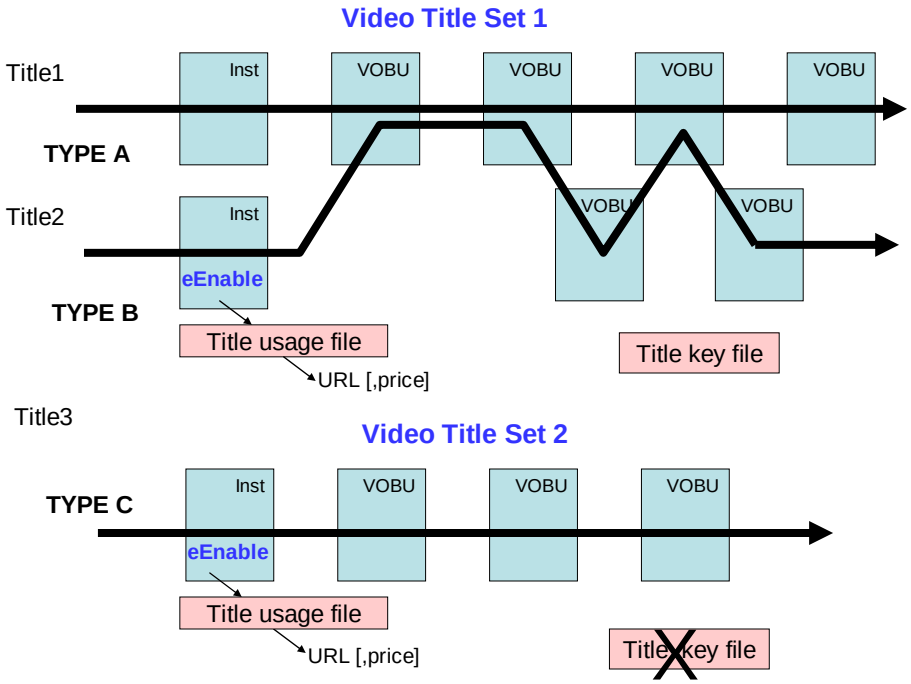



**Figure 5-1 – Example Titles (Using HD DVD)**

For the sake of clarity in this section, the precise definition of relevant terms is given as follows:

| | |
|---|---|
| Basic Title | A Title (on Pre-recorded or Recordable media) that is readily accessible by the set of Device Keys currently embedded in the device and the Encrypted Title Keys already present in the media. No on-line connection is required to access a Basic Title. All AACS compliant devices shall be able to access and playback Basic Titles. |
| Basic Playback | Fundamental behavior expected when playing any given Title. It refers to a baseline playback user experience that is analogous to the one currently provided by a standard DVD-Video player, i.e. menus, navigation, play title, scenes grouping, random playback, etc. Basic Playback is the only type of user experience that is available for Basic Titles |
| Enhanced Title | A Title that requires an on-line connection or extended function in the player in order to grant its access. |
| Basic Device | A device that can only provide a Basic Playback experience. A device that does have network connectivity but does not include the features defined in this chapter for accessing an Enhanced Title is considered a Basic Device. |
| Enhanced Device | A device that is capable of establishing an on-line connection and providing full access to both Basic and Enhanced Titles. An Enhanced Device, when the user has elected not to connect it to the Internet, shall operate as a Basic Device. |

| | |
|---|---|
| Remote Server | The remote computer that grants Enhanced Devices the right to play Enhanced Titles. The appropriate Remote Server for a particular Title will be identified in a Title Usage File. An Enhanced Device usually establishes an on-line connection to the Remote Server before playing Enhanced Titles. |
| Permission | A data record which is created during communication between an Enhanced Device and a Remote Server. The data record specifies whether that device is allowed to playback an Enhanced Title. |
| Title ID | Title IDs identify Titles within each Content ID. A single piece of content with a single Content ID may have many Titles (paths through the content) and therefore many Title IDs associated with it. Title IDs are defined by the given format groups and are generally small numbers. For example, the existing DVD standard defines that each Title Set may have up to 99 Titles numbered 1:99. In this case, the numbers 1 through 99 would be the Title IDs. |
| Content ID | The Content ID uniquely identifies the content to the Remote Server in an on-line transaction. The Content ID shall contain a registered number obtained from ISAN |

| | | |
|---|---|---|
| Content ID Type (4 bits) | Reserved (28 bits) | ISAN number: (96 bits) |

The Content ID shall be represented by a 128 bit value as follows

Content ID Type: 4 bits

1000 : ISAN number

Other values are reserved for future use.

Reserved: 28 bits (All zeros)

Content ID ISAN number: 96 bits. This value will contain a Versioned International Standard Audiovisual Number (V-ISAN, ISO 15706:2002 , http://www.isan.org)

All AACS-protected Titles have a corresponding entry in a Title Usage File (TUF). Access to Enhanced Titles may require an additional monetary transaction.

## 5.1 AACS On-line Enabled Content and AACS Stream Content

On-line Enabled Content and Streamed Content are very similar: in On-line Enabled Content, the content is already present on the media and an on-line transaction is needed to unlock it. In Streamed Content, the content is not present, and is delivered as a stream as part of the transaction. Note that Streamed Content should never be permanently stored on the media or in the device, that is, Streamed Content is always transient.

Content consists of one or more *Title Sets*, each of which consists of one or more *Titles*. All the Titles in a Title Set are encrypted with a single key. A single transaction enables the playing of a single Title. A Title is a particular path through the content. As shown in Figure 5-1, two Titles may share substantial elements, and it is perfectly normal for some elements within a priced Title to be common with elements in a free title. Thus, in

general, titles are not one-to-one with encryption keys, so the external permissions do not necessarily involve the delivery of more Title Keys, although they may. Regardless, players must obtain external permission to play a Title if it is denoted as an Enhanced Title, even if they can otherwise decrypt the content in that title.

AACS also defined the concept of a "logical disc" in order to be able to group all the Titles obtained in one on-line transaction from a specific URL. Such an example would be a "download and burn" operation where the user downloads a set of related Titles at once onto a particular physical media. Thus, on recordable media, it is not uncommon for a single piece of media to be associated with content from multiple Content IDs. In contrast, content on pre-recorded media is only associated with a single Content ID.

### 5.1.1 Permission Types

Three types of on-line permissions are defined, Default, Instant, and Cacheable. The following sub-sections provide precise definitions of each of these permission types. On-line Enabled Content can be associated with either an Instant or a Cacheable Permission. Streamed Content can only be associated with an Instant Permission.

#### 5.1.1.1 Default Permission

A Basic Title contains a Default Permission, the right to play. All titles in the Title Usage File are denoted as either Basic or Enhanced.

#### 5.1.1.2 Instant Permission

An Instant Permission is acquired from a Remote Server and allows the Enhanced Device to begin playing an Enhanced Title at the moment in time the Permission is acquired. The Permission must be re-acquired from the Remote Server before the device starts playing that Title again, and each acquisition may include a monetary transaction.

Fast forward and rewind within a title shall not require a new Permission. If the player leaves the title to play another title, or to display a navigation menu, and later is directed to play the title again, it must obtain a new Instant Permission.

#### 5.1.1.3 Cacheable Permissions

A Cacheable Permission is merely an Instant Permission with an additional keyword in the Title Usage File marking it as "cacheable". A device is allowed to store a Cacheable Permission and use it in the future without requiring any additional on-line connections to the Remote Server. It is the manufacturer's option to support Cacheable Permissions. There are no specific requirements for how much space a device must provide for caching Permissions. If the space is fully utilized, the device must treat all new Permissions as Instant Permissions until either more space is made available or some of the existing Permissions have been deleted.

If an Enhanced Device wishes to utilize "Cacheable Permissions", then the device must provide for storage of that Permission. If no storage is provided, then all Cacheable Permissions must be treated as "Instant Permissions". Difference classes of devices can store Cacheable Permissions in different ways.

- Persistent storage can be used to store Permissions until the Permission expires.
- Temporary storage can be used to store Permissions but the Permission will need to be re-acquired if the device loses power.
- Devices that do not provide any local storage for Cacheable Permissions will need to connect to the Remote Server to re-acquire a Cacheable Permission.

A Cacheable Permission may contain an expiration period after which the device must destroy the Permission and acquire a new Permission from the Remote Server. Devices that support Permission caches are not required to support Cacheable Permissions with expiration periods. In that case, however, the device shall treat the Permission with an expiration period as an Instant Permission. It is also possible that a device would lose its period timer when it powers off. Such a device shall forget the Cacheable Permission when it powers off.

Permissions may include an expiration time or a "do not play until" time. To acquire a Permission that contains a time, Enhanced Devices must provide a secure mechanism for determining the time that cannot be reset by the end-user, as described in the compliance section of the AACS license. In the case that a device does not support

secure time, it shall treat every time-related Permission as an Instant Permission, with one exception: a device, having once successfully acquired a “do not play until” Permission, may cache it indefinitely.

## 5.2 The Title Usage File

A Title Set must contain one Title Usage File (TUF). Fields contained in the TUF that may not be modified by compliant recorders are cryptographically bound to the content by including a hash of those fields in the content signature. Fields that may be modified by compliant recorders are cryptographically bound to the Title Key. The name and syntax of the TUF is Format-specific, as specified in the Format-specific books of this specification. The TUF denotes which Titles require the player to obtain external permission or keys before it plays them. If the Title is Enhanced, then the TUF contains additional information that identifies how a device acquires the necessary Permission to allow access to that Title. Each Title protected by AACS must be covered by an entry in the TUF.

Each On-line Enabled Content or Streamed Content Title is associated with a URL of the Remote Server which will permit it to be played.

The TUF shall identify the Content ID, and shall give information about each Title as explained below.

### 5.2.1 The Per-Title Information in the Title Usage File

The Title records shall, at a minimum, contain the following information. For preciseness of description, this information below is shown in a XML-like syntax; however, the actual syntax is Format-specific and may not be XML-like.

<title id=*titleID*
type=basic | enhanced
[url=*url*]
[cacheable [period=*hours*]
[after=*date/time*]
[before=*date/time*] ]
></title>

This record identifies the title. The “type” attribute indicates whether the Title is an Enhanced Title or a Basic Title. In some formats, the TUF may be the only indication that a Title is an Enhanced Title. Thus, players must check the TUF before playing any Title the first time, even if the player knows the Title Key.

If the Title is an Enhanced Title, the “url” attribute is mandatory and specifies the URL of the Remote Server that can provide the Permission to play the Title. The “url” attribute shall be omitted if the Title is a Basic Title.

If a Permission obtained for this Title may be cached, the Title Record will have the “cacheable” attribute. Titles whose Permissions are cacheable may be further optionally modified by the “period”, “after”, or “before” attributes. These attributes have the following precise meanings:

- The “period” attribute indicates the amount of time that the Permission may stay in the cache until it must be deleted. A player may always delete it earlier.
- The “after” attribute indicates that a player may not begin playing the title until the date and time specified.
- The “before” attribute indicates that a player may not begin playing the title after the date and time specified.

The dates and times in the “after” and “before” attributes shall specify global rather than local time. For example, the date and time might be specified in Greenwich Mean Time. Note that since Permissions govern the right to *start* playing the Title, it is irrelevant if a time-based permission expires during the play of a Title. The player shall continue to play the Title, including any user interactions within the Title such as rewind.

Players are not required to support any time-based conditions; the Remote Server is certainly capable of enforcing those conditions. A player that does not support a time-based condition shall ignore the “cacheable” attribute and treat any Permission obtained as an Instant Permission, with one exception: a player obtaining a Permission for a Title that is denoted as “after” in the TUF may cache that permission. If a player supports a time-based condition, it must have a Secure Clock for that purpose.

It is always optional for a device to implement a Secure Clock. It is also very possible that a device has a Secure Clock for the purpose of measuring elapsed time, but does not have a Secure Clock for the purpose of determining the calendar time (in which case the device shall fall back to the server for permissions with the "after" or "before" attributes). It is also possible that a device has a secure calendar clock set at manufacturing time, but after some years, due to expected clock drift, the device shall no longer treat its clock as a Secure Clock. Of course, a device that can periodically consult a trusted clock source can maintain a Secure Clock indefinitely. A trusted clock source may be, for example, a cryptographically secured time value from an Internet server, or an in-the-clear over-the-air broadcast signal such as WWV in the United States.

A Secure Clock is defined as a clock that can keep sufficiently accurate time even in the presence of attacks designed to modify the clock. Alternatively, it is a clock that can detect such attacks and will cease acting as a Secure Clock if tampering is suspected. In other words, the device shall discard all time-based permissions from its cache and cease caching new time-based permissions until and unless its tampered clock reacquires its Secure Clock state. The assumptions on the attackers' resources are defined in the AACS compliance rules. For calendar time purposes, a Secure Clock must keep time to within 10 minutes of Greenwich Mean Time. For elapsed time purposes, a Secure Clock must not gain or lose more than 1 minute in 24 hours.

### 5.2.2 Errors in the Title Usage File

Since the integrity of the TUF is guaranteed cryptographically, any syntax or logic errors in the TUF are due to authoring errors. Therefore, in general, the player action in the case of detecting syntax errors in the TUF is manufacturer-specific.

However, to remain compatible with potential future extensions to the TUF, the following shall not be treated as errors:

1. Any unknown attributes associated with a Title. There are two cases:
    a. If there are any unknown attributes in a Basic Title, they are ignored.
    b. If there are any unknown attributes in an Enhanced Title, the Title shall be treated as an Enhanced Title with Instant Permissions only, even if it has the "cacheable" attribute. In other words, the Remote Server shall handle any Enhanced Titles with unknown attributes.
2. Any non-zero values in reserved fields. The player shall ignore values in reserved fields. Likewise, it shall ignore any data outside of the fields associated with the individual Titles.

## 5.3 Connection Protocol

In general, the protocol a player uses to connect to the Internet and obtain a Permission is format-specific, and is described in the relevant Format-specific book in this specification. However, in the case that a given Format does not describe the connection protocol, players supporting this format shall implement the default protocol defined below. Players supporting formats who have defined their own protocols are not required to implement the default protocol.

Regardless of the protocol used, players shall use the following Application Programming Interface between the AACS secure layer in the player and the not-necessarily-secure Internet application layer in the player. This API is shown in the following figure:

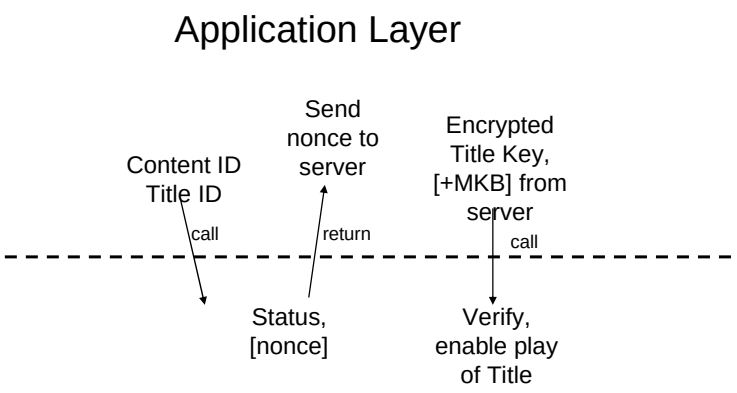



**Figure 5-2 – Transaction Protocol API**

The API works as follows:

1. The application layer desires to play an Enhanced Title (being asked to do so by user navigation). It calls the AACS layer, passing the Content ID and the Title ID of the Title it wants to play.
2. If the permission is already in the cache, AACS Layer sets the status to "Permission Already Obtained" and returns. The Enhanced Title will play.
3. If the permission is not in the cache, the AACS layer generates a 128-bit nonce (random number). The AACS layer returns the nonce and one of two status states:
    a. "Normal". This state is returned if the permission is an Instant Permission, or if it is a Cacheable Permission and the player knows it has not been previously obtained.
    b. "Potentially Lost". This state is returned if the Permission is a Cacheable Permission, and the player does not have a cache. This state is also returned if the player has a cache, but has deleted entries from the cache that could potentially cover this Content ID. This state is also returned if the Cacheable Permission has a time-based condition, and the player has not cached the Permission because it does not have a Secure Clock. This state is intended to allow the application layer to query the server and have a more user-friendly interaction with the user, in the case that the user has already purchased the Permission.
4. The application layer sends the nonce to the Remote Server, together with whatever other information is defined in the protocol. This additional information is usually the Content ID, the Title ID, and the financial information for the transaction.
5. If the transaction succeeds at the Remote Server it shall send an Encrypted Title Key by way of granting the permission. The formula of the Encrypted Title Key is called out in the specific Format books of this specification, with the addition that the Remote Server will have XORed the Title Key with the nonce and with a hash of the Content ID and Title ID before encrypting it. In general, this means that the Encrypted Title Key formula is of this form:

    $$\text{AES-128E}(K_{vu}, K_t \oplus \text{nonce} \oplus \text{AES\_H}(\text{Content ID} \,||\, \text{Title ID}) \oplus \ldots)$$

6. In addition, the Remote Server may send a new Media Key Block in the case that Media Key Block on the media is out-of-date. In this case, the Volume Unique Key $K_{vu}$ is calculated assuming the Media Key is the XOR of the Media Key on the media with the Media Key in the new Media Key Block.
7. The application layer passes the Encrypted Title Key to the AACS Layer, together with the new Media Key Block if there is one.

8. The AACS layer calculates the Title Key from the formula in step 5, and uses it to play the Enhanced Title, in lieu of any Title Key it finds on the media. In this calculation, the AACS layer must remember and use the nonce it created in the previous call; it shall not trust the application layer to pass the nonce to it.

This paragraph is informative: The Remote Server has a database which, at the very least, will contain the Title Keys that are missing on pre-recorded media. It is convenient that this database might contain other data, such as the Media Key and version number of the Media Key Block in use in this particular Content ID. The database might also contain the attributes found in the content's Title Usage File. So, in general, the protocol need not require the player to send the MKB or the TUF to the Remote Server.

### 5.3.1 Default Connection Protocol

Devices are only required to implement the Default Connection Protocol if all of the following conditions are true:

- The device is an Enhanced Device
- The device implements AACS for at least one Format that does not define a specific method or protocol for performing on-line transactions

The following subsections (5.3.1.1through 5.3.1.4) define the *Default Protocol* to be implemented by such devices.

#### 5.3.1.1 Platform Requirements for the Default Protocol

It is the choice of the device manufacturer which Internet connection method is supported, for example, Ethernet, IEEE 1394, wireless, etc. Regardless, an Enhanced Device implementing the Default Protocol must support:

1. DHCP client.
2. HTTPS client. In particular, the device is required to support Transport Layer Security (TLS) 1.0 with server-side authentication using RSA, AES-128, and SHA-1 only. (In other words, the supported cipher suite is TLS_RSA_WITH_AES_128_CBC_SHA.) The key length of RSA is 2048 bits. The 'e' of Public Key is $2^{16}+1$. The only certificate authority that the device needs to recognize is the AACS LA. The device connecting using HTTPS can assume that the server will have a certificate signed by AACS LA.

#### 5.3.1.2 User Accounts for Default Protocol

This paragraph is informative, describing a typical way in which On-line Enabled Content and Streamed Content features would be paid for by consumers. The end-user interested in invoking the On-line Enabled Content or Streamed Content functions is expected to set up an account with the content provider, or with a service center acting on the content owner's behalf. There may be many different ways for the user to register, for example, he may connect to a Web site, call an 800 number, or mail in a form provided with the media packaging. It can be assumed that registration information would be available in printed material with the media packaging, and/or playable on the media itself. In any event, the registration process does not involve the player. After registration, the user will have received an account number (a decimal number), and a personal identification number (PIN), also a decimal number.

The next two paragraphs are normative for the Default Protocol. The player prompts the user to enter the account number and PIN when certain navigation paths through the disc content require additional payment. As explained below, the additional payment requires the player to connect to a URL. At the player's option, the player may offer to remember the account number, to simplify future user interactions. In that case, the player shall assume that each domain name in a URL shall be associated with a single account number. Also at the player's option, the player may offer to remember the PIN associated with the account number. It is strongly recommended that players selecting this option shall make it part of any parental guidance features that they provide.

Player must always ask for positive acknowledgement from the user before proceeding through an additionally priced content, even if the player has remembered the necessary account information. Players must also provide a way for the user to delete the stored information, for example, when the user wants to sell the player.

#### 5.3.1.3 User Interface Requirements for the Default Protocol

If access to a Title requires a monetary transaction or the transmission of user account information to a Remote Server, then the Device must provide for a mechanism to notify the user that this is the case. It must also provide a way to cancel the transaction if so desired.

The players shall ask for user permission before performing a transaction using the Default Protocol. However, in the case of a free transaction, the player may, at its option, accept a stated user preference in lieu of an individual confirmation. Of course, if a free transaction requires an account/PIN the user would have authorized the player to have cached the account/PIN in order to avoid the confirmation.

#### 5.3.1.4 Default Connection Protocol Syntax

When a player needs an On-line Enabled Content Permission or a Streamed Content key, the player connects to the Internet and does an HTTPS POST to the URL associated with that Title. The post data it sends shall be in the following format, in readable ASCII:

```
version=1.0
title=id
nonce=nonce
contentId=contentID
[serialNo=serialNo]
playerId=playerId
[language=language]
[price=price
account=nnnnnn…nnnnnn
pin=nnnnn…nnnnnn]
```

Initially, the price and account/PIN information are omitted. Only if the initial transaction fails (as described below) would a subsequent request include this information. The Serial Number line shall be omitted if the disc does not have a unique machine-readable Serial Number. The “language” line may be omitted, but if the end user has expressed a language preference to the player, the player can use the “language” line to communicate that to the service center. The service center, in turn, may use that preference to format potential error messages. The language is denoted with the English name for the language, for example, “English”, “Japanese”, “Spanish”, “Mandarin”, etc. The number of the title shall be in decimal. All other IDs and the nonce shall be in readable hexadecimal, with ‘a-f’ in lower case. The price is in the form currency:nnn, e.g., USD1.00 means $1.00. The account number and PIN shall be decimal digits. Each line is ended by the line feed character ‘\n’ (0x0A). There shall be no imbedded white space characters within the lines.

The service center responds to the POST in one of three ways:

1. “go=< data>\n
   [date=yyyy/mm/dd hh:mm:ss+ts]
   [mkb=<mkb>\n]”
2. “price=<price>\n”
3. “error=<message>”

The first is the normal response permitting play. The data is readable hexadecimal of the Encrypted Title Key. The Title Key is encrypted as described in Section 5.3. This Title Key applies to both Streamed Content and On-line Enabled Content. It is often the case that the player already knows the Title Key. In that case, it shall ignore the old Title Key and use the Title Key decrypted from the “go” response.

Optionally, the service provider may follow the “go” line with a “date” line, for the benefit of players that are implementing cacheable periods, but cannot maintain the period when powered off. In this case, the player shall record the time when the original Permission was received. When the player powers on again and gets the original Permission a second time, it calculates the difference in time in order to determine how much time remains in the period.

Optionally, the service provider may also follow the “go” line with an “mkb” line, transmitting a new MKB. This might be used, for example, if the MKB on the media was seriously down-level. If the player receives a new MKB, it must recalculate the content unique key using a new Media Key that is the bitwise XOR of the Media Key on the media and the Media Key calculated from the new MKB in the response. That new content unique key shall be used to decrypt the data in the “go” line.

The second response ("price") occurs if the Permission is offered for sale. If the player receives the "price=" response, it shall stop for confirmation from the user before proceeding. If the user confirms the purchase, the player shall repeat the transaction, this time sending the price and the user's account information. The format for the price is the same currency:nnn format. If the price is offered in more than one currency, they are listed in a comma-delimited list surrounded by braces. For example:

```
{USD2.00,YEN150}
```

The Remote Server may respond with a "price" response, but indicate that the price is zero. The content owners may choose this option when they want to offer a free feature, in return for learning the identity of the consumer. The player shall treat this "price=0" feature identically to non-zero priced features; in other words, it shall ask for user confirmation before continuing. In this case, however, the player may use a previously stated user preference in lieu of asking for confirmation.

If the permission was previously purchased by the user, but had merely fallen out of the player's cache, the service center shall transmit the "go" response without demanding payment. However, service centers may use their own criteria in determining if a key has been previously purchased and therefore can be given again for free. For example, they might use the media ID, the player ID, the account number, or some combination of these things. Whatever heuristics the service center uses to determine that a permission request is a repeat, the service center shall guarantee that repeats due to connectivity problems shall not yield duplicate charges.

If the response from the service center begins "error=", the response shall be an explanatory response intended for human eyes, for example, "your credit card has expired". The first six characters are in ASCII; the remaining characters immediately after the '=' character are in Unicode. How the player handles this response is player specific.

## 5.4 AACS Network Download

"AACS Network Download" content is content that is prepared on a server to be downloaded directly bound to AACS media. In general, the client does not require AACS device keys and is outside of the scope of this specification. Of course, the device that finally plays the content will need a set of device keys.

AACS Network Download also refers to content that is being downloaded to a bulk storage device such as a hard disk, as long as the content itself is cryptographically bound to a piece of AACS media. In other words, content cryptographically bound to a piece of AACS media *is* AACS content, even if it is not physically stored on that media. Note that if AACS content is stored on bulk storage, the content itself need no longer be bound to AACS recordable media. It could be bound to pre-recorded media.

### 5.4.1 Basic Approach

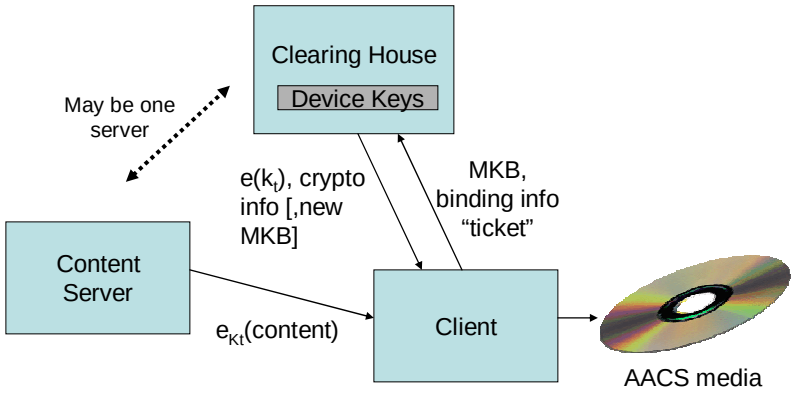



**Figure 5-3 – Example System for AACS Network Download**

Figure 5-3 shows an example system. It consists of a Content Server, a Clearing House, and a Client, although the Content Server and the Clearing House functions may be combined in a single box.

The Content Server delivers content prepared for recording on AACS media. This means that the content is encrypted with AES using a randomly selected title key, $K_t$, as specified for the given format in one of the other books in this specification. Although this figure shows a DVD recordable disc as an example, the target could be any licensed AACS media. The content, as stored on the Content Server, is not yet playable by AACS devices. It is missing an additional step, involving the Clearing House.

That missing step is the *media binding* operation. Four different binding methods are allowed and are defined in Section 5.5 below. The details of this media binding operation are different for the different binding methods, but fundamentally, the Clearing House receives the Media Key Block and other information from the Client and generates the necessary cryptographic information. This generated cryptographic information includes an Encrypted Title Key; that is, the Clearing House encrypts the Title Key in a key based on the Media Key. The Clearing House knows the Title Key for the content based on a relationship with the content owner. In general, the Clearing House sends additional cryptographic information that binds the content to a particular piece of media using one of the four defined binding methods. In the case of Pre-recorded media, if the media does not have a Pre-recorded Serial Number then any binding method that utilizes the Pre-recorded Serial Number would not be available.

The Client places the encrypted Title Key and the other cryptographic binding information along with the encrypted content from the Content Server on the AACS media, as specified in the corresponding AACS book.

How the Clearing House learns the Title Key is implementation-specific. For example, the Clearing House could have a data base associating pieces of content to Title Keys. Alternatively, the content from the Content Server may contain the Title Key, encrypted in a key only the Clearing House knows. We call that the *Prepared Title Key* to distinguish it from the Encrypted Title Key that is the final output of the Clearing House. In the latter case, that Prepared Title Key data would be transmitted from the Client to the Clearing House as part of the *ticket* data.

### 5.4.2 The Ticket

We use the term “ticket” to describe the data that passes from the Client to the Clearing House. The ticket serves as a “proof of purchase” or “proof of entitlement” to a piece of content. At a minimum, the ticket shall contain the following information:

1. The binding information to which the content will be bound.
2. The identification of the content. For example, this may be a Content ID or a Prepared Title Key.

That might be the entire ticket: for example, a system might operate by mailing special blank media to its subscribers. Possession of the special media is proof of entitlement, and, since the Clearing House would only bind to special media, no cryptographic authentication on the ticket would be needed. However, in most systems, the ticket is more involved, and may contain things like a ticket ID, a store ID, and/or a Client ID, and may be digitally signed to prove authenticity.

How the ticket is delivered to the Client is outside the scope of this specification. For example, it may be delivered by an on-line store as part of a financial transaction. In that case, the consumer’s client software would transmit the binding information when it purchased the content.

After verifying that the ticket is authentic, the Clearing House performs the media binding operation, confident that the user is entitled to the content.

Note that because the binding information is included in the ticket, no replay attack is possible. If the client fails to receive the Encrypted Title Key or any of the cryptographic information for any reason, the media binding operation can be safely repeated. The resend does not constitute an additional copy, because it is bound to the same thing.

### 5.4.3 Content Delivery

All methods of delivering the content are within this specification. For example, the encrypted content may be delivered ahead of time, before it is actually purchased, and the purchase would involve only the delivery of the Encrypted Title Key. Alternatively, the delivery of the content may wait until the user has a valid ticket.

In some cases, the content may not even be delivered by electronic download. For example, the content may be delivered on pre-recorded media. In that case, the client software would copy the pre-recorded content onto recordable AACS media (if it were not already there), and then connect to the Clearing House to obtain the Encrypted Title Key to make it playable.

### 5.4.4 Client-to-Clearing-House Protocol

The particular syntax of the Client to Clearing House is implementation specific. Regardless of the syntax, the protocol must proceed as follows:

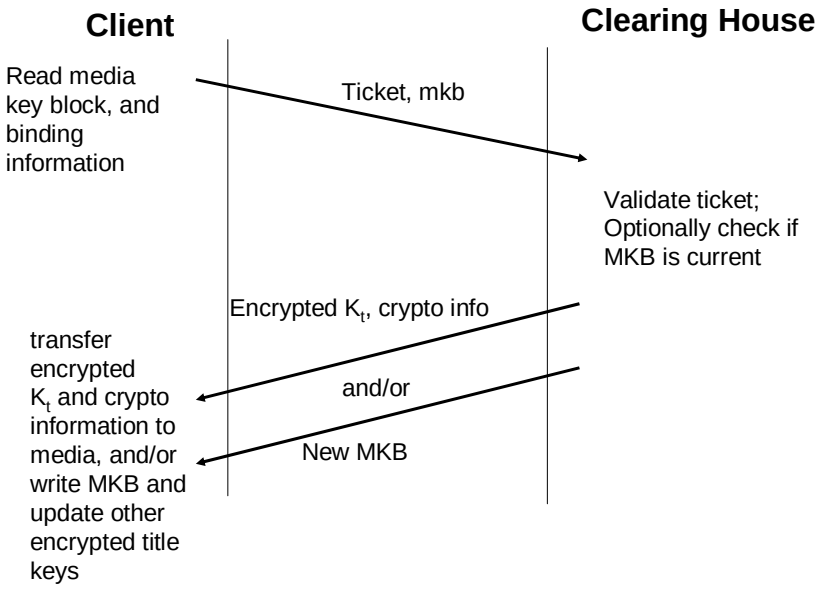



**Figure 5-4 – Normal Client-to-Clearing-House Flow**

The client connects to the Clearing House, transmitting the ticket and the Media Key Block on the user's media. The Clearing House normally responds with the Encrypted Title Key and the cryptographic information, which the Client records on the media.

The Clearing House may want to update the Media Key Block on a given piece of media. This specification does not require that the Clearing House support this function; however, the Clearing House operates on behalf of the content owners, and the owners have a vested interest in having Media Key Blocks be at the latest level. Therefore, a Clearing House might be required to support updating the Media Key Block by the content owners.

Instead of or in addition to responding with the Encrypted Title Key, it may respond with the Media Key Block itself. The Client writes the Media Key Block in the appropriate file on the media. If the Encrypted Title Key has not yet been transmitted, the Client simply retries the original request, this time transmitting the new Media Key Block. This will be acceptable to the Clearing House, and the modified request will succeed.

#### 5.4.4.1 Updating Other Encrypted Title Keys

However, if the media already contains other content of the same type, the new Media Key Block will destroy it unless those Encrypted Title Keys are updated. If the Client has a set of device keys, it can perform the update itself. However, there are many advantages if the Client can remain secret-free, so the Clearing House can also perform this transformation.

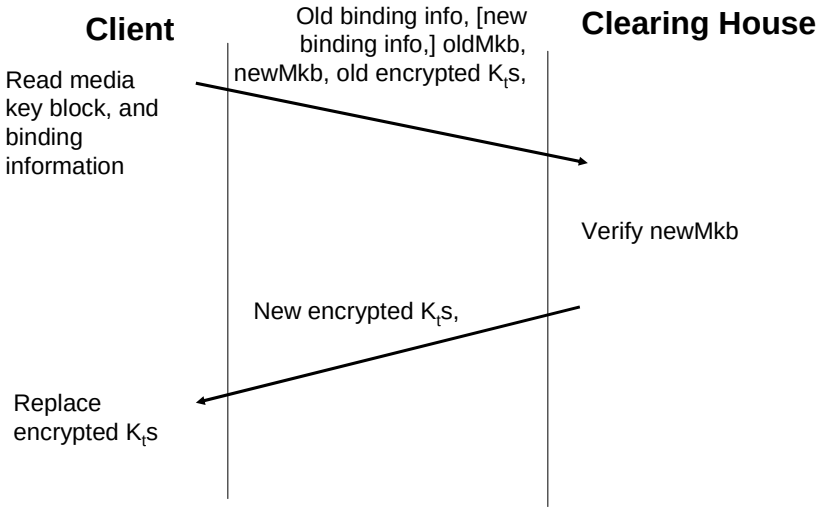



**Figure 5-5 – Updating Existing Encrypted Title Keys**

If the Clearing House is performing the update of the existing title keys, the protocol proceeds as shown in Figure 5-5. The Client transmits the old binding information, the old Media Key Block, the new Media Key Block, and the encrypted title keys. In the case of recordable media, the Client must also transfer the new binding information in the form of the new Binding Nonce, since Encrypted Title Keys cannot be written on the media without writing a new Binding Nonce.

The Clearing House shall verify that the new Media Key Block is one that it is currently using. In other words, the Clearing House must not act as a general purpose transformation engine from one Media Key Block to another. It is *not* required, however, to verify that the binding information corresponds to binding information used in a previously validated ticket.

It is an error if the new Media Key Block is out-of-date or is not one that originated from the Clearing House. The Media Key Block should not be out of date if the Client is acting in a timely manner; the Media Key Block would have been the one just delivered. Nonetheless, if an error occurs, the Client shall simply restart the original transaction, using the old Media Key Block. This will force the Client and the Clearing House to synchronize on Media Key Blocks.

### 5.4.4.2 Caching Media Key Blocks

It is permissible for the Clearing House to cache Media Key Blocks and Media Keys, and so doing will somewhat reduce the amount of data that the Client usually needs to transfer to the Clearing House. Instead of transmitting the entire Media Key Blocks, a system may be designed for the Client to simply substitute the 16-byte verification data instead of the entire Media Key Block. (The verification data is the data part of the Verification Record in the Media Key Block.) The protocols above work exactly the same, except for that substitution.

However, the protocols all have a possible additional response from the Clearing House: the Clearing House can respond that it does currently have the Media Key Block in its cache. In that case, the protocols proceed as illustrated in Figure 5-6.

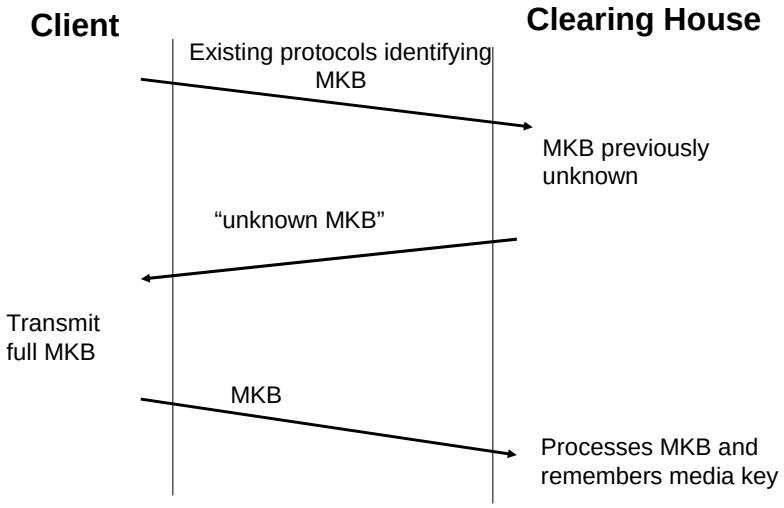



**Figure 5-6 – Client-to-Clearing-House Protocol when an MKB is MKB is not known**

In this case, the Clearing House responds with a message that indicates that the Media Key Block is unknown to it. The Client shall read the missing data from the media and transmit it. The Client then re-initiates the original protocol, which should now succeed.

The Clearing Houses must treat Media Keys as highly confidential data as described in their license.

### 5.4.5 The Clearing House

The function of a Clearing House is to transform Title Keys so that they are bound to particular pieces of media. The Clearing House shall obtain Title Keys based on their relationships with content owners. The Clearing House shall *not* obtain Title Keys from existing pieces of media to copy them to new pieces of media. The Clearing House must not act as a copying center, copying content from one piece of media to another, *even if the copy control information on the source media permits a copy*.

However, the Clearing House may transform title keys within a single piece of media, as part of updating a Media Key Block, as described in Section 5.4.4.1 above.

## 5.5 AACS Media Binding

AACS content is content that is bound to AACS media even if the content is stored in bulk storage and not actually stored on AACS media. AACS defines four methods of binding for this purpose. The source of content that could utilize these binding methods may be via Network Download as defined in Section 5.4 or an alternative source such as a Pre-recorded media. The four defined binding methods are as follows.

1. **Media Binding**. The unique Media ID and Binding Nonce on recordable media, or the Serial Number and Volume ID on the pre-recorded media, are transmitted from the Client to the Clearing House. The Clearing House generates a MAC on the Media ID or the Serial Number and returns it as part of the binding. In the case of recordable media, the Clearing House encrypts the Title Key with the Protected Area Key as described in the recordable book of this specification. In the case of pre-recorded media, the Clearing House encrypts the Title Key with the Volume Unique Key as described in the pre-recorded book of this specification.
2. **Content Binding**. In this case, the downloaded content is bound to *any* copy of a particular content item, not to a particular piece of media. The downloaded content itself may be delivered encrypted using the Title Key already present on the media, in which case no transaction with the Clearing House

is necessary. Alternatively, for pre-recorded media only, the Client may deliver the pre-recorded media's Volume ID to the Clearing house, and the Clearing House will provide an Encrypted Title Key, encrypted in the Volume Unique Key. The following sentence is informative: Note that this binding permits sharing of the downloaded content amongst all users who have the original content, so this binding method should not be used for high-value downloaded content.

3. **Device/Content Binding**. In this case, the downloaded content is bound to a particular device in addition to being bound to any copy of a particular content item. In this case, a nonce is generated by the device and stored in a way that can not be modified by the user. This nonce N is combined with the content item's Media Key and is used to produce a device-and-content unique key as follows:

   $K_{dc} = AES\text{-}G(K_m, N)$

   The player, acting as a Client, sends the nonce to the Clearing House, and the Clearing House will provide an Encrypted Title Key, encrypted in $K_{dc}$.
4. **Device/Media Binding**. In this case, the most restrictive binding, the downloaded content is bound to both a particular piece of media, and a particular device. The device, acting as a Client, creates a nonce for a device-and-content key as in the Device/Content Binding case and sends the nonce to the Clearing House. The Client also sends the Media ID in the case of recordable media and the Serial Number in the case of pre-recorded media. The Clearing House responds with a MAC on the Media ID or Serial Number, and an Encrypted Title Key, encrypted in $K_{dc}$.

If specified by instructions associated with the content, devices may use AACS cryptography to provide a secure "per-Content-Provider" partitioning of local bulk storage, as may be required by some Format groups. The player calculates a per-Content-Provider key $K_{cp}$, as follows:

$$K_{cp} = AES\text{-}G(K_d, ID_{cp} \| 00000000000000000000000000000000_{16})$$

The $K_d$ is a unique device key provided by the device (and is not an AACS key), and $ID_{cp}$ is the Content Provider ID from the Content Certificate, as described in the Pre-recorded book of this specification. Prior to computing $K_{cp}$, a device must verify the signature of the Content Certificate and verify that it corresponds to the actual content on the media as described in the Pre-recorded book. The device can then use $K_{cp}$ to encrypt and decrypt the partition of its local storage that is assigned for that Content Provider. A device may also use the verified AACS Content Provider ID as a partition ID and restrict access to that partition except while playing verified AACS media from that Content Provider.

This page is intentionally left blank.

# A Appendix
# Calculating the “uv” Values of Device Keys

## A.1 Calculating the “uv” Values of Device Keys

It is not necessary for a device to store the uv values associated with its device keys. It is sufficient for a device to store only its device node; from that, it can calculate the *uv* values of its device keys. The following working Java program illustrates this calculation. The calculation determines the root of every sub-tree for which the device knows every key.

This program lists all possible such keys. In most AACS applications, the number of keys will actually be less, because the keys in large sub-trees (high *u*) are not actually populated. This program lists keys in the reverse order of the size of the sub-trees. If your AACS application has less keys, just ignore the later *uv* values.

```
package com.aacsla;

import java.util.Enumeration;
import java.util.Vector;

public class ListKeys
{

	/**
	 * @param args "deviceNode treeheight"
	 */
	public static void main(String[] args)
	{
		if (args.length != 2) {
			System.out.println("Format: deviceNode treeheight");
			return;
		}
		long deviceNode;
		if (args[0].startsWith("0x")) {
			deviceNode = Integer.parseInt(args[0].substring(2),16);
		} else {
			deviceNode = Integer.parseInt(args[0]);
		}
		int height = Integer.parseInt(args[1]);
		deviceNode |= 1L << height; // we turn on a high bit for formatting
purposes only
		Vector uvs = listKeys(deviceNode, height);
		for (Enumeration i = uvs.elements(); i.hasMoreElements(); ) {
			Object uv = i.nextElement();
			System.out.println(uv);
		}
	}

	/**
	 * The recursive function actually listing the keys
	 */
	public static Vector listKeys(long deviceNode, int height)
	{
		Vector r;
		if (height > 1) {
			r = listKeys(deviceNode, height - 1);
		} else {
			r = new Vector();
		}
		Node u = new Node(deviceNode, height);
		while (--height >= 0) {
			int mask = 1 << height;
```

```
                Node v = new Node(deviceNode ^ mask, height);
                r.addElement(new UV(u, v));
            }
            return r;
        }

        /**
         * Holds and formats a node in a tree
         */
        static private class Node
        {
            private final static String x = "xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx";
            private long path;
            private int height;

            private Node(long path, int height)
            {
                this.path   = path;
                this.height = height;
            }

            public String toString()
            {
                String r = Long.toString(path, 2);
                // here is where we strip out the high order bit we turned on
                return r.substring(1, r.length() - height)
                       + x.substring(0, height);
            }
        }

        /**
         * Holds and formats a subset difference (node u
         * minus node v)
         */
        static private class UV
        {
            private Node u,v;

            private UV(Node u, Node v)
            {
                this.u = u;
                this.v = v;
            }

            public String toString()
            {
                return ("U = " + u + "; V = " + v);
            }
        }
}
```