Content Protection for eXtended Media Specification

Introduction and Common Cryptographic Elements

Intel Corporation

International Business Machines Corporation

Panasonic Corporation

Toshiba Corporation

Revision 0.85 Preliminary Release

September 27, 2010

NOT FOR LICENSE OR IMPLEMENTATION AT THIS TIME

This page is intentionally left blank.

Preface

Notice

Intellectual Property

Implementation of this specification requires a license from the 4C Entity, LLC.

Contact Information

Please address inquiries, feedback, and licensing requests to the 4C Entity, LLC:

- Licensing inquiries and requests should be addressed to 4C-Services@4CEntity.com.
- Feedback on this specification should be addressed to 4C-Services@4CEntity.com.

The URL for the 4C Entity, LLC web site is http://www.4CEntity.com.

This page is intentionally left blank.

Table of Contents

List of Figures

This page is intentionally left blank.

List of Tables

This page is intentionally left blank.

Chapter 1
Introduction

1. Introduction

1.1 Purpose and Scope

The *Content Protection for eXtended Media Specification* (CPXM) defines a renewable method for protecting content recorded on a number of physical media types. This series of CPXM Specifications are different from the CPRM Specifications. A main difference is the cipher algorithm and the use of a technology that implements broadcast encryption and the associated Media Key Block (MKB). The AES cipher is used in CPXM instead of the C2 cipher in CPRM. Therefore, the MKB and Device Key spaces are also different. MKBs and Device Keys for CPRM cannot be used for CPXM. The specification is organized into several "books". This document, the *Introduction and Common Cryptographic Elements* book, provides a brief overview of CPXM, and defines cryptographic procedures that are common among its different uses. Other books provide additional details specific to using CPXM protection for different applications and media types. Other books of the CPXM Specification available at or around the time of this publication are:

- SD Memory Card Book

Books covering other media types are expected to be available in the future (see Section 1.5 below). CPXM is an integral part of an overall system for protecting content against unauthorized copying, known as the Content Protection System Architecture (see the corresponding reference in Section 1.4).

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 the 4C Entity, LLC is responsible for establishing and administering the content protection system based in part on this specification.

This revision of this document is for evaluation purpose only.

1.2 Overview

The CPXM technology is designed to meet the following criteria:

- It meets the content owners' requirements for robustness and system renewability.
- It is applicable for both audio and video content.
- It is equally suitable for implementation on PCs and CE devices.
- It is applicable to different media types.

The system is based on the following technical elements:

- Key management for interchangeable media
- Content encryption
- Media based renewability

Figure 1-1 shows a simplified illustrative example of how the system operates. The actual details of component storage and cryptographic key management will vary with different types of SD Memory Cards and other supported media, as well as with different applications, as described in the other books of this specification.

Step 1a. The 4C Entity, LLC provides secret device keys to the device manufacturer for inclusion into each device produced.

Step 1b. Media manufacturers place a Media Identifier and Media Key Block generated by the 4C Entity, LLC on each piece of compliant media.

Step 2. When compliant media is placed within a compliant drive or player/recorder, a secret Media Key is generated by the device using its secret keys and the Media Key Block stored on the media itself. The same secret Media Key is generated regardless of which compliant device is used to access the media.

Step 3. Content stored on the media is encrypted/decrypted by a secret Title Key. The Title Key is encrypted and stored on the media using a key derived from a one-way function of the Media Key and Media ID. The copy control information (CCI) associated with the content is also encrypted by the Title Key. Both encrypted Title Key and encrypted CCI are stored in Protected Area which requires to authenticate the device to access.

Again, actual details of key management can vary among different applications, as described in the other books of this specification.

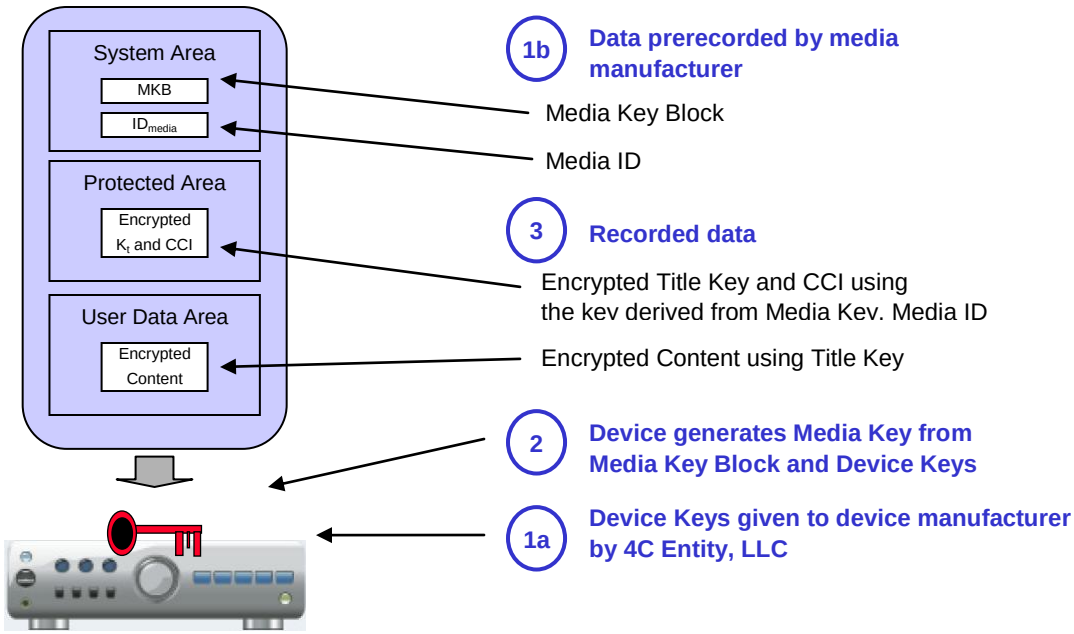

Figure 1-1 – CPXM Illustrative Example

1.3 Organization of this Document

This document is organized as follows:

- Chapter 1 provides an introduction and overview of CPXM.
- Chapter 2 describes common CPXM cryptographic functions based on the AES cipher algorithm.
- Chapter 3 describes a common CPXM cryptographic key management procedure, using a Media Key Block.

1.4 References

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

4C Entity, LLC, *CPXM License Agreement* (Unpublished)

4C Entity, LLC, *Content Protection System Architecture White Paper, Version 0.81*

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

National Institute of Standards and Technology (NIST), *Recommendation for Random Number Generation Using Deterministic Random Bit Generators (Revised), NIST Special Publication 800-90, March 2007*.

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), *Advanced Encryption Standard (AES), FIPS Publication 197, November 26, 2001.*

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 Future Directions

With its robust cryptography, key management, and renewability mechanisms, it is expected that CPXM will develop and expand, through additions to this specification, to address content protection for additional media types, application formats, and usage models.

1.6 Notation

1.6.1 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., $3C2_{16}$).

1.6.2 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.

1.6.3 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.
x / y	Division of x by y.
floor(x)	Truncated 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

1.7 Abbreviations and Acronyms

The following is an alphabetical list of abbreviations and acronyms used in this document:

4C	4 Companies (IBM, Intel, Panasonic, and Toshiba)
AES	Advanced Encryption Standard
ATA	AT Attachment
CBC	Cipher Block Chaining
CCI	Copy Control Information
CE	Consumer Electronics
CPRM	Content Protection for Recordable Media
CPXM	Content Protection for eXtended Media
ECB	Electronic Codebook
FIPS	Federal Information Processing Standards
ID	Identifier
LLC	Limited Liability Company
lsb	Least Significant Bit
MKB	Media Key Block
msb	Most Significant Bit
PC	Personal Computer
SD	Secure Digital
XOR	Exclusive-OR

Chapter 2
CPXM Common Cryptographic Functions

2. CPXM Common Cryptographic Functions

This chapter describes common cryptographic functions that are used by CPXM for various applications and media types. The functions are described here in isolation; their specific uses as part of CPXM encryption, key management, and renewability mechanisms are described elsewhere in this document, as well as in the other books of this specification.

2.1 AES Block Cipher Algorithm

Common cryptographic functions used for CPXM are based on the AES block cipher. A description of the AES block cipher algorithm is provided in the FIPS Publication 197, as referred to in Section 1.4. That specification describes two basic operational modes of the AES cipher: Electronic Codebook (ECB) mode and Cipher Block Chaining (CBC) mode. The remainder of this section describes notation that will be used in this document and in other books of this specification to refer to those two modes of operation.

2.1.1 AES Block Cipher in Electronic Codebook (ECB) Mode

In this document and in other books of this specification, encryption with the AES cipher in Electronic Codebook (ECB) mode is represented by the function

AES_E(k, d)

where k is a 128-bit key, d is a 128-bit value to be encrypted, and AES_E returns the 128-bit result.

Decryption using the AES cipher in ECB mode is represented by the function

AES_D(k, d)

where k is a 128-bit key, d is a 128-bit value to be decrypted, and AES_D returns the 128-bit result.

2.1.2 AES Block Cipher in Cipher Block Chaining (CBC) Mode

The AES cipher is used in Cipher Block Chaining (CBC) mode for encryption and decryption of content protected by CPXM. In this document and in other books of this specification, encryption with the AES cipher in CBC mode is represented by the function

AES_ECBC(k, d)

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

Decryption using the AES cipher in CBC mode is represented by the function

AES_DCBC(k, d)

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

The size of the frame of data to be encrypted or decrypted (i.e. how often a new CBC cipher chain is started) depends on the particular application format, and is defined for each in the corresponding books of this specification. The initialization vector used at the beginning of a CBC encryption or decryption chain is a constant and given by the 4C Entity, LLC as Confidential Information, as defined in the CPXM License Agreement.

2.2 AES Hash Function

CPXM uses a hashing procedure based on the AES encryption algorithm. This procedure is called the AES Hash Function, and is represented by the function

AES_H(d)

where d is input data of arbitrary length, and AES_H returns the 128-bit result.

Prior to hashing, the data to be hashed (d) is padded using the standard SHA-1 method as described in the following sentences. The message or data file is 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 an integer 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 $128 \times n$. The 64-bit integer is the length of the original message in bits. The length of padding is at least 65 bits ("1" || the 64-bit integer) and at most 192 bits ("1" || 127 "0"s || the 64-bit integer.) 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 d' is divided into n 128-bit blocks, represented as d_1' ,d_2' ,…d_n', which are used in the hashing procedure as shown in Figure 2-1.

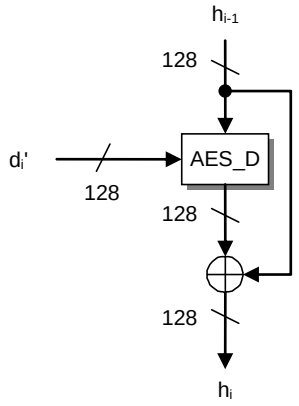

Figure 2-1 – AES Hash Function

A 128-bit fixed initial value h_0 is provided by the 4C Entity, LLC for media types and applications where the AES Hash Function is used.

The following are calculated iteratively for i from 1 to n:

$h_i = AES\_D(d_i', h_{i-1}) \oplus h_{i-1}'$.

The value h_n is the final result of the hash, i.e. $AES\_H(d) = h_n$.

2.3 AES One-way Function

CPXM uses a cryptographic one-way function based on the AES encryption algorithm. This function is called the AES One-way Function, and is represented by

$AES\_G(d_1, d_2)$

where d_1 is a 128-bit input value, d_2 is a 128-bit input value, and AES_G returns the 128-bit result.

Figure 2-2 depicts the one-way function.

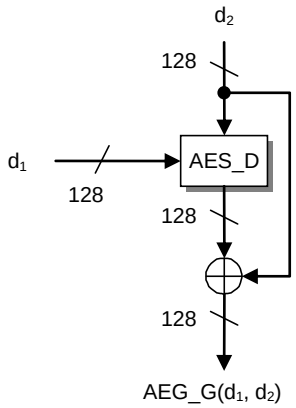

Figure 2-2 – AES One-way Function

The one-way function result is calculated as

$AES\_G(d_1, d_2) = AES\_D(d_1, d_2) \oplus d_2$.

2.4 Random Number Generators

This section describes a random number generator and a pseudo-random number generator, both of which are based on the AES One-way Function. Unless explicitly noted otherwise, one or plural of the following random/pseudo random number generators shall be used: (1) Pseudorandom number generator based on a design described in either Section 2.4.1 or Section 2.4.2 and described below (2) Pseudorandom number generators defined in NIST Special Publication 800-90 (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).

2.4.1 AES Random Number Generator

Figure 2-3 shows the AES Random Number Generator, which is a random number generator based on the AES One-way Function that uses a non-correlated input in every cycle.

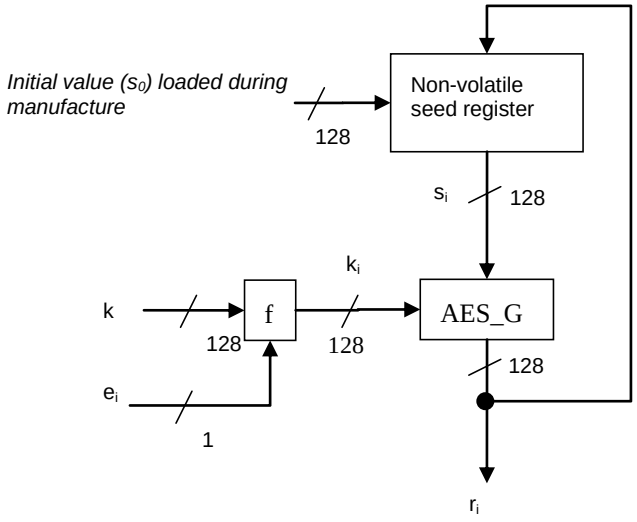

Figure 2-3 – AES Random Number Generator

Manufacturers need to generate a unique value, s_0, for each device or medium. During manufacture, the s_0 is loaded into the non-volatile seed register. Thereafter, 128-bit random numbers r_i (i=0,1,…) are generated as

$r_i = AES\_G(k_i, s_i)$,

where $k_i = f(k, e_i)$

and $s_{i+1} = r_i$.

The function $f(k, e_i)$ returns the value k after its least significant bit is exclusive-ORed with e_i.

The constant k is a 128-bit value generated individually for each device by a physically random process. This may be taken from the random number provided for each device by the 4C Entity, LLC. The 1-bit value e_i is taken from a source of run-time entropy, such as the least significant bit of a free-running counter having a frequency significantly higher than the random number sample rate.

Unless explicitly noted otherwise, a device shall treat its k value as Highly Confidential, as defined in the CPXM License Agreement.

2.4.2 AES Pseudo-random Number Generator

Figure 2-4 shows the AES Pseudo-random Number Generator, which is a pseudo-random number generator based on the AES One-way Function that generates an output sequence with a period of length 2^{128}.

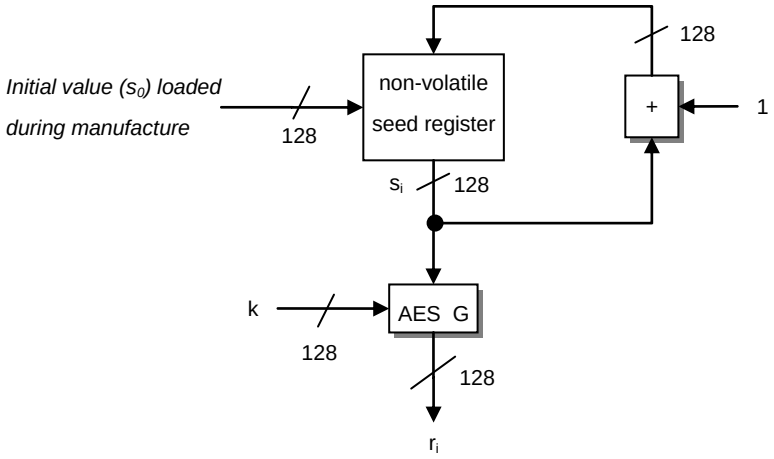

Figure 2-4 – AES Pseudo-random Number Generator

Manufacturers need to generate a unique value, s_0, for each device or medium. During manufacture, the s_0 is loaded into the non-volatile seed register. Thereafter, 128-bit random numbers r_i (i=0, 1, …) are generated as

$r_i = AES\_G(k, s_i)$

where $s_{i+1} = [s_i + 1]_{lsb\_128}$

The constant k is a 128-bit value generated individually for each device by a physically random process. This may be taken from the random number provided for each device by the 4C Entity, LLC. Unless explicitly noted otherwise, a device shall treat its k value as Highly Confidential, as defined in the CPXM License Agreement.

Chapter 3
CPXM Common Cryptographic Key Management

3. CPXM Common Cryptographic Key Management

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 and media revocation. The procedure is described here in isolation; its use as part of the overall protection system is described elsewhere in this specification.

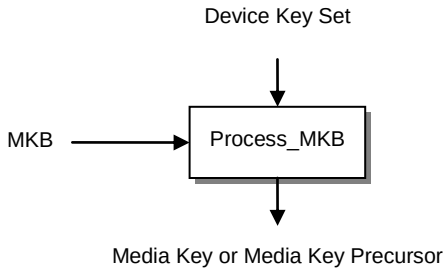

Figure 3-1 – Common CPXM Cryptographic Key Management Procedure

The 4C Entity, LLC licenses a Device Key Set which includes Device Keys (K_{d_0},K_{d_1},…,K_{d_n-1}) used to decrypt one or more elements of a Media Key Block (MKB), in order to extract the secret Media Key (K_m^0) or the Media Key Precursor(K_m^{-1}). The result which is calculated by Process MKB depends on the type of Device Key used. 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^0)	128 bits
Media Key Precursor (K_m^{-1})	128 bits

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

3.1 Device Keys

Each compliant device or medium is given a Device Key Set when manufactured. A Device Key Set consists of a Device Node, a set of Device Keys (K_d) and the UV Descriptor (UV) associated with each Device Key, {Device Node, {K_{d_0}, UV_0}, {K_{d_1}, UV_1}, ..., {K_{d_n-1}, UV_{n-1}}}. Table 3-2 shows these elements and the size.

There are two types of Device Keys which are called Host Device Key and Media Device Key. A Host Device Key is given to a host device and a Media Device Key is given to a medium. 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 4C Entity, LLC., and are used by the device to process the MKB. The Host Device Key set produces the Media Key Precursor (K_m^{-1}), which can then be used to calculate the Media Key (K_m^0), whereas the Media Device Key set produces the Media Key (K_m^0). The set of Host Device Keys shall either be unique per device, or shared by multiple devices, however the set of Media Device Keys shall be unique per medium. The CPXM License Agreement describes details and requirements associated with these alternatives. A device and medium shall treat its Device Keys as Highly Confidential, as defined in the CPXM License Agreement.

Table 3-2 Elements of Device Key Set

Elements	Size
Device Keys (K_{d_0}, K_{d_1},…, K_{d_n-1})	128 bits each
UV Descriptor (UV_0, UV_1, ..., UV_{n-1})	48 bits each
Device Node	40 bits

3.2 Media Key Block (MKB)

The Media Key Block (MKB) enables system renewability. The MKB is generated by 4C Entity, LLC. All compliant Host Devices using their set of secret Host Device Keys calculate the same K_m^{-1} and K_m^0 when using the same MKB. All compliant Media Device using their set of secret Media Device Keys calculate the same K_m^0 as Host Devices will from the same MKB. 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 Host Device or Media Device with the compromised set of Device Keys to be unable to calculate the correct K_m^{-1}/K_m^0 or K_m^0 respectively. 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. Figure 3-2 shows the calculation of these secret keys.

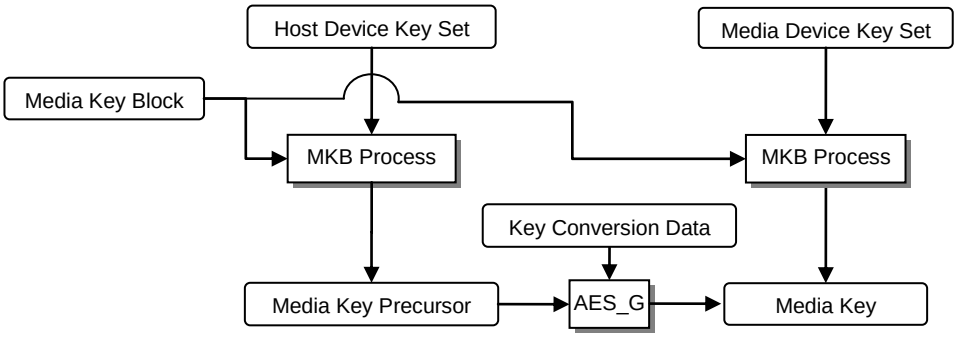

Figure 3-2 – Calculation of Secret Keys

3.2.1 Subset-Difference Tree Related Definitions

This section describes defined terms mentioned in the tree calculations.

3.2.2 Calculation of Subsidiary Device Keys and Processing Keys

For the purpose of processing an MKB to calculate K_m^0 or K_m^{-1}, Device Keys are used to calculate subsidiary Device Keys and Processing Keys using the AES_G3 function. The AES_G3 is the AES Cipher based one way calculation and outputs 384 bits. From a Device Key or subsidiary Device Key, AES_G3 produces the subsidiary Device Key for the left child of the current node, the Processing Key and the subsidiary Device Key for the right child of the current node. By using the AES_G3 function, the device calculates all subsidiary Device Keys that it needs from the few Device Keys that it stores at manufacturing time. Figure 3-3 depicts overview of AES_G3 function

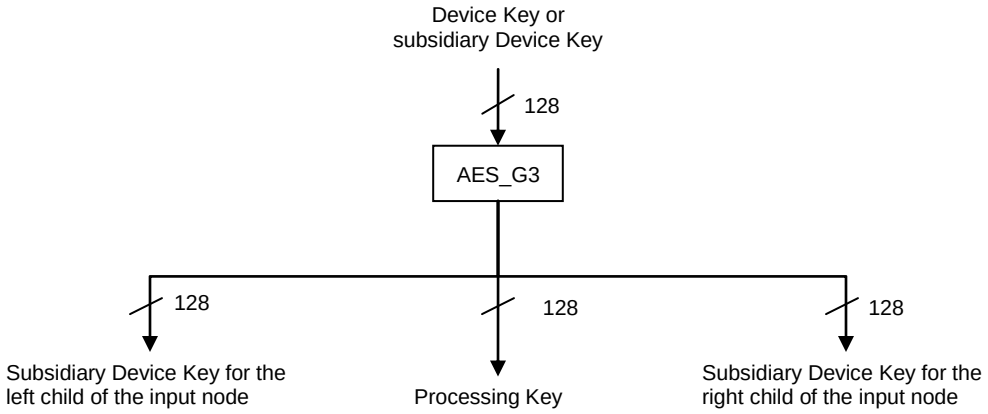

Figure 3-3 – Overview of AES_G3 function

3.2.3 Storing Device Keys

Each Host Device or Media Device is given its Device Keys and a 39-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 sub-tree 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 40-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 long v_mask = 0xFFFFFFFFFF;
while ((uv & ~v_mask) == 0) v_mask <<= 1;
```

By the same token, CPXM distinguishes between Device Numbers and Device Node Numbers, and uses the latter in the key order format. A Device Node Number is the *uv* number of the Device Number. 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 $FFFFFFFFFE_{16}$. In other words, the Device Node Number is the Device Number shifted left by 1, with the low-order bit set. For example, when a 39-bit Device Number is $0010010001101000101011001111000100110101_2$ ($123456789A_{16}$), the Device Node Number is $0010010001101000101011001111000100110101_2$ ($2468ACF135_{16}$).

3.2.4 Calculation of Media Key and Media Key Precursor

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 corresponds 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 either has the Device Key, or is able to derive the subsidiary Device Key, associated with that subset-difference. It finds the appropriate stored Device Key as follows: assuming the Explicit Subset-Difference Record value is *uv*, m_u, and m_v, and the stored Device Key has *uv*', m'_u, and m'_v, the appropriate Device Key is the one that meets the following condition:

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

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

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

Media Devices calculate the 128 bit Media Key (K_m^0) using that Processing Key and the appropriate 16 bytes of encrypted key data C found in the appropriate Media Key Data Record in the MKB and the AES hash of the 16 bytes of the *Type and Version* Record of the MKB described in Section 3.2.5.2 and the 40-bit uv number as follows:

K_m^0 = AES_F0(Processing Key, C, Type and Version Record, uv number)

where AES_F0 is the AES Cipher based calculation and exact notation of this function is provided by 4C Entity, LLC.

Host Devices calculate the Media Key K_m^0 by first calculating the Media Key Precursor K_m^{-1}.using the same calculation above, which yields the K_m^{-1} instead of K_m^0 as follows:

K_m^{-1} = AES_F1(Processing Key, C, Type and Version Record, uv number)

where AES_F1 is the AES Cipher based calculation and exact notation of this function is provided by 4C Entity, LLC.

Host Devices then calculate the Media Key K_m^0 and a constant Key Conversion Data (KCD) using the AES_G one-way function as follows:

K_m^0 = AES_G(K_m^{-1}, KCD)

where KCD is a constant value provided by 4C Entity, LLC.

Note that using the calculations above, Host and Media Devices both derive the same Media Key K_m^0 from the same MKB using their respective Device Key Set.

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, Host Devices calculate K_m^{-1} and K_m^{0}, while Media Devices calculate K_m^{0} by processing Records of the MKB one-by-one, in order, from first to last. The device shall not make any assumptions about the length of Records, and shall instead use the Record Length field value to go from one Record to the next. If a device encounters a Record with a Record Type field value it does not recognize, that is not an error; it shall ignore that Record and skip to the next. Likewise, if a Record Length indicates a record is longer than the device expects, that is also not an error; it shall ignore the additional record data.

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

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, 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 3-3 – *Type and Version* Record Format

Bit / Byte	7	6	5	4	3	2	1	0
0	Record Type: 10_{16}							
1	Record Length: 000010_{16}							
2								
3								
4	MKBType: 00301003_{16}							
5								
6								
7								
8	Version Number							
9								
10								
11								
12	Reserved							
13	Reserved							
14	Application ID							
15								

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 4C Entity, LLC 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.

A 2-byte Application ID indicates which application uses the MKB.

For CPXM applications, the MKBType field is set to 00301003_{16}.

3.2.5.2 Verify Media Key Record

Table 3-4 – *Verify Media Key* Record Format

Bit / Byte	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 corresponding records which are 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

$$D_v = AES\_E\ (K_m^0,\ 0123456789ABCDEF_{16} \parallel XXXXXXXXXXXXXXXX_{16})$$

where $XXXXXXXXXXXXXXXX_{16}$ is an arbitrary 8-byte value, and K_m^0 is the correct 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. The device shall verify the correctness of the MKB by observing the following condition:

$$[AES\_D(K_m^0,\ D_v)]_{msb\_64} == 0123456789ABCDEF_{16}$$

where K_m^0 is the Media Key.

3.2.5.3 Explicit Subset-Difference Record

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

Bit / Byte	7	6	5	4	3	2	1	0
0	Record Type: 04_{16}							
1	Record Length							
2								
3								
4	10_2		U Mask (0)					
5	UV Number (0)							
…								
9								
10	10_2		U Mask (1)					
11	UV Number (1)							
...								
15								
16	. . .							
Length-1								

In this record, each subset-difference is encoded with 6 bytes called the UV Descriptor. The first 2-bits shall be set to 10_2. The mask for *u* is given by the least significant 6 bits of the first byte. The value of these 6 bits is the number of contiguous low-order bits in the mask which are set to zero. For example, the value 01_{16} denotes a mask of $FFFFFFFFFE_{16}$; value $0A_{16}$ denotes a mask of $FFFFFFFC00_{16}$.

The last 5 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 an entry whose high-order two bits are not set to 10_2, without finding an applicable subset, it may conclude it is revoked. In other words, if the first byte is not of the form $10xxxxxxx_2$, this marks the end of the list. The device's action in this case is manufacturer-specific. However, it is common for proactively-renewed host 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 is always a multiple of 4 bytes. Thus, there may be unused bytes at the end of the record.

3.2.5.4 Subset-Difference Index Record

Table 3-6 – *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)							
…								
8								
9	Data Offset (0)							
10								
11								
12	Data Offset (1)							
13								
14								
15	Data Offset (2) – Data Offset (N)							
…								
Length-1								

This is a speed-up record which may be ignored by devices not wishing to take advantage of it. It is a lookup table which allows devices to quickly find their subset-difference in the *Explicit Subset-Difference* Record, without processing the entire record. This *Subset-Difference Index* Record is always present, and always precedes the *Explicit Subset-Difference* Record in the MKB, although it does not necessarily immediately precede it. Furthermore, the *Subset-Difference Index* Record is guaranteed to be within the first one megabyte of the 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 shall 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 Data Offsets. These Data Offsets refer to the byte offset within the following *Explicit Subset-Difference* Record, with 0 being the start of the record. Devices whose device number is between 0 and Span-1 shall begin processing the *Explicit Subset-Difference* Record at Data Offset (0). Devices whose number is between Span and 2 × (Span-1), shall begin processing the *Explicit Subset-Difference* Record at Data Offset (1), and so on. Equivalently, if a device's number is *d*, the corresponding Data Offset is Data Offset floor(*d*/Span) and it finds its Data Offset within the *Explicit Subset-Difference* Record at offset 3 × floor(*d* / Span) + 9 in this record. For example, when *d* is $1270D89F2B_{16}$ and Span is 0100000000_{16}, the Data Offset (18) starting at offset 45_{10} is used.

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

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

3.2.5.5 Media Key Data Record

Table 3-7 – *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 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 22 bytes: 6 bytes for its "*uv*" data and 16 bytes for the Media Key Data. On average there are 1.28 encryptions per revocation.

3.2.5.6 End of Media Key Block Record

Table 3-8 – *End of Media Key Block* Record Format

Bit / Byte	7	6	5	4	3	2	1	0
0	Record Type: 02_{16}							
1	Record Length: 000014_{16}							
2								
3								
4	Check Data for MKB							
...								
19								

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^0 value it has calculated up to that point as the final K_m^0 for that MKB (pending possible checks for correctness of the key, as described previously).

This record includes 16 bytes of the Check Data for MKB field to verify the integrity of the MKB. The Check Data for MKB field is calculated by following formula.

Check Data for MKB = AES_H(MKB data up to but not including the End of MKB record)

Note that MKB data shall include all records including those with Record Type fields that the device might not recognize.

The length of this record is always 20 bytes.

Figure 3-4 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.

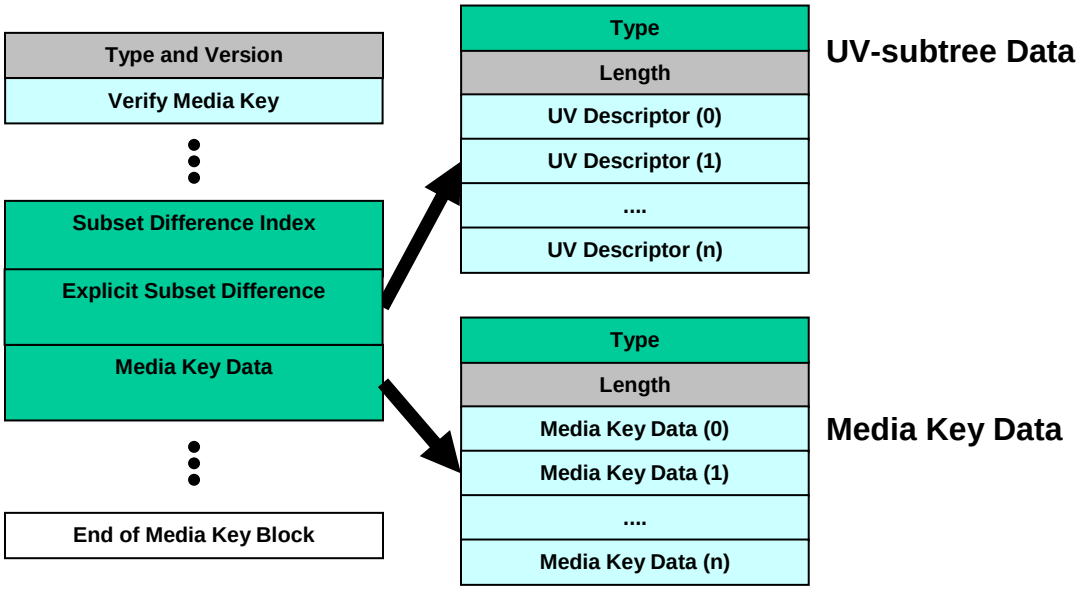

Figure 3-4 – Example of Media Key Block Showing a Valid Order of Records

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 Media book portion of this specification.