Content Protection for Recordable Media Specification

Introduction and Common Cryptographic Elements

Intel Corporation

International Business Machines Corporation

Panasonic Corporation

Toshiba Corporation

Revision 1.1

December 15, 2010

This page is intentionally left blank.

Preface

Notice

THIS DOCUMENT IS PROVIDED "AS IS" WITH NO WARRANTIES WHATSOEVER, INCLUDING ANY WARRANTY OF MERCHANTABILITY, NONINFRINGEMENT, FITNESS FOR ANY PARTICULAR PURPOSE, OR ANY WARRANTY OTHERWISE ARISING OUT OF ANY PROPOSAL, SPECIFICATION OR SAMPLE. IBM, Intel, Panasonic, and Toshiba disclaim all liability, including liability for infringement of any proprietary rights, relating to use of information in this specification. No license, express or implied, by estoppel or otherwise, to any intellectual property rights are granted herein.

Intellectual Property

Implementation of this specification requires license from the 4C Entity, LLC. Note that use of the AES-based technology also requires execution of an addendum to the 4C license agreement.

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 Recordable Media Specification* (CPRM) defines a renewable method for protecting content recorded on a number of physical media types. The specification is organized into several "books." This document, the *Introduction and Common Cryptographic Elements* book, provides a brief overview of CPRM, and defines cryptographic procedures that are common among its different uses. Other books provide additional details specific to using CPRM protection for different applications and media types. Other books of the CPRM Specification available at or around the time of this publication are:

- DVD Book
- Portable ATA Storage Book
- SD Memory Card Book.

Books covering other media types are expected to be available in the future (see Section 1.5 below). CPRM 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.

1.2 Overview

The CPRM technology is designed to meet the following criteria:

- It meets the content owners' requirements for robustness and system renewability.
- It is applicable for audio, video and other commercial 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 DVD 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 Content Key derived from a one-way function of a secret Title Key and the copy control information (CCI) associated with the content. 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. Again, actual details of key management can vary among different applications, as described in the other books of this specification.

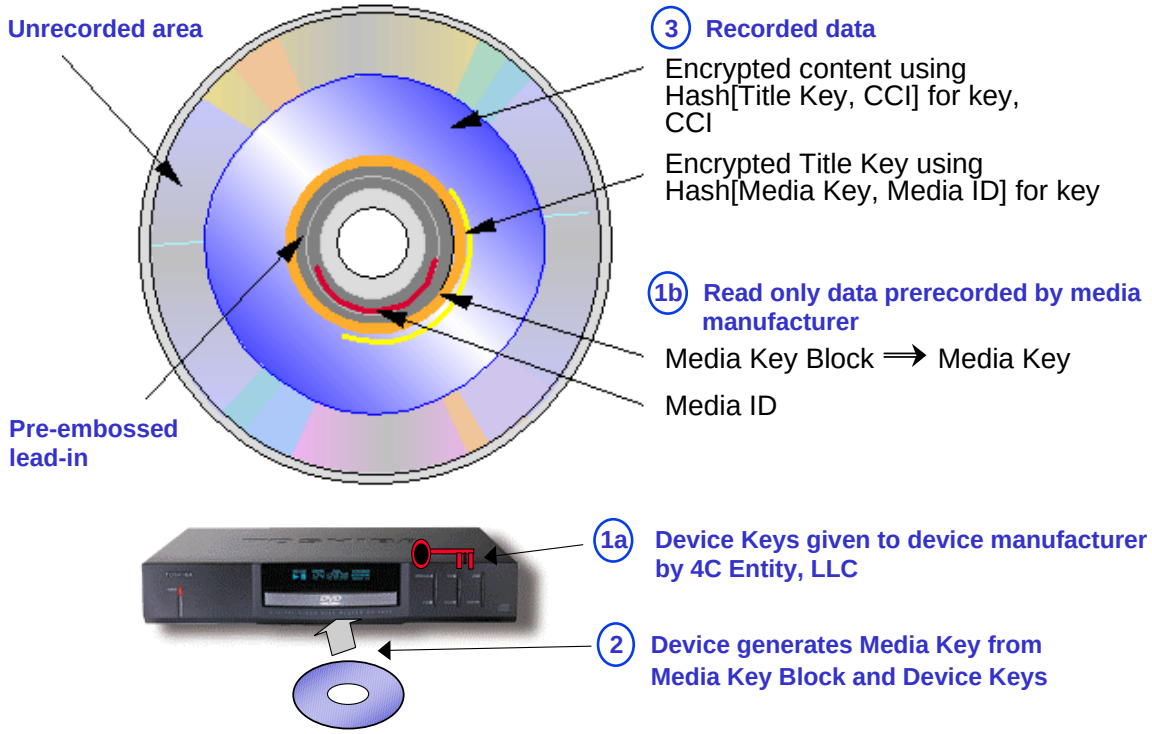

Figure 1-1 – CPRM Illustrative Example

1.3 Organization of this Document

This document is organized as follows:

- Chapter 1 provides an introduction and overview of CPRM.
- Chapter 2 describes common CPRM cryptographic functions based on the C2 cipher algorithm.
- Chapter 3 describes a common CPRM cryptographic key management procedure, using a Media Key Block and Media Identifier.
- Chapter 4 describes common CPRM enhanced cryptographic functions based on the AES cipher algorithm.

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, *CPRM License Agreement*

4C Entity, LLC, *C2 Block Cipher Specification, Revision 1.0*

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

National Institute of Standards and Technology (NIST), *Security Requirements for Cryptographic Modules, FIPS Publication 140-1, April 14, 1982*

Secure Digital Music Initiative (SDMI), *SDMI Portable Device Specification Version 1.0*

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

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 Revision 1a, April 2010*

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

National Institute of Standards and Technology (NIST), *Recommendation for Block Cipher Modes of Operation: The CMAC Mode for Authentication, NIST Special Publication 800-38B, May* 2005

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*

1.5 Future Directions

With its robust cryptography, key management, and renewability mechanisms, it is expected that CPRM 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 mod y	The modulo of x by y.

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:

AES	Advanced Encryption Standard
ATA	AT Attachment
C-CBC	Converted Cipher Block Chaining
C2	Cryptomeria Cipher
CBC	Cipher Block Chaining
CCI	Copy Control Information
CE	Consumer Electronics
CMAC	Cipher-based Message Authentication Code
CPRM	Content Protection for Recordable Media
CTR	Counter
DVD	Digital Versatile Disc
ECB	Electronic Codebook
FIPS	Federal Information Processing Standards
ID	Identifier

LLC	Limited Liability Company
lsb	Least Significant Bit
MAC	Message Authentication Code
MKB	Media Key Block
msb	Most Significant Bit
PC	Personal Computer
SDMI	Secure Digital Music Initiative
SHA	Secure Hashing Algorithm
XOR	Exclusive-OR

Chapter 2
CPRM Common Cryptographic Functions

2. Introduction

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

2.1 C2 Block Cipher Algorithm

Common cryptographic functions used for CPRM are based on the C2 block cipher. A description of the C2 block cipher algorithm is provided in a separate specification, referred to in Section 1.4. That specification describes two basic operational modes of the C2 cipher: Electronic Codebook (ECB) mode and Converted Cipher Block Chaining (C-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 C2 Block Cipher in Electronic Codebook (ECB) Mode

In this document and in other books of this specification, encryption with the C2 cipher in Electronic Codebook (ECB) mode is represented by the function

C2_E(k, d)

where k is a 56-bit key, d is 64-bit data value to be encrypted, and C2_E returns the 64-bit result.

Decryption using the C2 cipher in ECB mode is represented by the function

C2_D(k, d)

where k is a 56-bit key, d is a 64-bit data value to be decrypted, and C2_D returns the 64-bit result.

2.1.2 C2 Block Cipher in Converted Cipher Block Chaining (C-CBC) Mode

The C2 cipher is used in Converted Cipher Block Chaining (C-CBC) mode for encryption and decryption of content protected by CPRM. In this document and in other books of this specification, encryption with the C2 cipher in C-CBC mode is represented by the function

C2_ECBC(k, d)

where k is a 56-bit key, d is a frame of data to be encrypted, and C2_ECBC returns the encrypted frame.

Decryption using the C2 cipher in C-CBC mode is represented by the function

C2_DCBC(k, d)

where k is a 56-bit key, d is a frame of data to be decrypted, and C2_DCBC returns the decrypted frame.

The size of the frame of data to be encrypted or decrypted (i.e. how often a new C-CBC cipher chain is started) depends on the particular application format, and is defined for each in the corresponding books of this specification.

2.2 C2 Hash Function

CPRM uses a hashing procedure based on the C2 encryption algorithm. This procedure is called the C2 Hash Function, and is represented by the function

C2_H(d)

where d is input data of arbitrary length, and C2_H returns the 64-bit result.

For the purpose of calculating the hash value for input data d, d is padded as follows. Padded data d' is formed by first always appending a single "1" bit to d, and then appending from zero to sixty-three "0" bits as needed to make the total length of d' a multiple of 64 bits. Table 2-1 shows examples with input data d of various lengths.

Table 2-1 – Examples of Padding Input Data

Length of input data d (in bits)	Formation of padded data d'
126	$d' = d \,\|\| \, 10_2$
127	$d' = d \,\|\| \, 1_2$
128	$d' = d \,\|\| \, 8000000000000000_{16}$

The padded data d' is divided into n 64-bit blocks, represented as $d_1', d_2', \ldots d_n'$, which are used in the hashing procedure as shown in Figure 2-1.

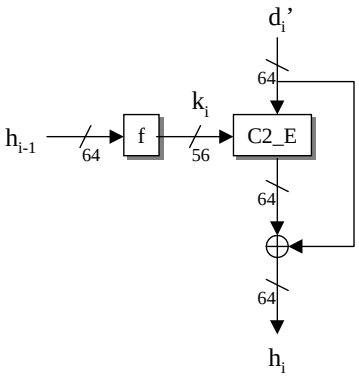

Figure 2-1 – C2 Hash Function

A conversion function f is defined as

$f(x) = [x]_{lsb\_56}$

where x is a 64-bit input data value.

A 64-bit fixed initial value h_0 is provided by the 4C Entity, LLC to licensees of CPRM for media types and applications where the C2 Hash Function is used.

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

$k_i = f(h_{i-1})$

and

$h_i = C2\_E(k_i, d_i') \oplus d_i'$.

The value h_n is the final result of the hash, i.e. $C2\_H(d) = h_n$.

2.3 C2 One-way Function

CPRM uses a cryptographic one-way function based on the C2 encryption algorithm. This function is called the C2 One-way Function, and is represented by

$C2\_G(d_1, d_2)$

where d_1 is a 56-bit input data value, d_2 is a 64-bit input data value, and C2_G returns the 64-bit result.

Figure 2-2 depicts the one-way function.

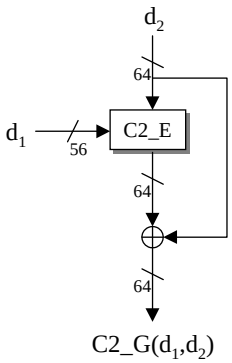

Figure 2-2 – C2 One-way Function

The one-way function result is calculated as

$C2\_G(d_1, d_2) = C2\_E(d_1, d_2) \oplus d_2$.

2.4 Random Number Generators

This section describes a random number generator and a pseudorandom number generator for generating random value up to 64-bit, both of which are based on the C2 One-way Function. Unless explicitly noted otherwise, one of these designs, or a design of equal or higher quality that passes the tests described in FIPS-140 section 4.11.1 or designs described in Section 4.5 shall be used for CPRM. Note that for generating random value more than 64-bit, random number generators described in Section 4.5 shall be used.

2.4.1 C2 Random Number Generator

Figure 2-3 shows the C2 Random Number Generator, which is a random number generator based on the C2 One-way Function that uses a non-correlated input in every cycle.

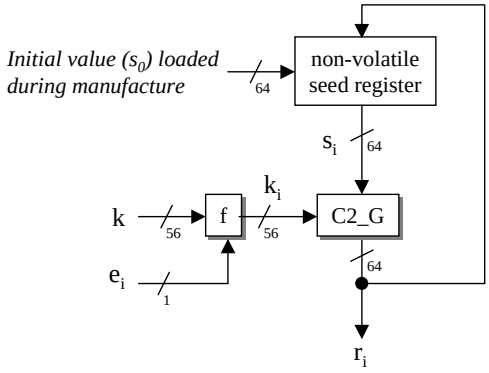

Figure 2-3 – C2 Random Number Generator

During manufacture, an arbitrary 64-bit initial value s_0 is loaded into the non-volatile seed register. Thereafter, 64-bit random numbers r_i (i=0,1,…) are generated as

$r_i = C2\_G(k_i, s_i)$,

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 fixed input k is a 56-bit value generated individually for each device by a physically random process, and 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 CPRM License Agreement.

2.4.2 C2 Pseudorandom Number Generator

Figure 2-4 shows the C2 Pseudorandom Number Generator, which is a pseudorandom number generator based on the C2 One-way Function that generates an output sequence with a period of length 2^{64}.

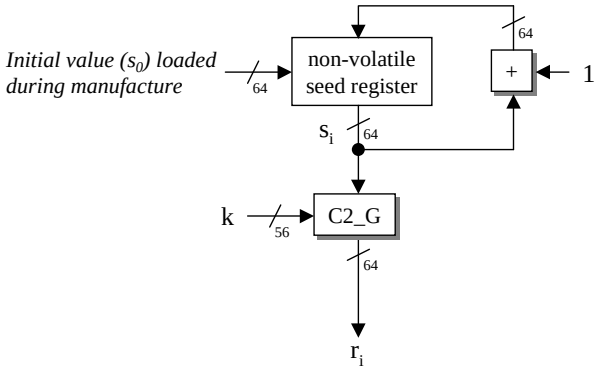

Figure 2-4 – C2 Pseudorandom Number Generator

During manufacture, an arbitrary 64-bit initial value s_0 is loaded into the non-volatile seed register. Thereafter, 64-bit random numbers r_i (i=0,1,…) are generated as

$r_i = C2\_G(k, s_i)$

where $s_{i+1} = [s_i + 1]_{lsb\_64}$.

The fixed input k is a 56-bit value generated individually for each device by a physically random process, and 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 CPRM License Agreement.

Chapter 3
CPRM Common Cryptographic Key Management

3. Introduction

This chapter describes a CPRM common cryptographic key management procedure, depicted in Figure 3-1, which uses a Media Key Block to provide renewability, and a Media Identifier for individual media identification. The procedure is described here in isolation; its use as part of CPRM for different media types and applications is described in the other books of this specification.

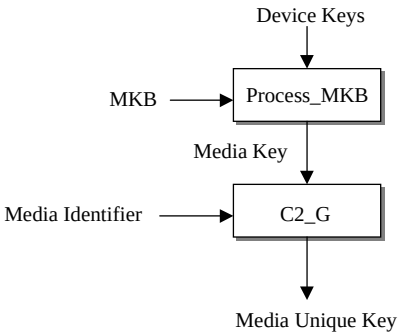

Figure 3-1 – Common CPRM Cryptographic Key Management Procedure

- 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).
- K_m and a Media Identifier (ID_{media}) are combined, using the C2 One-way Function, to produce a Media Unique Key (K_{mu}).

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})	56 bits each
Media Key Block (MKB)	Variable, multiple of 4 bytes
Media Key (K_m)	56 bits
Media Identifier (ID_{media})	64 bits
Media Unique Key (K_{mu})	56 bits

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

3.1 Calculation of the Media Key (K_m)

3.1.1 Device Keys

Each CPRM compliant device is given a set of secret Device Keys when manufactured. These keys are provided by the 4C Entity, LLC, and are for use in processing the MKB to calculate K_m. Key sets may either be unique per device, or used commonly by multiple devices. The CPRM License Agreement describes the details and requirements associated with these two alternatives.

Each device receives n Device Keys, which are referred to as K_{d_i} (i=0,1,…,n-1). For each Device Key there is an associated Column and Row value, referred to as C_{d_i} and R_{d_i} (i=0,1,…,n-1) respectively. Column and Row values start at 0. For a given device, no two Device Keys will have the same associated Column value (in other words, a device will have at most one Device Key per Column). It is possible for a device to have some Device Keys with the same associated Row values. The number of Device Keys that are given to each device and the range of Column and Rows values that are possible are defined separately for each device type in the corresponding book of this specification.

A device shall treat its Device Keys as Highly Confidential, and their associated Row values as confidential, as defined in the CPRM License Agreement.

3.1.2 Media Key Block (MKB)

CPRM's cryptographic key management scheme uses the Media Key Block (MKB) to enable system renewability. The MKB is generated by the 4C Entity, LLC, 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 calculate a different K_m than is computed by the remaining compliant devices. In this way, the compromised Device Keys are "revoked" by the new MKB.

An MKB 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. The Record Type and Record Length fields are never encrypted. Subsequent fields in a Record may be encrypted (by the C2 cipher in ECB mode), depending on the Record Type.

Using its Device Keys, a device calculates K_m by processing Records of the MKB one-by-one, in order, from first to last. Except where explicitly noted otherwise, a device must process every Record of the MKB. 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, it ignores that Record and skips to the next. For some Records, processing will result in the calculation of a K_m value. Processing of subsequent Records may update the K_m value that was calculated previously. After processing of the MKB is completed, the device uses the most recently calculated K_m value as the final value for K_m (i.e. the output of Process_MKB in Figure 3-1).

If a device correctly processes an MKB using Device Keys that are revoked by that MKB, the resulting final K_m will have the special value 00000000000000_{16}. This special value will never be an MKB's correct final K_m value, and can therefore always be taken as an indication that the device's keys are revoked. If a device calculates this special K_m value, it shall stop the authentication/playback/recording session in progress, and shall not use that K_m value in any subsequent calculations. Other 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.

The following subsections describe the currently defined Record types, and how a device processes each.

3.1.2.1 Verify Media Key Record

Table 3-2 shows the format of a *Verify Media Key* Record.

Table 3-2 – *Verify Media Key* Record Format

Bit / Byte	7	6	5	4	3	2	1	0
0	Record Type: 81_{16}							
1	Record Length: $00000C_{16}$							
2								
3								
4	Verification Data (D_v): C2_E(K_m, $DEADBEEF_{16}$ || $XXXXXXXX_{16}$)							
:								
11								

A properly formatted MKB shall have exactly one *Verify Media Key* Record as its first Record. Bytes 4 through 11 of the Record contain the value

D_v = C2_E(K_m, $DEADBEEF_{16}$ || $XXXXXXXX_{16}$)

where K_m is the correct final Media Key value, and $XXXXXXXX_{16}$ is an arbitrary 4-byte value.

The presence of the *Verify Media Key* Record in an MKB is mandatory, but the use of the Record by a device is not mandatory.

As an optimization, a device may attempt to decrypt D_v using its current K_m value during the processing of subsequent Records, checking each time for the condition

$[C2\_D(K_m, D_v)]_{msb\_32} == DEADBEEF_{16}$

where K_m is the current Media Key value.

If this condition is true, the device has already calculated the correct final K_m value, and may therefore stop processing the MKB.

Also (or alternatively), a device could check the same condition after processing the entire MKB, in order to determine if it has calculated the correct final K_m. Failure to calculate the correct K_m after processing the entire MKB could be the result of data or calculation errors, or of the device's keys having been revoked (or both). Note that these two cases can generally be distinguished, since a device with revoked keys that correctly processes the MKB will calculate an incorrect final K_m with the special value 00000000000000_{16}.

3.1.2.2 Calculate Media Key Record

Table 3-3 shows the format of a *Calculate Media Key* Record.

Table 3-3 – *Calculate Media Key* Record Format

	Bit / Byte	7	6	5	4	3	2	1	0
	0	Record Type: 01_{16}							
	1	Record Length							
	2								
	3								
	4	Reserved							
	5								
	6	Revision							
	7								
	8	Column							
	9	Generation: 000001_{16}							
	10								
	11								
Encrypted Key Data	12	Encrypted Key Data for Row 0 (D_{ke_0})							
	:								
	19								
	20	Encrypted Key Data for Row 1 (D_{ke_1})							
	:								
	27								
	28	.							
	:	.							
	Length-1	.							

A properly formatted MKB shall have exactly one *Calculate Media Key* Record. Devices must ignore any *Calculate Media Key* Records encountered after the first one in an MKB. The use of the Reserved field is currently undefined, and it is ignored. The Revision field contains a 16-bit value, which is set to 0000_{16} in initial MKBs, and is incremented each time the revocation information released in MKBs is updated. The use of the Revision field is defined in Section 3.1.3, and is not mandatory. The Generation field shall contain 000001_{16} for the first generation. The Column field indicates the associated Column value for the Device Key to be used with this Record, as described below. Bytes 12 and higher contain Encrypted Key Data (possibly followed by some padding bytes at the end of the Record, not shown in Table 3-3). The first eight bytes of the Encrypted Key Data correspond to Device Key Row 0, the next eight bytes correspond to Device Key Row 1, and so forth.

Before processing the Record, the device checks that both of the following conditions are true:

Generation == 000001_{16}

and

the device has a Device Key with associated Column value C_{d_i} == Column, for some i.

If either of these conditions is false, the device ignores the rest of the Record.

Otherwise, using the value i from the condition above, and $r = R_{d\_i}$, $c = C_{d\_i}$, the device calculates:

$$K_m = [C2\_D(K_{d\_i}, D_{ke\_r})]_{lsb\_56} \oplus f(c,r)$$

where K_{d_i} is the i^{th} Device Key's value and D_{ke_r} is the 64-bit value starting at byte offset $r \times 8$ within the Record's Encrypted Key Data. Unless explicitly noted otherwise in another book of this specification, f(c,r) represents the 56-bit value

$$f(c,r) = 0000_{16} \,||\, c \,||\, 0000_{16} \,||\, r$$

where c and r are left-padded to lengths of 8 and 16 bits respectively, by prepending zero-valued bits to each as needed. The resulting K_m becomes the current Media Key value.

It is not necessary for a first generation device to verify that Record Length is sufficient to index into the Encrypted Key Data. First generation devices are assured that the Encrypted Key Data contains a value corresponding to their Device Key's associated Row value.

3.1.2.3 Conditionally Calculate Media Key Record

Table 3-4 shows the format of a *Conditionally Calculate Media Key* Record.

Table 3-4 – *Conditionally Calculate Media Key* Record Format

	Bit / Byte	7	6	5	4	3	2	1	0
	0	Record Type: 82_{16}							
	1	Record Length							
	2								
	3								
Encrypted Conditional Data (D_{ce})	4	$DEADBEEF_{16}$ (encrypted)							
	:								
	7								
	8	Column (encrypted)							
	9	Generation: 000001_{16} (encrypted)							
	10								
	11								
Doubly Encrypted Key Data	12	Doubly Encrypted Key Data for Row 0 (D_{kde_0})							
	:								
	19								
	20	Doubly Encrypted Key Data for Row 1 (D_{kde_1})							
	:								
	27								
	28	.							
	:	.							
	Length-1	.							

A properly formatted MKB may have zero or more *Conditionally Calculate Media Key* Records. Bytes 4 through 11 of the Record contain Encrypted Conditional Data (D_{ce}). If decrypted successfully, as described below, bytes 4 through 7 contain the value $DEADBEEF_{16}$, byte 8 contains the associated Column value for the Device Key to be used with this Record, and bytes 9 through 11 contain a Generation value of 000001_{16} for the first generation. Bytes 12 and higher contain Doubly Encrypted Key Data (possibly followed by some padding bytes at the end of the Record, not shown in Table 3-4). The first eight bytes of the Doubly Encrypted Key Data correspond to Device Key Row 0, the next eight bytes correspond to Device Key Row 1, and so forth.

Using its current K_m value, the device calculates Conditional Data (D_c) as:

$D_c = C2\_D(K_m, D_{ce})$.

Before continuing to process the Record, the device checks that all of the following conditions are true:

$[D_c]_{msb\_32} == DEADBEEF_{16}$

and

$[D_c]_{lsb\_24} == 000001_{16}$

and

the device has a Device Key with associated Column value $C_{d\_i} == [D_c]_{31:24}$ for some i.

If any of these conditions is false, the device ignores the rest of the Record.

Otherwise, using the value i from the condition above, and $r = R_{d\_i}$, $c = C_{d\_i}$, the device calculates:

$d = C2\_D(K_m, D_{kde\_r})$

where D_{kde_r} is the 64-bit value starting at byte offset $r \times 8$ within the Record's Doubly Encrypted Key Data,

and then uses the resulting value d to calculate:

$K_m = [C2\_D(K_{d\_i}, d)]_{lsb\_56} \oplus f(c,r)$

where K_{d_i} is the i^{th} Device Key's value. Unless explicitly noted otherwise in another book of this specification, f(c,r) represents the 56-bit value

$f(c,r) = 0000_{16} \,||\, c \,||\, 0000_{16} \,||\, r$

where c and r are left-padded to lengths of 8 and 16 bits respectively, by prepending zero-valued bits to each as needed. The resulting K_m becomes the current Media Key value.

3.1.2.4 End of Media Key Block Record

Table 3-5 shows the format of an *End of Media Key Block* Record.

Table 3-5 – *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: 000004_{16}							
2								
3								

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).

3.1.3 Media Key Block Extension

In most cases, an MKB is placed on CPRM compliant media by the manufacturer, in such a way that it cannot be altered or replaced. If a set of Device Keys is subsequently compromised, new media can be released containing an updated MKB that causes the newly compromised set of Device Keys to calculate an incorrect K_m, thereby "revoking" its ability to work with the new media. For some CPRM applications on some media types, Recording Devices may accelerate this revocation process by writing additional MKB Records (referred to collectively as an MKB Extension) onto a writable area of the media. This MKB Extension is then treated as a part of the original MKB for the purpose of calculating the Media Key (K_m), as depicted in Figure 3-2.

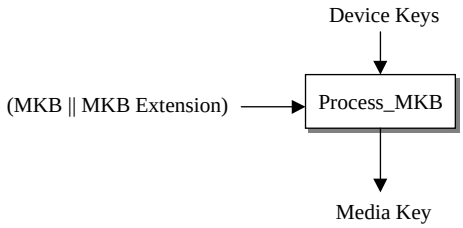

Figure 3-2 – Calculation of Media Key from MKB and MKB Extension

In this chapter, the MKB originally placed on the media by the manufacturer will be referred to as the Static MKB. The combination of a Static MKB and an MKB Extension will be referred to as an Extended MKB (i.e. Extended MKB = Static MKB || MKB Extension). In other books of this specification where applications that support MKB Extensions are described, the term MKB may be used to refer to either a Static MKB alone, or an Extended MKB.

MKB Extensions allow the use of system renewal information that comes from sources other than newly manufactured media, such as via electronic music distribution. The other books of this specification indicate which applications support MKB Extensions. The remainder of this section describes the additional rules and procedures for writing MKB Extensions and processing Extended MKBs, for those cases where this feature is supported.

3.1.3.1 Writing an MKB Extension

Even for CPRM applications where MKB Extensions are supported, the writing of an MKB Extension onto the media by a Recording Device is not mandatory. If an MKB Extension is not already present on the media, the Recording Device may write one. If an MKB Extension is already present, the Recording Device may replace it with another MKB Extension.

Before writing an MKB extension, the Recording Device obtains a recent MKB, formatted as described in Section 3.1.2, from an external source, such as via electronic music distribution. This MKB is referred to either as the New MKB (since it is typically more recent than the Static MKB placed on the media by the manufacturer) or as the MKB for MKB Extension, and is used in creating the MKB Extension as described below.

For media types where the integrity of the Static MKB is verified in a robust manner, the Recording Device may use a comparison of the Revision fields in the *Calculate Media Key* Records of both the Static and New MKB in deciding whether to write/replace an MKB Extension. If the Revision field value in the New MKB is not the higher of the two, then the New MKB is not truly more recent, and thus the Recording Device may choose to save time by not using the New MKB to form an MKB Extension for that media.

For media types where the integrity of the Static MKB is not verified robustly, the decision to write an MKB Extension must be made without regard to Revision fields, since otherwise the Revision field of the Static MKB might be forged maliciously in order to prevent MKB Extensions being written. Since MKB Extensions do not contain a *Calculate Media Key* Record (as described below), they cannot "enable" more Device Key sets than

were enabled by the Static MKB. Therefore, it is always safe to write an MKB Extension. The comparison of Revision fields, where used, is only a performance optimization.

A Recording Device uses the following steps when writing (or replacing) an MKB Extension:

1) Calculate the Media Key (K_m) of the Static MKB:

   The Recording Device calculates the K_m value for the media's Static MKB in the usual way.

2) Modify the New MKB:

   The Recording Device modifies the New MKB by changing its *Calculate Media Key* Record into a *Conditionally Calculate Media Key* Record, as follows:

   a) Change the Record Type field value from 01_{16} to 82_{16}.

   b) Change the value in bytes 4 through 7 (the Reserved and Revision fields) to $DEADBEEF_{16}$.

   c) Change the value in bytes 4 through 11 to an Encrypted Conditional Data (D_{ce}) value:

      Treating bytes 4 through 11 as a Conditional Data (D_c) value, use the K_m value from Step 1 to calculate an Encrypted Conditional Data (D_{ce}) value as

      $D_{ce} = C2\_E(K_m, D_c)$,

      and then replace D_c with D_{ce}.

   d) Change Encrypted Key Data to Doubly Encrypted Key Data:

      Starting at byte 12, replace every Encrypted Key Data (D_{ke_i}) value with a corresponding Doubly Encrypted Key Data (D_{kde_i}) value, calculated using K_m from Step 1 above as follows:

      $D_{kde\_0} = C2\_E(K_m, D_{ke\_0})$,

      $D_{kde\_1} = C2\_E(K_m, D_{ke\_1})$,

      …

      In this step, the Recording Device shall not make any assumptions about the number of Encrypted Key Data values in the Record, and shall instead use the Record Length field to determine how many Encrypted Key Data values are to be replaced. In other words, starting at byte 12, the Recording Device encrypts data in sequential 8-byte blocks until the end of the Record is reached. If the Record ends with a block of data less than 8 bytes long, that block is left unmodified.

3) Write modified New MKB as MKB Extension:

   The New MKB, as modified in Step 2 above, is written to the media as the MKB Extension (possibly replacing a previous MKB Extension). The specific location or file where the MKB Extension is stored is defined for each application and media type in the corresponding book of this specification. Note that MKB Extensions may be stored as normal read/write files.

Writing/replacing an MKB Extension changes the K_m value, and thus the K_{mu} value, for the application(s) using that MKB Extension. Therefore, if CPRM protected content is already present on the media when an MKB Extension is to be written, the Recording Device must decrypt any data belonging to the affected application(s) that was previously encrypted with the old K_{mu}, and re-encrypt it using the new K_{mu}, as part of the process outlined above. Typically, applications that support MKB Extensions use K_{mu} to encrypt title keys, and in turn use those title keys to encrypt the content itself. In such cases, only the encrypted title keys must be decrypted and re-encrypted as just described. Note that if multiple applications share a common MKB Extension on a given media (stored in a common location or file), a convention should be used that allows a Recording Device to locate the title keys for all of those applications when it writes an MKB Extension. Alternatively, different applications may use different MKB Extensions (stored in separate locations or files). Such details are described for each application in the corresponding books of this specification.

3.1.3.2 Processing an Extended MKB

All Playback and Recording Devices used for applications where MKB extensions are supported must recognize and use an MKB Extension if one is present. This applies even to such a Recording Device that is not capable of writing MKB extensions. Since writing MKB Extensions is not mandatory, even for applications where they are supported, MKB processing shall proceed as usual if no MKB Extension is present (i.e. it is not considered an error if an MKB Extension is not present).

When an MKB Extension is present, devices treat the Static MKB and MKB Extension as if they were a single MKB (referred to as the Extended MKB) for the purpose of calculating K_m. The Records of the Static MKB are processed first, followed by the Records of the MKB Extension. The procedure is identical to that described in Section 3.1.2, with the following exceptions:

- The Extended MKB has two *Verify Media Key* Records and two *End of Media Key Block* Records, one each in the Static MKB and MKB Extension. When processing the Static MKB, the *End of Media Key Block* Record (or *Verify Media Key* Record, if it is used to stop processing early) is used as a signal to begin processing the MKB Extension.
- Recording Devices shall always use the *Verify Media Key* Record in the MKB Extension to verify calculation of the correct K_m value. If the final K_m value is not successfully verified, the Recording Device shall erase the MKB Extension. This ensures that an invalid MKB Extension does not permanently disable the media for CPRM use.

Note that devices processing an Extended MKB must ignore any additional *Calculate Media Key* Records encountered after the first one in the Static MKB. The MKB Extension should not contain a *Calculate Media Key* Record, and if one is encountered there, a device must ignore it.

3.1.4 Pseudo-code for Processing a Media Key Block

To help clarify the procedure for calculating the Media Key (K_m) from a Media Key Block, this section provides pseudo-code examples for processing both Static and Extended Media Key Blocks. The pseudo-code provided here shall not be considered definitive; other methods of processing Media Key Blocks that meet the requirements described in this chapter are possible.

The pseudo-code assumes that the following subroutines are available:

- **decrypt**(*key, data*) - returns the "double word" (8 byte) result from decryption using C2 in ECB mode. (Parameters *'key'* and *'data'* are also double words.)
- **getByte**() - returns the next byte in the Media Key Block.
- **getDoubleWord**() - returns the next 8 bytes in the Media Key Block.
- **skip**(*bytes*) - advances the current position in the Media Key Block by that number of bytes.

Furthermore, there are external arrays '*deviceKey*' (16 double words), and '*row*' (16 integers). These arrays are indexed by Column values, and contain, respectively, the Device Key values and associated Row values assigned to the particular device (this example assumes a case where the MKB has 16 Columns defined, and the device is assigned one Device Key for each Column).

Note that if the Media Key Block is for some reason found incorrectly formatted (e.g. the *End of Media Key Block* Record is missing, or a Record Length value is out of range), the **getByte**() or **getDoubleWord**() functions might return an "end-of-file" indication. For some media types, the MKB will have an associated length indicator (stored outside of the MKB), which could be used to prevent the **skip**() routine from advancing past the end of the MKB.

The following pseudo-code fragment illustrates the two steps in processing a Media Key Block when an MKB Extension is present:

```
double word mediaKey;
set input source to the Static MKB;
mediaKey = processMediaKeyBlock(nil);
if there is an MKB Extension then
        set input source to the MKB Extension;
        mediaKey = processMediaKeyBlock(mediaKey);
endif
```

The following is the “processMediaKeyBlock” routine:

```
procedure processMediaKeyBlock(
        double word mediaKey) /* incoming media key, (or nil) */
        returns double word;  /* media key or 'nil' */
{

        integer recordType;    /* type of each record */
        integer column;        /* column in device key matrix */
        integer length;        /* length of the record */
        double word verificationData; /* for verifying media key */
        double word buffer;    /* temporary buffer */

        do forever {

                recordType = getByte();
                if no more data in the media key block
                then
                        return mediaKey; /* missing End record -- ignore */
                endif

                /* read record length, and set length to remaining bytes: */
                length = (getByte() << 16) + (getByte() << 8) + getByte();
                length = length – 4;

                if length >= 8
                then
                        buffer = getDoubleWord();
                        length = length - 8;
                else if length < 0
                        then
                                length = 0; /* ignore bad length */
                endif
                endif


                switch based on recordType {

                case 0x82: /* Conditionally Calculate Media Key record */

                        buffer = decrypt(mediaKey, buffer);

                        if the first four bytes of buffer are not 0xDEADBEEF
                        then
                                exit switch;
                        endif

                        /* join next case below: */

                case 0x01: /* Calculate Media Key record */

                        column = the fifth byte of the buffer
                        /* (column numbers start at 0) */
                        if the last three bytes of buffer are not 0x000001
                           OR column >= 16
```

```
                then
                        exit switch; /* ignore, not an error */
                endif

                if (row[column]+1)*8 > length
                then
                        exit switch; /* ignore, not enough data,
                                        not an error */
                endif

                /* skip the cells up to the one I'm interested in: */
                skip(row[column] * 8); /* note rows start at 0! */

                /* get the cell and update the length */
                buffer = getDoubleWord();
                length = length - 8 - row[column] * 8;

                if recordType == 0x82
                then
                        buffer = decrypt(mediaKey, buffer);
                else
                        if mediaKey is not nil
                        then
                                exit switch; /* must enforce only one CMK record! */
                        endif
                endif

                mediaKey = decrypt(deviceKey[column], buffer);

                /* Verifying the media key as shown below is not mandatory.
                   However, recording devices must try to verify the media key
                   in MKB Extensions, and if ultimately unsuccessful, delete the MKB
                   Extension (case not shown here). */

                buffer = decrypt(mediaKey, verificationData);
                if the first four bytes of buffer are 0xDEADBEEF
                then
                        return mediaKey;
                endif

                exit switch;

        case 0x02: /* End of Media Key Block record */
                return mediaKey;

        case 0x81: /* Verify Media Key record */
                verificationData = getDoubleWord();
                exit switch;

        default case: /* it is important to ignore unknown records, for the future! */
                exit switch;

        }

        skip(length); /* advance to next record */

    }
}
```

The function returns 'nil' in the case of certain errors, although most errors are deliberately ignored. Since the Media Key value '0' can be used to detect that the device has been revoked, ‘nil’ might be some condition other than ‘0.’

3.2 Calculation of the Media Unique Key (K_{mu})

3.2.1 Media Identifier (ID_{media})

Each piece of CPRM compliant media (or in some cases, playback device) shall contain an individual identifier that is readable. This identifier does not need to be secret, but must be stored in a manner that prevents it from being altered or replaced, unless otherwise specified elsewhere in this specification. If an identifier is 64 bits long, its value can be used directly as ID_{media}. For cases where the identifier is not 64 bits long, a function shall be defined in the corresponding book of this specification to convert that identifier to a 64-bit value to be used as ID_{media}.

3.2.2 Media Unique Key (K_{mu})

CPRM's cryptographic key management uses a Media Unique Key (K_{mu}) to bind encrypted content to the media (or in some cases, device) on which it will be played back. K_{mu} is calculated using the ID_{media} and the previously calculated K_m, as follows:

$$K_{mu} = [C2\_G(K_m, ID_{media})]_{lsb\_56}.$$

The specific use of K_{mu} in the process of calculating encryption and decryption keys for content varies with different applications and media types, as described in the corresponding books of this specification.

Chapter 4
Common Cryptographic Functions for Enhanced CPRM

4. Introduction

This chapter specifies additional common cryptographic functions based on the AES cipher for implementing AES extensions to CPRM for various applications and media types, which is denoted AES-CPRM. The original version of CRPM is denoted C2-CPRM.

The AES-CPRM protects Content, Title Keys/Content Keys/User Key, CCI/Usage Rules and other data specified in this specification as protected with the AES cipher. Unless explicitly stated in this specification, the remaining data and procedures, such as authentication between a device and an SD Memory Card, continue to be protected by the C2-CPRM.

4.1 AES Block Cipher Algorithm

Common cryptographic functions used for AES-CPRM are based on the AES block cipher. A description of the AES block cipher algorithm is provided in FIPS Publication 197, as referenced in Section 1.4. Three block cipher modes, Electronic Codebook (ECB) mode, Cipher Block Chaining (CBC) mode and Counter (CTR) mode, which are used for AES-CPRM are provided in NIST Special Publication 800-38A, as referenced in Section 1.4. 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 three modes of operation.

4.1.1 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.

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

The AES cipher may be used in Cipher Block Chaining (CBC) mode for encryption and decryption of content protected by AES-CPRM. 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 specified in the corresponding books of this specification as Confidential Information, as defined in the CPRM License Agreement.

4.1.3 AES Block Cipher in Counter (CTR) Mode

The AES cipher may be used in Counter (CTR) mode for encryption and decryption of content protected by AES-CPRM. In this document and in other books of this specification, encryption with the AES cipher in CTR mode is represented by the function

AES_ECTR(k, d)

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

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

AES_DCTR(k, d)

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

The size of the frame of data to be encrypted or decrypted depends on the particular application format, and is defined for each in the corresponding books of this specification. The value of initial counter block used at the beginning of a CTR encryption or decryption chain (or method to generate the initial counter value) is specified in the corresponding books of this specification. This information is considered Confidential Information, as defined in the CPRM License Agreement. The counter for CTR mode shall be incremented by one (1) for every block encountered using the Standard Incrementing Function as specified in Appendix B.1 of *NIST Special Publication 800-38A* , unless otherwise specified in the other corresponding application books of this specification. Given a sequence of counters, T_1, T_2, …, T_n, the each value of counter block shall be calculated as follows (T_1 is the value of initial counter block):

$T_{j+1} = (T_j + 1) \bmod 2^{128}$ for $j = 1, 2, \ldots, n\text{-}1$

4.2 AES Hash Function

AES-CPRM 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 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 × 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 4-1.

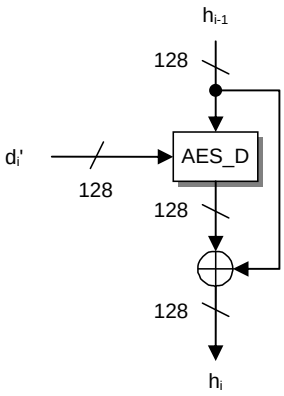

Figure 4-1 – AES Hash Function

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. $AES\_H(d) = h_n$.

4.3 AES One-way Function

AES-CPRM 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 4-2 depicts the one-way function.

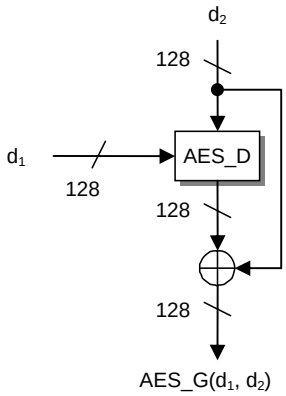

Figure 4-2 – AES One-way Function

The one-way function result is calculated as

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

4.4 Message Authentication Code (CMAC)

AES-CPRM uses a Cipher-based Message Authentication Code (CMAC) based on the AES encryption algorithm to generate a Message Authentication Code (MAC), as defined in the National Institute of Standards and Technology Special Publication 800-38B. This function is called the CMAC, and is represented by

M = CMAC(k, d)

where k is a 128-bit key to be used to create the MAC, d is the data to be authenticated in arbitrary length, and M is a resulting MAC in 128-bit length.

4.5 Random Number Generators

This section describes a random number generator and a pseudorandom number generator for generating random values of more than 64 bits (i.e. including 128 bits), both of which are based on the AES One-way Function. Unless explicitly noted otherwise, one or more of the following random/pseudorandom number generators shall be used: (1) Pseudorandom number generator based on a design described in either Section 4.5.1 or Section 4.5.2 as described below; (2) Pseudorandom number generators defined in NIST Special Publication 800-90; (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.

4.5.1 AES Random Number Generator

Figure 4-3 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.

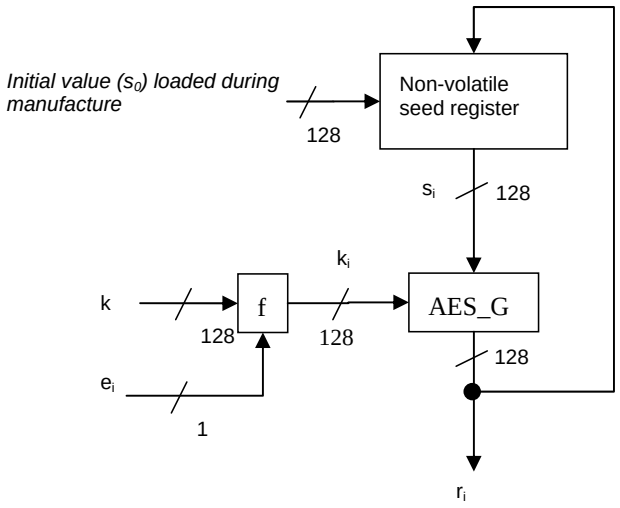

Figure 4-3 – AES Random Number Generator

Manufacturers need to generate a unique value, s_0, for each device. 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 CPRM License Agreement.

4.5.2 AES Pseudorandom Number Generator

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

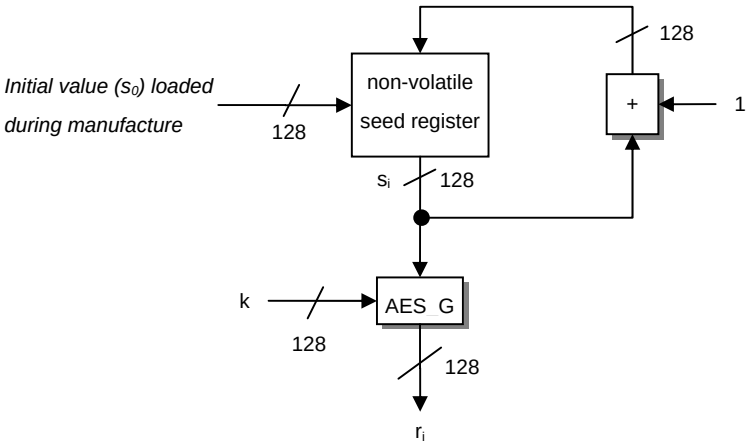

Figure 4-4 – AES Pseudorandom Number Generator

Manufacturers need to generate a unique value, s_0, for each device. 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 CPRM License Agreement.

This page is intentionally left blank.