Content Protection for Prerecorded Media Specification

Introduction and Common Cryptographic Elements

Intel Corporation
International Business Machines Corporation
Matsushita Electric Industrial Co., Ltd.
Toshiba Corporation

Revision 1.0
January 17, 2003

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, MEI, 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 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 cppm-licensing@4Centity.com.
- Feedback on this specification should be addressed to cppm-comment@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

Notice ..... **iii**

Intellectual Property ..... **iii**

Contact Information ..... **iii**

1. INTRODUCTION **1-1**

1.1 Purpose and Scope **1-1**

1.2 Overview **1-1**

1.3 Organization of this Document **1-2**

1.4 References **1-2**

1.5 Future Directions **1-3**

1.6 Notation **1-3**
1.6.1 Numerical Values 1-3
1.6.2 Bit and Byte Ordering 1-3
1.6.3 Operations 1-3

1.7 Abbreviations and Acronyms **1-4**

2. CPPM COMMON CRYPTOGRAPHIC FUNCTIONS **2-1**

2.1 C2 Block Cipher Algorithm **2-1**
2.1.1 C2 Block Cipher in Electronic Codebook (ECB) Mode 2-1
2.1.2 C2 Block Cipher in Converted Cipher Block Chaining (C-CBC) Mode 2-1

2.2 C2 One-way Function **2-2**

3. CPPM COMMON CRYPTOGRAPHIC KEY MANAGEMENT **3-1**

3.1 Device Keys **3-2**

3.2 Media Key Block (MKB) **3-2**
3.2.1 Verify Media Key Record 3-3
3.2.2 Calculate Media Key Record 3-4
3.2.3 Conditionally Calculate Media Key Record 3-5
3.2.4 End of Media Key Block Record 3-6

3.3 Pseudo-code for Processing a Media Key Block **3-7**

This page is intentionally left blank.

List of Figures

Figure 1-1 – CPPM Illustrative Example 1-2

Figure 2-1 – C2 One-way Function 2-2

Figure 3-1 – Common CPPM Cryptographic Key Management Procedure 3-1

This page is intentionally left blank.

List of Tables

Table 3-1 – Common Cryptographic Key Management Elements 3-1
Table 3-2 – *Verify Media Key* Record Format 3-3
Table 3-3 – *Calculate Media Key* Record Format 3-4
Table 3-4 – *Conditionally Calculate Media Key* Record Format 3-5
Table 3-5 – *End of Media Key Block* Record Format 3-6

This page is intentionally left blank.

Chapter 1
Introduction

1.

1.1 Purpose and Scope

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

- DVD Book, which describes protection for the DVD-Audio format.

Books covering other media types are expected to be available in the future (see Section 1.5 below). CPPM 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 CPPM 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 read-only 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 1. The 4C Entity, LLC provides secret device keys to the device manufacturer for inclusion into each device produced.

Step 2. The media manufacturer places a Media Key Block generated by the 4C Entity, LLC, and a title-specific identifier (placed so that it cannot be copied) on each piece of media containing protected content.

Step 3. The protected content on the media is encrypted by a Content Key, which is derived from a one-way function of the secret Media Key, the non-copyable identifier, and the copy control information (CCI)

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

Step 4. When media containing protected content is placed in a compliant drive or player, the device uses its keys and the Media Key Block stored on the media to calculate the secret Media Key. Using the Media Key, the device calculates the Content Key described in Step 3, for use in decrypting and playing the content.

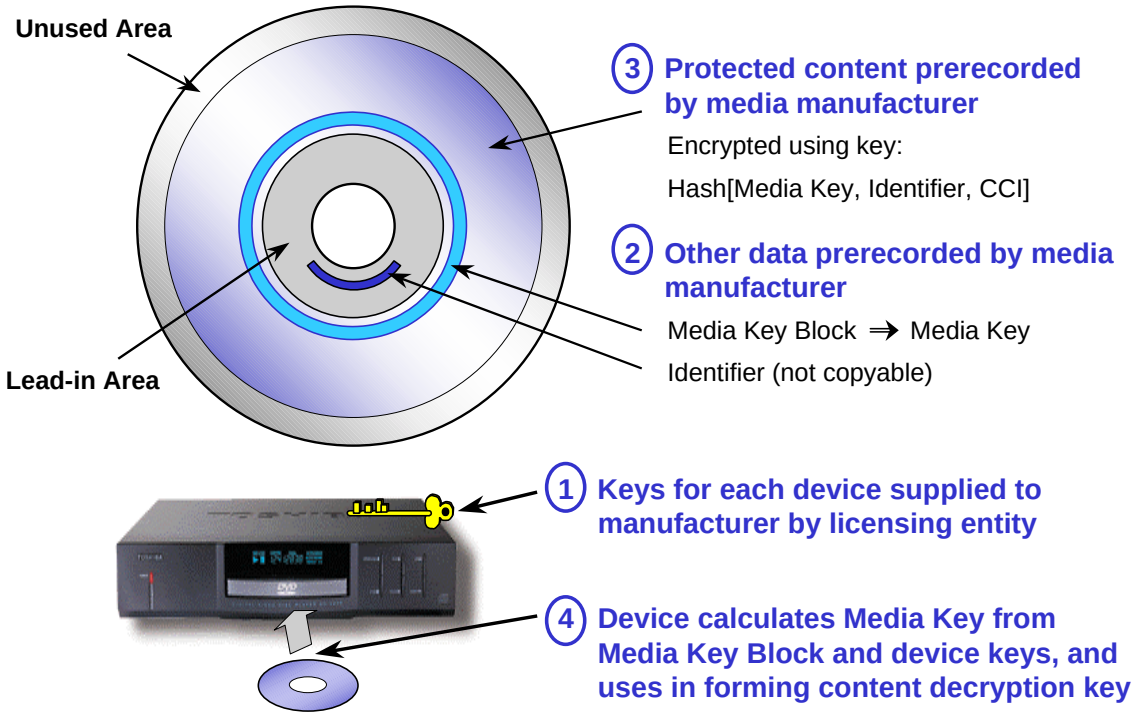

Figure 1-1 – CPPM Illustrative Example

1.3 Organization of this Document

This specification is organized as follows:

- Chapter 1 provides an introduction and overview of CPPM.
- Chapter 2 describes common CPPM cryptographic functions based on the C2 cipher algorithm.
- Chapter 3 describes a common CPPM 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, *CPPM 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

1.5 Future Directions

With its robust cryptography, key management, and renewability mechanisms, it is expected that CPPM will develop and expand, through additions to this specification, to address content protection for additional media types, application formats, and usage models. An extension under consideration at the time of this release is:

- Protection for read-only content prerecorded on SD memory card media.

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.

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, MEI, and Toshiba)
ASCII	American Standard Code for Information Interchange
C-CBC	Converted Cipher Block Chaining
C2	Cryptomeria Cipher
CCI	Copy Control Information
CE	Consumer Electronics
CPPM	Content Protection for Prerecorded Media
DVD	Digital Versatile Disc
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
CPPM Common Cryptographic Functions

2. Introduction

This chapter describes common cryptographic functions that are used by CPPM for various applications and media types. The functions are described here in isolation; their specific uses as part of CPPM 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 CPPM 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 CPPM. 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 One-way Function

CPPM 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 2-1 depicts the one-way function.

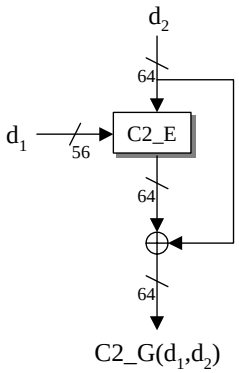

Figure 2-1 – C2 One-way Function

The one-way function result is calculated as

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

Chapter 3
CPPM Common Cryptographic Key Management

3. Introduction

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

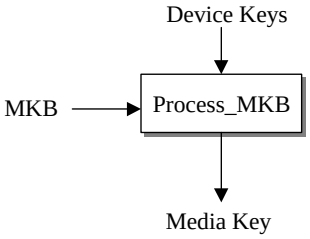

Figure 3-1 – Common CPPM Cryptographic Key Management Procedure

Device Keys (K_{d_0},K_{d_1},…,K_{d_n-1}) are used to decrypt one or more elements of a Media Key Block (MKB), in order to extract a secret Media Key (K_m). Table 3-1 lists the elements involved in this process, along with their sizes.

Table 3-1 – Common Cryptographic Key Management Elements

Key or Variable	Size
Device Keys (K_{d_0},K_{d_1},…,K_{d_n-1})	56 bits each
Media Key Block (MKB)	Variable, multiple of 4 bytes
Media Key (K_m)	56 bits

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

3.1 Device Keys

Each CPPM compliant Playback 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 CPPM 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 CPPM license agreement.

3.2 Media Key Block (MKB)

CPPM'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 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.

3.2.1 Verify Media Key Record

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

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

Byte \ Bit	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}.

3.2.2 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 uses of the Reserved and Revision fields are currently undefined for CPPM, and they are ignored. 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, D_{ke_r} is the 64-bit value starting at byte offset $r \times 8$ within the Record's Encrypted Key Data, and 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.

3.2.3 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, and 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.

3.2.4 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

Bit / Byte	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).

3.3 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 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 is the “processMediaKeyBlock” routine:

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

        double word mediaKey = nil;   /* for media key calculation */
        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. */

                        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’.