Content Protection for Recordable 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.

Copyright © 1999-2003 by International Business Machines Corporation, Intel Corporation, Matsushita Electric Industrial Co., Ltd., and Toshiba Corporation. Third-party brands and names are the property of their respective owners.

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 cprm-licensing@4Centity.com.
- Feedback on this specification should be addressed to cprm-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 ............................................................................................................... 1-3
   
   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. **CPRM 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 Hash Function** ............................................................................................... 2-2
   
   2.3 **C2 One-way Function** .......................................................................................... 2-3
   
   2.4 **Random Number Generators** .............................................................................. 2-3
      
      2.4.1 C2 Random Number Generator ......................................................................... 2-3
      2.4.2 C2 Pseudo-random Number Generator ............................................................... 2-4

3. **CPRM COMMON CRYPTOGRAPHIC KEY MANAGEMENT** .............................. 3-1
   
   3.1 Calculation of the Media Key ($K_m$) ....................................................................... 3-2
      
      3.1.1 Device Keys ........................................................................................................ 3-2
      3.1.2 Media Key Block (MKB) .................................................................................... 3-2
      3.1.2.1 Verify Media Key Record ............................................................................... 3-3
      3.1.2.2 Calculate Media Key Record .......................................................................... 3-4
      3.1.2.3 Conditionally Calculate Media Key Record .................................................... 3-5
3.1.2.4  End of Media Key Block Record ................................................................. 3-6
3.1.3  Media Key Block Extension ............................................................................. 3-7
3.1.3.1  Writing an MKB Extension .......................................................................... 3-7
3.1.3.2  Processing an Extended MKB ..................................................................... 3-9
3.1.4  Pseudo-code for Processing a Media Key Block .............................................. 3-9

3.2  Calculation of the Media Unique Key (K_{mu}) .................................................... 3-12
3.2.1  Media Identifier (ID_{media}) ........................................................................ 3-12
3.2.2  Media Unique Key (K_{mu}) ........................................................................... 3-12
List of Figures

Figure 1-1 – CPRM Illustrative Example ................................................................................................. 1-2
Figure 2-1 – C2 Hash Function ............................................................................................................ 2-2
Figure 2-2 – C2 One-way Function ..................................................................................................... 2-3
Figure 2-3 – C2 Random Number Generator ....................................................................................... 2-3
Figure 2-4 – C2 Pseudo-random Number Generator ............................................................................. 2-4
Figure 3-1 – Common CPRM Cryptographic Key Management Procedure ........................................ 3-1
Figure 3-2 – Calculation of Media Key from MKB and MKB Extension ................................................. 3-7
This page is intentionally left blank.
List of Tables

Table 2-1 – Examples of Padding Input Data ..................................................................................................... 2-2
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 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 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 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.

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

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

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., 1010₂). Hexadecimal numbers are represented as a string of hexadecimal (0..9, A..F) digits
followed by a subscript 16 (e.g., 3C2₁₆).

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)
- **ATA**: AT Attachment
- **C-CBC**: Converted Cipher Block Chaining
- **C2**: Cryptomeria Cipher
- **CCI**: Copy Control Information
- **CE**: Consumer Electronics
- **CPRM**: Content Protection for Recordable 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
- **SDMI**: Secure Digital Music Initiative
- **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

\[ C_2\_E(k, d) \]

where \( k \) is a 56-bit key, \( d \) is 64-bit data value to be encrypted, and \( C_2\_E \) returns the 64-bit result.

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

\[ C_2\_D(k, d) \]

where \( k \) is a 56-bit key, \( d \) is a 64-bit data value to be decrypted, and \( C_2\_D \) 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

\[ C_2\_ECBC(k, d) \]

where \( k \) is a 56-bit key, \( d \) is a frame of data to be encrypted, and \( C_2\_ECBC \) returns the encrypted frame.

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

\[ C_2\_DCBC(k, d) \]

where \( k \) is a 56-bit key, \( d \) is a frame of data to be decrypted, and \( C_2\_DCBC \) 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>
<thead>
<tr>
<th>Length of input data ( d ) (in bits)</th>
<th>Formation of padded data ( d' )</th>
</tr>
</thead>
<tbody>
<tr>
<td>126</td>
<td>( d' = d \parallel 10_2 )</td>
</tr>
<tr>
<td>127</td>
<td>( d' = d \parallel 1_2 )</td>
</tr>
<tr>
<td>128</td>
<td>( d' = d \parallel 8000000000000000_16 )</td>
</tr>
</tbody>
</table>

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.

A conversion function \( f \) is defined as

\[ f(x) = \lfloor x \rfloor_{lsb, 56} \]

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.

\[ C2_G(d_1, d_2) = C2_E(d_1, d_2) \oplus d_2. \]

2.4 Random Number Generators

This section describes a random number generator and a pseudo-random number generator, 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, shall be used for CPRM.

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.
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 Pseudo-random Number Generator

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

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](image)

- Device Keys ($K_{d_0}, K_{d_1}, \ldots, K_{d_{n-1}}$) 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>
<thead>
<tr>
<th>Key or Variable</th>
<th>Size</th>
</tr>
</thead>
<tbody>
<tr>
<td>Device Keys ($K_{d_0}, K_{d_1}, \ldots, K_{d_{n-1}}$)</td>
<td>56 bits each</td>
</tr>
<tr>
<td>Media Key Block (MKB)</td>
<td>Variable, multiple of 4 bytes</td>
</tr>
<tr>
<td>Media Key ($K_m$)</td>
<td>56 bits</td>
</tr>
<tr>
<td>Media Identifier ($ID_{media}$)</td>
<td>64 bits</td>
</tr>
<tr>
<td>Media Unique Key ($K_{mu}$)</td>
<td>56 bits</td>
</tr>
</tbody>
</table>

The remainder of this section describes this common cryptographic key management procedure in detail.
3.1 Calculation of the Media Key (Km)

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 Km. 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 Kd_i (i=0,1,…,n-1). For each Device Key there is an associated Column and Row value, referred to as Cd_i and Rd_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 Km. 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 Km 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 Km 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 Km value. Processing of subsequent Records may update the Km value that was calculated previously. After processing of the MKB is completed, the device uses the most recently calculated Km value as the final value for Km (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 Km will have the special value 0000000000000000_16. This special value will never be an MKB’s correct final Km value, and can therefore always be taken as an indication that the device’s keys are revoked. If a device calculates this special Km value, it shall stop the authentication/playback/recording session in progress, and shall not use that Km 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

<table>
<thead>
<tr>
<th>Byte</th>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

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

\[ D_v = C2_E(K_m, \text{DEADBEEF}_{16} || \text{XXXXXXXX}_{16}) \]

where \( K_m \) is the correct final Media Key value, and \( \text{XXXXXXXX}_{16} \) 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

\[ [C2_D(K_m, D_v)]_{\text{msb,32}} = \text{DEADBEEF}_{16} \]

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 \( 0000000000000000_{16} \).
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

<table>
<thead>
<tr>
<th>Bit</th>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>16</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>16</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>9</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>10</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>11</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>12</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>13</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>14</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>15</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

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 000016 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 00000116 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 == 00000116

and

the device has a Device Key with associated Column value \( C_{d,1} = \text{Column} \), 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

<table>
<thead>
<tr>
<th>Bit</th>
<th>Byte</th>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>8</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>9</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>10</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>11</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>12</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>13</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>14</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>15</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>16</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>17</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>18</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>19</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>20</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>21</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>22</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>23</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>24</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>25</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>26</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>27</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>28</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>29</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>30</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>31</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

---

**Encrypted Conditional Data (D_{c, i})**

- **Record Type:** \( 82_{16} \)
- **Record Length:** \( \) (encrypted)
- **DEADBEEF_{16}** (encrypted)
- **Column** (encrypted)
- **Generation:** \( 000001_{16} \) (encrypted)
- **Doubly Encrypted Key Data for Row 0** (\( D_{ke, 0} \))
- **Doubly Encrypted Key Data for Row 1** (\( D_{ke, 1} \))
- **Doubly Encrypted Key Data 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:

- \([D_c]_{msb\_32} == \text{DEADBEEF}_{16}\)
- \([D_c]_{lsb\_24} == \text{000001}_{16}\)
- 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>
<thead>
<tr>
<th>Byte</th>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

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](image)

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 as the New MKB (since it is typically more recent than the Static MKB placed on the media by the manufacturer), 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 \textit{Calculate Media Key} Record into a \textit{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:

\[ D_{kde},_0 = C2_E(K_m, D_{ke},_0), \]

\[ D_{kde},_1 = C2_E(K_m, D_{ke},_1), \]

\[ \vdots \]

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.
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 `getBytes()` 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:

```c
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:

```c
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 0x0000001
      OR column >= 16
```
then
    exit switch; /* ignore, not an error */
endif

if (row[column]+1)*8 > length
    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
    buffer = decrypt(mediaKey, buffer);
else
    if mediaKey is not nil
        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
    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:

$$K_{mu} = \text{C2}_G(K_m, ID_{media})_{lsb, 56}.$$  

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.
This page is intentionally left blank.