Content Protection for Recordable Media Specification

Portable ATA Storage Book

Intel Corporation

International Business Machines Corporation

Matsushita Electric Industrial Co., Ltd.

Toshiba Corporation

Revision 0.92

May 30, 2001

This page intentionally left blank.

Preface

Notice

This document is an intermediate draft and is subject to change without notice. Adopters and other users of this specification are cautioned that products based on it may not be interoperable with the final version or subsequent versions thereof.

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.

Change History

- Revision 0.90. Initial version.
- Revision 0.92. The names of the commands were changed to conform to recommendations from the Secure CompactFlash Working Group. The command byte and Identify Device status bits were changed based on new assignments from the CompactFlash Association.

This page intentionally left blank.

Table of Contents

Chapter 1
Introduction

1. Introduction

1.1 Purpose and Scope

The *Content Protection for Recordable Media Specification* (CPRM) defines a robust and renewable method for protecting content stored on a number of physical media types. The specification is organized into several "books". The *Introduction and Common Cryptographic Elements* book provides a brief overview of CPRM, and defines cryptographic procedures that are common among its different uses. This document (the *Portable ATA Storage Book*) specifies additional details for using CPRM technology to protect content stored on removable or externally portable ATA devices with integrated controllers, namely:

- CompactFlash™ storage cards
- IBM Microdrive™ hard disk drives
- Other such devices as authorized by the 4C Entity, LLC.

In the remainder of this document, such devices will be referred to collectively as "CPRM Portable ATA Storage".

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 Document Organization

This specification is organized as follows:

- Chapter 1 provides an introduction.
- Chapter 2 lists abbreviations and acronyms used in this document.
- Chapter 3 describes the use of CPRM to protect content stored on ATA media.
- Chapter 4 describes the "Madison" application format for protecting audio content.

1.3 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, *CPRM Specification: Introduction and Common Cryptographic Elements, Revision 0.94*

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

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

1.4 Future Directions

In future revisions, the use of CPRM for other application formats on ATA storage may also be described.

1.5 Notation

Except where specifically noted otherwise, this document uses the same notations and conventions for numerical values, operations, and bit/byte ordering as described in the *Introduction and Common Cryptographic Elements* book of this specification.

Chapter 2
Abbreviations and Acronyms

2. Alphabetical List of Abbreviations and Acronyms

The following abbreviations and acronyms are used in this document:

4C	4 Companies (IBM, Intel, MEI, and Toshiba)
AAC	Advanced Audio Codec
ATA	PC/AT Attachment (the AT in IBM's PC/AT stood for "Advanced Technology").
C-CBC	Converted Cipher Block Chaining
C2	Cryptomeria Cipher
CCI	Copy Control Information
Cmd.	Command
CPRM	Content Protection for Recordable Media
CGMS	Copy Generation Management System
ECB	Electronic Codebook
FAT	File Allocation Table
FM	Flash Media
ID	Identifier
LCM	Licensed Compliant Module
LLC	Limited Liability Company
lsb	Least Significant Bit
MKB	Media Key Block
msb	Most Significant Bit
PC	Personal Computer
PD	Portable Device
PIO	Programmed Input/Output
PM	Portable Media
SDMI	Secure Digital Music Initiative
XOR	Exclusive-OR

This page is intentionally left blank.

Chapter 3
CPRM for Portable ATA Storage

3. CPRM for Portable ATA Storage

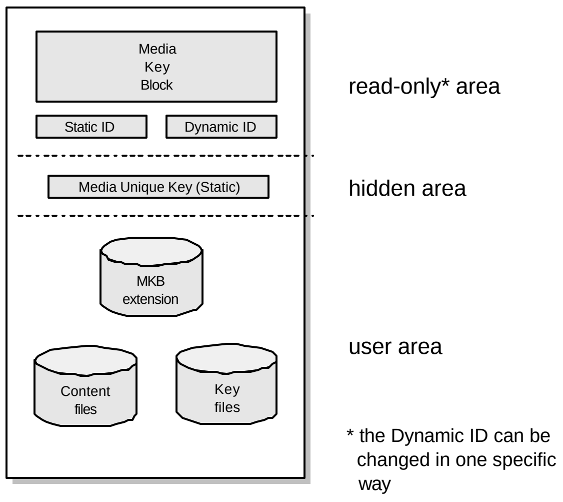

Figure 1. CPRM Portable ATA Storage

Figure 1 shows the important CPRM components on the portable ATA storage. Many of the components are simple files in the user area. The Media Key Block, Static Media ID, and Dynamic Media ID must be read-only; after being set during the manufacturing process, they must not be settable outside of the storage controller. There is one exception: there is a special command that increments the Dynamic Media ID as described below. The Static Media Unique Key must be kept as a secret.

3.1 Device Requirements

Each CPRM compliant recording or playing device for CPRM portable ATA storage must follow the protocols for accessing devices described in this specification.

In addition, each device is given a set of 16 secret Device Keys, denoted $K_{d\_1},K_{d\_2},\ldots,K_{d\_16}$. The actual keys are provided by the 4C Entity, LLC, and are used for processing the MKB to calculate the Media Key (K_m), as described in the *Introduction and Common Cryptographic Elements* 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.

3.2 Media ID

CPRM Portable ATA Storage devices use an “increment ID” technique to securely support check-in/check-out and move applications (as defined in SDMI) without having to explicitly authenticate the host. That means that the authentication is one-way; the host must authenticate the storage, but the reverse is not true.

The attack against check-in/check-out applications is the so-called "save/restore" attack. The attacker, for example, saves a copy of the user area prior to checking-in a piece of content. Although the compliant host erases the content, the attacker simply does a bit-for-bit restore of the user area to keep an extra copy. This attack is defeated by having the compliant host change the media ID when checking in any piece of content. Since changing the media ID changes the media unique key, any saved copy becomes useless. Of course, the freedom to set the media ID to an arbitrary value would allow media IDs to be cloned. Therefore the host is only allowed to increment the media ID by one, via a special command described below, and is not allowed to otherwise modify it.

Some applications are not interested in check-in/check-out. For them, the changing of media ID can be a nuisance. For this reason, the CPRM Portable ATA Storage device provides *two* media IDs: a Dynamic Media ID and a Static Media ID. Applications can use either ID.

The Static Media ID is derived from the Serial Number (words 10-19) and the Model Number (words 27-46) in the returned device identification data. The actual 64-bit CPRM Static Media ID is calculated as follows:

1. The 20-byte Serial Number is appended to the 40-byte Model Number.
2. Four more bytes of x'00' are appended.
3. The resulting 64-byte buffer is XORed together, 8 bytes at a time. The result is the 64-bit Static Media ID.

Storage devices must have unique serial numbers, and the serial number must be read-only, once set during manufacturing.

The Dynamic Media ID is a 64-bit number, initialized to a statistically unique (random) number at manufacturing time. Manufacturers are not required to check for duplicates. The Dynamic Media ID is returned in the READ KEY MANAGEMENT STRUCTURE command, explained below:

3.2.1 Identify Device Command (ECh)

This is an existing command. The following reserved bit is newly defined for CPRM.

Word 162 bit 0: Shall be set to 1 if CPRM is supported.

Manufacturers of ATA storage devices may find it convenient to include the CPRM Feature Set commands in all their control units, but only certain of their products might actually have media key blocks and a media unique key. Obviously, a storage device without a media key block cannot be used to protect content. In this case, the device, if it has licensed components such as C2 or the increment media ID function, must return "command abort" to the commands below.

3.2.2 Read Key Management Structure Command (B9h Features 00h-7Fh)

All CPRM commands use command byte B9h. The different commands are distinguished by the Features register.

Protocol

PIO data-in.

Inputs

The Features register bit 7 shall be set to zero.

Register	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
Features	0	c4						
Sector Count	c3							
Sector Number	c2							
Cylinder Low	c1							
Cylinder High	c0							
Device/Head	na			DEV	0			
Command	B9h							

Features, Sector Count, Sector Number, Cylinder Low, Cylinder High

Are treated as a 39-bit challenge number concatenating c4 to c3 to c2 to c1 to c0.

Device/Head -

DEV shall indicate the selected device. The low order 4 bits shall be zero to denote CPRM.

Outputs

Register	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
Error	na	UNC	na	na	na	ABRT	na	na
Sector Count	na							
Sector Number	na							
Cylinder Low	na							
Cylinder High	na							
Device/Head	obs	na	obs	DEV	0			
Status	BSY	DRDY	DF	na	DRQ	na	na	ERR

Device/Head register -

DEV shall indicate the selected device.

Status register -

BSY shall be cleared to zero indicating command completion.

DRDY shall be set to one.

DF (Device Fault) shall be cleared to zero.

DRQ shall be cleared to zero.

Error register -

ABRT shall be set to one if this command is not supported or if the low 4 bits of the Device/Head register are not 0.

UNC shall be set if an uncorrectable read error reading the Key Management structure.

Device/Head register -

DEV shall indicate the selected device.

The READ KEY MANAGEMENT STRUCTURE command returns a 512 byte data structure via PIO data-in transfer. The content of this data structure indicates the dynamic media ID, the size of the media key block, and the response to the challenge number explained below.

The format of the READ KEY MANAGEMENT PROTECTION data structure is shown in table 1.

Table 1 - Key Management Structure for CPRM

Word	Content
0	Key Management Scheme
1-4	Dynamic Media ID
5	Size of Media Key Block
6-9	Challenge Response
10-254	Reserved
255	Integrity Word
	15-8 checksum
	7-0 signature

Word 0: Key Managment Scheme

This word shall be set to 0001h, denoting the CPRM key management scheme.

Words 1-4: Dynamic Media ID

The Dynamic Media ID is a 64-bit number, initialized to a statistically unique (random) number at manufacturing time. Manufacturers are not required to check for duplicates. Word 1 is the most significant word; word 4 is the least significant word.

Word 5: Size of Media Key Block

The size of the media key block, in 512-byte sectors.

Words 6-9: Challenge Response

The 64-bit response to the challenge. Word 6 is the most significant word; word 9 is the least significant word.

The storage device calculates a Challenge Response as shown in Figure 2. The challenge allows the host to verify that it is connected to a valid CPRM device, and not a circumvention program masquerading as one.

This command transfers a 39-bit random challenge number to the storage device. The challenge is expanded to the 56-bit quantity $C = 0_b \| c4 \| c3 \| c2 \| c1 \| c0 \| 0000_{16}$. The storage controller will calculate the response and place it in words 6-9 of the READ KEY MANAGEMENT STRUCTURE data. The response is:

$$R = C2\_G(K_{mus}, C2\_G(C, ID_d))$$

ID_d is the Dynamic Media ID, words 1-4. K_{mus} is the static media unique key (the static media unique key is the one resulting from the Static Media ID, i.e., it is $K_{mus} = [C2\_G(K_m, ID_s)]_{lsb56}$.

The storage controller does not have to calculate the static media unique key. Instead, it can be stored in a non-user-accessible area during manufacturing.

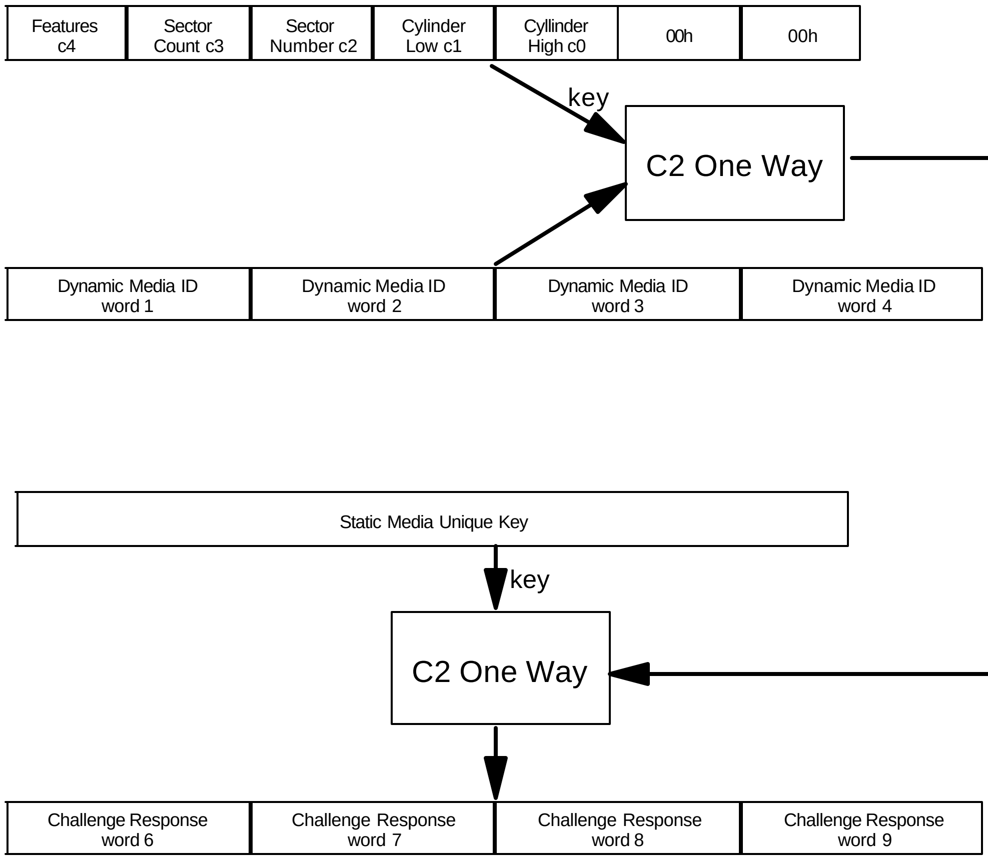

Figure 2: CPRM Challenge Response Calculation

Word 255: Integrity Word

Bits 7:0 of this word shall contain the value A5h. Bits 15:8 of this word shall contain the data structure checksum. The data structure checksum shall be the two's complement of the sum of all byte in words 0 through 254 and the byte consisting of bits 7:0 of word 255. Each byte shall be added with unsigned arithmetic, and overflow shall be ignored. The sum of all bytes is zero when the checksum is correct.

Note that PIO data-in protocol transfers data in words (16 bits). The more significant byte of the most significant word is the first byte of the Dynamic Media ID and the Challenge Response, for the purpose of performing C2 functions. The less significant byte is the second byte of the Dynamic Media ID and the Challenge Response, and so on. Observe that if the host processor is using an architecture with "little endian" byte ordering, and simply stores the words in memory as it receives them, it will get the wrong result.

3.2.3 Change Key Management Value Command (B9h, Features 81h)

CPRM Portable ATA Storage devices shall implement the CHANGE KEY MANAGEMENT VALUE command. This is a new command. In CPRM, the "key management value" that is changed is the Dynamic Media ID (words 1-4 in the READ KEY MANAGEMENT STRUCTURE response). It is incremented by one. It allows the ATA storage device to support SDMI check-in and move applications.

Protocol

Non-data.

Inputs

The Features register shall be set to 81h.

Register	7	6	5	4	3	2	1	0
Features	81h							
Sector Count	na							
Sector Number	na							
Cylinder Low	6Eh							
Cylinder High	B2h							
Device/Head	na			DEV	0			
Command	B9h							

Device/Head -

DEV shall indicate the selected device.

Cylinder Low –

Shall be set to 6Eh.

Cylinder High –

Shall be set to B2h.

Outputs

If the device does not support this command, or if the Cylinder Low is not 6Eh, or if the Cylinder High is not B2h, or if the low 4 bits of the Device/Head register are not 0, the device shall return command aborted.

Register	7	6	5	4	3	2	1	0
Error	na	UNC	na	na	na	ABRT	na	na
Sector Count	na							
Sector Number	na							
Cylinder Low	na							
Cylinder High	na							
Device/Head	obs	na	obs	DEV	na			
Status	BSY	DRDY	DF	na	DRQ	na	na	ERR

Error register -

ABRT shall be set to one if this command is not supported, or if the CPRM feature is disabled, or if Cylinder Low is not 6Eh, or if Cylinder High is not B2h.

UNC shall be set to one if there was a uncorrected error while incrementing the media ID.

Device/Head register -

DEV shall indicate the selected device. The low order 4 bits shall be zero to denote CPRM.

Status register -

BSY shall be cleared to zero indicating command completion.

DRDY shall be set to one.

DF (Device Fault) shall be set to one if a device fault has occurred.

DRQ shall be cleared to zero.

ERR shall be set to one if an Error register bit is set to one.

This command increases the value of the Dynamic Media ID by one per command. The Dynamic Media ID is returned in the READ KEY MANAGEMENT STRUCTURE command.

3.3 The Media Key Block

The read-only media key block is only accessible by the special command defined below. There is also a media key block extension. It is named **MKB.EXT** in the **CPRM** directory. It is a normal read/write user file and is accessed without the use of special commands.

The size of the read-only media key block will not exceed 1 MB (2048 sectors). No row number will exceed 16,383.

3.3.1 Read Keying Material Command (B9h, Features 80h)

CPRM Portable ATA Storage devices shall implement the READ KEYING MATERIAL command. In CPRM, the “keying material” is the media key block. This is a new command.

Protocol

PIO data-in

Inputs

The Features register shall be set to 80h.

Register	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
Features	80h							
Sector Count	count							
Sector Number	sector offset – low							
Cylinder Low	sector offset – high							
Cylinder High	na							
Device/Head	na			DEV	na			
Command	B9h							

Sector Count

the number of MKB sectors to be transferred. 00h denotes 256 sectors.

Cylinder Low, Sector Number –

These two registers are treated as a 16-bit number to specify the sector offset within the media key block to begin the transfer from. Cylinder Low is the more significant byte.

Device/Head -

DEV shall indicate the selected device. The low order 4 bits shall be zero to denote CPRM.

Outputs

If the device does not support this command, the device shall return command aborted.

Register	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
Error	na	UNC	na	IDNF	na	ABRT	na	na
Sector Count	na							
Sector Number	sector offset – low							
Cylinder Low	sector offset – high							
Cylinder High	na							
Device/Head	obs	na	obs	DEV	0			
Status	BSY	DRDY	DF	na	DRQ	na	na	ERR

Device/Head register -

DEV shall indicate the selected device.

Status register -

BSY shall be cleared to zero indicating command completion.

DRDY shall be set to one.

DF (Device Fault) shall be cleared to zero.

DRQ shall be cleared to zero.

ERR shall be set to 1 if any bit is set in the error register.

Error register -

ABRT shall be set to one if this command is not supported or if the low 4 bits of the Device/Head register are not 0.

IDNF shall be set if the requested range is out of bounds for the MKB.

UNC shall be set if an uncorrectable read error.

Cylinder Low, Sector Number

These registers are treated as a 16-bit number specifying the failing sector offset, in the case that UNC is set. Otherwise, the values are undefined.

This command transfers ‘count’ sectors of the Media Key Block, starting at the sector offset specified, to the host by using PIO data-in protocol.

Note that PIO data-in protocol transfers data in words (16 bits). The more significant byte of the first word is the first byte of the Media Key Block. The less significant byte is the second byte of the Media Key Block, and so on. Observe that if the host processor is using an architecture with “little endian” byte ordering, and simply stores the words in memory as it receives them, it will get the wrong result.

3.4 Authentication

The host uses authentication to verify that it is speaking to a legitimate storage device, and that the important CPRM values returned from the storage device have not been modified by an interloper. The host challenges the storage device, and the storage device calculates the correct response. The challenge/response is are both carried out using the single READ KEY MANAGEMENT STRUCTURE command.

3.4.1 Authentication Protocol

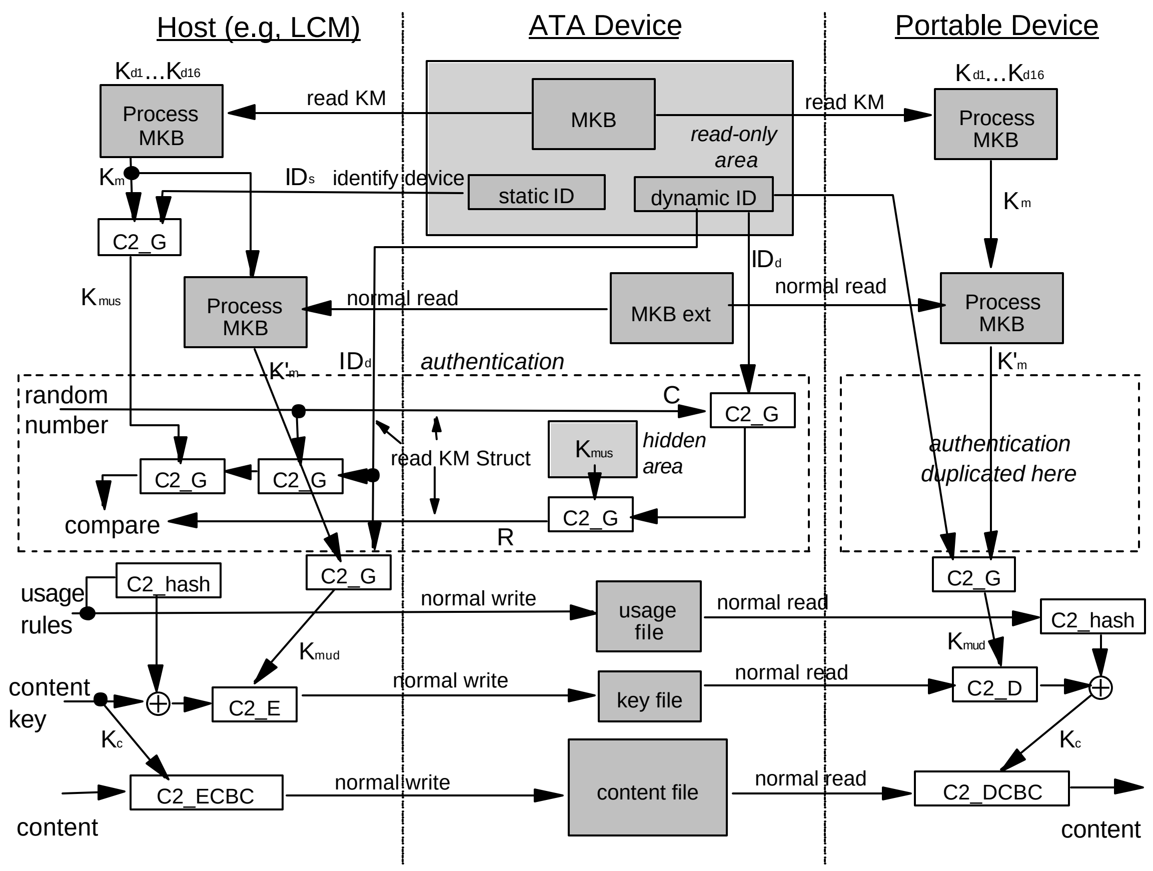

Figure 3: Example Recorder/Player Interaction

Figure 3 shows an example interaction between a recorder (like a PC), a CPRM Portable ATA Storage device, and a player (like a portable device). For simplicity, the player's authentication is not shown. It is exactly like the host's.

Connecting hosts shall use the READ KEY MANAGEMENT STRUCTURE command to authenticate the storage device, as follows:

1. The host executes the IDENTIFY DEVICE command and securely remembers the storage device's Model Number and Serial Number.. From the Model Number and Serial Number it calculates the Static Media ID as described in Section 3.2.
2. The host executes one or more READ KEYING MATERIAL commands to read the media key block. It processes the media key block in the normal way, yielding the media key.
3. The host calculates the static media unique key from the media key and the Static Media ID in the usual way (i.e., it is $K_{mus} = [C2\_G(K_m, ID_s)]_{lsb56}$).
4. The host generates a random 39-bit number and executes the READ KEY MANAGEMENT STRUCTURE command, passing the 39-bit number as a challenge. It is not required that the random number generator in this case be unpredictable to a potential attacker; it is enough that it produces a long (2^{39}) series.

5. The host reads the Dynamic Media ID and Challenge Response in the READ KEY MANAGEMENT STRUCTURE command in step 4.
6. Using the static media unique key it calculated in step 3, the host verifies the quantity
   $C2\_G(K_{mus}, C2\_G(C, ID_d))$
   is the same as the storage device's Challenge Response.
7. The host should not proceed if the Challenge Response is incorrect.

The host shall execute the authentication protocol whenever it reads the Static Media ID, the Dynamic Media ID, and/or the media key block. It shall repeat it any time it executes an CHANGE KEY MANAGEMENT VALUE command, and verify that the Dynamic Media ID has been incremented by one.

3.5 Key Files

CPRM for Portable ATA Storage applications support check-in/check-out and move operations by changing the Dynamic Media ID after each check-in or move. This changes the dynamic media unique key. This means that *every* such application's keys must be updated when *any one* application changes the media ID. To make this feasible, the content keys are stored in a standard format, in a standard place.

Changing the media key block extension, **\CPRM\MKB.EXT**, has the same problem, and requires the same solution.

The solution is for all such applications to use the common format to find and fix each other's keys when they change either the Dynamic Media ID or the **MKB.EXT**. The format is as follows: content keys are stored in files in the sub-directory **KEYS** in the **CPRM** directory. The key files are multiples of 8 bytes long. Each 8 bytes is simply one 56-bit content key concatenated with a random pad byte, in that order, and the result is encrypted with the media unique key using the C2 cipher in ECB mode. The names of the key files can be anything. Applications should not put other types of files in the **\CPRM\KEYS** directory. When an application changes the dynamic media unique key, it reads every key file, decrypts it with the old media unique key, and re-encrypts it with the new media unique key.

Note that, in a multi-tasking system, an application should obtain a software mutex before making any changes to the key files. It should wait to read the Dynamic Media ID and **MKB.EXT** until it has obtained the mutex.

If applications do not use the Dynamic Media ID, they must not follow this key file naming convention, nor can they use the common **\CPRM\MKB.EXT** file. They are free to define an alternative media key block extension file.

3.5.1 Check-in Protocol

When a CPRM for Portable ATA Storage application checks in a piece of content, it changes the Dynamic Media ID, and re-encrypts content keys in the **\CPRM\KEYS** directory with the new dynamic media unique key. In doing so, it must be sure not to re-encrypt *any* occurrences of the checked-in content's key value, since otherwise a user may be able to restore a saved copy of the content and still play it (by renaming files, etc. as needed). Note then that the content key value itself is used as the definitive identifier for the content. When an application checks out content, it must securely remember the content key value that was used. When it checks that content back in, it must ensure that *all* occurrences of its content key value in the **\CPRM\KEYS** directory are either deleted or changed to new random values before being re-encrypted with the new media unique key. The details of this protocol are described below.

There is a very small chance that a content key picked at random will equal a key already in use on the media, causing confusion at check-in time. Therefore, at check-out time, before generating a new content key in the **\CPRM\KEYS** directory, applications shall check to make sure the proposed key does not duplicate an existing value. For the purpose of checking for duplications, an application shall compare both the content key and its associated random pad byte. Therefore, applications do not have to decrypt every existing key; they may simply compare the cipher text of existing keys with the cipher text of the proposed new key.

The description below assumes that the application has previously executed the authentication protocol and therefore has read the media key block and has an authenticated static media unique key. It also may have had an authenticated Dynamic Media ID; however, this shall not be relied upon since that ID might have changed in the meantime.

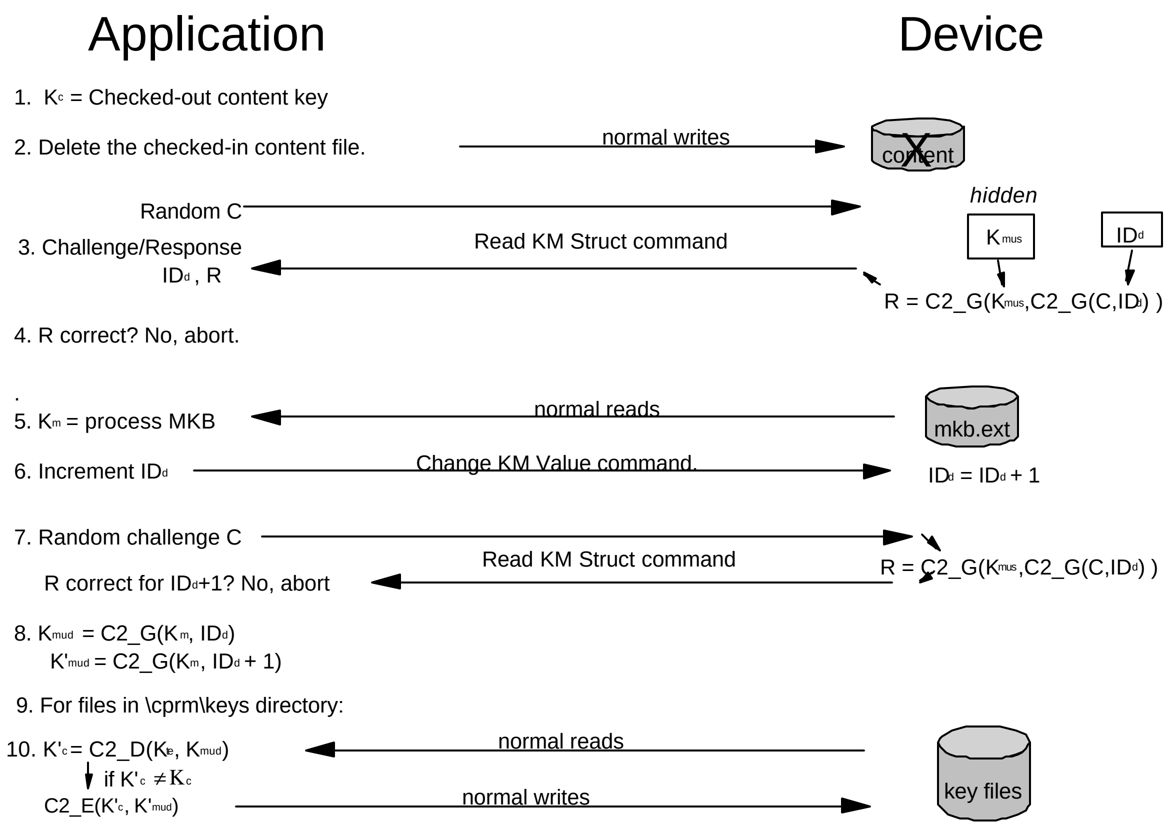

Figure 4. The Check-in Protocol

The application proceeds as follows:

1. The application denotes the content key and associated random pad byte it used to check-out the content as "K_c".
2. The application deletes the content file.
3. The application executes the READ KEY MANAGEMENT STRUCTURE command with a random challenge.
4. The application verifies the Challenge Response is correct. If not, it aborts. If so, it remembers the Dynamic Media ID.
5. The application reads the **\CPRM\MKB.EXT** file, if present, and processes it, obtaining a media key.
6. The application executes the CHANGE KEY MANAGEMENT VALUE command.
7. The application executes the READ KEY MANAGEMENT STRUCTURE command with a new random challenge and verifies that the Challenge Response is correct, given the incremented Dynamic Media ID. If not, it aborts.

8. Using the media key from step 5, the application calculates two dynamic media unique keys, the old one from before the CHANGE KEY MANAGEMENT VALUE command, and the new one after the CHANGE KEY MANAGEMENT VALUE command.
9. The application iterates through the **\CPRM\KEYS** directory, and for each file, it performs the following:
10. It reads the file, decrypting each 8 bytes in ECB mode using the old media unique key from step 8. It re-encrypts the file, 8 bytes at a time in ECB mode, using the new media unique key from step 8. Along the way, any occurrences of K_c, are invalidated[1].

[1] There are several effective ways to invalidate a key: the fundamental principle is that K_c must **at no time** be encrypted with the new media unique key, to avoid a "pull card" attack. The simplest way is to leave the 8 bytes untouched (as they would be if the file were deleted). Other ways would be to set the key (or the cipher text) to random data, or to zeros.

Chapter 4
"Madison" Audio Format

4. "Madison" Audio Format

This chapter describes the use of CPRM for the "Madison" format for audio applications. In this format, the file system is a very simple FAT file system. All relevant files (except key files) are in the **MADISON** directory in the root. If any subdirectories are found, they are ignored. Likewise, if a file's name does not match a recognized Madison name, it is ignored. Thus the same media can be used for other applications. The music itself and associated data are in separate normal read/write files. The name of each file is the decimal track number prefixed by the letters "**TRK**". The extension of the file is **".INI**". Thus the sequential tracks are defined by the files **TRK1.INI, TRK2.INI, TRK3.INI**, and so on. The **INI** file contains human-readable ASCII text. Here is an example **INI** file:

```
[Track]
AlbumTitle=My List of Songs
AlbumArtist=My Favorite Song Writers
TrackTitle=Song1
TrackArtist= Song 1 Performer
TrackID=1234567890ABC.qwerty.5

[Files]
SongTrack=SONG1.TRK
Lyrics=SONG1TRK.TXT
TrackArt=ARTWORK1.GIF
KeyFile=\CPRM\KEYS\SONG1.KEY
```

The **INI** files contain both pointers to other files and metadata about the track. To a player, the essential files are the **SongTrack** and **KeyFile** files. The other files, and the metadata in the **[Track]** section, are optional to enhance the user display, and may be omitted by some hosts. The minimal **INI** file has the **[Files]** section, with **SongTrack** and **KeyFile** keywords only.

4.1 Music Host Requirements

It is anticipated that connected applications will write a media key block extension (**\CPRM\MKB.EXT**) when recording a track or tracks. The media key block extension accelerate the revocation process and greatly reduce the effectiveness of the "virtual device" attack.

After the host downloads some music, it should guarantee that all the track numbers in all the **TRK*n*.INI** names are consecutive, starting at "1", and are in the order the user wants them played. If necessary, it must rename existing files so that this is true. All other files on the media must be left unchanged.

4.2 Portable Player Requirements

Of course, the end-user may use another non-compliant program to delete, rename, or otherwise reorganize the **INI** files after the compliant application is done. The player shall gracefully handle missing files or non-consecutive tracks, even though the compliant host itself will leave the media in a predictable state. Often, there is a correspondence between the **INI** file and the song and key files. For example, the **\MADISON\TRK1.INI** file might have a SongFile of **\MADISON\SONG1.TRK** and a key file of **\CPRM\KEYS\SONG1.KEY**. This is just a coincidence, and the player must *not* count on it. Once the user has reorganized the order of the tracks (especially by non-compliant software), the correspondence will be broken.

4.3 Format of the SongTrack File

The music file is kept in the standard "RIFF" format. The first 12 bytes of the file are the RIFF header, as follows:

'R'	**'I'**	**'F'**	**'F'**
length(LSB)	**length(LSB+1)**	**length(LSB+2)**	**length(MSB)**
'C'	**'P'**	**'R'**	**'M'**

The length field indicates the length of the file in bytes, minus this 12 byte header. The rest of the file is divided into "RIFF chunks". Devices should ignore chunks with identifiers they do not understand. The chunks are defined by 8 byte headers, as follows:

Identifier
Length (LSB first)

The first identifier is the ASCII string "fmt " (note the trailing blank). The length field indicates the size of the remainder of the chunk, in this case 6 bytes. The 6 bytes are the ASCII string "C2EPAC" denoting C2 encrypted EPAC compressed music. The second chunk has the identifier "data". It contains the encrypted EPAC data. Only this **data** chunk is encrypted. All headers, identifiers, and other chunks are in the clear.

The encrypted music chunk is encrypted either with the track key, or with the track key XORed with a hash of the **INI** file calculated with the C2 Hash Function. The hash is used if and only if the **INI** file contains a [Usage] section (described below), to give a cryptographic guarantee that any usage conditions will remain intact, or else the music will not play. The C2 hash need not be calculated if there is not a [Usage] section.

The encryption frame for the encrypted music is 4096 bytes. This means that every 4096 bytes the C-CBC chaining is broken, and the next block is encrypted starting with the original key. The last frame's size may be less than 4096 bytes, but shall nevertheless be a multiple of 8 bytes.

4.4 Encoding the Usage Rules

The special usage rules are placed in the **INI** file, in the [Usage] section. If there is no [Usage] section (the likely case), the default usage rules allow unlimited play of the associated music in the portable player, and the CCI is "copy no more". There are two types of keywords that might be found in a [Usage] section. The first type allows additional permissions, and can be ignored if the player does not understand the keyword. This type of keyword always begins with the first character '*'. The second type of keyword restricts the default usage rules. If the player does not understand or does not support the keyword, it **must not play** the music. This type of keyword never begins with a '*' character. Thus if the portable player only supports the normal usage rules, it must parse the [Usage] section. Along the way, if it finds a keyword that does not begin with '*', it must ignore the track.

Here are the currently defined keywords:

expires=	The date (mm/dd/yyyy) after which the music must not be played.
begins=	The date (mm/dd/yyyy) before which the music must not be played.
remaining=	The remaining counts (decimal number) of plays. The device must (logically) decrement the count after 30 seconds of the track has played.
***moveable**	The music is allowed to be moved to a system that did not originally download it.

Although the concepts behind the usage rules are simple, implementing them securely is so difficult that most players are not expected to support them. To implement the “expires” or “begins” keyword, the player must have a real time clock that is not easily settable by the user. To implement the “remaining” keyword, the player must understand the CHANGE KEY MANAGEMENT VALUE command, so that the user cannot reset the count by restoring the flash memory to a previous state. (The “*moveable” keyword is a flag important to compliant recording applications and has no semantics in the player.)