US 20080137865A1

(19) **United States**

(12) **Patent Application Publication**

Sabet-Sharghi et al.

(10) **Pub. No.: US 2008/0137865 A1**

(43) **Pub. Date: Jun. 12, 2008**

(54) **SYSTEM, METHOD, AND DEVICE FOR PLAYING BACK RECORDED AUDIO, VIDEO OR OTHER CONTENT FROM NON-VOLATILE MEMORY CARDS, COMPACT DISKS, OR OTHER MEDIA**

(76) Inventors: **Farshid Sabet-Sharghi**, Los Altos Hills, CA (US); **Bahman Qawami**, San Jose, CA (US); **Robert C. Chang**, Danville, CA (US)

Correspondence Address:
WEAVER AUSTIN VILLENEUVE SAMPSON LLP
ATTN: SANDISK, P.O. BOX 70250
OAKLAND, CA 94612-0250

(21) Appl. No.: **11/901,577**

(22) Filed: **Sep. 18, 2007**

Related U.S. Application Data

(63) Continuation of application No. 10/006,554, filed on Dec. 6, 2001.

(60) Provisional application No. 60/251,731, filed on Dec. 7, 2000.

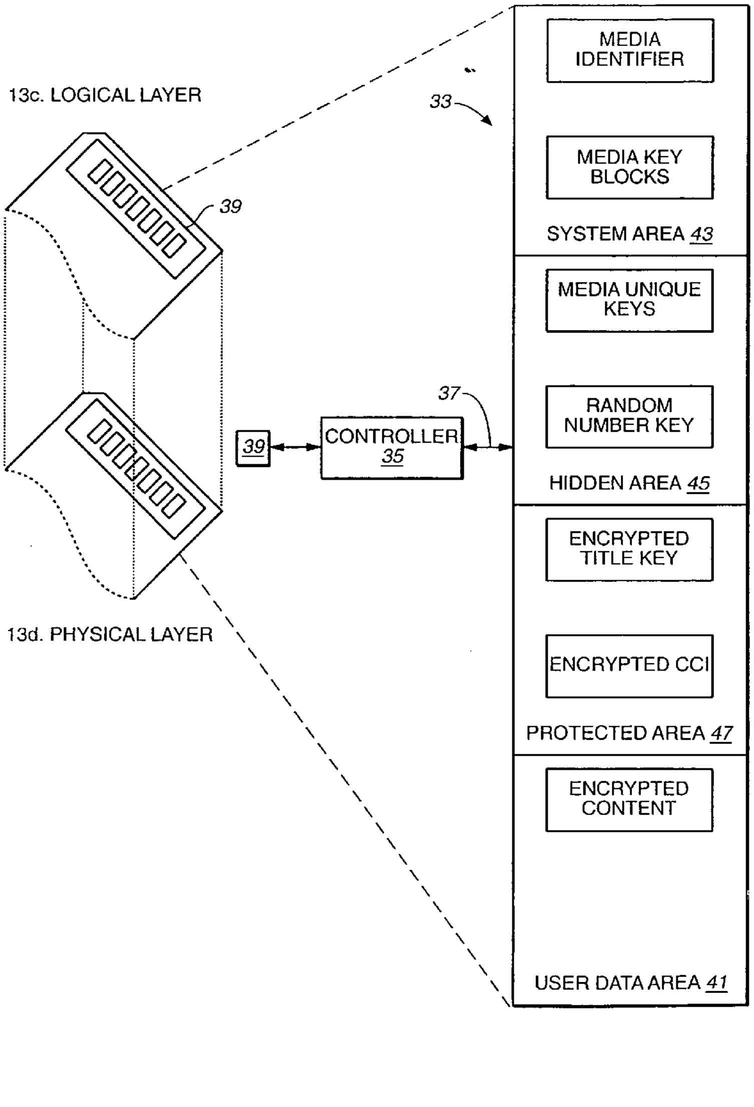
Publication Classification

(51) **Int. Cl.**
G06F 12/14 (2006.01)
H04L 9/00 (2006.01)

(52) **U.S. Cl.** **380/277**; 713/193

(57) **ABSTRACT**

A secure software package for original equipment manufacturers to run in electronic devices in order to access and dynamically decrypt encrypted audio video or other content from a memory storage device such as a memory card, optical or hard disk such that the user interface of the device need only send simple commands and the decrypted content is output.



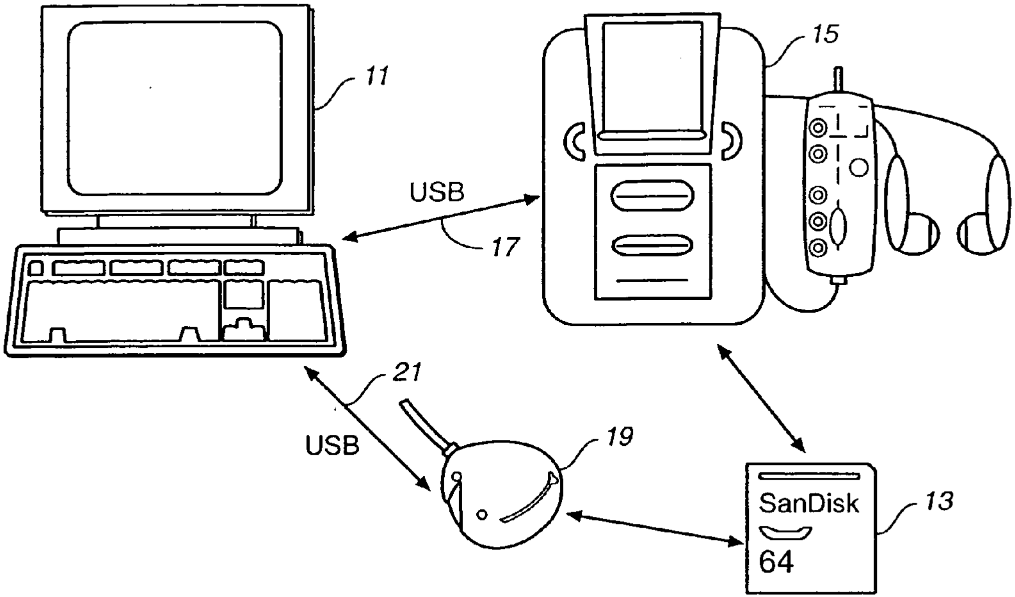

11
15
USB
17
21
USB
19
SanDisk
64
13

FIG._1

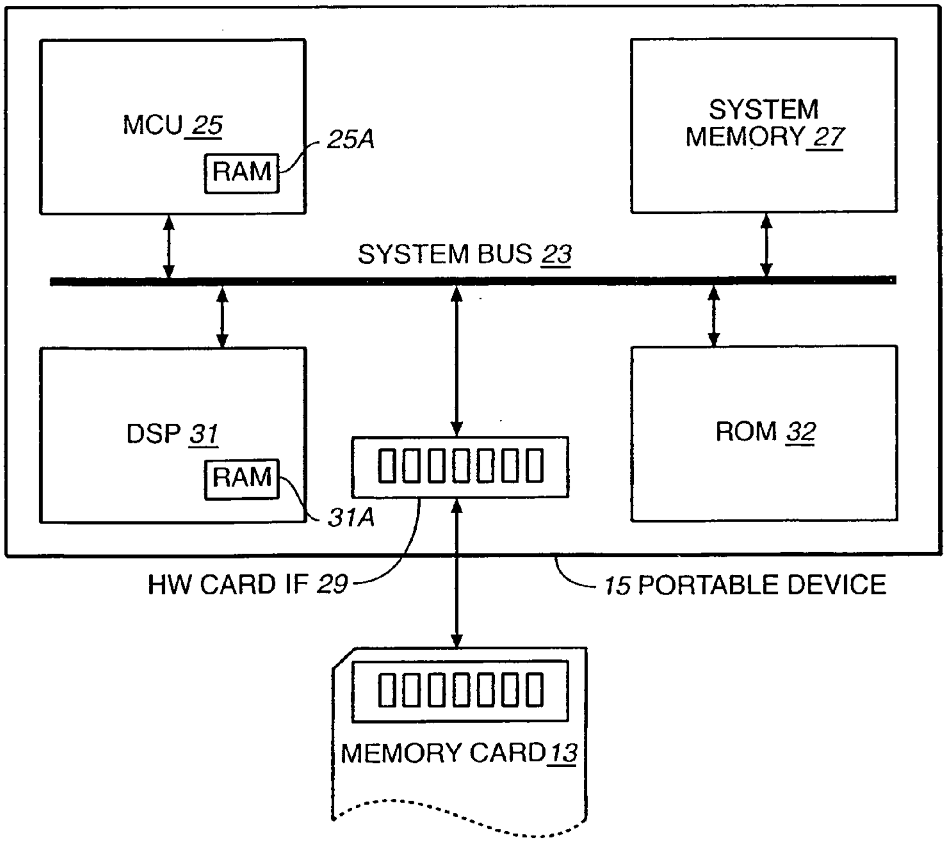

MCU 25
RAM
25A
SYSTEM MEMORY 27
SYSTEM BUS 23
DSP 31
RAM
31A
ROM 32
HW CARD IF 29
15 PORTABLE DEVICE
MEMORY CARD 13

FIG._2

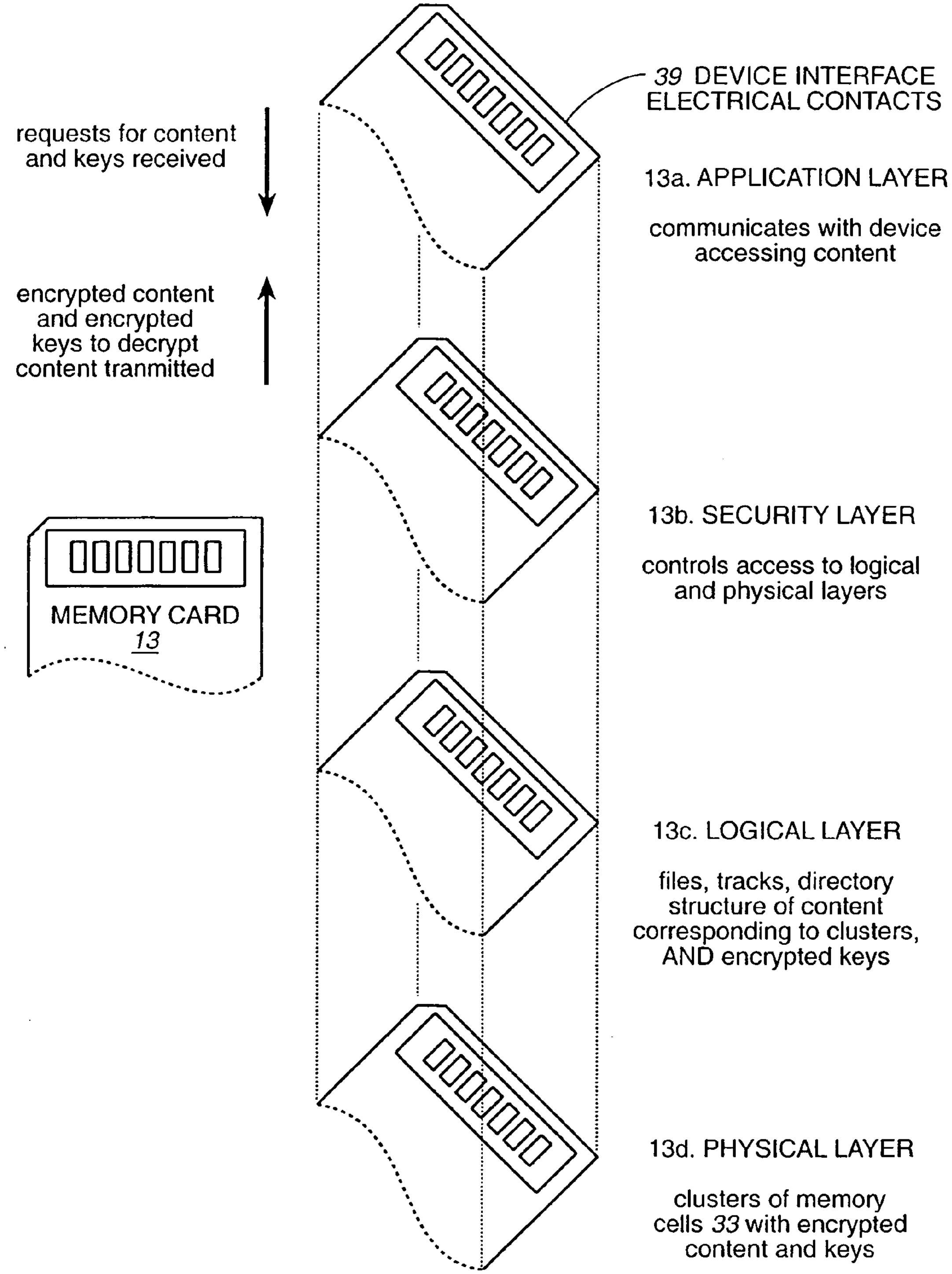

39 DEVICE INTERFACE ELECTRICAL CONTACTS
requests for content and keys received
encrypted content and encrypted keys to decrypt content tranmitted
MEMORY CARD 13
13a. APPLICATION LAYER
communicates with device accessing content
13b. SECURITY LAYER
controls access to logical and physical layers
13c. LOGICAL LAYER
files, tracks, directory structure of content corresponding to clusters, AND encrypted keys
13d. PHYSICAL LAYER
clusters of memory cells 33 with encrypted content and keys

FIG._3A

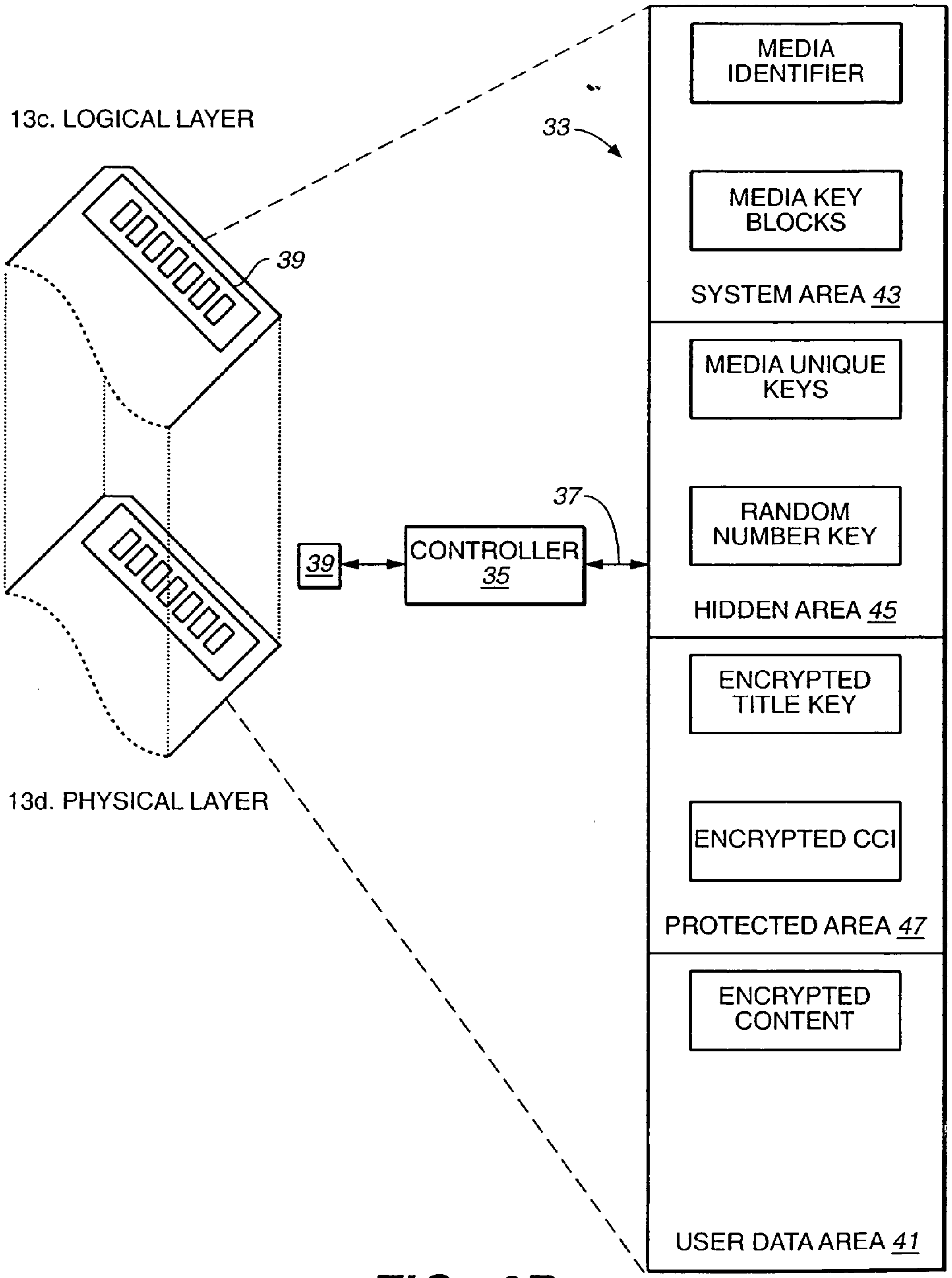

13c. LOGICAL LAYER
33
MEDIA IDENTIFIER
MEDIA KEY BLOCKS
SYSTEM AREA 43
39
MEDIA UNIQUE KEYS
37
RANDOM NUMBER KEY
39
CONTROLLER 35
HIDDEN AREA 45
ENCRYPTED TITLE KEY
13d. PHYSICAL LAYER
ENCRYPTED CCI
PROTECTED AREA 47
ENCRYPTED CONTENT
USER DATA AREA 41

FIG._3B

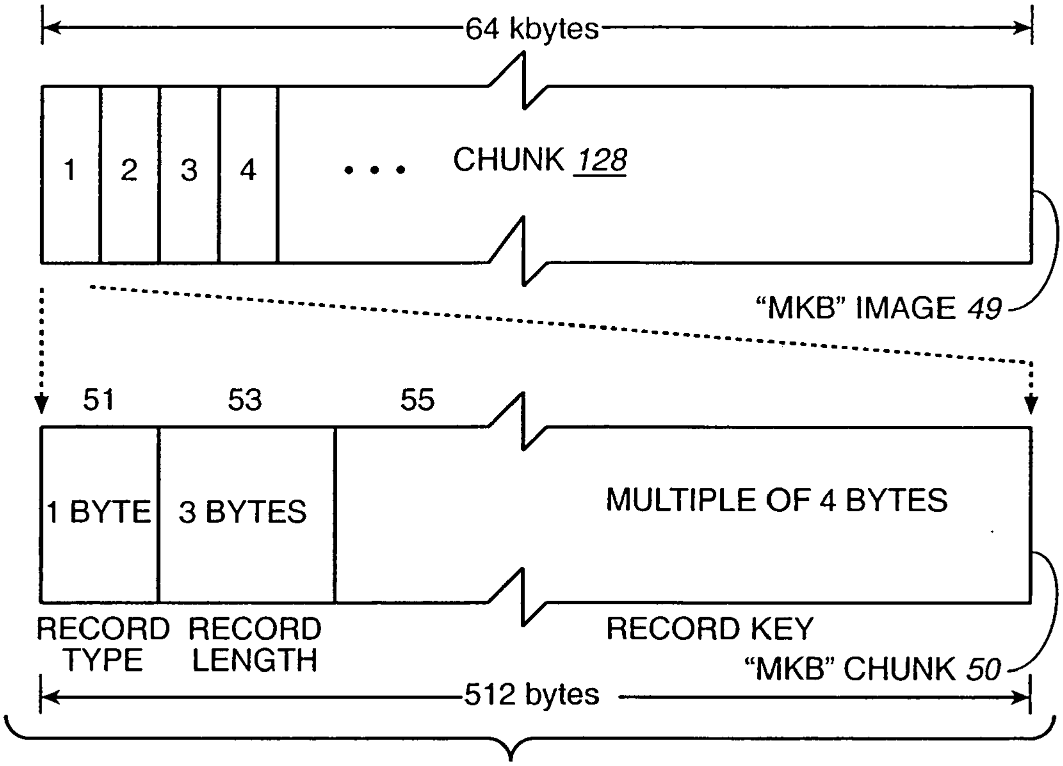

64 kbytes
1
2
3
4
• • •
CHUNK 128
"MKB" IMAGE 49
51
53
55
1 BYTE
3 BYTES
MULTIPLE OF 4 BYTES
RECORD TYPE
RECORD LENGTH
RECORD KEY
"MKB" CHUNK 50
512 bytes

FIG._4

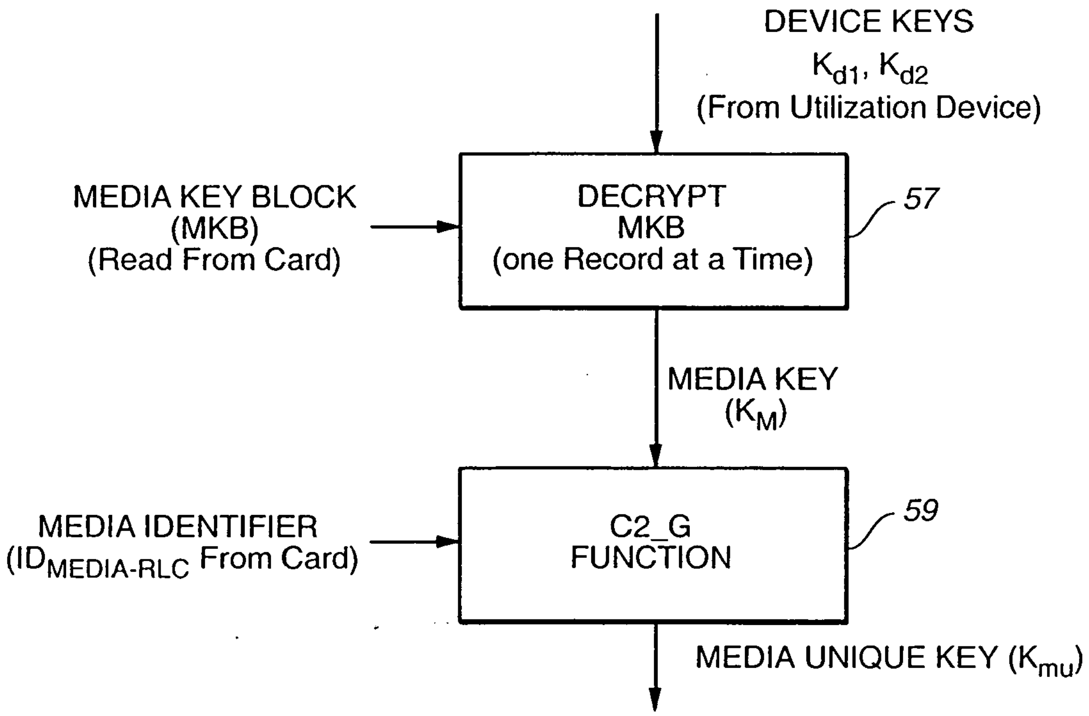

DEVICE KEYS
K_{d1}, K_{d2}
(From Utilization Device)
MEDIA KEY BLOCK
(MKB)
(Read From Card)
DECRYPT
MKB
(one Record at a Time)
57
MEDIA KEY
(K_M)
MEDIA IDENTIFIER
($ID_{MEDIA-RLC}$ From Card)
C2_G
FUNCTION
59
MEDIA UNIQUE KEY (K_{mu})

FIG._5

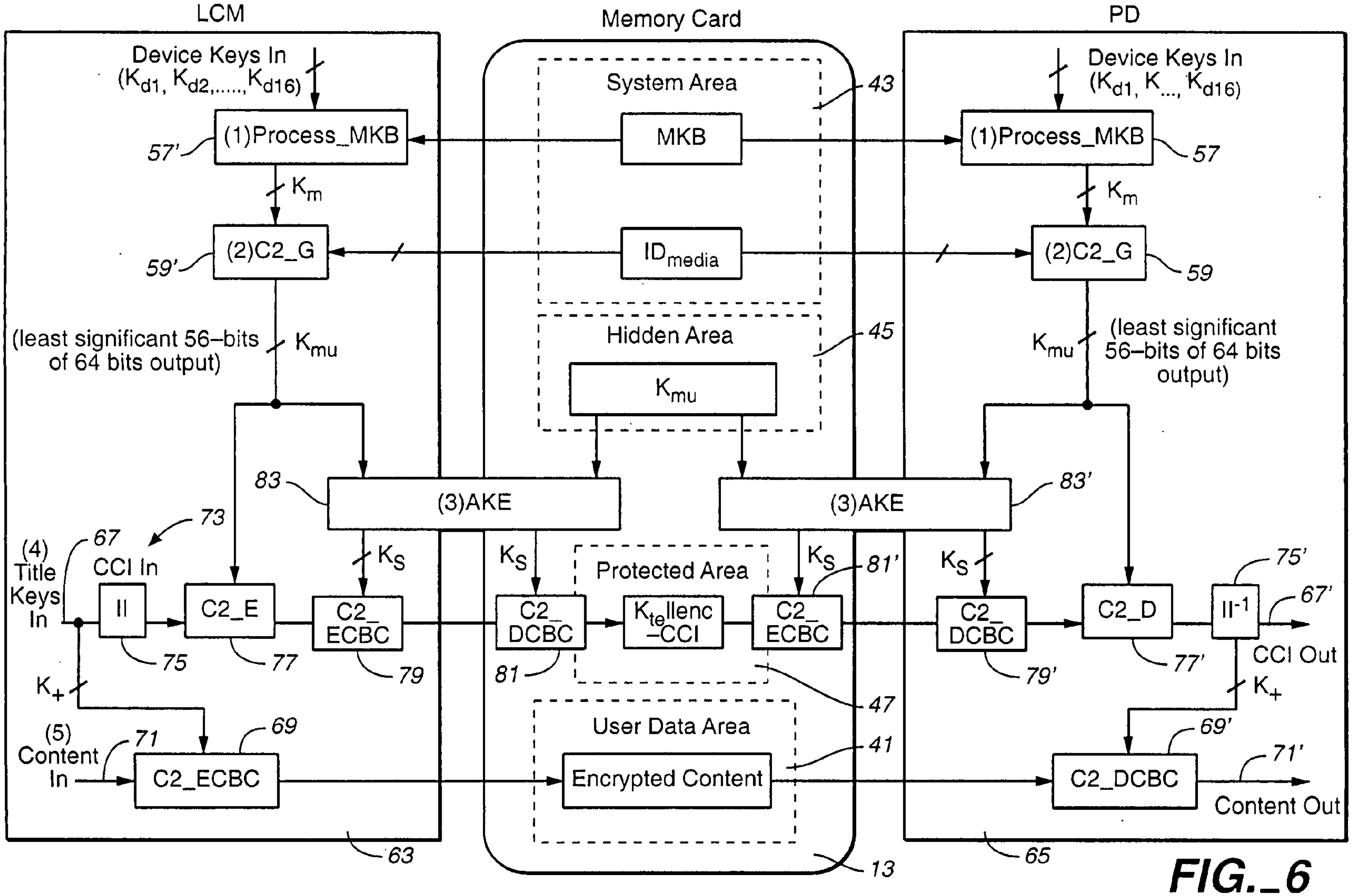

LCM
Memory Card
PD
Device Keys In
(K_{d1}, $K_{d2,....,}$$K_{d16}$)
(1)Process_MKB
57'
K_m
(2)C2_G
59'
(least significant 56–bits of 64 bits output)
K_{mu}
System Area
43
MKB
ID_{media}
Hidden Area
45
K_{mu}
Device Keys In
(K_{d1}, $K_{...}$, K_{d16})
(1)Process_MKB
57
K_m
(2)C2_G
59
K_{mu}
(least significant 56–bits of 64 bits output)
83
(3)AKE
(3)AKE
83'
73
67
(4) Title Keys In
CCI In
II
75
C2_E
77
K_S
C2_ ECBC
79
K_S
C2_ DCBC
81
Protected Area
K_{te}llenc –CCI
K_S
81'
C2_ ECBC
47
K_S
C2_ DCBC
79'
C2_D
77'
75'
II^{-1}
67'
CCI Out
K_+
K_+
(5) Content In
71
C2_ECBC
69
User Data Area
41
Encrypted Content
69'
C2_DCBC
71'
Content Out
63
13
65

FIG._6

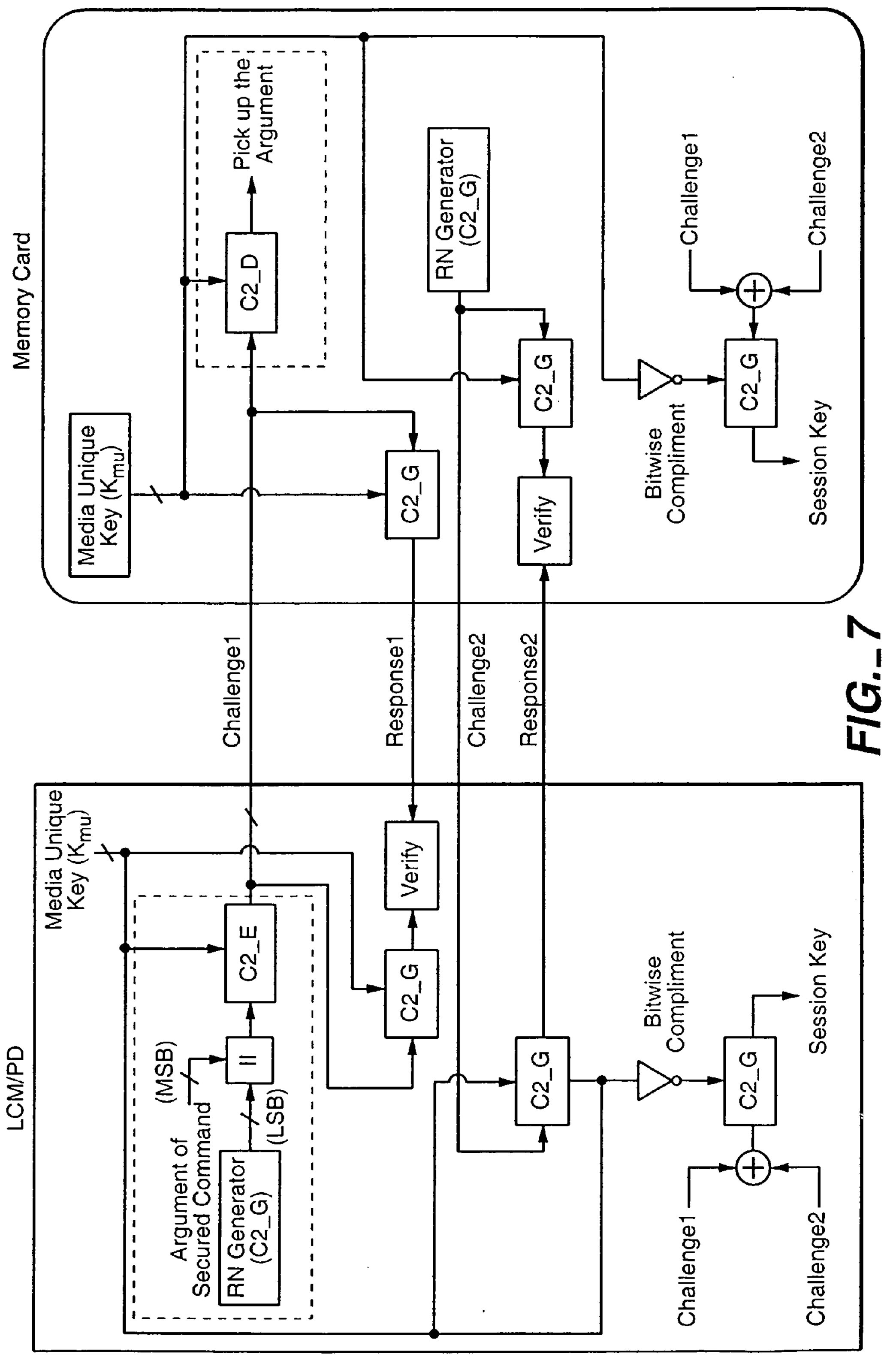

Memory Card
Media Unique Key (K_{mu})
C2_D
Pick up the Argument
C2_G
RN Generator (C2_G)
C2_G
Verify
Bitwise Compliment
C2_G
Challenge1
Challenge2
Session Key
Challenge1
Response1
Challenge2
Response2
LCM/PD
Media Unique Key (K_{mu})
Argument of Secured Command
(MSB)
(LSB)
RN Generator (C2_G)
II
C2_E
C2_G
Verify
C2_G
Bitwise Compliment
C2_G
Challenge1
Challenge2
Session Key

FIG._7

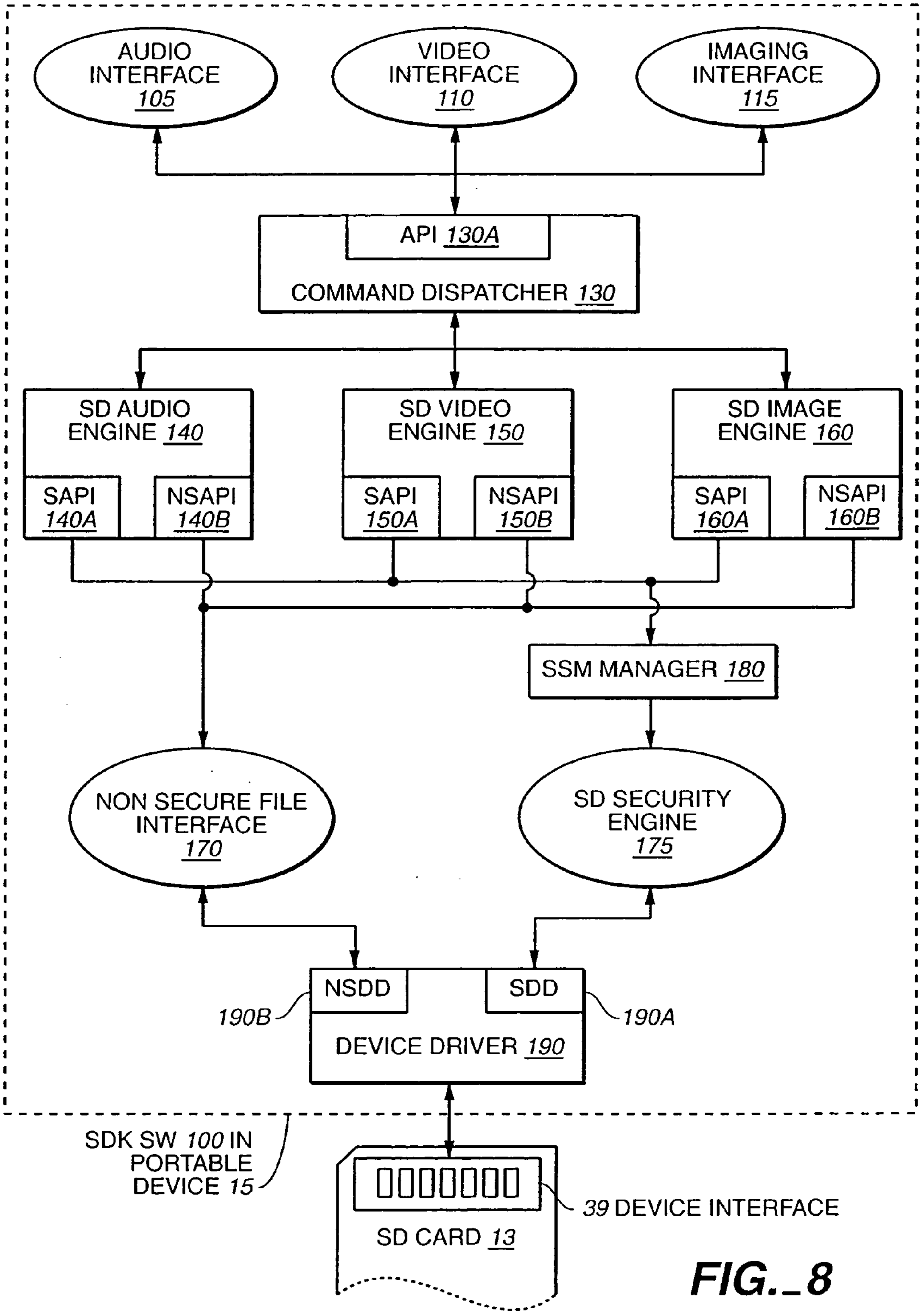

AUDIO INTERFACE 105
VIDEO INTERFACE 110
IMAGING INTERFACE 115
API 130A
COMMAND DISPATCHER 130
SD AUDIO ENGINE 140
SAPI 140A
NSAPI 140B
SD VIDEO ENGINE 150
SAPI 150A
NSAPI 150B
SD IMAGE ENGINE 160
SAPI 160A
NSAPI 160B
SSM MANAGER 180
NON SECURE FILE INTERFACE 170
SD SECURITY ENGINE 175
NSDD
SDD
190B
190A
DEVICE DRIVER 190
SDK SW 100 IN PORTABLE DEVICE 15
SD CARD 13
39 DEVICE INTERFACE

FIG._8

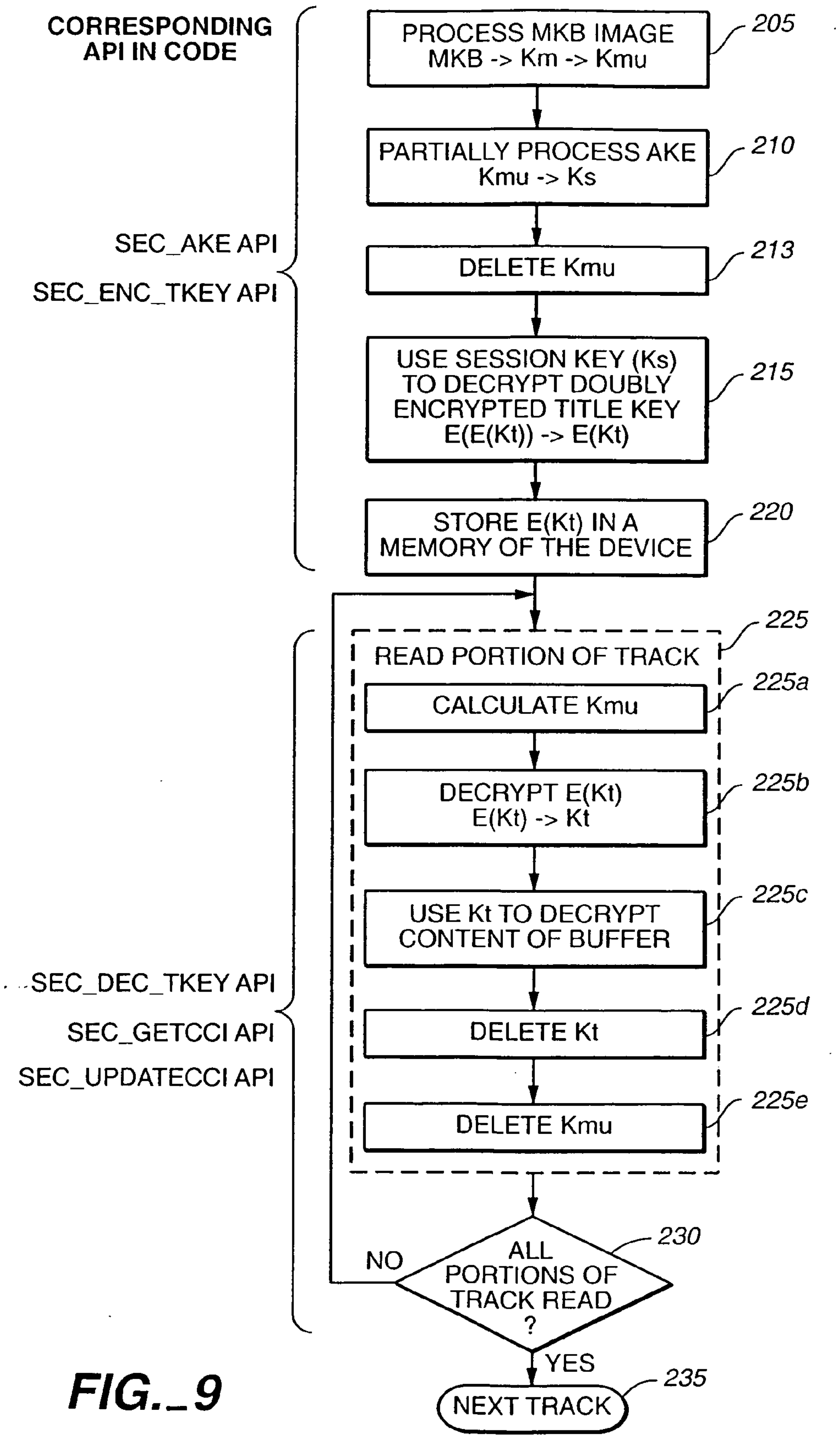

CORRESPONDING
API IN CODE
SEC_AKE API
SEC_ENC_TKEY API
PROCESS MKB IMAGE
MKB -> Km -> Kmu
205
PARTIALLY PROCESS AKE
Kmu -> Ks
210
DELETE Kmu
213
USE SESSION KEY (Ks)
TO DECRYPT DOUBLY
ENCRYPTED TITLE KEY
E(E(Kt)) -> E(Kt)
215
STORE E(Kt) IN A
MEMORY OF THE DEVICE
220
225
READ PORTION OF TRACK
CALCULATE Kmu
225a
DECRYPT E(Kt)
E(Kt) -> Kt
225b
USE Kt TO DECRYPT
CONTENT OF BUFFER
225c
DELETE Kt
225d
DELETE Kmu
225e
SEC_DEC_TKEY API
SEC_GETCCI API
SEC_UPDATECCI API
NO
ALL
PORTIONS OF
TRACK READ
?
230
YES
NEXT TRACK
235

FIG._9

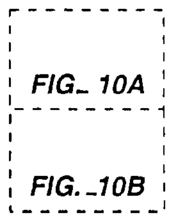
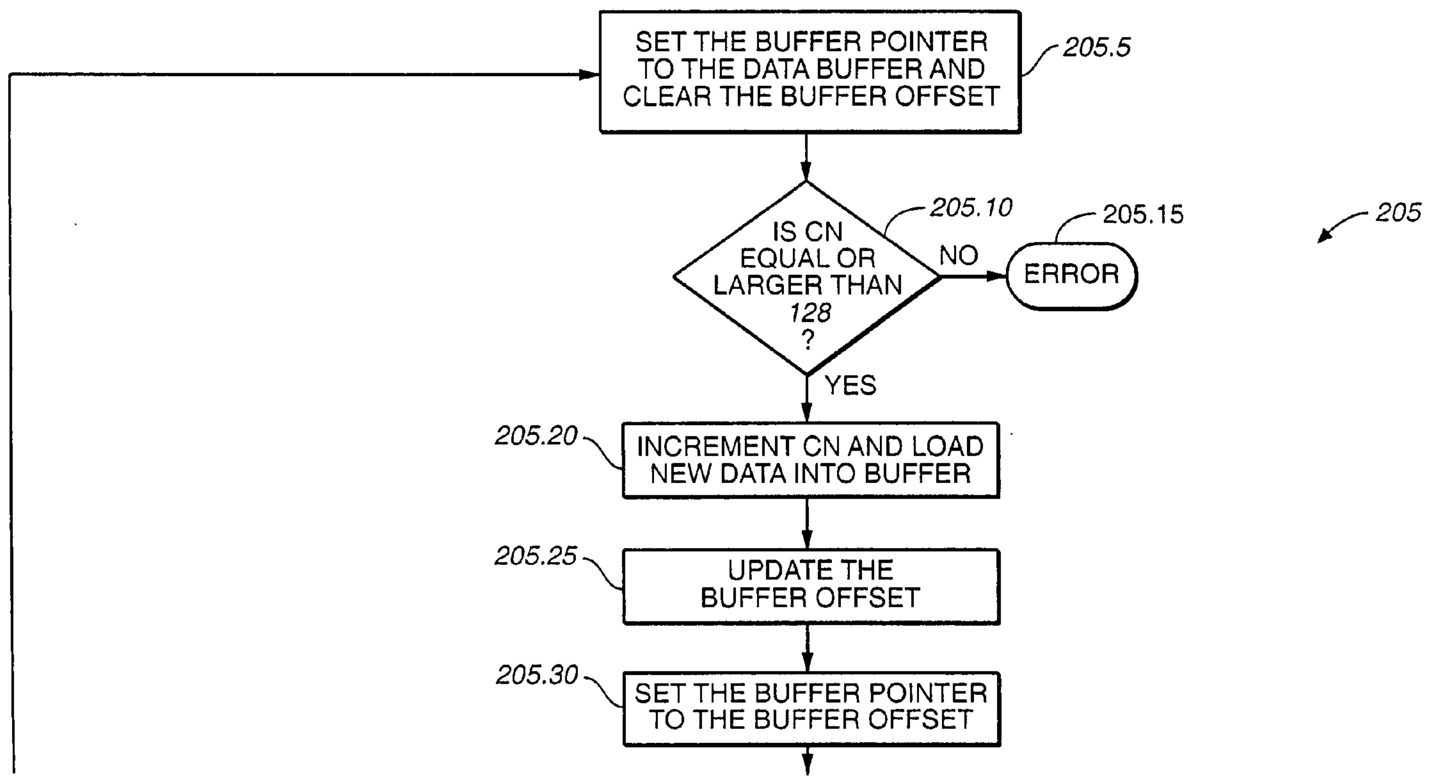


FIG._10

FIG._10A


205
SET THE BUFFER POINTER TO THE DATA BUFFER AND CLEAR THE BUFFER OFFSET
205.5
IS CN EQUAL OR LARGER THAN 128 ?
205.10
NO
ERROR
205.15
YES
205.20
INCREMENT CN AND LOAD NEW DATA INTO BUFFER
205.25
UPDATE THE BUFFER OFFSET
205.30
SET THE BUFFER POINTER TO THE BUFFER OFFSET

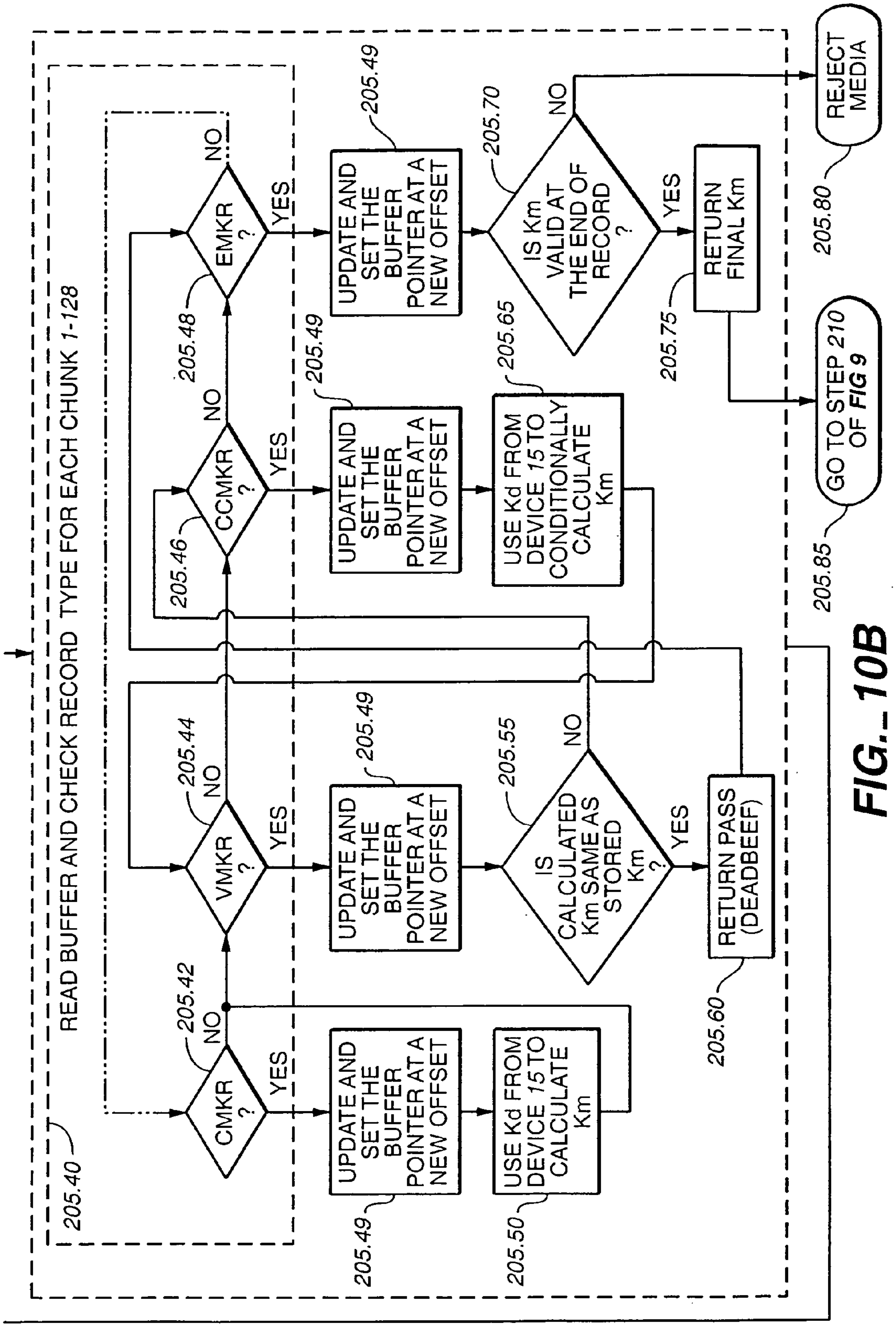

READ BUFFER AND CHECK RECORD TYPE FOR EACH CHUNK 1-128
205.40
CMKR ?
205.42
NO
YES
UPDATE AND SET THE BUFFER POINTER AT A NEW OFFSET
205.49
USE Kd FROM DEVICE 15 TO CALCULATE Km
205.50
VMKR ?
205.44
NO
YES
UPDATE AND SET THE BUFFER POINTER AT A NEW OFFSET
205.49
IS CALCULATED Km SAME AS STORED Km ?
205.55
NO
YES
RETURN PASS (DEADBEEF)
205.60
CCMKR ?
205.46
NO
YES
UPDATE AND SET THE BUFFER POINTER AT A NEW OFFSET
205.49
USE Kd FROM DEVICE 15 TO CONDITIONALLY CALCULATE Km
205.65
EMKR ?
205.48
NO
YES
UPDATE AND SET THE BUFFER POINTER AT A NEW OFFSET
205.49
IS Km VALID AT THE END OF RECORD ?
205.70
NO
YES
RETURN FINAL Km
205.75
REJECT MEDIA
205.80
GO TO STEP 210 OF FIG 9
205.85

FIG._10B

SYSTEM, METHOD, AND DEVICE FOR PLAYING BACK RECORDED AUDIO, VIDEO OR OTHER CONTENT FROM NON-VOLATILE MEMORY CARDS, COMPACT DISKS, OR OTHER MEDIA

CROSS REFERENCE TO RELATED APPLICATIONS

[0001] This application is a continuation of U.S. patent application Ser. No. 10/006,554 filed Dec. 6, 2001 to Farshid Sabet-Sharghi et al., entitled "System, Method, and Device for Playing Back Recorded Audio, Video or Other Content From Non-Volatile Memory Cards, Compact Disks, or Other Media", which claims the benefit of U.S. Provisional Patent Application Ser. No. 60/251,731 filed Dec. 7, 2000 to Farshid Sabet-Sharghi et al., entitled "Secure Software System for Playing Back Recorded Audio, Video or Other Content From Non-Volatile Memory Cards, Compact Disks or Other Media".

[0002] This application is related to U.S. Pat. No. 7,227,952, issued on Jun. 5, 2007 to Bahman Qawami et al. entitled "System, Method, and Device for Playing Back Recorded Audio, Video or Other Content From Non-Volatile Memory Cards, Compact Disks or Other Media" and application Ser. No. 11/809,222 filed May 31, 2007 to Bahman Qawami et al., entitled "System, Method, and Device for Playing Back Recorded Audio, Video or Other Content From Non-Volatile Memory Cards, Compact Disks or Other Media".

[0003] These applications are hereby incorporated by this reference in their entirety.

[0004] Source code is submitted on a compact disc according to 37 CFR 1.52 as an appendix containing the following files, each of which is hereby incorporated by this reference in its entirety: Sd_security\Sd_oem\Makefile, Nov. 05, 2001, 2KB; Sd_security\Sd_oem\Readme, Nov. 05, 2001, 3KB; Sd_security\Sd_oem\Sd_oem.c, Nov. 05, 2001, 6KB; Sd_security\Sd_oem\Sd_oem.h, Nov. 05, 2001, 1KB; Sd_security\Sd_oem\Sd_oem.inc, Nov. 05, 2001, 1KB; Sd_security\Sd_oem\Sdtypes.h, Nov. 05, 2001, 3KB; Sd_security\Sd_oem\vssver.scc, Nov. 05, 2001, 1KB; Sd_security\Security\Tstsampl\Dotest.c, Nov. 05, 2001, 8KB; Sd_security\Security\Tstsampl\Makefile, Nov. 05, 2001, 4KB; Sd_security\Security\Tstsampl\Readme, Nov. 05, 2001, 3KB; Sd_security\Security\Tstsampl\Regress.c, Nov. 05, 2001, 26 KB; Sd_security\Security\Tstsampl\Sdls.c, Nov. 05, 2001, 10KB; Sd_security\Security\Tstsampl\Sdrm.c, Nov. 05, 2001, 5KB; Sd_security\Security\Tstsampl\Securmmc.c, Nov. 05, 2001, 6KB; Sd_security\Security\Tstsampl\Tstsampl.inc, Nov. 05, 2001, 1KB; Sd_security\Security\Tstsampl\vssver.scc, Nov. 05, 2001, 1KB; Sd_security\Security\Err.h, Nov. 05, 2001, 1KB; Sd_security\Security\Fsentry.c, Nov. 05, 2001, 7KB; Sd_security\Security\keyInfo.h, Nov. 05, 2001, 84KB; Sd_security\Security\Makefile, Nov. 05, 2001, 3KB; Sd_security\Security\Readme, Nov. 05, 2001, 4KB; Sd_security\Security\Scdrv.c, Nov. 05, 2001, 29 KB; Sd_security\Security\Scdrv.h, Nov. 05, 2001, 5KB; Sd_security\Security\Scfs.c, Nov. 05, 2001, 13KB; Sd_security\Security\Scfs.h, Nov. 05, 2001, 4KB; Sd_security\Security\Sdsec.h, Nov. 05, 2001, 5KB; Sd_security\Security\Sdsys.c, Nov. 05, 2001, 2KB; Sd_security\Security\Security.c, Nov. 05, 2001, 64KB; Sd_security\Security\Smanager.c, Nov. 05, 2001, 7KB; Sd_security\Security\Smanager.h, Nov. 05, 2001, 2KB; Sd_security\Security\Ssmapi.c, Nov. 05, 2001, 3KB; Sd_security\Security\vssver.scc, Nov. 05, 2001, 1KB; Sdaudlib\HostFunc.c, Nov. 05, 2001, 3KB; Sdaudlib\Inpoutp.c, Nov. 05, 2001, 1KB; Sdaudlib\mssccprj.scc, Nov. 05, 2001, 1KB; Sdaudlib\plmInfo.h, Nov. 05, 2001, 16KB; Sdaudlib\Sd_plm.h, Nov. 05, 2001, 5KB; Sdaudlib\Sd_tkm.h, Nov. 05, 2001, 4KB; Sdaudlib\Sd_types.h, Nov. 05, 2001, 2KB; Sdaudlib\Sdapi.h, Nov. 05, 2001, 2KB; Sdaudlib\Sdaudapi.c, Nov. 05, 2001, 91KB; Sdaudlib\Sdaudapi.h, Nov. 05, 2001, 8KB; Sdaudlib\Sdaudlib.dsp, Nov. 05, 2001, 4KB; Sdaudlib\Sdaudlib.dsw, Nov. 05, 2001, 1KB; Sdaudlib\vssver.scc, Nov. 05, 2001, 1KB.

BACKGROUND

[0005] 1. Field of the Invention

[0006] This invention relates generally and specifically to secure playback of digital audio, video or other content from memory cards, compacts disks or other media.

[0007] 2. Related Art

[0008] The potential of electronic distribution of copyrighted music over the Internet, by other communication systems or through retail kiosks, is being limited by concerns about unauthorized copying of the music. This is also the case for other audio, as well as video, content. The content is typically provided to the ultimate customer in encrypted form, and the customer records the encrypted content files onto some storage media, such as a personal computer memory, a memory of a portable playing device, a writable compact disk (CD) or a non-volatile memory card. Providers of the content would like to eliminate the possibility of unauthorized copying of the content but have to be satisfied with taking steps that minimize the amount of copying that occurs. This includes providing protection of the content on the recording media. The protection of content stored on non-volatile memory cards is described herein, as specific examples, but the same content protection techniques can be applied to compact disks or other recordable media.

[0009] There are several commercially available non-volatile memory cards that are suitable for use as the content data storage media. One is the CompactFlash (CF) card, another is the MultiMediaCard (MMC), and yet another is the Secure Digital (SD) memory card that is closely related to the MMC card. All three, and others, are available in various storage capacities from SanDisk Corporation of Milpitas, Calif., assignee of the present application. The physical and electrical specifications for the MMC are given in "The MultiMediaCard System Specification" that is updated and published from time-to-time by the MultiMediaCard Association ("MMCA") of Cupertino, Calif. Versions 2.11 and 2.2 of that Specification, dated June 1999 and January 2000, respectively, are expressly incorporated herein by this reference. The MMC products are also described in a "MultiMediaCard Product Manual," Revision **2**, dated April 2000, published by SanDisk corporation, which Manual is expressly incorporated herein by this reference. Certain aspects of the electrical operation of the MMC products are also described in patent applications of Thomas N. Toombs and Micky Holtzman, Ser. No. 09/185,649, now U.S. Pat. Nos. 6,279,114, and 09/186,064, now U.S. Pat. No. 6,901,457, both filed Nov. 4, 1998, and assigned to SanDisk Corporation. The physical card structure and a method of manufacturing it are described in U.S. Pat. No. 6,040,622, assigned to SanDisk Corporation. These patents are also expressly incorporated herein by this reference.

[0010] The newer SD Card is similar to the MMC card, having the same in plan view. A primary difference between them is that the SD Card includes additional data contacts in order to enable faster data transfer between the card and a host. The other contacts of the SD Card are the same as those of the MMC card in order that sockets designed to accept the SD Card will also accept the MMC card. The electrical interface with the SD card is further made to be, for the most part, backward compatible with the MMC product described in version 2.11 of its specification referenced above, in order that few changes to the operation of the host need be made in order to accommodate both types of card. The electrical interface of the SD Card, and its operation, are described in co-pending patent application Ser. No. 09/641,023, filed Aug. 17, 2000, now U.S. Pat. No. 6,820,148, which application is incorporated herein in its entirety by this reference.

SUMMARY OF THE INVENTION

[0011] Encrypted content is difficult to access, and memory cards or compact disks with encrypted content each have specific structures that require specific commands and routines to access encrypted and unencrypted content. The software of the present invention is a simple solution that any original equipment manufacturer (OEM) can install and run on a myriad of different types of devices having a myriad of different types of microprocessors. These devices range from personal computers to portable devices to car stereos, and include any device from which one would like to access content that may be encrypted. The portable devices may be portable audio players or cell phones or portable organizers or generally any microprocessor controlled portable device. The storage media may be flash memory or any type of recordable disk. The devices may have a simple or powerful microprocessor with a small or large amount of memory. The software utilizes and requires only a small buffer for encryption purposes and is designed to run efficiently even in environments with limited processing power and memory. It can be run by any type of general purpose microprocessor, or special purpose microprocessors such as a DSP, or an ASIC. Additionally, computationally demanding portions of the software, such as the encryption and decryption (security) engine may be executed by the DSP, while other portions of the software may be executed by another microprocessor or an ASIC.

[0012] The software has audio, video and image interfaces to receive commands for each of the respective types of files. These interfaces can organize playback and recording, including managing playlists and other convenient features. Thus, whatever the device, it need only issue a command to an interface and the software will take care of reading or writing data from the secure media, and decoding and decompressing the data from any well known audio, video or image file formats within the audio video or image engines.

[0013] The encryption and decryption takes place in an isolated module that is very difficult to access and thus isolated from any attempts from unauthorized persons wishing to access encryption keys in order to copy the files from the media or the device. Content is only decrypted in small portions, and a method of dynamic key generation and deletion minimizes exposure of decrypted keys.

BRIEF DESCRIPTION OF THE FIGURES

[0014] FIG. **1** is an illustration of the devices used to read and write information on a secure media.

[0015] FIG. **2** is a schematic diagram of a device used to access the secure media.

[0016] FIG. **3**A is an abstract illustration of the layers of the secure media.

[0017] FIG. **3**B is an illustration of the physical and logical structure of the memory cells of the secure media.

[0018] FIG. **3**C is an illustration of the track structure and the component parts of a track.

[0019] FIG. **4** is an illustration of a media key block (MKB) image broken into its component chunks.

[0020] FIG. **5** is an illustration of a portion of the authentication and decryption process.

[0021] FIG. **6** is an illustration of the authentication and encryption process.

[0022] FIG. **7** is a schematic of the authentication key exchange process shown in FIG. **6**.

[0023] FIG. **8**A is a block diagram of the software of the present invention.

[0024] FIG. **8**B is a block diagram illustrating the modules of the software of the present invention.

[0025] FIG. **8**C is a flow chart overview of the pre-play and play process according to the present invention, with the related API modules/calls shown.

[0026] FIG. **8**D is an expanded flow chart of audio content initialization phase in FIG. **8**C.

[0027] FIG. **8**E is an illustration of information blocks created during the pre-play process **805** of FIG. **8**C.

[0028] FIG. **9** is a flow chart overview of the playback of an audio track according to the present invention.

[0029] FIG. **10** is a flow chart of the processing of an MKB image seen in FIG. **4**, a step of FIG. **9**.

DETAILED DESCRIPTION OF THE INVENTION

[0030] Encrypted content is difficult to access, and memory cards or compact disks with encrypted content each have specific structures that require specific commands and routines to access encrypted and unencrypted content. The software of the present invention is a simple solution that any original equipment manufacturer (OEM) can install and run on a myriad of different types of devices having a myriad of different types of microprocessors. These devices range from personal computers to portable devices to car stereos, and include any device from which one would like to access content that may be encrypted. The portable devices may be portable audio players or cell phones or portable organizers or generally any microprocessor controlled portable device. The storage media may be flash memory or any type of recordable disk. The devices may have a simple or powerful microprocessor with a small or large amount of memory. The software utilizes and requires only a small buffer for encryption purposes and is designed to run efficiently even in environments with limited processing power and memory. It can be run by any type of general purpose microprocessor, special purpose microprocessors such as a DSP, or an ASIC. Additionally, computationally demanding portions of the software, such as the encryption and decryption (security) engine may be executed by the DSP while other portions of the software may be executed by another microprocessor or an ASIC. The source code referred to in the Cross Reference section forms a part of this application, and is hereby expressly incorporated in its entirety by this reference.

[0031] With reference to FIG. **1**, an exemplary system is described in which content protection is applied to audio content such as music. A host computer device **11** may be a personal computer (PC), as shown, a kiosk located in a retail store to distribute music or other content, or the like. An SD memory card **13** is used in this example to store music. The card **13** is insertable into a utilization device, in this case a portable device (PD) **15** that operates from batteries to play the music or other audio content recorded on the card **13** through personal earphones. The music may be stored on the card **13** when inserted into the device **15** by connecting the device **15** to the host **11**, such as through a computer universal serial bus (USB) connection **17**. Alternatively, if the player device **15** is not provided with the capability of recording content onto the card **13**, or if it is otherwise desirable, a card writer/reader **19** may be connected to the computer through a USB connection **21**, and the card **13** inserted into it for recording music on the card. The card **13** is then removed from the writer/reader **19** and inserted into the portable device **15** to play the audio content recorded on the card. The host **11** is termed a licensed compliant module (LCM) when it includes the software necessary to write to and read from the card **13** content data in accordance with the security and authentication protocols of the 4C Entity and the SD Group.

[0032] The electronic system within the example portable utilization device **15** is illustrated in FIG. **2**. Operably connected together through a bus **23** are a computing unit (MCU) **25**, preferably with some non-volatile flash memory **25**A, system memory **27**, which is preferably a high speed random access memory (RAM), and interface circuits **29** for connecting with the memory card **13**. The USB connection **17** is also optionally provided to the MCU **25**. A digital signal processor (DSP) **31** is also included, when needed, for decompressing and/or decrypting content data, such as audio or video data, that is stored in a compressed and/or encrypted form. DSP **31** also has its own RAM memory **31**A included as part of the processor. DSP **31** may or may not be included. Furthermore, if a DSP processor is included, it may perform the functionality of MCU **25**, and thus MCU **25** may therefore be eliminated. Read only memory (ROM) **32** can store part or all of the software of the invention. Software instructions and data in ROM **32** can be executed or read directly from ROM **32** or first shadowed into any RAM memory included in the circuitry of the device.

[0033] Specifications for the protection of content on recordable media have been jointly established by Intel Corporation, International Business Machines Corporation, Matsushita Electric Industrial Co., Ltd. and Toshiba Corporation (4C Entity). Particularly relevant here are the following three publications of the 4C Entity, which are expressly incorporated herein by this reference: "Content Protection for Recordable Media Specification, Introduction and Common Cryptographic Elements," Revision 0.94, October, 2000, "Content Protection for Recordable Media Specification, SD Memory Card Book," Revision 0.95, May, 2001, and "C2 Block Cipher Specification," Revision 0.9, January, 2000, and "Content Protection for Recordable MediaSpecification, DVD Book," Revision 0.95, May, 2001. Additional detailed specifications for implementing these 4C Entity specifications on SD memory cards have been established by Matsushita Electric Industrial Co., Ltd. (MEI), SanDisk Corporation and Toshiba Corporation (SD Group).

[0034] Referring to FIG. **3**A, a memory card **13** can be thought of as having four distinct layers. Such layers may also be present in other types of secure media.

[0035] At its most basic level, data is stored in memory cells arranged in clusters on the physical layer **13***d* of memory card **13**. The data is encrypted or secure if it is copyrighted material or otherwise worthy of encryption. Keys used to encrypt and decrypt the secure content are also encrypted and stored in a secure area of the physical layer.

[0036] The software of the present invention runs within a device to allow the device to store and retrieve encrypted information without the manufacturer (OEM) having to program very specific instructions to access the memory cells containing encrypted data and keys. It contains methods of sending the encrypted data to the device, decrypting the data within the device, and decompressing and playing audio, video and image files upon requests from the device. In short, a device need only send a command such as "play track." The software will accept the command, retrieve the encrypted data stored in the memory cells, retrieve the encrypted keys, organize and decrypt the data, decompress and format it, and play the song back.

[0037] Logical layer **13***c* contains the organizational structure for the memory cells and clusters of physical layer **13***d*. The two layers **13***c* and **13***d* contain and logically structure the memory of card **13**. As card **13** is a secure card, security layer **13***b* controls and limits access to the secure data housed in the layers below.

[0038] Application layer **13***a* is the part of memory card **13** that communicates with a device accessing the content stored in the card. It does this through a device interface or contacts **39**. Memory card **13** preferably includes a controller that manages the operation of the card and functionality of the application layer **13** together with control of all layers **13***a-d* of the card.

[0039] The physical and logical structure of a recording media, the SD card **13**, according to the foregoing specifications, and corresponding to layers **13***c* and **13***d* of FIG. **3**A, is illustrated in FIG. **3**B. The card includes an array of memory cells **33** and a memory controller **35**. User data, commands and status signals are communicated between the controller **35** and the memory array **33** over a circuit **37**. The controller **35** communicates with a host device connected to a socket in which the card is inserted through a series of electrical contacts **39** on the card.

[0040] The memory cells of the array **33** are divided into the four non-overlapping areas of cells that are individually designated to store different types of data. A largest storage capacity area **41** is designated to store user data, in this case, encrypted audio, video or other data. The user data may or may not also include unencrypted data. A system area **43** of the memory stores a 64-bit media identifier (IDmedia) of the card manufacturer, and 16 media key blocks (MKB) provided by the 4C Entity, each MKB having a maximum size of 4k bytes, all being pre-recorded by the card manufacturer. One of the 16 MKBs is specified for use with audio user data, another for use with video user data, another for use of image data, and so on. The system area **43** is a write-protected area that is accessible for reading from outside of the card. A hidden area **45** carries 16 pre-recorded media unique keys (Kmu) corresponding to the 16 distinct media key blocks (MKB) stored in the system area **43**. The hidden area **45** is a write-protected area that is accessible only by the memory card itself. A protected area **47** is a read/write area that is accessible only after a successful explicit mutual authentication has occurred. Randomly picked title keys (Kt) and copy control information (CCI) are stored in the protected area **47** in an encrypted form. Each piece (file) of content stored in the user data area **41** is encrypted with a unique title key that is also stored in an encrypted form in the protected area **47**. The title keys and CCI stored in the protected area **47** are concatenated and encrypted together by the media unique key, which is unique for each memory card and stored in its hidden area **45**.

[0041] The file system of the user data area **41** is typically an ordinary FAT file system. The FAT system describes what memory clusters make up what tracks and the various subcomponents of the tracks. Audio or video tracks within user data area **41** may comprise multiple files as illustrated in FIG. 3C. Audio files are referred to as audio objects (AOB's) and picture files are referred to a picture objects (POB's). A track may comprise both. Track **300**, for example is composed of AOB **304** and AOB **308**, and track **302** is composed of AOB **306** and the last track xxx is composed of AOB xxx and POB xxx. Each AOB or POB is also broken down into sub components. AOB **304** is shown broken down into AOB blocks **312**, which are further broken down into AOB elements **316** with header **318**. Each element may be stored in one or more memory clusters of memory card **13**. AOB elements are divided into the lowest component level, AOB frames **320**. Depending on the encoding and compression of the content, each two seconds may comprise a varying number of frames. A time search table (TMSRT) has information about the number of frames and data size which corresponds to "every two seconds" of playback. This information is used when an audio or video track and the component AOB's elements and frames are accessed for fast forward and rewind. Also, as will be discussed later with regard to FIGS. **9** and **10**, at title key (Kt) is an a decrypted state only for the time it takes to access this "every two seconds" of content, although anywhere from less than one to ten seconds of content may be decrypted at a time. For further detail, please refer to the CPRM Specification, SD Memory Card Book, which was previously incorporated by reference.

[0042] The media key block (MKB), as stored in the system area **43** of the card memory, contains a sequence of contiguous records, one such record being illustrated in FIG. **4**. The entire MKB image **49** is 64 Kbytes. It is broken into 128 chunks of 512 bytes, and chunk **1**, which contains all or part of the first record, and is labeled MKB chunk **50** in the figure, is enlarged to show its component parts. Chunk **50** may also contain multiple records. A first field **51** contains the record type, a second field **53** the total length of the record, and the remaining field **55** the key itself. The data in the record type and length fields **51** and **53** are not encrypted. Each record of the MKB is a multiple of 4 bytes in total length. As illustrated by a block **57** of FIG. **5**, the MKB key records are decrypted by device keys stored in the portable device (PD), licensed compliant module (LCM) or other device that utilizes a memory card for reading or programming content data stored on it. Device keys Kd**1**, Kd**2**, Kd**3** . . . are written into a memory of the utilization device, such as non-volatile flash memory within the MCU **25** of the portable audio player of FIG. **2**, by the manufacturer of the device. The device keys are provided to device manufacturers by the 4C Entity, and are maintained in confidence. The number of device keys which are stored in a given utilization device depends upon the type of the device.

[0043] The utilization device (PD, LCM or other device) which performs the processing of FIG. **5** calculates the media key Km as part of the decryption of block **57**, which is discussed in further detail with regard to FIGS. **9** and **10**. Each record (FIG. **4**) of the MKB read from the system area of an inserted memory card is usually processed in this manner. After processing of the MKB is completed, the most recently calculated Km value is taken as the secret media key output of the block **57**. This media key Km and the media identifier IDmedia are combined by use of a C2 one-way function, as indicated by a block **59** of FIG. **5**, to produce the media unique key Kmu. Additional details of this processing may be had by reference to the 4C Entity publications referenced previously.

[0044] FIG. **6** illustrates all of the authentication and encryption processing that takes place when either recording audio content onto, or playing audio content from, a memory card **13** having the memory space allocation of FIG. **3**. Processing that takes place in a personal computer or other LCM **63** is illustrated for recording audio or other content onto the card **13**. Similarly, the processing of a portable audio or other utilization device **65** is shown for reading the recorded content from the card **13**. Included in both is the processing described with respect to FIG. **5**, the processing blocks **57** and **59** being part of the utilization device **65** and corresponding processing blocks **57'** and **59'** being part of the content recording system **63**.

[0045] As part of recording content, an arbitrarily assigned title key Kt is input at a line **67** for use by an encryption module **69** to encrypt one file (piece) of audio or other content input at line **71**. The encrypted file is then stored in the user data area **41** of the memory card **13**. In order to make the title key available for decrypting the recorded content, an encrypted version of the title key (Kt) is stored in the protected card memory area **47**, as previously described. An encrypted version of the title key (Kt) is also stored in either system memory **27**, RAM memory **25**A of MCU **25**, or RAM memory **31**A of DSP **31**. Storing the encrypted title key (Kt) in a memory of the device eliminates the need to access protected card memory area **47**. This is significant because it saves considerable time and processing capacity in comparison to accessing the protected area **47** for each read. This will be discussed later with regard to FIG. **9**. The title key Kt and copy control information CCI are encrypted by a series of encryption modules **75**, **77** and **79** in the LCM **63**, and a module **81** on the memory card **61**. The media unique key Kmu is used by the module **77**. An authentication key exchange (AKE) module **83** combines the media unique keys Kmu as calculated by the module **59'** and stored in the hidden area **45** of the card **61**, to generate a session key Ks that is used by each of the modules **79** and **81**. In order for the utilization device **65** to decrypt the recorded encrypted content, corresponding modules, indicated with the same reference numbers but with a prime (') added, are utilized to perform an inverse of the encryption process.

[0046] FIG. **7** illustrates a technique for accessing the protected area **47** of a memory card, utilizing an authentication and key exchange (AKE) challenge-response protocol between a card and some LCM or utilization device. When this authentication is successful, the card and the other module or device share a secure common session key Ks. Additional details of the forgoing processing and protocols may be had by reference to the 4C Entity publications previously identified.

[0047] FIGS. **8**A and **8**B illustrate an embodiment of a software system designed to run in a portable device or LCM in order to access information encrypted with the aforementioned processes. The SanDisk software, SW **100**, is a complete turn-key software solution that enables OEM music players and recorders to readily support secure media including the secure digital (SD) memory card. SW **100** shown within portable device **15** in order to access SD card **13**. SW **100** may also be installed in any licensed compliant module such as a personal computer. As seen in FIG. **8**A, at its highest level, SW**100** receives calls from device **15**, particularly a user interface of device **15**, retrieves encrypted content from the Secure Digital card **13**, and returns decrypted content to the device. Thus only simple calls are required to execute many complicated processes. The complicated processes of retrieving encrypted content stored in memory locations of card **13**, and then subsequently decrypting and formatting the content are handled by SW **100**.

[0048] Performing accesses to the authentication area of the SD Memory Card requires using secret device keys that OEMs must license from the 4C Entity, as mentioned previously. Protecting these key values and restricting their exposure within SDK SW **100** software layers is one of the central considerations in the software design. Isolation of these keys (and other resultant values such as session keys) within a single internal module while enabling a secure media such as the SD memory card device driver to perform operations dependent on these values is achieved in a robust and secure interface methodology. Once again, the SD memory card is used to illustrate the invention; however, the invention can be used on any secure media such as CDs or other secure memory that may be in a card or even in a remotely located storage device.

[0049] FIG. **8**B illustrates the layered structure of SW **100** in more detail. Audio interface **105**, video interface **110**, and imaging interface **115** are the points of communication to the device. These interfaces provide a single point of communication for the device and generally receive simple commands from the device so that the device does not have to get involved with the intricacies of getting encrypted data from a secure media, then decrypting and processing the data. All of these complex processes are handled by SW **100**. Interfaces **105**, **110**, and **115** also manage the arrangement of playback such as managing playlists and the correlation of images such as that of an artist with the songs of the artist or the various playlists. Application programming interface (API) **130**A resides within command dispatcher (CD) **130**. CD **130** and API **130**A receive commands from interfaces **105**, **110**, and **115**, relay information to the interfaces, and organize all of the processes that take place in the SW **100**—the processes of device **15** related to the playback and recording of content stored on the secure media, with all of the requisite encryption, decryption, and compression algorithms.

[0050] SD audio engine (SDAE) **140**, SD video engine (SDVE) **150**, and SD image engine (SDIE) **160** respectively process audio, video, and image content residing on the secure media, upon receipt of instructions from CD **130**. This means SDAE **140** can process any of the well known formats for audio, such as AAC, WMA, and MP3. Likewise, SDVE **150** can process any of the well known formats for video clips such as Windows media files or real networks files MPEGs or any other well known type of video files. Finally, SDIE **160** can process any well known type of image files such as TIF, GIF, JPEG, bitmaps, etc. Each interface has a secure API (SAPI) and a non-secure API (NSAPI). The content processed may or may not be encrypted. Encrypted content is accessed through SAPIs **140**A, **150**A, and **160**A. These SAPIs communicate with SanDisk security manager (SSM) **180**. All commands having to do with secure content are channeled through SSM **180**. Secure digital security engine (SDSE) **175**, which will be described later in further detail, handles all encryption and decryption processes. Keys used to authenticate the media and decrypt the content are contained within and handled exclusively by SDSE **175**. Unencrypted content residing on the card is accessed through NSAPI **140**B, **150**B, and **160**B. These NSAPIs communicate with a non-secure file interface (NSFI) **170** in order to access unencrypted content on the media.

[0051] In order to read or write data in the storage media, NSFI **170** and SDSE **175** communicate with device driver **190**. Device driver **190** in the example of the SD card manages and drives signals to and from the device interface **39**'s contacts of the SD card **13**. Device driver **190** will be tailored to the specific type of device interface **39** of various devices or media. In the case of a memory card device, driver **190** manages and drives signals to and from contacts located on device **15**. In the case of optical media, device driver **190** may manage and drive signals from various hardware components including an optical pick-up unit. Alternatively, in the case of a hard disk drive (hdd), device driver **190** will manage and drive the required hdd signals. Device driver **190** contains a secure device driver interface (SDDI) **190**A, and a non-secure device driver interface (NSDDI) **190**B. SDDI **190**A and NSDDI **190**B are isolated from each other within device driver **190**. SDDI **190**A communicates exclusively with SDSE **175**, while NSDDI **190**B communicates exclusively with NSFI **170**.

[0052] Device keys and other values central to the SD-Audio security scheme are housed within one restricted security software module, SD security engine (SDSE) **175**. All manipulation of these values is solely restricted to this module. Values are never passed in or out to software layers above SDSE **175**. All requests for the security services involving these keys are controlled and monitored by SSM **180** that shields this security module. Beneath the security module, the SD Memory Card device driver **190** carries out security accesses. Requests for these driver services are made via a private driver security interface, secure device driver interface (SDDI) **190**A, that is only known to the security module. SDSE **175** uses this interface **190**A to perform special security commands such as Get Media Key Block (MKB). Non-secure device driver interface (NSDDI) **190**B also utilizes device driver **190** to access any unencrypted files in user area **41** of card **13**.

[0053] The security of SW**100** architecture resides in the security of its keys. Secret "soft keys" are not stored in temporary secure areas for a long period of time, since this increases the possibility of compromising the keys and thus the encrypted content. SW **100** utilizes a scheme within SDSE **175** of dynamically generating the needed keys (or "soft keys") and deleting them when there is no immediate need for those specific keys.

[0054] Operation of SW **100** is now described in more detail. SW **100**, in particular, command dispatcher **130**/API **130**A have a number of API routines that can be called upon to perform a certain function. Although there are many routines, only 22 of the routines are accessed externally by device **15**. These routines are accessed by calls, which are also referred to as commands. In order to retrieve the content in memory card (or other media) **13**, the device need only send one of the 22 calls and the content will be retrieved, decrypted if necessary, and decoded. In the case of audio, for example, the device need only send the "play" call, and the music will start.

[0055] The following listed APIs allow applications to interface to device compliant with the Secure Digital (SD) standard. Although implementation of the invention is illustrated with the SD standard, the present invention can be used with many different standards.

TABLE 1

API Routines/Calls

Function of call/API routine	Call name/API routine (as seen in appended source code)
1. Initialize audio system	SdInitAudioSystem
2. Mount media (if necessary)	SdMountAudio
3. Unmount audio (if necessary)	SdUnMountAudio
4. Check the free space available	SdDriveFreeSpace
5. Eject media	SdEjectCard
6. Get number of playlists	SdGetPlayListCount
7. Get playlists	SdGetPlayLists
8. Get the track title (x)	SdGetTrackTitle
9. Get the track information	SdGetTrackInfo
10. Open the track	SdOpenTrack
11. Play the track	SdPlayTrack
12. Go to the next track	SdNextTrack
13. Stop playback	SdStopPlay
14. Pause playback	SdPauseTrack
15. Resume playback	SdResumeTrack
16. Reset playlist	SdResetPlayList
17. Fast forward/rewind playback (+/−)	SdForward
18. Add track index to playlist	SdAddTKItoPLM
19. Delete track index from playlist	SdDelTKItoPLM
20. Delete track index from track manager	SdDelTKItoTMG
21. Covert MP3 to internal playback format	SdConvertMP3ToSA1
22. Convert AAC to internal playback format	SdConvertAACToSA1

[0056] The principle API routines which can be called by device **15** will now be described in detail. Reference will be made to FIGS. **8**A-**8**E.

[0057] As can be seen in FIG. **8**C, there is a pre-play process **805** and a play process **810**. The blocks of FIG. **8**E are created during the processes of FIG. **8**C. The related API modules are executed when called upon by the user interface of device **15**. In the case of audio playback, calls to the modules are sent by the user interface of device **15** to the audio interface **105** as seen in FIG. **8**B. The primary calls/modules that carry out the functionality listed in the flowchart are indicated on the right. Many of these modules execute other internally and externally accessed modules, and the list is not meant to be exhaustive, but is meant to be a reference to the software code on compact disc that was previously incorporated by reference and forms a part of this application. For further detail please refer to the software code.

[0058] The device is first powered up in step **803**, after which the pre-play process **805** commences. The pre-play process has two major phases: a power up initialization phase **805**.**10**, and an audio content initialization phase **805**.**20**. Audio content initialization phase will be described in further detail with regard to FIG. **8**D.

[0059] Generally speaking, in pre-play process **805** the device and media are initialized and certain information from the media is read from the media and stored in a buffer of a RAM memory of device **15**. As seen previously in FIG. **2** this RAM memory may either be system memory **27**, RAM memory **31**A of DSP **31**, or RAM memory **25**A of MCU **25**. In step **805**A SW **100** will get the drive number of the media. In some applications there may be more than one memory card or disk being accessed by device **15**. In this step it will get all the drive numbers in order that content on each of the drives can be properly accessed. This is accomplished with API routine SdInitAudioSystem, and can either be called upon by device **15** or can be internally called by SW **100** as part of a pre-play routine. SW **100** will then initialize SSM **180** and SDSE **175** within SW **100**. This is necessary before any encrypted keys and content from the media can be processed. SW **100** will also initialize the playlist and track manager in card **13**.

[0060] In step **805**B, SW **100** will initialize and verify the media. In the case of the SD card illustrated here, the MKB process of FIGS. **5**-**7** will be performed. This MKB process can also be executed during step **805**A, and if previously executed it will not be executed again in step **805**B. For further detail of this process please see FIGS. **5**-**7** and the 4C documents incorporated earlier. This process will also be discussed in greater detail with regard to FIG. **10**. In step **805**B, media information values will be copied from the card **13** and stored in locations of media information block **850** of FIG. **8**E in a RAM memory of device **13**. This is accomplished with API routine SdMountAudio, and can either be called upon by device **15** or can be internally called by SW **100** as part of a pre-play routine. Thus values for playlist general information (pTGInfo), the validity number of the media (SanDisk), the drive number (drivenum), the security system of the media (security) and the mounting status of the media (mounted) will be filled in their respective locations. These locations can then be subsequently read from the RAM of device **15** when called upon by any number of API calls without having to read them from card **13**.

[0061] After the power-up initialization **805**.**10** is completed, audio content initialization **805**.**20** commences. Generally speaking, during audio content initialization **805**.**20**, information specifying location and sequencing of the encrypted audio content of an individual track and multiple audio tracks (playlists) are copied from the card (or other media) **13** into a small buffer in a RAM of device **15**. This information, shown in blocks in FIG. **8**E, is thus quickly and easily accessible within the device and does not need to be constantly read from or updated to card **13** during the subsequent play process **810**.

[0062] Referring to FIG. **8**D, the audio content initialization phase **805**.**20** will be described in more detail. This phase creates a number of structures that act as a local roadmap or directory to the encrypted content on memory card (or other media) **13**.

[0063] In step **805**C, device **15** calls API module SdGetPlayListCount. This call, and all of the following calls, are generally sent from the software of a user interface of device **15** to one of the interface modules of SW**100**. In this illustration of audio playback the call is sent from the user interface to audio interface **105**. In the case of video playback, the call would be sent to video interface **110** and in the case of image reproduction, the call would be sent to imaging interface **115**. The call is then relayed to command dispatcher **130** which contains the API modules within API **130**A.

[0064] In step **805**D, SdGetPlayListCount will fill in the values for the Playlist Info block **860** by copying the information from card **13** into a RAM memory of device **15**. It will select the appropriate authorized drive(s) by referring to media info block **850**. The total number of playlists for all authorized drives will be copied into a RAM of device **15**.

[0065] In step **805**E, device **15** calls API module SdGetPlaylist.

[0066] In step **805**F, SdGetPlaylist will fill in the values for the playlist info block **860** by copying the information from card **13** into a RAM memory of device **15**. It will select the appropriate authorized drive where the playlist info resides by referring to media info block **850**. The total playback time of the selected or default playlist in milliseconds (pListTime), the number of tracks in the playlist (tracksInPlist), the index number corresponding to the current playlist (index), the playlist name string length (Length), and the playlist name (pListName) will be filled into their respective locations of Playlist Info block **860**.

[0067] In step **805**G device **15** calls API module SdGetTrackInfo.

[0068] In step **805**H, SdGetTrackInfo will fill in the values for the track information block **870** by copying the information from card **13** into a RAM of device **15**. It will select the appropriate authorized drive where the playlist info resides by referring to media info block **850**. It will select the tracks within each playlist by referring to the Playlist info block **860**. The total track time (trackTime) in milli-seconds including the related track units ("TKI's") in the track, the total track size in bytes (bytesize), including the related TKI's, the number of TKI's in the track (tkisInTrack), the track number (tracknum), the index corresponding the current track (index), and the track information from the media (trkInformation) will be filled into their respective locations.

[0069] In step **805**I device **15** calls API module SdOpenTrack.

[0070] In step **805**J, SdOpenTrack fills in some of the values for the Track Gen Info block **880** by copying the information from card **13** into a RAM of device **15**. It will select the appropriate drive by referring to media info block **850**, and it will select the tracks within the appropriate playlists and tracks by referring to Playlist Info block **860** and Track Info block **870**, the total playback time of the playlist in milliseconds (pListTime), the current playlist number (plistnum), the track number to be played (tracknum), the first AOB block for the track (firstAOB), and the current AOB being decrypted (currentAOB).

[0071] In step **805**K SdOpenTrack fills Track Index Info block **875** by copying the information from card **13** into a RAM of device **15**. It will select the authorized drive where the playlist info resides by referring to media info block **850** and playlist info block **860**, and it will select the proper tracks within the proper playlists by referring to Playlist info block **860** and Track Info block **870**.

[0072] After Track info block **870** is created, in step **805**L, SdOpenTrack will fill in the remaining values of Track General Info Block **880** by copying the information from card **13** into a RAM of device **15**. The following values will be filled into their respective locations of block **880**: a verification number for the media (SanDisk), an operation command (CMD), the audio format such as MP3, AAC, or WMA (audioformat), the codec sampling frequency (sampfreq), the application attribute, e.g., music, book image, etc. (appAtrib), the size of the audio object in bytes (size AOB), the last AOB block for the track(lastAOB), the total number of AOB's for the track (countAOB), the current position of sync position in AOB (syncword) also known as the header, the seek position within the AOB(seekposAOB), the elapsed time of the track in milliseconds (trkElapsedTime), the total play time of the track in milliseconds (trkTotalTime), the total track size in bytes including related TKI's (bytesize), the playtime of each element in milliseconds (elementplaytime), the forward seek time (fwTime), the time to the next track (fwNext), the number of the tracks in the playlist (tracksInPlist), the size of the current element (elementsize), the offset within the current element (element offset), the current elements in the AOB (currentelement), the total number of elements n the AOB (totalelements), and the file handle of the AOB (fdAOB). In a different embodiment of the invention, step **805**J will completely fill the values of Track General Info block **880** and step **805**K will be eliminated. Track Index Info block **875** is a subset of Track Gen Info block **880** and is designed to save space and processing time. It is meant to be referred to by the user interface of device **15** in the event that it is just browsing the information. Once the user interface has selected a particular track for playback, Track Gen info block **880** will be filled, including the subset of information contained in block **875**.

[0073] SdOpenTrack and can either be called upon by device **15** or can be internally called by SW **100** as part of a pre-play routine.

[0074] Having the blocks and the information of the blocks contained in a memory of the device is an advantage because if there is any failure in the playback process, it is not necessary to reset the media, i.e., perform steps **805**A or **805**B of power up initialization **805.10**. Also, it should normally not be necessary to read the information needed for playback from card **13**. The information in the blocks can be used to access the next content (audio, video etc.) frame because the information in the blocks **850**, **860**, **870**, **875**, and **880** is used as a pointer to the content contained in the next frame. The blocks of FIG. **8**E detail the location within memory card **13** of the files, elements and frames that make up and audio or video track are located within memory card **13**, as was earlier described with regard to FIG. **3**C.

[0075] The pre-play process of step **805** can be triggered by a number of calls (the numbers in parenthesis indicate the call in Table 1). As seen in FIG. **8**C, the external calls that will trigger audio content initialization **805.20** are: SdOpenTrack, SdGetPlaylistCount, SdGetPlaylist, and SdGetTrackInfo. SdOpenTrack (**10**) is internally called by SdNextTrack (**12**), SdStopPlay (**13**), and SdResetPlaylist (**16**). API modules SdGetPlaylistCount, SdGetPlaylist, and SdGetTrackInfo can also be called internally by SdOpenTrack. Generally, it will be called upon by device **15** for such device functions as displaying the track time, rewinding, fast forwarding, changing playlists, changing graphic user interface displays, or deleting tracks. Once the pre-play process **805** is complete, the play process **810** can commence.

[0076] In play process **810**, calls that will initiate, stop, or pause playback of one or more audio or video tracks are received by the audio interface **105**, video interface **110**, or imaging interface **115** of FIG. **8**B in step **810**A. These calls can be seen in FIG. **8**C next to play process **810** and are SdPlayTrack, SdNextTrack, SdStopPlay, SdPauseTrack, SdResumeTrack, SdResetPlayList, SdForward, SdAddTKI-toPLM, SdDelTKItoPLM, SdDelTKItoTMG, SdConvertMP3ToSA**1**, and SdConvertAACToSA**1**.

[0077] Regardless of how many API modules are executed, either internally or when called upon by the device, two primary modules will always be required in order to play an audio track. These modules are SdOpenTrack (**10**) and SdPlayTrack (**11**). SdOpenTrack (**10**) and SdPlayTrack (**11**) will read the information in Track General Info block **880** in order to access the encrypted content in the memory locations of clusters of memory card **13**.

[0078] SdOpenTrack (**10**) is internally called by SdNextTrack (**12**), SdStopPlay (**13**), and SdResetPlaylist (**16**). Generally, it will be called upon by device **15** for such device functions as displaying the track time, rewinding, fast forwarding, changing playlists, changing graphic user interface displays, or deleting tracks.

[0079] SdPlayTrack (**11**) is the core API that plays the music or video track. It is generally used by a device when the user wants to play the current track, the next track, or when he wants to rewind or fast forward within a track. It is called upon by other API's such as SdNextTrack (**12**) SdResumeTrack (**15**) and SdForward (**17**). SdPlayTrack finds the AOB for the selected track, checks the audio format (MP3, AAC, or WMA etc.) and decodes the track.

[0080] Referring to FIGS. **8**B, **9**, and **10**, playback of an encrypted track, step **810**B of FIG. **8**C, will now be described.

[0081] If encrypted content is desired, then commands are issued to/from device **15** and SW **100** which require the OEM's 4C-licensed device keys to be used. All processing of these keys is solely limited to the SDSE **175** module which is housed beneath the SSM **180**. If non secure or non-encrypted content is requested, NSFI **170** and NSAPI's **140**B, **150**B, and **160**B and NSDD **190**B will access the content.

[0082] When SSM **180** receives a request for security services, it carries it out by passing the command request packet to the process_security function within SDSE **175**. Key values are never contained within the request packets or exposed at software layers above SDSE **175**.

[0083] When needed internally by SDSE **175**, device keys are retrieved via a function call into an OEM-supplied library. The library of SDSE **175**, security.lib, contains the following APIs designed to reduce the time that a decrypted key resides in the secure area of the system:

[0084] 1) SEC_AKE API;

[0085] 2) SEC_ENC_TKEY API;

[0086] 3) SEC_DEC_TKEY API;

[0087] 4) SEC_GETCCI API;

[0088] 5) SEC_UPDATECCI API.

[0089] The functionality and the structure of SW **100** are described in the text of this application and more specifically, the functionality of APIs **1**-**5** above are shown within the flowchart of FIG. **9**. The APIs are shown next to the corresponding functions that they implement. Further detail of the implementation of these APIs, as well as all of SW **100**, can be seen in the source code that is submitted in an appendix of this application.

[0090] Once obtained, the device key is combined with the Media Key Block (MKB) from the SD Memory Card to form the "media key." This value is kept within SDSE **175** for use in processing subsequent requests. Note, however, the "unique media key" (Kmu) is never retained inside SDSE **175**. This value, which forms the basis for all security accesses, is always calculated on a real-time basis (and never cached) as an extra security precaution. Detailed description of the processing of the keys within SDSE **175** follows.

[0091] The encryption process is in general terms designed to stop unauthorized copying of the content located on the secure media. There are many aspects of the invention that achieve this. First, an entire file, for example, a song, is never decrypted at once and stored into memory where it may be vulnerable. The portable device allocates a buffer and SDSE **175** reads chunks of encrypted content at a time, decrypts it, and then writes over the same buffer over and over again until the end of the file.

[0092] As was seen in FIGS. **6** and **7**, the media unique key (Kmu) and title key (Kt) are the keys finally used to decrypt the content. There are many ways to protect the title key. One is to store the keys in a very secure area of device **15**, another is to read the title key from the protected area **47** of card **13** each time the encrypted buffer is read and decrypted. FIG. **9** is a flow chart depicting the preferred method.

[0093] Returning to FIG. **9**, in step **205**, an MKB image, which, as seen in FIG. **4**, is **64** kilobytes, is read to process the media key (Km), as seen in FIG. **6**, to yield the media unique key (Kmu). This step is further detailed in FIG. **10** which will be described later. After mutual authentication of the device and the media is complete in step **205**, the AKE process is undergone to yield a session key (Ks) that can only be used during that session (as long as the device is turned on or is in an active state) in step **210**. The AKE process can be seen by referring once again to FIG. **6**. In step **213**, the media unique key (Kmu) is deleted. In step **215**, the session key (Ks) is used to decrypt the doubly encrypted title key E(E(Kt)) stored in protected area **47** of memory card **13**. The result is a singly encrypted title key (E(Kt)). In step **220**, this encrypted title key (E(Kt)) is stored in a memory of the device **15**. The (E(Kt)) may be stored in system memory **27**, RAM memory **25**A of MCU **25**, or RAM memory **31**A of DSP **31**. The title key Kt is specific for each title, referred to as a track in the realm of audio and on FIG. **9** used to illustrate the invention. Each track may be made of multiple files, for example, in the case of a long classical song. For large video clips, a title may comprise many files. Thus, for all subsequent reading and decryption of the encrypted content of the track, the title key need not be retrieved from the memory card because it is stored in a local memory, and precious time and computing resources can be saved, while at the same time, the title key remains encrypted for security purposes.

[0094] In step **225**, a portion of the track is played back. This portion may be in any of the files that comprise the track. In step **225***a*, the media unique key (Kmu) is calculated once again. In step **225***b*, the encrypted title key stored in local memory is decrypted. Then, in step **225***c*, the title key is used to decrypt the content from the buffer of device **15** containing content from the user area **41** of card memory card **13**. Immediately after the buffer is decrypted, the title key is deleted in step **225***d* and the media unique key is deleted in step **225***e*. The order of steps **225***d* and **225***e* is not important, but it is important that both keys are only exposed for the time it takes to read a portion of the track. This portion may be anywhere from a fraction of a second of playback (decrypted, decompressed, and decoded) content, audio or otherwise, to about ten seconds. Preferably it is two seconds. The time it takes to read the portion is dependent on many factors including the processing speed and the buffer size of the device. As discussed previously, SW **100** can be executed by either the MCU **25** or DSP **31** and stored in any of the memory **27**, **25**A, **31**A or **32** of device **15**, thus, the processing times can vary. This is repeated until all portions of the track are read as seen in step **230**. Once all portions have been read the system can move on to the next track, as shown in step **235**, if playback is to continue. This may be the case, for example, if the user has chosen to play an entire playlist.

[0095] When the all portions of track have been read and the reading of the next track is to commence, the process will begin again at step **215** and will retrieve the next doubly encrypted title key from the protected area **47** of card **13**. This is generally the case if the user has set the device in motion to play an entire playlist that includes multiple tracks. If the session is closed (i.e., device **15** has been turned on or off), then a new session key will have to be generated and the process will initiate at step **210**. If memory card is removed or freshly inserted, the device and media will have to be re-authenticated and the process will begin again at step **205** in order to read a track.

[0096] FIG. **10** describes the operation of processing the Media Key Block, step **205** of FIG. **9** described above. As was seen in FIG. **4**, an MKB image **49** is 64 Kbytes in length. Reading the entire image **49** at once to calculate the MKB would be inefficient, requiring a large RAM and long processing times. The present system reduces RAM requirements and decreases processing time. The MKB image **49** is divided into chunks **1** through **128**. Each chunk is **512** bytes and may contain one of four different types of records of the MKB: the verify media key record (VMKR) known as 0×81; the calculate media key record (CMKR) known as 0×01; the conditionally calculate media key record (CCMKR) known as 0×82; or the end media key record (EMKR) known as 0×02. These records are described in the Content Protection for Recordable Media (CPRM) Specification of the 4C Entity, referenced above.

[0097] In this example, the chunk length and the buffer length are the same. However, the buffer length and chunk length can both range from 256 bytes to 4096 bytes. Each record is examined to perform specific operations based on the record type and certain data will be saved for later to obtain the Media Key. The record length is added to the total length of the buffer offset every time a record is identified. The chunk number is calculated by dividing the total length with the chunk length. The chunk number is the index to the Media Key Block of a selected chunk data. The remainder of the total length is the offset to the selected chunk data. The row and column are used to figure out where the encrypted media key and the conditional encrypted media key are. Those encrypted keys are saved and the decryption C2 cipher in Electronic Codebook Mode algorithm is performed to obtain the Media Key. This Media Key is then verified for a correct final Media Key (Km).

[0098] The number of reads, T, required per MKB chunk for obtaining the Media Key (Km) from the MKB associated with the number of records is shown below:

$$\text{Number of Records} < T < (\text{Number of records*2})$$

[0099] T: Number of times required for accessing MKB chunks

[0100] Each record has different length and data values. The information of each record can be obtained within two reads. Since there are four records, between 4 and 8 reads will be necessary to process the MKB chunk and obtain the records.

[0101] Therefore, the number of reads, T, are:

$$4<T<8$$

[0102] Suppose that it takes N ms to access 512-byte of MKB data. It will take (128*N)ms to access an entire 64K MKB image to obtain the Media Key from the first method. It only takes, from the second method, (8*N)ms, as the worst case scenario, to obtain the Media Key. Thus, there is a considerable time saved using this scheme. On the average, to obtain the Media Key (Km), the number of reads would be in the range of 4 to 6, and the time necessary would be proportionately less than shown above.

[0103] Step **205** of FIG. **9**, expanded here in FIG. **10**, is performed until a final media key is produced in step **205.75** or the media is rejected in step **205.80**. Not all of the 128 chunks need to be read, and not all of the 512 bytes per chunk need to be read in order to calculate the media key. Processing MKB data is an operation that requires requesting a chunk of data at a time, pointing to the desired location within that specific chunk and computing the obtained values. Not all MKB data is needed. The algorithm depicted in FIG. **10** will provide a mathematical calculation to figure out exactly what chunk of MKB data is needed, what record should be processed and where the encrypted data is located.

[0104] In step **205.5**, the buffer pointer is set to the data buffer and the buffer offset is cleared. Next, in step **205.10**, the chunk number is checked to see if it is equal to or larger than the maximum chunk number. If it is, an error will be returned in step **205.15**. If it is not, the chunk number will be incremented and new data will be loaded into the buffer in step **205.20**. Then the buffer offset will be updated in step **205.25**. Thus, the pointer can be set to the correct location (the chunk number plus offset). In step **205.30**, the buffer pointer is set to the buffer offset. In step **205.40** the buffer is read starting at the offset where the pointer is located. The system will then determine what type of record it is reading. As seen in step **205.40**, the system will first check what type of record is being read, and what record length is associated with that record. The actions that will follow differ depending upon the record type and length. The record length of each record will be used to determine where the buffer pointer should be located in reading the subsequent record. This is reflected by steps **205.49**, updating the buffer offset and setting the buffer pointer at the new offset.

[0105] If the record is a CMKR as shown in step **205.42**, then the system updates the buffer chunk number and offset to the correct MKB location where the encrypted media key (Km)is located in step **205.49**. Each card has 16 MKBs. Thus, the system will get the offset where the encrypted media key is, go to the specific MKB chunk number, allocate buffer (16 blocks×512 bytes), and go to the offset within each block to read the encrypted media key. Then the system uses a device key (Kd) supplied from device **15** to decrypt (calculate) the media key in step **205.50**. Once the media key has been calculated the next step is to verify the media key.

[0106] If the record is a VMKR as evaluated in step **205.44**, the media key that was previously calculated, either on the first attempt in step **205.50**, or in a subsequent attempt in step **205.65**, will be compared to a reference media key (Km) in step **205.55**. In order to do this, reference media key will first be stored locally. If the key is the same a pass will be returned, which in hex is DEADBEEF, and the system will not need to conditionally calculate the media key. In order to figure out where to start reading the next record, the record length of the VMKR is used to move the buffer pointer to the next record. If it is not the same it then it will be calculated again when a CCMKR record is read in step **205.46**. When this record is read, the media key will be calculated once again in step **205.65** after the buffer point has been set to read at the updated buffer offset in step **205.49**, and then it will be subsequently verified when the next VMKR is read. The maximum number of times the CCMKR is calculated may be set by the system and preferably one.

[0107] The first calculation takes place when a CMKR is found. If it is successfully calculated, as determined during the verification process initiated when a VMKR is found, then there will be no need to conditionally calculate the media key (Km). If the verification is unsuccessful then when a CCMKR is found the media key (Km) will be recalculated and re-verified. This means that there are two chances to calculate the media key. Finally, if the record is an EMKR as evaluated in step **205.48**, then in step **205.75** the system will verify that at the end of the record a valid media key (Km) is present, and in step **205.75** the final media key (Km) will be produced, after the buffer pointer is set at a the proper offset for this type of record in step **205.49**. If, however, a valid media key is not returned in step **205.70**, the media will be rejected in step **205.80**. If the final media key is returned in step **205.70**, the processing will continue at step **210** of FIG. **9**, as shown by step **205.85**. Thus the MKB process is complete.

[0108] Functions within SDSE **175** perform security accesses such as Get MKB by using a secure device driver interface (SDDI) **190**A to device driver **190**. This same device driver, SDDI **190**A also makes use of functions within SDSE **175** which it can call directly. For example, prior to issuing a read of the authentication area, SDDI **190***a* must first call the sec_ake function within SDSE **175**. The sec_ake function will in turn call back into SDDI **190**A. This "dual calling relationship" which facilitates the isolation of the device key within SDSE **175** is unique to SW **100***s* implementation of the SD-Audio standards.

[0109] Since SDSE **175** handles all key-oriented processing, and these values are needed when certain SD commands are received by the audio interface **105**, video interface **110**, or image interface **115**, the device driver must make use of functions within SDSE **175** which it can call directly. When carrying out the functions, SDSE module **175** must in turn call back into the device driver **190**'s private security interface, SDDI **190**A. This "dual calling relationship" allows interwoven requests between SDSE **175** and device driver **190**, thus enabling key values to be isolated within the security module.

[0110] The SDSE **175** software layer invokes security device driver services via the private interface by initiating a security driver request packet and calling the security driver interface entry point passing a request packet pointer.

[0111] In order to clarify the appended source code which has been incorporated by reference, the following tables are provided.

[0112] The request packet (defined in sdapi.h) consists of a data type SSMSERVE which is defined as follows:

TABLE 2

Variable	Variable name
Typedef struct __mySecuredDrv	
{	
Data buffer	UCHAR *buffer
Number of data blocks	UINT16 noBlocks
Application unique Number	UINT16 mkb__ID
Start address	UINT16 lba
Security flag	INT16 securityFlag
Drive number	INT16 driveNo
Command index	INT16 opCode
}	

[0113] Command index (INT16 opCode) holds the command for the service being requested. Supported commands include:

TABLE 3

Command	Functional Code Routine
Device identify	#define SDDRV__IDENT 0
Security identify	#define SDDRV__SECIDENT 1
Secure read	#define SDDRV__SECRD 2
Secure write	#define SDDRV__SECWR 3
Secure erase	#define SDDRV__SECERASE 4
Read MKB	#define SDDRV__RDMKB 5
Get MID	#define SDDRV__GETMID 6
Set challenge	#define SDDRV__SETCHALGE 7
Get challenge	#define SDDRV__GETCHALGE 8
Set response	#define SDDRV__SETRESP 9
Get response	#define SDDRV__GETRESP 10
Change size of protected area	#define SDDRV__CHANGESA 11

[0114] Security device driver service requests are issued from the SDSE **175**module. For example, the Generate Challenge **1** function sends challenge **1** as follows:

TABLE 4

Generate Challenge 1

Command	Operation
Call security routine	SDSECURITYDRV mySecDrv
Set drive number	mySecDrv.driveNo = (INT16)drv
Set memory address within media	mySecDrv.lba = 0
Number of data blocks	mySecDrv.noBlocks = 1
Set challenge	mySecDrv.opCode = SDDRV__SETCHALGE
Send challenge 1	mySecDrv.buffer = Chlg1
Call to device driver	scDDHandler(&mySecDrv)

[0115] Because all key manipulation is confined to SDSE **175**, SSDI **190**A must rely on SDSE **175** functions to perform Authentication Key Exchange (AKE) or for decrypting data that has been transferred across the bus (note that all data sent across the bus is first encrypted using the "session key" which is generated from each AKE.)

[0116] When performing the AKE, SDSE **175** must send commands to the SD Memory Card **13**, thus, it must in turn call into SDDI **190**A. This calling relationship is outlined in the diagram of FIG. **7** which depicts the steps necessary to process a read of the authentication area.

[0117] Notice that the sec_ake function within the SDSE **175**, when called by the security SDDI **190**A, performs four calls back into the security device driver via the private driver interface. These four requests consist of: SDDRV_SETCHALGE, SDDRV_GETCHALGE, SDDRV_SETRESP, and SDDRV_GETRESP. This enables the security module to carry out the requisite set challenge/get challenge, set response/get response steps seen in FIG. **7**. The resultant session key is stored within the security module. This is used to decrypt data when the security device driver calls into the SDSE **175**'s bus_decrypt function to get information from SDDI **190**A.

[0118] The system and method of the present invention are advantageous over prior techniques in many ways. The present invention provides a turnkey solution for original equipment manufacturers to access encrypted content without having to have any knowledge of the memory structure of the storage media. The decryption process by itself is very complex. Furthermore, simply reading and writing to a memory card or compact disk is complex in and of itself. All a manufacturer needs to do is send a simple command such as "play" or "next track" and return the decrypted content from whatever the memory device happens to be.

[0119] Device keys and resultant session keys are manipulated in a very isolated and protected software layer. These are never exposed in upper layers. Even the lower device driver layer is not given direct access to the keys. Device keys are retrieved from an OEM-supplied library when generating the media key. This key is retained within the security engine, but the media unique key (Kmu) which is the heart of the security scheme is never stored. A private interface to the security engine enables the security engine to gain low-level access to the memory card while keeping the exposure of all security-related keys (e.g., device keys, media keys, session keys) confined within the security engine. A "dual calling relationship" allows the security engine and the security device driver to make interwoven use of each other's services.

[0120] While particular embodiments of the present invention and their advantages have been shown and described, it should be understood that various changes, substitutions, and alterations can be made therein without departing from the spirit and scope of the invention as defined by the appended claims. For example, although usage of an SD memory card has been shown to illustrate the functioning of the invention, the invention can be used on any media having encrypted content. It can also be utilized by any type of device. Furthermore, encrypted content can be decrypted from any type of memory device, whether it be fixed or removable, and whether it be solid state or rotating. The content is not limited to audio or video, but can be any content worthy of encryption.

What is claimed is:

1. A method of securing a storage device and providing content to an authorized entity yet protecting the content from unauthorized duplication:

providing a flash memory array within the storage device;

providing a freely accessible read/write area within the memory array, the freely accessible read/write area being a first distinct area;

providing a read/write area within the memory array that is accessible only after a successful mutual authentication between a host device and the storage device comprising the flash memory array has occurred, the read/write area within the memory array that is accessible only after a successful mutual authentication being a second distinct area within the memory array;

providing a write protected area within the memory array that is accessible only by the storage device itself, the write protected area being a third distinct area within the memory array;

providing a write protected area that is accessible for reading from outside of the storage device, the write protected area being a fourth distinct area within the memory array; and

storing the content in an encrypted format in the freely accessible read/write area.

2. The method of claim **1**, further comprising storing a media unique key in the write protected area within the memory array that is accessible only by the storage device itself.

3. The method of claim **2**, further comprising storing title keys necessary to access content in the freely accessible read/write area within the write protected area within the memory array that is accessible only by the storage device itself.

4. The method of claim **3**, further comprising:

providing copy control information within the read/write area within the memory array that is accessible only after a successful mutual authentication; and

concatenating and encrypting the title keys and the copy control information together with the media unique key stored in the write protected area within the memory array that is accessible only by the storage device itself.

5. The method of claim **1**, further comprising storing a media identifier unique to each storage device within the write protected area that is accessible for reading from outside of the storage device.

6. The method of claim **5**, further comprising storing media key blocks within the write protected area that is accessible for reading from outside of the storage device.

7. The method of claim **6**, further comprising transferring the media key blocks from the write protected area that is accessible for reading from outside of the storage device to a portable device, the portable device having a key able to decrypt the transferred media key blocks.

8. The method of claim **7**, further comprising transferring a piece of the encrypted content from the freely accessible read/write area to the portable storage device, the portable storage device having an encrypted version of the title key used to encrypt the piece of the encrypted content.

9. A system for providing and reproducing content for an authorized entity yet protecting the content from unauthorized duplication, the system comprising:

a host device for reproducing the content;

a storage device comprising a flash memory array;

a freely accessible read/write area within the memory array, the freely accessible read/write area being a first distinct area;

a read/write area within the memory array that is accessible only after a successful mutual authentication between the host device and the storage device has occurred, the read/write area within the memory array that is accessible only after a successful mutual authentication being a second distinct area within the memory array;

a write protected area within the memory array that is accessible only by the storage device itself, the write protected area being a third distinct area within the memory array and having records ; and

a write protected area that is accessible for reading from outside of the storage device, the write protected area being a fourth distinct area within the memory array, the content being freely accessible but only reproducible if the host device can successfully mutually authenticate with the storage device and access the second distinct read/write area.

* * * * *