US 20080215847A1

(19) **United States**

(12) **Patent Application Publication**

Holtzman et al.

(10) **Pub. No.: US 2008/0215847 A1**

(43) **Pub. Date: Sep. 4, 2008**

(54) **SECURE YET FLEXIBLE SYSTEM ARCHITECTURE FOR SECURE DEVICES WITH FLASH MASS STORAGE MEMORY**

(75) Inventors: **Micky Holtzman**, Kfar-Vradim (IL); **Hagai Bar-El**, Rehovot (IL); **Ronen Greenspan**, Kiryat Yam (IL); **Royan Shapiro**, Tel Aviv (IL)

Correspondence Address:
BRINKS HOFER GILSON & LIONE/SanDisk
P.O. BOX 10395
CHICAGO, IL 60610 (US)

(73) Assignee: **SanDisk Corporation and Discretix Technologies Ltd.**

(21) Appl. No.: **12/122,412**

(22) Filed: **May 16, 2008**

Related U.S. Application Data

(62) Division of application No. 11/317,339, filed on Dec. 22, 2005.

(60) Provisional application No. 60/717,164, filed on Sep. 14, 2005.

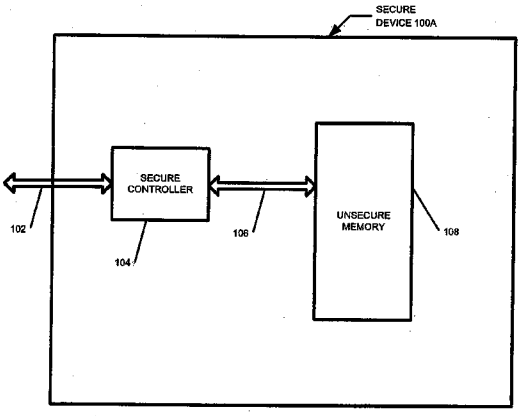
Publication Classification

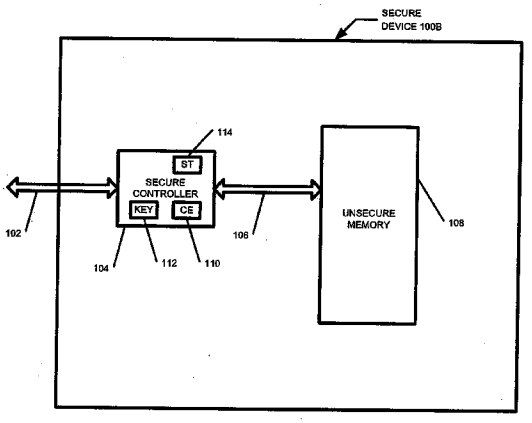
(51) **Int. Cl.**
G06F 12/02 (2006.01)

(52) **U.S. Cl.** **711/173**; 711/E12.002

(57) **ABSTRACT**

A device with mass storage capability that uses a readily available non secure memory for the mass storage but has firmware (and hardware) that provides security against unauthorized copying of data. This is true even though the firmware itself is stored in the non secure mass storage memory, and therefore potentially vulnerable to hacking. An indication of the authenticity of the firmware must be present before it will be executed by the device. This protects the device contents from unauthorized duplication or tampering. Additional functionality can be added to the device with additional firmware applications, and the authenticity of those additional applications will also be verified before they will be executed. This further prevents unauthorized copying or tampering of secure content through any mechanisms that may be unscrupulously introduced. Any data within the mass storage memory may also be encrypted.





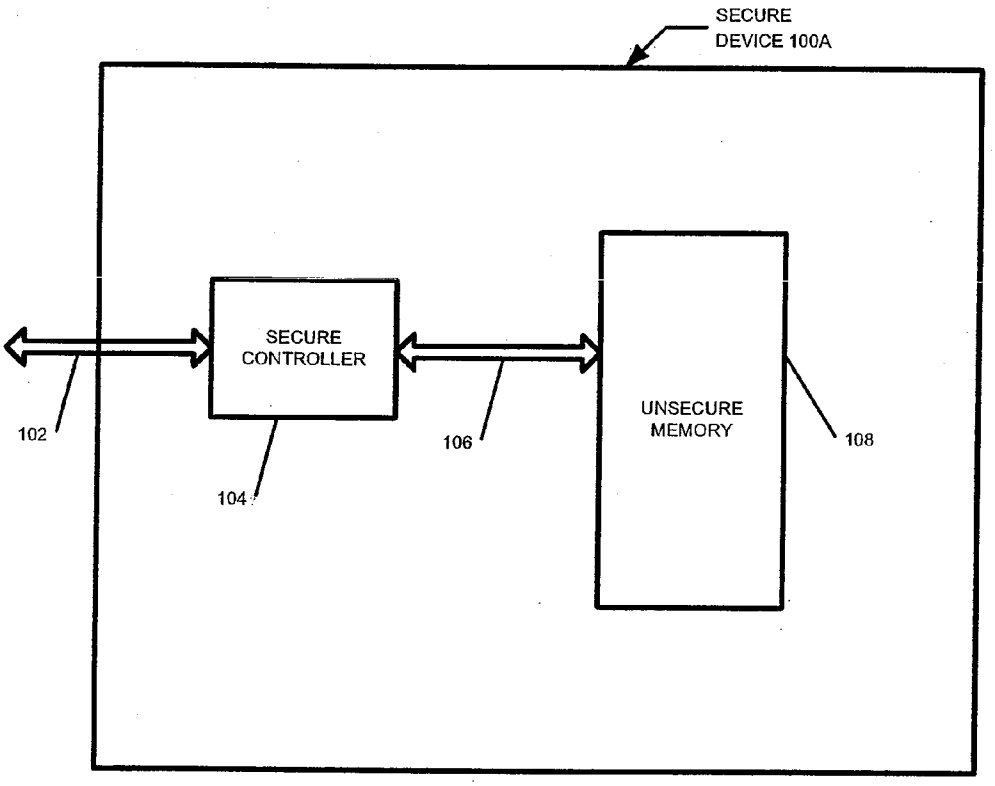

SECURE
DEVICE 100A
SECURE
CONTROLLER
UNSECURE
MEMORY
102
104
106
108

FIG. 1A

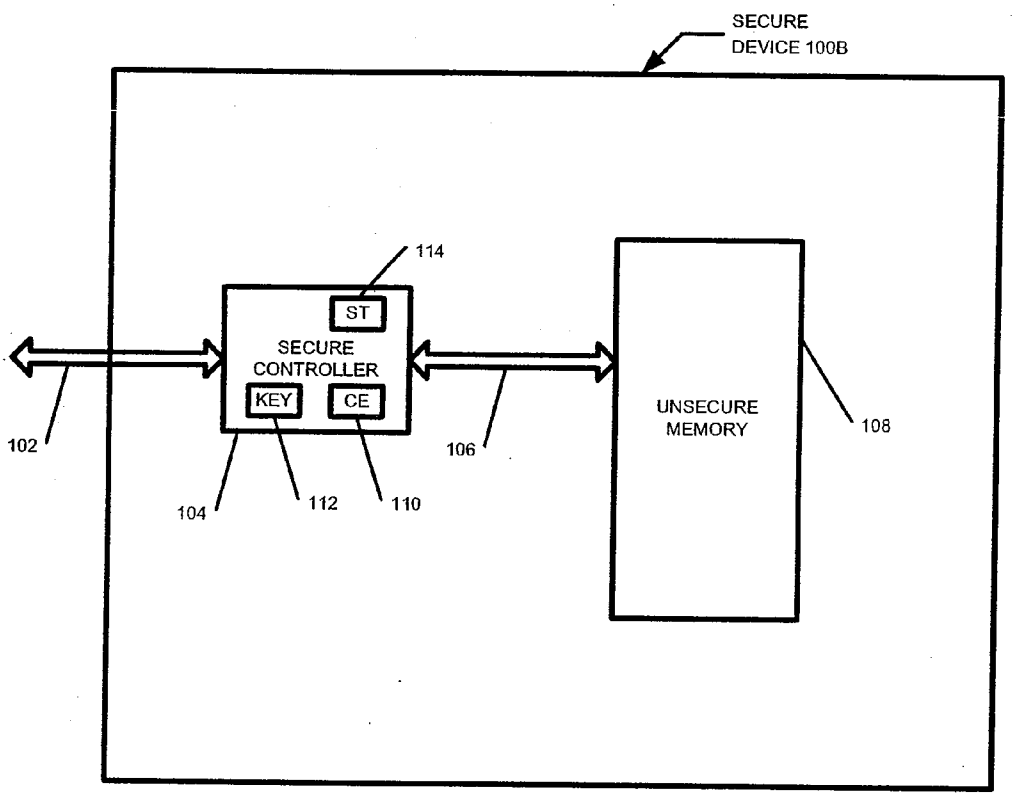

SECURE
DEVICE 100B
114
ST
SECURE
CONTROLLER
KEY
CE
UNSECURE
MEMORY
102
104
112
110
106
108

FIG. 1B

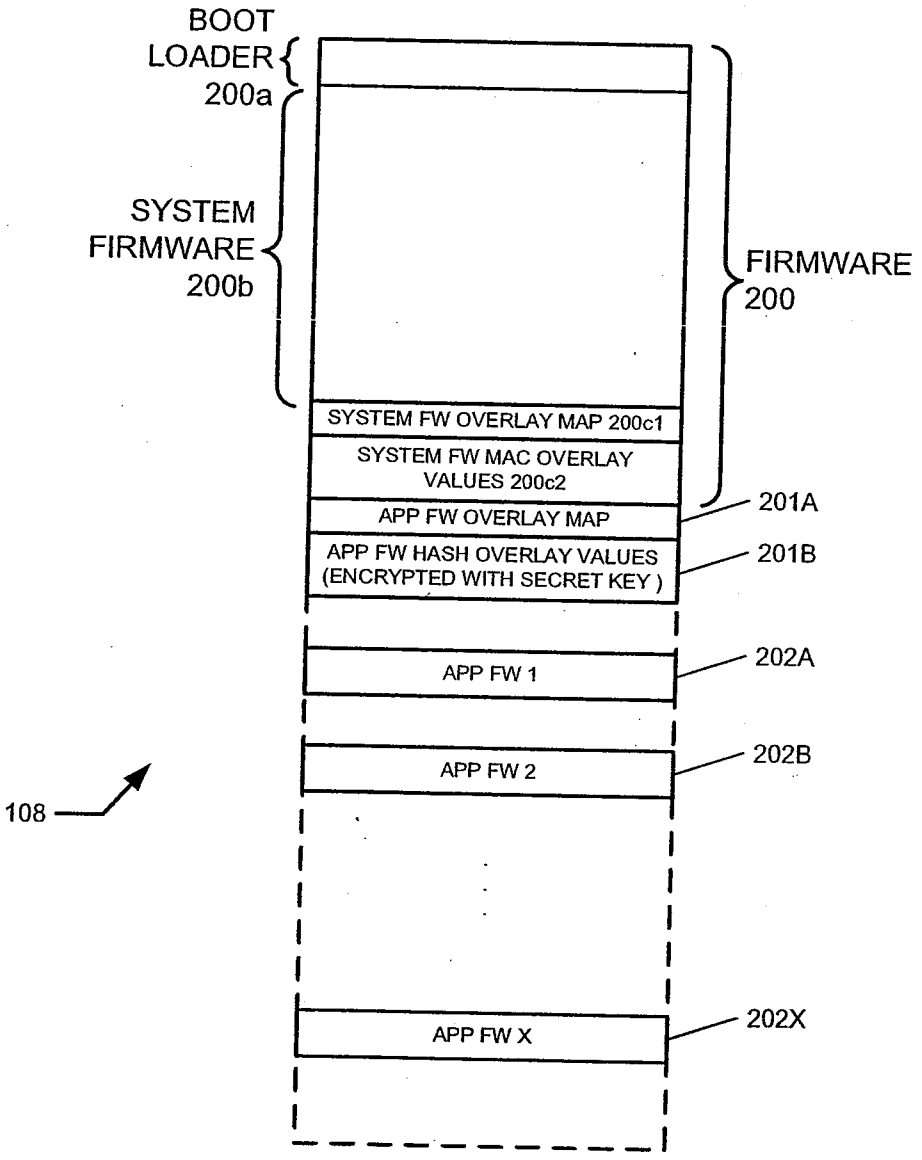
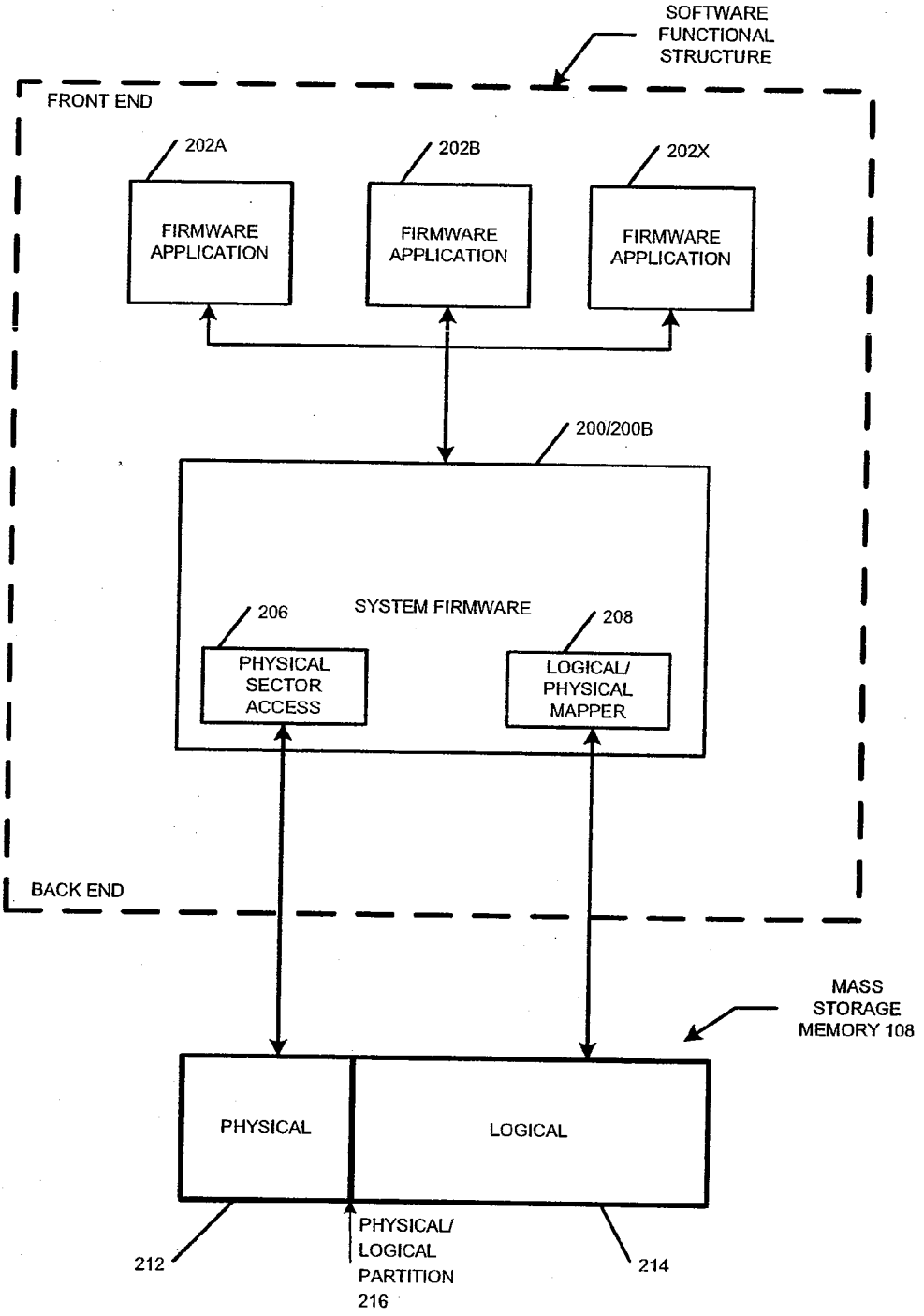

BOOT LOADER 200a
SYSTEM FIRMWARE 200b
FIRMWARE 200
SYSTEM FW OVERLAY MAP 200c1
SYSTEM FW MAC OVERLAY VALUES 200c2
APP FW OVERLAY MAP
201A
APP FW HASH OVERLAY VALUES (ENCRYPTED WITH SECRET KEY)
201B
APP FW 1
202A
APP FW 2
202B
108
APP FW X
202X

FIG. 2

FIG. 3


SOFTWARE FUNCTIONAL STRUCTURE
FRONT END
202A
202B
202X
FIRMWARE APPLICATION
FIRMWARE APPLICATION
FIRMWARE APPLICATION
200/200B
SYSTEM FIRMWARE
206
208
PHYSICAL SECTOR ACCESS
LOGICAL/ PHYSICAL MAPPER
BACK END
MASS STORAGE MEMORY 108
PHYSICAL
LOGICAL
212
PHYSICAL/ LOGICAL PARTITION 216
214

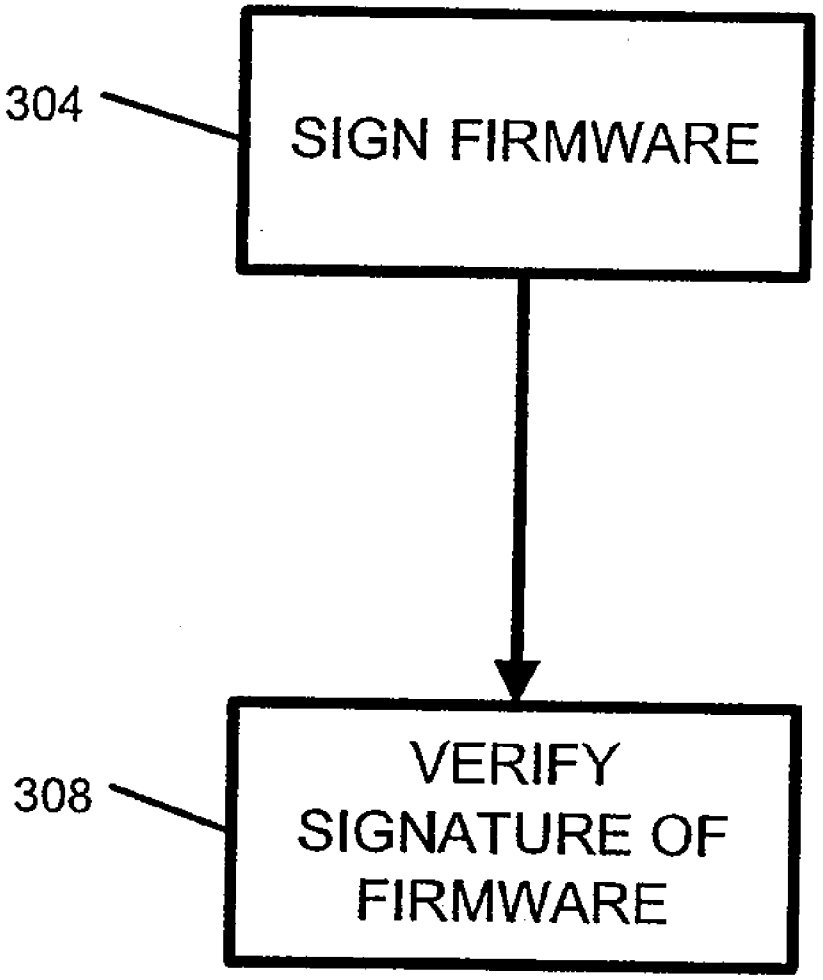


FIG. 4

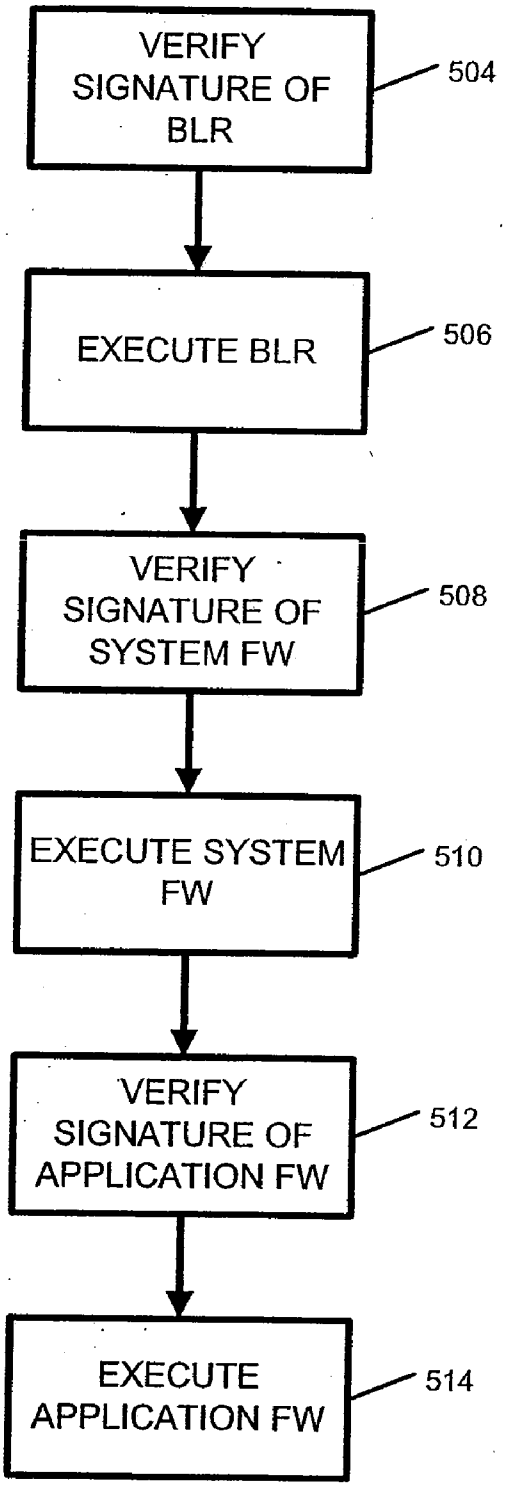

VERIFY SIGNATURE OF BLR
504
EXECUTE BLR
506
VERIFY SIGNATURE OF SYSTEM FW
508
EXECUTE SYSTEM FW
510
VERIFY SIGNATURE OF APPLICATION FW
512
EXECUTE APPLICATION FW
514

FIG. 5

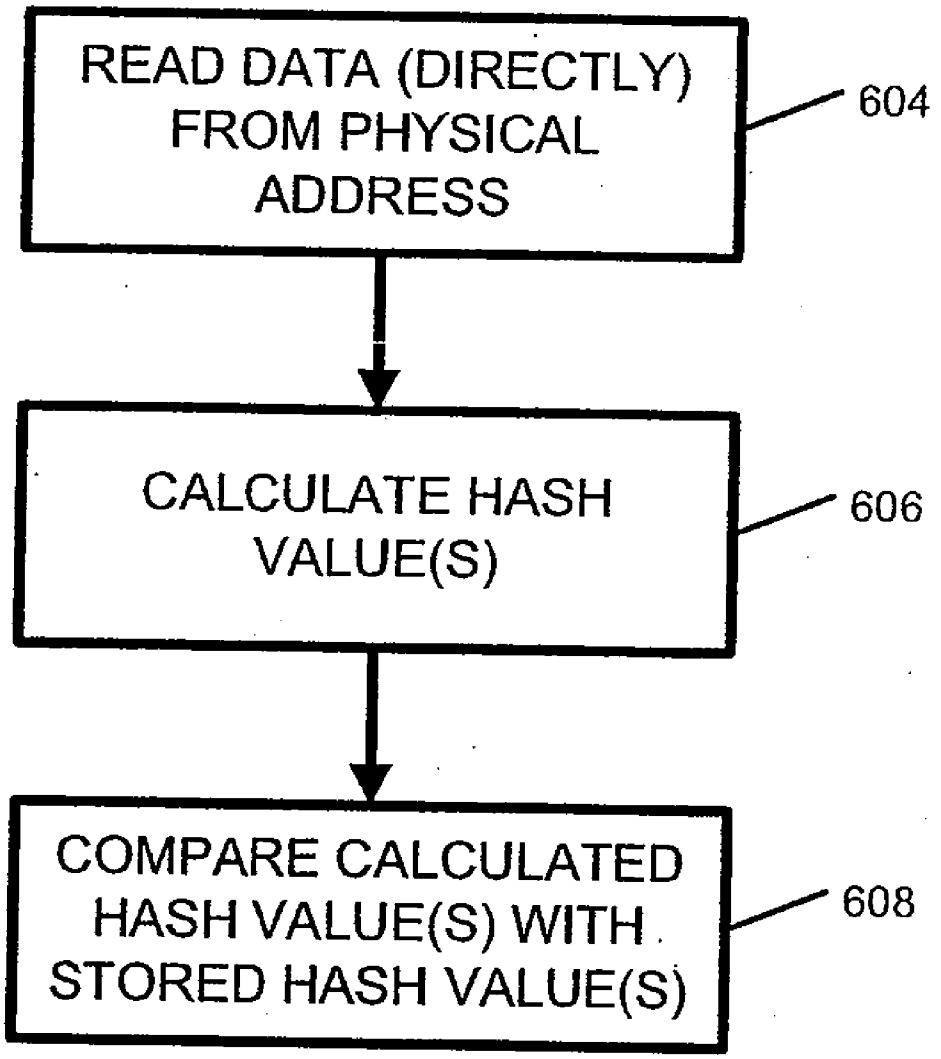

READ DATA (DIRECTLY) FROM PHYSICAL ADDRESS
604
CALCULATE HASH VALUE(S)
606
COMPARE CALCULATED HASH VALUE(S) WITH STORED HASH VALUE(S)
608

FIG. 6

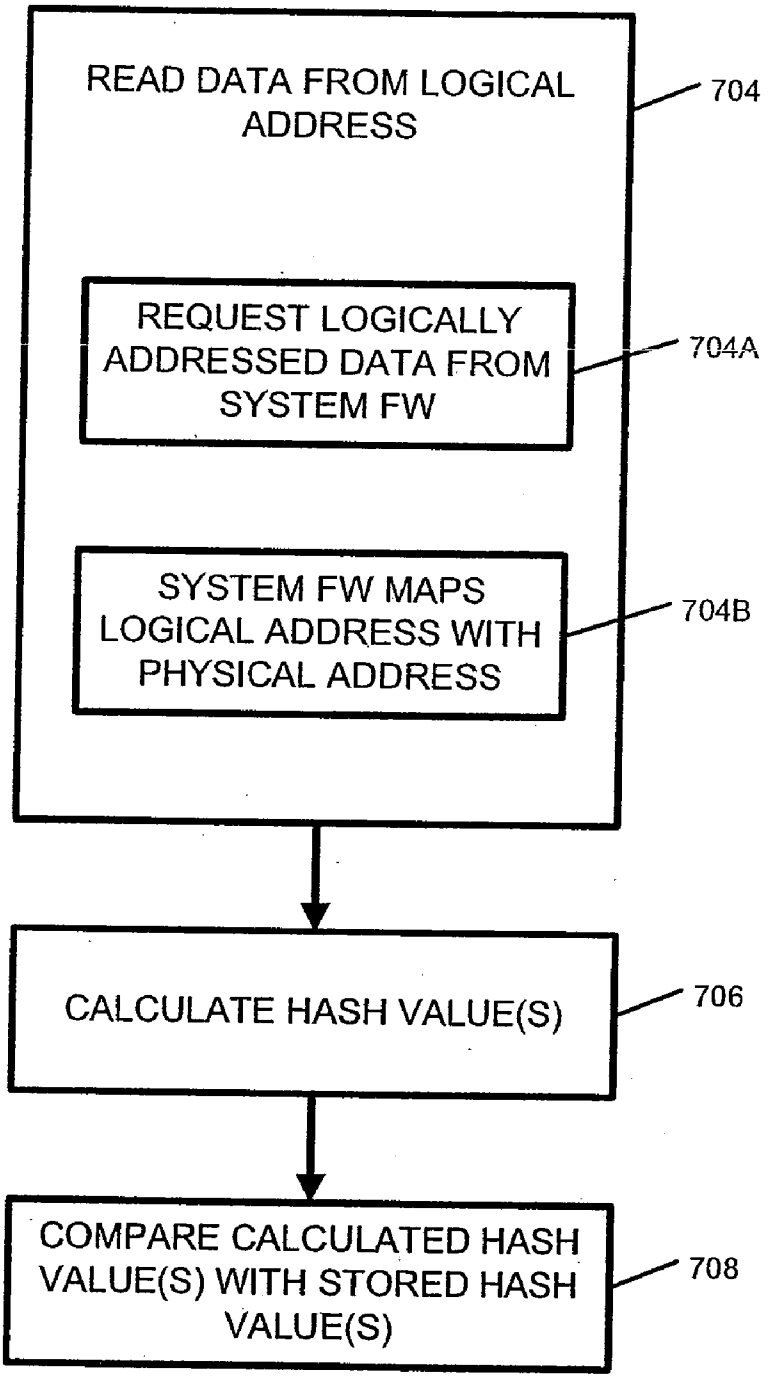

READ DATA FROM LOGICAL ADDRESS
704
REQUEST LOGICALLY ADDRESSED DATA FROM SYSTEM FW
704A
SYSTEM FW MAPS LOGICAL ADDRESS WITH PHYSICAL ADDRESS
704B
CALCULATE HASH VALUE(S)
706
COMPARE CALCULATED HASH VALUE(S) WITH STORED HASH VALUE(S)
708

FIG. 7

SECURE YET FLEXIBLE SYSTEM ARCHITECTURE FOR SECURE DEVICES WITH FLASH MASS STORAGE MEMORY

CROSS REFERENCE TO RELATED APPLICATIONS

[0001] This application claims priority to provisional Application No. 60/717,164 entitled "Secure Yet Flexible System Architecture for Secure Devices With Flash Mass Storage Memory" filed Sep. 14, 2005 to Micky Holtzman et al.

[0002] This application is related to the following applications, each of which is hereby incorporated by this reference in its entirety: "Methods Used in a Secure Yet Flexible System Architecture for Secure Devices With Flash Mass Storage Memory" to Micky Holtzman et al., Attorney Docket No.: SNDK.470US2; "Method of Hardware Driver Integrity Check Of Memory Card Controller Firmware" to Micky Holtzman et al., application Ser. No. 11/284,623, Attorney Docket No. SNDK.408US 1; "Hardware Driver Integrity Check Of Memory Card Controller Firmware" to Micky Holtzman et al., application Ser. No. 11/285,600, Attorney Docket No. SNDK.408US2; "Methods Used in a Secure Memory Card With Life Cycle Phases" to Micky Holtzman, et al. Attorney Docket No. SNDK.383US2; and "Secure Memory Card With Life Cycle Phases" to Micky Holtzman et al., Attorney Docket No. SNDK.383US3.

FIELD OF THE INVENTION

[0003] The present application is generally related to the operation of flash based mass storage devices, and in particular those with copy protection of secure content.

BACKGROUND OF THE INVENTION

[0004] Flash based mass storage devices are used to store very large amounts of content, such as pictures and music or software programs. Examples of these mass storage devices include memory cards, universal serial bus ("USB") flash drives, flash based music and/or video players, and other portable computing devices that rely on flash for the mass storage of content or files.

[0005] User files are frequently updated and modified. This is particularly the case when dealing with photos, music, and documents. Flash memory has a limited number of read/write cycles, and a great deal of research and development has gone into distributing the cycles among the flash memory cells in order to maximize the lifespan and reliability of the devices. For instance, wear leveling techniques such as those taught in U.S. Pat. No. 6,230,233 entitled "Wear Leveling Techniques For Flash EEPROM Systems" to Lofgren et al., U.S. Pat. No. 5,268,870 entitled "Flash EEPROM System and Intelligent Programming and Erasing Methods Therefore" to Harari, PCT Publication No. WO2004040578A2 entitled "Wear Leveling In Non-Volatile Storage Systems" to Chang et al., and U.S. Patent Publication No. 20040083335A1, entitled "Automated Wear Leveling In Non-Volatile Storage Systems" to Gonzalez et al., which are hereby incorporated by this reference in their entireties, are commonly implemented in these devices. These techniques generally involve changing the logical/physical mapping so that physical locations of the memory are used roughly the same amount.

[0006] In addition, as the usage of flash based mass storage devices is proliferating, the number of different things that can be done with them is also increasing.

[0007] Therefore, there exists a need for a new device operating system architecture that provides flexibility to store and run a wide range of firmware that can be updated and changed to accommodate the range of increasing functionality. In addition to being flexible, this architecture must provide a highly secure and reliable environment for both firmware and content. As is always the case, all of this should be done for the lowest possible cost, using standard components whenever possible.

SUMMARY OF INVENTION

[0008] The present invention allows a device to be both secure in operation and flexible in terms of functionality. This means functionality can be tailored to users' desires and added over time all the while maintaining a high level of security. Therefore the device can be used to store confidential and limited access information such as transactional data and copyrighted artistic works.

[0009] The present invention also allows for the device to boot quickly and reliably while still providing for reliable long term data storage through the use of wear leveling techniques where appropriate.

[0010] Firmware that is not authentic, and that may potentially compromise the security of the device will not be executed. An indication of the authenticity is verified before execution. In a preferred embodiment, multiple different levels of such indications are provided and are associated with the particular controller of the device that created the indications. In this preferred embodiment, one or more of the different levels of indications can be verified. Without the properly associated indication the firmware will not be executed.

[0011] Another aspect of the present invention is that this security is achieved despite the fact that the device utilizes readily available memory without built in security for the mass storage of the data, including the firmware.

BRIEF DESCRIPTION OF THE DRAWINGS

[0012] FIG. **1**A is a schematic diagram of secure device **100**A, an embodiment of the present invention.

[0013] FIG. **1**B is a schematic diagram of secure device **100**B, an embodiment of the present invention.

[0014] FIG. **2** is a diagram illustrating various pieces of firmware in a portion of memory space **108**.

[0015] FIG. **3** is a schematic diagram illustrating software structure and hardware access according an embodiment of the present invention.

[0016] FIG. **4** is a flowchart illustrating some steps of firmware integrity verification.

[0017] FIG. **5** is a flowchart of operation of an embodiment of the present invention.

[0018] FIG. **6** is a flowchart illustrating integrity checking of physically stored data such as the firmware **200**.

[0019] FIG. **7** is a flowchart illustrating integrity checking of logically stored data such as user files and the application firmware **202**A, B, . . . X.

DETAILED DESCRIPTION OF THE PREFERRED EMBODIMENTS

[0020] Devices incorporating flash memory for mass storage purposes must store large amounts of content that is

written and read relatively often. For instance, digital photo and music libraries are regularly updated by users of such devices. With the increase of protected content and the desire to protect content generally, such devices must also provide robust security to prevent unauthorized copying of such "secure" or protected content. At the same time, security should not come at the cost of flexibility. The present invention provides for a device that allows functionality to be added over time, while maintaining a high level of security. This flexibility is essential in a world where devices are expected to provide ever increasing functionality.

[0021] A secure device is one that protects the contents of the device from unauthorized copying or alteration. Secure content includes any content or data that it is desirable to safeguard from unauthorized copying or tampering. In addition to billing, transaction and other traditionally confidential personal information, artistic content must also be secured from access and copying by those other than the owner or other authorized persons.

[0022] Depending on the architecture of the device, a hacker may try to gain access to the content via data buses, or by directly accessing the mass storage memory. In some prior devices, directly accessing the memory storage unit was difficult as the memory storage unit was often protected by placing it an environment that was logistically hard to access. For instance, Smart Cards utilized programmable read only memories (PROMS) that incorporated a small amount of non volatile memory that was made secure in part by physically isolating it from access.

[0023] However, it is desirable to utilize unsecure mass storage memory, that is, among other things, more standardized, readily available, and/or economical. An unsecure memory or storage unit is one where authorization is not required in order to gain (read/write) access to the (encrypted or unencrypted) data stored therein, or one where there are no built in protection mechanisms that prevent copying of the stored data. While this memory may be packaged in a multi functional package with other non-memory components such as a processor, it is commonly in the form of a dedicated memory package with one or more memory chips.

[0024] Typically, a device or system incorporating mass storage flash memory utilizes a processor to control the data storage and retrieval operations of the memory. Such a processor is part of a controller and is often referred to as a controller. A controller executes software instructions to control the device. The software that runs and controls the hardware of a device is often referred to as firmware. The firmware is typically executed from random access memory (RAM) after having been copied from some other memory where it is normally stored. Shadowing or copying to RAM is advantageous because although flash is easily updated it is slower and not inherently executable because it does not have random access capability, and because read only memory is not easily updated.

[0025] In the case where some amount of security is to be provided in the firmware, there must be some mechanism to prevent execution of the other than the proper firmware that has the requisite security mechanisms. This is especially true when the firmware is stored in an unsecure memory. As mentioned above, it is the firmware that controls the operation of the device, and therefore it is not a simple matter to have the firmware essentially protect itself. Nor is it a simple matter to protect execution of compromised or unauthentic firmware when such firmware is stored in an otherwise unsecure memory package.

[0026] The present invention provides for a secure system with mass storage capability even though it uses unsecure memory for the mass storage unit. Furthermore, it creates a secure system where the firmware for running the secure system is stored in the unsecure memory. In order to be able to store the firmware in the unsecure mass storage memory, the present invention employs a system that prevents execution of inauthentic firmware.

[0027] Reference will now be made to preferred embodiments depicted in the figures. FIG. **1**A illustrates secure device ("SD") **100**A, an embodiment of the present invention. SD **100**A comprises a secure controller **104** and unsecure memory **108**.

[0028] Memory **108** is preferably flash type memory and is used for mass storage purposes. This means that the memory is used for general purpose storage of user files, such as audio, video, and picture files, among other things. It is a principal memory storage unit of device **108** and can be used to store any type of file a user wishes to store in it. It is designed to allow a user to frequently update and access his library of files. A mass storage memory is generally larger than other random access memory ("RAM") and read only memory ("ROM") that SD **100**A may also comprise (not shown) in this and other embodiments. Also, as a general file storage device, a mass storage memory is distinct from code storage devices that are designed to store comparatively small amounts of operating code that are infrequently updated. A ROM or flash memory may be used as a code storage device, but it should be understood that a code storage device is different in purpose and generally in size than a mass storage device.

[0029] SD **100**A also comprises a data or memory bus **106** and a host bus **102**. SD **100**A may be a complete electronic device such as a digital camera or music player, cellular telephone etc. It may also have the form factor of a memory card or universal serial bus ("USB") drive designed to be used in conjunction with any type of processor controlled electronic device. For simplicity in describing SD**100**A and the other embodiments depicted in the figures, the embodiments may often be referred to as a memory card, but it should be understood that such reference is to a preferred embodiment and should not limit the scope of the present invention which is defined by the appended claims. Currently, the preferred form factor for a memory card in which the present invention is especially useful is the well known Secure Digital ("SD") Card.

[0030] Data and commands are communicated to and from SD**100**A via host bus **102**. The host, which is not shown, may be a personal computer or other electronic device. Secure controller **104** controls the read and write operations to and from unsecure memory **108** via memory bus **106**. In doing so, it also limits access to the contents of the unsecure memory **108**. As mentioned above, the firmware that runs the device is stored in unsecure memory **108**. This firmware, which will be described in more detail later with regard to FIGS. **2-7**, in conjunction with controller **104**, provides the security that makes device **100**A a secure device. Therefore, it is essential that the firmware that is executed by secure controller **104** is authentic, or the security of the system could be compromised.

[0031] Ensuring the authenticity of the firmware is much more difficult when it is in an unsecure memory. However, given that the unsecure memory **108** is used for mass storage purposes, it is quite large and is easily updated. Therefore, it makes sense to use the capacity of the unsecure memory to store the firmware. This may eliminate or a least reduce the size of a code storage device dedicated to storing the firmware. Alternatively it reduces the need for such storage within the controller. This cost saving is important in a competitive market. There are 3 main paths to the contents stored in memory **108**: reading the contents of the memory **108** directly; monitoring the signals on bus **102**; and monitoring the signals on bus **106**. Even though any or all of the information in the unsecure memory **108** or on buses **102** and **106** may be in an encrypted format, there is always the danger that the encryption key(s) could be compromised. If the firmware were to be compromised and replaced with another firmware that lacked the security features of the authentic firmware, and then executed by the system, restricted or limited access files and private data on the mass storage memory could be copied or tampered with. For example, a user's banking or social security information could be copied or altered without authorization, with obvious negative ramifications. In another example, copyrighted or otherwise protected content could also be copied without authorization. Digital rights management schemes could be thwarted. As another example, cryptographic codes or user passwords could also be compromised.

[0032] FIG. **1**B illustrates secure device **100**B. Secure controller **104** comprises cryptographic engine **110**, one or more encryption keys **112** stored in a non volatile memory of controller **104**, and an indication **114** of the device operating state that is also stored in a non volatile memory of controller **104**. In certain embodiments of the invention, numerous states or life cycle phases are entered and passed through during the life of the card. Depending on the phase, logic in the card enables or disables the encryption engine, controls access to hardware (before and after card assembly) and software testing mechanisms, and controls key generation. These phases not only allow both the hardware and software of the card to be thoroughly tested before and after manufacture, but also make it virtually impossible to access the encrypted keys and thus the encrypted content when the card is in a secure phase, the operating phase that the card is in when it is shipped to the user. For more information on the states or life cycle phases please refer to an application having attorney docket No. SNDK.383US3 "Secure Memory Card With Life Cycle Phases" to Micky Holtzman et al., which is hereby incorporated by this reference in its entirety.

[0033] The cryptographic engine **110** is hardware based and can encrypt and/or decrypt data as it passes through secure controller **104**. For example, data encrypted with a first encryption algorithm as it arrives at the controller from host bus **102** can be decrypted and then encrypted with a second algorithm before it is sent to flash memory **108** via data bus **106**. Of course, data encrypted in memory **108** can be decrypted by engine **110** and passed in a decrypted state over host bus **102** although it is preferably in an encrypted format as it passes over host bust **102** so as to avoid potential unauthorized copying of the data.

[0034] The cryptographic engine **110**, also referred to as encryption engine **110**, may comprise numerous sub engines and is capable of utilizing numerous encryption standards and algorithms. Examples of the various encryption techniques and algorithms include: Message Authentication Codes ("MACs"); Data Encryption Standard ("DES"), Triple DES, Advanced Encryption Standard ("AES"), RSA and Diffie-Helman that are often used in a Public Key Infrastructure ("PKI"), and other hash based encryption such as SHA-1 and MD5. The encryption engine may use other currently available algorithms and techniques and others yet to be developed or well accepted, and the aforementioned list is only meant to provide some examples.

[0035] A Message Authentication Code is a hash computed from a message and some secret data. It is difficult to forge without knowing the secret data. The MAC is computed using an algorithm based on the DES or AES ciphers, which use a secret key. The secret key **112**, or one or more keys derived from the secret key are stored in controller **104**, and therefore the hash or message authentication code created by the controller is associated with that controller, and cannot be duplicated by another controller. Therefore hash values from a particular controller are associated with the controller and can act as a type of signature of the controller and device, because the signature is unique and cannot be duplicated.

[0036] Although the aforementioned standards and various other algorithms and/or standards are well known to those skilled in cryptography, the following publications are informative and are hereby incorporated by reference in their entireties: *RFC* 3566—*The AES-XCBC-MAC-96 Algorithm and Its Use With IPsec* by Sheila Frankel, NIST—National Institute of Standards and Technology, 820 West Diamond Ave, Room 677, Gaithersburg, Md. 20899, available at http://www.faqs.org/rfcs/rfc3566.html; *Performance Comparison of Message Authentication Code (MAC) Algorithms for the Internet Protocol Security (IPSEC)* by Janaka Deepakumara, Howard M. Heys and R. Venkatesan, Electrical and Computer Engineering, Memorial University of Newfoundland, St. John's, NL, Canada, A1B3S7 available at http://www.engr.mun.ca/~howard/PAPERS/necec_2003b.pdf; and *Comments to NIST concerning AES Modes of Operations: A Suggestion for Handling Arbitrary-Length Messages with the CBC MAC* by John Black, University of Nevada, Reno, Phillip Rogaway, University of California at Davis, available at http://csrc.nist.gov/CryptoToolkit/modes/proposedmodes/xcbc-mac/xcbc-mac-spec.pdf.

[0037] FIG. **2** is an illustration of the memory space of the flash memory **108** that includes firmware **200** that runs devices **100**A or **100**B. The system firmware **200** comprises a boot loader (BLR) portion **200***a* that resides in flash memory **108** and is preferably not changeable, and system firmware **200***b* that resides in flash memory **108** and can be changed from time to time if necessary. The size of system firmware **200** is larger than the RAM module it is executed from, so the system firmware is divided into smaller portions referred to as overlays. Each overlay preferably has its own hash value and within system firmware **200** is a table **200***c* of those hash values. Table **200***c* is not loaded as part of system firmware **200***b*, but the pre-stored values are compared with calculated values as will be discussed in more detail below. Any hash value can be used, but MAC or SHA-1 values are currently preferable. Generally, SHA-1 digests may alternatively be used in place of MAC values, and vice versa. The advantage of using MAC values is that they are associated with the hardware and the key of the hardware that created them. While SHA-1 values can be created for a given data set simply based upon the data itself, MAC values cannot be recreated without the key, and thus provide for more robust security.

Specifically, because key **104** (or a key derived therefrom) stored in the non volatile memory of encryption engine **110** must be used to create the MAC values, another processor cannot be utilized to recreate the MAC values. For example, a hacker cannot use another processor outside of the system to duplicate the firmware and the associated MAC values.

[0038] As a further security precaution, the hash values themselves can be encrypted one or more times. In the example of MAC values, a MAC entry that protects the MAC table **200***c***2** is created so even if a hacker finds a way to switch or alter the firmware and recalculate the appropriate MACs, he is still facing a problem because he must calculate the MAC of MACs (or MAC of SHA-1s). Furthermore, in one embodiment the MAC of MACs is again encrypted and stored in another (different) memory field, for example the non volatile memory of encryption engine **110** or the controller **104**. This multi-level distributed hierarchy ensures that the signatures cannot be forged, or rather, that a forged signature will not be accepted as authentic. As an illustration, if one were to access the flash memory **108** and replace the firmware and table **200***c*, the system would then check one level up the hierarchy and see if the MAC of table **200***c* indicates that table **200***c* has not been tampered with. If the stored MAC of the table does not match the calculated MAC, this indicates a problem with the authenticity. However, if the MAC of table **200***c* has also been altered to match the replaced table **200***c*, then the system would verify the signature in error. This error is avoided by storing a copy of the MAC of table **200**C in another (inaccessible) memory, and comparing the copy in the other (inaccessible) memory with the value in the flash memory **108**. If the values do not match, this indicates an authenticity problem. Although only a few levels were illustrated, this multi-level distributed structure may have numerous levels and incorporate numerous different memories depending on the size and complexity of the firmware to be protected.

[0039] This multi-level distributed hierarchy employed in conjunction with the overlay structure of the firmware also results in a very efficient and rapid authentication process. Dividing the firmware into overlays and signing each overlay greatly speeds up the overall authentication process. This is because it is much faster to verify the signature of a smaller amount of code. In order to calculate a hash value, all of the data for which the hash is to be calculated must be read. The larger the portion of firmware to be read, the longer it will take to calculate the signature, and then verify that the signature is authentic. Calculating the signature for a large amount of data is potentially very time consuming and inefficient.

[0040] Also stored within the flash memory are various firmware applications **202**A . . . X, shown as APP FW 1, 2 . . . X, and, of course, user files (not shown). The firmware applications may be configured differently for various product configurations. The number and type of these applications will vary from one product to another. The firmware applications are also preferably divided into overlays if the applications are larger than the RAM module. A map of the application firmware overlays **201**A indicates the location in memory of the various overlays. A table of hash values (SHA-1 digests or MAC values etc.) **201**B for the various firmware applications, encrypted with a secret key, which may be secret key **104** or a key derived from secret key **104**, is also stored in the flash memory. A firmware application is akin to other applications that run on a base system, e.g. a word processing application in the Windows® environment running on the Windows® operating system.

[0041] As discussed in the background, flash memory cells have a limited lifetime and the cells degrade with each read and write operation. Therefore data in the flash memory is generally moved from time to time in order to distribute the read and write operations evenly among the cells and distribute the "wear" evenly amongst the cells. This wear leveling, along with all read/write operations, is controlled by the firmware **200**, and in particular by the system firmware **200**B. In order to be able to easily move data, the data is logically stored. This means that a logical address is mapped to a physical address, and that while the logical address remains the same, it can be mapped to a different physical address. Again, this logical to physical mapping is carried out by the system firmware.

[0042] It presents some difficulty if the firmware is in charge of moving itself. This is especially true when the firmware is responsible for copy protection of the data in the flash memory, and should therefore preferably be verified as authentic before execution. Also, while it is true that the system firmware may be updated from time to time, it will be written very infrequently when compared with other data stored in the flash memory **108**. Therefore, the firmware **200**, including the boot loader **200***a* is physically (without logical mapping) written to and read from flash memory **108**.

[0043] The application firmware provides additional functionality not present in the system firmware, and may be loaded into the device at any time. It is unknown how much application firmware may be loaded into the device, and when each application may be loaded. Therefore space within the physical partition is not allocated and the application firmware is stored in the logical partition **214** and logically addressed like any other user files and data in the flash memory **108**.

[0044] FIG. **3** illustrates the functional structure of the software of the device and how it accesses the mass storage memory **108**. As mentioned before, the preferred embodiments comprise flash type memory for mass storage memory **108** and for simplicity, during this description of the preferred embodiments the terms may be used interchangeably. The portion of the software that is concerned with flash memory operations is referred to generally as the back end, while the portion of the software that involves the applications and the user interface is known as the front end. Firmware applications **202**A, **202**B . . . **202**X run on top of firmware **200** which includes system firmware **200**B. Although the BLR **200***a* is a separate component of firmware **200**, the BLR bootstraps the system firmware and may in essence generally be thought of as part of system firmware **200**. The system firmware **200** has physical sector address routines or block **206** and logical/physical mapper or mapping routines **208**. The mass storage memory **108** is partitioned into physical storage area **212** and logical storage area **214**. Physical/logical partition **216** is used to illustrate the division or partitioning of the mass storage memory **108** into areas **212** and **214**. Each of areas **212** and **216** can be further partitioned into smaller areas, and it is common in the art to use the term partitions to refer to these smaller areas also. The physical sector access routines or functional block **206** controls reading and writing in the physical area or partition **212**, and the logical/physical mapper block controls reading and writing in the logical storage area **214**.

[0045] Firmware **200**, including system firmware **200**B, is stored in physical area **212**. Application firmware **202**A . . . X is stored in logical area **214** where the user files are also stored. The application firmware and all other data in logical area **214** is moved around from time to time by the wear leveling routines of the system firmware.

[0046] The authenticity of all of the firmware is preferably checked before it is executed. This is done because, as discussed earlier, the mass storage memory **108** does not have its own built in protection mechanisms. The flowchart of FIG. **4** applies to any piece of firmware, including application firmware. In step **304**, the firmware is signed. This is typically done at the time of loading of the firmware, but a signed record can be updated by overwriting the record with a new one. The signature comprises one or more hash values of at least a portion of the firmware. The hash values are preferably of the MAC variety, because, as discussed earlier, a MAC value is created with a key used with and/or stored within the controller that created the MAC value, and cannot be recreated by another processor. Each portion or piece of firmware may be signed using a different key. For example, BLR **200**A may be signed with a first key, while system firmware **200**B is signed with a second key. Various portions (e.g. overlays) of firmware **200** can also be signed with various different keys. As another example, each piece of application firmware **202**A . . . X can be signed with a different key.

[0047] Hash values for BLR **200**A are stored and calculated in a unique process that is described in copending application entitled "Hardware Driver Integrity Check Of Memory Card Controller Firmware" to Micky Holtzman et al. having attorney docket number SNDK.408US2. Please refer to that application for further information on that process.

[0048] In one preferred embodiment involving the aforementioned life cycle phases or states, the firmware can only be signed in certain states, and unsigned firmware cannot be executed. In particular, in state **150** (not shown), which is the secure operating state that the device will generally be in while in the hands of the consumer, firmware update and signing will not be allowed. This prevents installation of substitute firmware that may not be authentic. The system in that embodiment only enables the encryption engine to sign the firmware in states other than the secure state. In other embodiments, updating of the firmware is allowed in the field (i.e. while in state **150**) as long as the firmware is signed before it is loaded and that signature can be verified by the card. The source of the firmware can also be identified and verified, as well as verifying the signature of the firmware itself. The firmware should be supplied by a trusted entity before it is loaded, and in a preferred embodiment the trust is established using a public key infrastructure ("PKI") certificate. This certificate could be in addition to or alternatively in lieu of the hash based signature. For example, if trust is established (by the certificate in this illustrative embodiment) then the encryption engine would sign the firmware. As another added precaution, a secure connection can be established with the supplier of the firmware. The secure connection would be encrypted to protect the data passing between the device and the supplier. The secure connection would preferably be encrypted according to the aforementioned AES standard, but could employ any known encryption standard.

[0049] As mentioned previously, the system firmware is broken up into overlays of smaller size so that each overly can be loaded into RAM for execution. A map **200***c***1** of the various overlays is stored in the flash memory. Each overlay is individually signed. A table **200***c***2** of the signatures, which are preferably MAC values, is also stored in the flash memory as part of firmware **200**. The system or device **200** allocates sufficient room in the RAM for complete table **200***c***2** to be loaded, and the entire table is loaded and resident in RAM during operation of the device.

[0050] Each firmware application **202**A . . . X is also broken up into overlays, and each overlay is likewise signed. Currently, as with the system firmware, it is preferable to calculate key dependent hash (e.g. MAC) values to sign the firmware applications, although as mentioned previously, other hash values may be used. Table **201**B contains the signatures for each application firmware overlay in map **201**A. A one sector buffer is pre-allocated in the RAM as a workspace for the application firmware signatures.

[0051] Although it is preferable to sign each overlay of any of the firmware because this prevents replacement of a piece of firmware that may have critical decision making functionality, any amount of firmware can rely on one signature. For example, although not preferred, one signature could be used for all the firmware. Furthermore, the size of the overlays or portions to be signed may also vary. Referring again to FIG. **4**, after the firmware is signed in step **304** as described above, each signature is verified in step **308**. When each piece, e.g. each overlay, of the firmware is read from the flash memory, it passes through the encryption engine **110**, and the hash value of the piece is created "on the fly" by the encryption engine. This calculated value is compared to the stored value, and if the values do not match there is a problem with the authenticity of the piece of the firmware. If there is a match, then the next level of the hierarchical structure described earlier with regard to FIG. **2** will preferably be checked. Preferably all the levels will be checked as will the copy stored in the additional memory. As mentioned previously, this distributed hierarchical structure assures that the firmware and signature are authentic.

[0052] FIG. **5** is a flowchart illustrating firmware execution. In step **504**, the system verifies the signature of the boot loader portion ("BLR"). This can be done as mentioned above in regard to the system firmware, but is preferably done in another process described in a co-pending application entitled "Hardware Driver Integrity Check of Memory Card Controller Firmware" to Micky Holtzman et al. with attorney docket No. SNDK.408US2. After the signature of the BLR has been verified it is executed in step **506**. Next in step **508** the system verifies the signature of the system firmware. It then executes it in step **510**. If any application firmware is present, its signature is verified in step **512** and then once verified it is executed in step **514**. This is done for each piece of application firmware. As mentioned above, any of the verification steps **506**, **508**, and **510** are preferably done for each overlay of the entity being verified before or as it is loaded into the RAM.

[0053] FIG. **6** is a flowchart illustrating the reading and verification of firmware **200**, which is stored in the physical storage area **212**. This corresponds to steps **504** and **508** of FIG. **5**. In step **604** the data (firmware in this case) is read from the physical address where it is stored. Again, this physical read is performed without any logical mapping beforehand. Next, in step **604**, hash value(s) are created for the firmware.

These value(s) are temporarily stored in a register of the controller. Next in step **608** the value(s) calculated in step **606** are compared with the stored value(s). In the embodiments described, the stored value(s) are in tables in the flash memory, and may themselves be encrypted.

[0054] FIG. **7** is a flowchart illustrating the reading and verification of the firmware applications **202**A . . . X. In step **704** the data is read from the logical address where it is stored. This comprises requesting logically addressed data from the system firmware in step **704**A. It also comprises step **704**B where the system firmware then maps the logical address with its corresponding physical address at that given time. After the data (firmware in this case) is read, hash values are calculated for the overlay or other quantity of firmware in step **706**. Then in step **708** the calculated has value(s) are compared with the stored hash value(s). Again, in the embodiments described the stored hash values are in tables in the flash memory and may themselves be encrypted.

[0055] Although the various aspects of the present invention have been described with respect to exemplary embodiments thereof, it will be understood that the present invention is entitled to protection within the full scope of the appended claims.

1. A flash memory based device comprising:
a mass storage repository comprising flash memory;
a first level of the firmware stored in a physical access partition of the mass storage repository;
a second level of the firmware stored in a logical access partition of the mass storage repository;
a logical to physical interface routine that maps logical addresses to physical addresses in the logical access partition; and
a controller that executes the first level of firmware in the physical access partition without use of the logical to physical interface routine, but executes the second level of firmware through the logical to physical interface routine.

2. The flash memory based device of claim **1**, wherein the logical to physical interface routine is part of the first level of firmware.

3. The flash memory based device of claim **2**, wherein a location of the second level of firmware is mapped by the first level of firmware.

4-26. (canceled)

* * * * *