

(19)



(11) Veröffentlichungsnummer:

(11) Publication number: **EP 1 934 879 A0**

(11) Numéro de publication:

Internationale Anmeldung veröffentlicht durch die
Weltorganisation für geistiges Eigentum unter der Nummer:

WO 2007/033321 (art. 153 des EPÜ).

International application published by the World
Intellectual Property Organisation under number:

WO 2007/033321 (art. 153 of the EPC).

Demande internationale publiée par l'Organisation
Mondiale de la Propriété sous le numéro:

WO 2007/033321 (art. 153 de la CBE).

(19) World Intellectual Property Organization
International Bureau



(43) International Publication Date
22 March 2007 (22.03.2007)

PCT

(10) International Publication Number
WO 2007/033321 A2

(51) International Patent Classification:

G06F 21/00 (2006.01) G06F 9/445 (2006.01)

Sigalit St., Kiryat Yam (IL). **SHAPIRO, Rony** [IL/IL]; 5
Glicksberg St., Tel Aviv (IL).

(21) International Application Number:

PCT/US2006/035839

(74) Agents: **PARSONS, Gerald, P.** et al.; Parsons, Hsue & de
Runtz LLP, 595 Market Street, Suite 1900, San Francisco,
California 94105 (US).

(22) International Filing Date:

13 September 2006 (13.09.2006)

(25) Filing Language:

English

(26) Publication Language:

English

(30) Priority Data:

60/717,164 14 September 2005 (14.09.2005) US
11/317,341 22 December 2005 (22.12.2005) US
11/317,339 22 December 2005 (22.12.2005) US

(81) Designated States (unless otherwise indicated, for every
kind of national protection available): AE, AG, AL, AM,
AT, AU, AZ, BA, BB, BG, BR, BW, BY, BZ, CA, CH, CN,
CO, CR, CU, CZ, DE, DK, DM, DZ, EC, EE, EG, ES, FI,
GB, GD, GE, GH, GM, HN, HR, HU, ID, IL, IN, IS, JP,
KE, KG, KM, KN, KP, KR, KZ, LA, LC, LK, LR, LS, LT,
LU, LV, LY, MA, MD, MG, MK, MN, MW, MX, MY, MZ,
NA, NG, NI, NO, NZ, OM, PG, PH, PL, PT, RO, RS, RU,
SC, SD, SE, SG, SK, SL, SM, SV, SY, TJ, TM, TN, TR,
TT, TZ, UA, UG, US, UZ, VC, VN, ZA, ZM, ZW.

(71) Applicants (for all designated States except US): **SAN-
DISK CORPORATION** [US/US]; 601 McCarthy Blvd.,
Milpitas, California 95035 (US). **DISCRETIX TECH-
NOLOGIES LTD.** [IL/IL]; 45 Hamelacha St., Etgarim
Bldg. Industrial Zone, P.o. Box 8718, 42502 Netanya (IL).

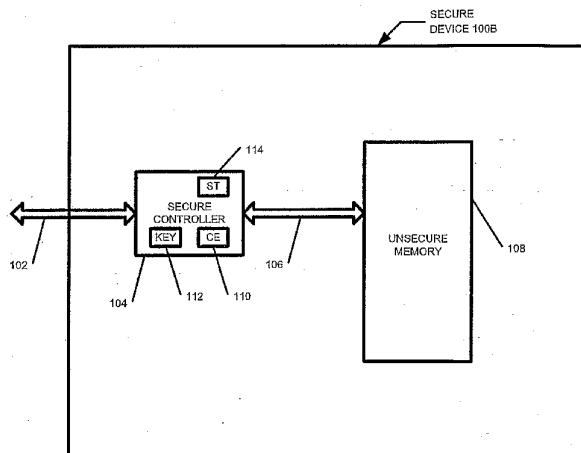
(72) Inventors; and

(75) Inventors/Applicants (for US only): **HOLTZMAN,
Michael** [IL/US]; 7602 Barnhart Place, Cupertino, Cali-
fornia 95014 (US). **BAR-EL, Hagai** [IL/IL]; Druyan 4,
76574 Rehovot (IL). **GREENSPAN, Ronen** [IL/IL]; 2

(84) Designated States (unless otherwise indicated, for every
kind of regional protection available): ARIPO (BW, GH,
GM, KE, LS, MW, MZ, NA, SD, SL, SZ, TZ, UG, ZM,
ZW), Eurasian (AM, AZ, BY, KG, KZ, MD, RU, TJ, TM),
European (AT, BE, BG, CH, CY, CZ, DE, DK, EE, ES, FI,
FR, GB, GR, HU, IE, IS, IT, LT, LU, LV, MC, NL, PL, PT,
RO, SE, SI, SK, TR), OAPI (BF, BJ, CF, CG, CI, CM, GA,
GN, GQ, GW, ML, MR, NE, SN, TD, TG).

[Continued on next page]

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



(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.



Published:

— without international search report and to be republished
upon receipt of that report

For two-letter codes and other abbreviations, refer to the "Guidance Notes on Codes and Abbreviations" appearing at the beginning of each regular issue of the PCT Gazette.

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

FIELD OF THE INVENTION

[0001] 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

[0002] 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.

[0003] 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. Patent No. 6,230,233 entitled "Wear Leveling Techniques For Flash EEPROM Systems" to Lofgren et al., U.S. Patent 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.

[0004] 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.

[0005] 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

[0006] 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.

[0007] 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.

[0008] 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.

[0009] 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

[0010] FIG. 1A is a schematic diagram of secure device 100A, an embodiment of the present invention.

[0011] FIG. 1B is a schematic diagram of secure device 100B, an embodiment of the present invention.

[0012] FIG. 2 is a diagram illustrating various pieces of firmware in a portion of memory space 108.

[0013] FIG. 3 is a schematic diagram illustrating software structure and hardware access according an embodiment of the present invention.

[0014] FIG. 4 is a flowchart illustrating some steps of firmware integrity verification.

[0015] FIG. 5 is a flowchart of operation of an embodiment of the present invention.

[0016] FIG. 6 is a flowchart illustrating integrity checking of physically stored data such as the firmware 200.

[0017] FIG. 7 is a flowchart illustrating integrity checking of logically stored data such as user files and the application firmware 202A, B,...X.

DETAILED DESCRIPTION OF THE PREFERRED EMBODIMENTS

[0018] 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.

[0019] 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.

[0020] 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 in 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.

[0021] 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.

[0022] 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.

[0023] 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.

[0024] 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.

[0025] Reference will now be made to preferred embodiments depicted in the figures. FIG. 1A illustrates secure device ("SD") 100A, an embodiment of the present invention. SD 100A comprises a secure controller 104 and unsecure memory 108.

[0026] 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 100A 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.

[0027] SD 100A also comprises a data or memory bus 106 and a host bus 102. SD 100A 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 SD100A 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.

[0028] Data and commands are communicated to and from SD100A 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 100A 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.

[0029] 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 at 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.

[0030] FIG. 1B illustrates secure device 100B. 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.

[0031] 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.

[0032] 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.

[0033] 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.

[0034] 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>.

[0035] FIG. 2 is an illustration of the memory space of the flash memory 108 that includes firmware 200 that runs devices 100A or 100B. The system firmware 200 comprises a boot loader (BLR) portion 200a that resides in flash memory 108 and is preferably not changeable, and system firmware 200b 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 200c of those hash values. Table 200c is not loaded as part of system firmware 200b, 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.

[0036] 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 200c2 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 200c, the system would then check one level up the hierarchy and see if the MAC of table 200c indicates that table 200c 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 200c has also been altered to match the replaced table 200c, then the system would verify the signature in error. This error is avoided by storing a copy of the MAC of table 200C 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.

[0037] 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.

[0038] Also stored within the flash memory are various firmware applications 202A...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 201A indicates the location in memory of the various overlays. A table of hash values (SHA-1 digests or MAC values etc...) 201B 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.

[0039] 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 200B. 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.

[0040] 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 200a is physically (without logical mapping) written to and read from flash memory 108.

[0041] 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.

[0042] 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 202A, 202B...202X run on top of firmware 200 which includes system firmware 200B. Although the BLR 200a 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.

[0043] Firmware 200, including system firmware 200B, is stored in physical area 212. Application firmware 202A...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.

[0044] 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 200A may be signed with a first key, while system firmware 200B 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 202A...X can be signed with a different key.

[0045] Hash values for BLR 200A 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.

[0046] 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.

[0047] As mentioned previously, the system firmware is broken up into overlays of smaller size so that each overlay can be loaded into RAM for execution. A map 200c1 of the various overlays is stored in the flash memory. Each overlay is individually signed. A table 200c2 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 200c2 to be loaded, and the entire table is loaded and resident in RAM during operation of the device.

[0048] Each firmware application 202A...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 201B contains the signatures for each application firmware overlay in map 201A. A one sector buffer is pre-allocated in the RAM as a workspace for the application firmware signatures.

[0049] 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.

[0050] 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.

[0051] 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 hash value(s). In the embodiments described, the stored value(s) are in tables in the flash memory, and may themselves be encrypted.

[0052] FIG. 7 is a flowchart illustrating the reading and verification of the firmware applications 202A...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 704A. It also comprises step 704B 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.

[0053] 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.

IT IS CLAIMED:

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. A device with mass storage capabilities, comprising:
 - mass storage comprising flash memory cells;
 - a controller having encryption hardware;
 - random access memory; and
 - a first level of firmware stored in the mass storage flash memory cells that controls writing data to and reading data from the flash memory cells;
 - wherein the authenticity of the first level of firmware is verified with hash values that are associated with the controller of the device.
5. The device of claim 4, wherein a key within the controller is needed to create the hash values, and wherein the hash values are associated with said key of the controller.

6. The device of claim 4, wherein the first level of firmware is stored in an encrypted format.
7. The device of claim 6, wherein the firmware is encrypted with the encryption hardware using a key stored in a non volatile memory of the controller.
8. The device of claim 4, further comprising a first table of the hash values associated with the first level of firmware, the table stored within the mass storage flash memory cells.
9. The device of claim 8, further comprising a second table, the second table comprising one or more hash values of the data of the first table.
10. The device of claim 9, wherein one or more of the hash values of the second table are associated with the controller of the device.
11. The device of claim 9, wherein a copy of the second table is stored in a non volatile memory of the controller.
12. The device of claim 11, wherein if the copy of the second table does not match the second table then the authenticity will not be verified. .
13. The device of claim 4, further comprising a second level of firmware comprising one or more firmware applications, the second level of firmware stored in the mass storage flash memory cells.
14. The device of claim 13, further comprising a third table, the third table comprising hash values associated with the second level of firmware.
15. The device of claim 14, wherein the third table is encrypted.

16. The device of claim 15, wherein the third table is encrypted with the controller encryption hardware using a key stored in a non volatile memory of the controller.
17. The device of claim 14, wherein one or more of the hash values of the third table are correlated to a controller of the device, and wherein the device will not function unless one or more of the hash values is correlated to the controller.
18. A device comprising:
a mass storage repository comprising flash memory;
random access memory;
system firmware stored in a first area of the mass storage repository; and
application firmware stored in a second area of the mass storage repository,
wherein the system firmware controls reading and writing of data in the second area,
including reading and writing of the application firmware.
19. The device of claim 18, further comprising an encryption mechanism that verifies the system firmware and the application firmware.
20. The device of claim 18, further comprising encryption hardware, the encryption hardware calculating hash values traceable to the particular hardware that created the values.
21. A secure system comprising:
a non-secure mass storage memory;
a secure controller; and
firmware comprising security mechanisms of the secure system, said firmware stored in the non-secure mass storage memory,
wherein the controller is operable to verify the integrity of at least a portion of the firmware without using the security mechanisms of the firmware, thereby preventing unauthorized alteration of the firmware although it is stored in the non-secure mass storage memory.

22. The secure system of claim 21, wherein the firmware is correlated with the controller of the system, and wherein the device will not function unless correlation between the firmware and the controller is established.
23. The secure system of claim 21, wherein the firmware comprises a group of layers, and wherein the authenticity of at least one layer is verified by hardware.
24. The system of claim 21, further comprising a plurality of states, one or more of the plurality used during production, and one or more of the states used for secure operation by a consumer.
25. The system of claim 24, wherein if the system is in a secure state a source of the firmware must be identified before the firmware can be stored in the non-secure mass storage memory.
26. The system of claim 24, wherein one or more of the plurality of states may be used for fault analysis, and wherein in a fault analysis state of the plurality, firmware from an unidentified source may be stored in the non-secure mass storage memory, and wherein the secure controller will not allow operation of the system in a secure state of the plurality.
27. A method of operating flash memory based storage device comprising:
providing a flash memory array;
reserving a first portion of the flash memory array;
storing application firmware in a second portion of the flash memory array,
said storing comprising:
 assigning a first logical address to the data;
 mapping the first logical address to a first physical address; and
 from time to time, re-mapping the first logical address to a second physical address;
storing system firmware in the first portion of the flash memory array, wherein the system firmware is stored to a physical address without first mapping it from a

logical address; conditionally executing, with a processor of the flash memory based storage device, the system firmware, without first mapping it from a logical address; and

conditionally executing, with a processor of the flash memory based storage device, the application firmware, after first mapping it from a logical address.

28. The method of claim 27 further comprising storing user content in the second the second portion of the flash memory array.

29. The method of claim 28 further comprising reading said user content by first mapping the content from a logical address.

30. The method of claim 27 wherein storing the system firmware in the first portion of the flash memory array further comprises storing an indication of the authenticity of the firmware.

31. The method of claim 30 further comprising reading the indication of the authenticity of the system firmware.

32. The method of claim 31 further comprising determining the authenticity of the system firmware.

33. The method of claim 32 wherein said system firmware is executed by accessing the physical address without first mapping it from a logical address, only if the firmware is first determined to be authentic.

34. The method of claim 28 further comprising determining the authenticity of the application firmware by reading a stored indication of the authenticity of the application firmware.

35. The method of claim 34 wherein the application firmware is executed only if the application firmware is first determined to be authentic.

36. The method of claim 34 wherein the indication is bound to a controller of the device.

37. The method of claim 30 wherein the indication is created with an encryption engine.

38. The method of claim 37 wherein the system firmware comprises a first portion, and wherein the signature for the first portion is created with a first cryptographic key.

39. The method of claim 37 wherein the system firmware comprises a second portion, and wherein the signature for the second portion is created with a second cryptographic key.

40. A method of operating flash memory based device that utilizes the flash memory for mass storage purposes, the device having firmware for operating the device stored within the flash memory, the method comprising:

accessing a first level of the firmware stored in a physical format in the flash memory; and

accessing a second level of the firmware stored in a logical format in the flash memory.

41. The method of claim 40 wherein a logical to physical mapping of the second level of the firmware is updateable by the firmware.

42. The method of claim 41 wherein the logical to physical mapping of the second level of the firmware is controlled by the first level of the firmware.

43. A method of making or operating a mass storage device, comprising:
providing a flash memory bank for the mass storage of content and software:

providing a controller that manages operations of the device;
writing firmware within the flash memory bank with the controller of the device;
signing the firmware with one or more signatures specific to the controller when it is written by the controller; and
verifying the one or more signatures of the firmware.

44. The method of claim 43 further comprising executing the firmware if the one or more signatures are verified.

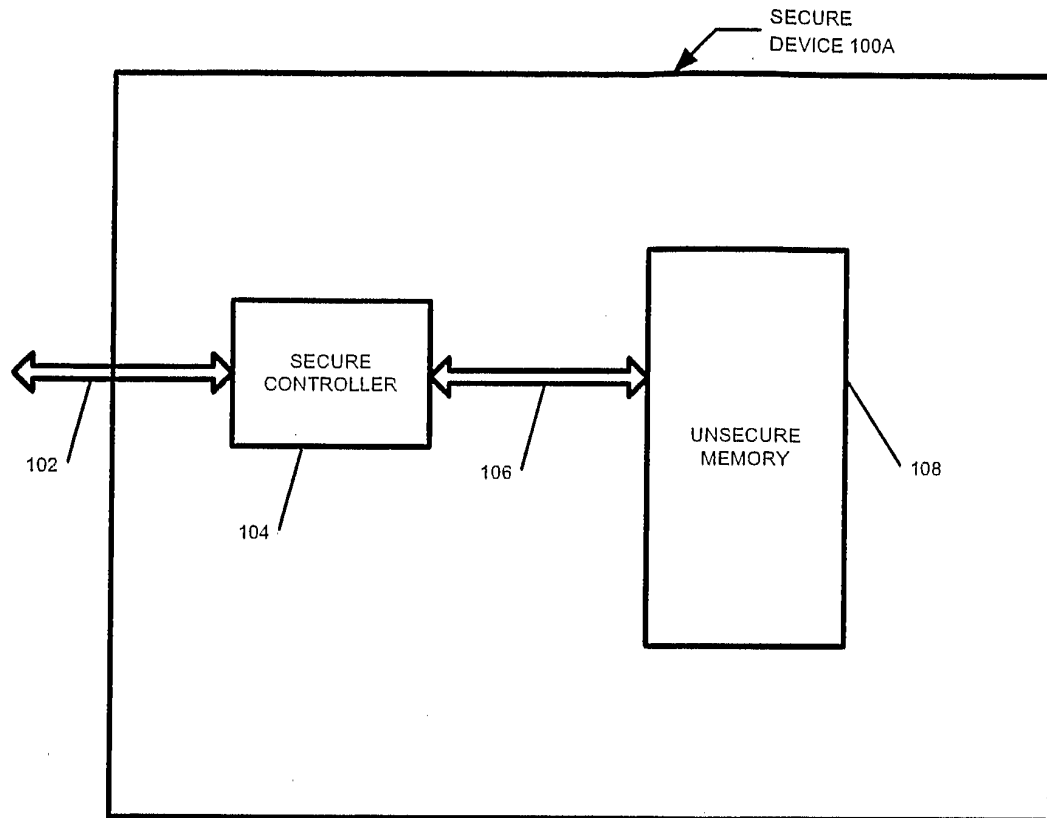


FIG. 1A

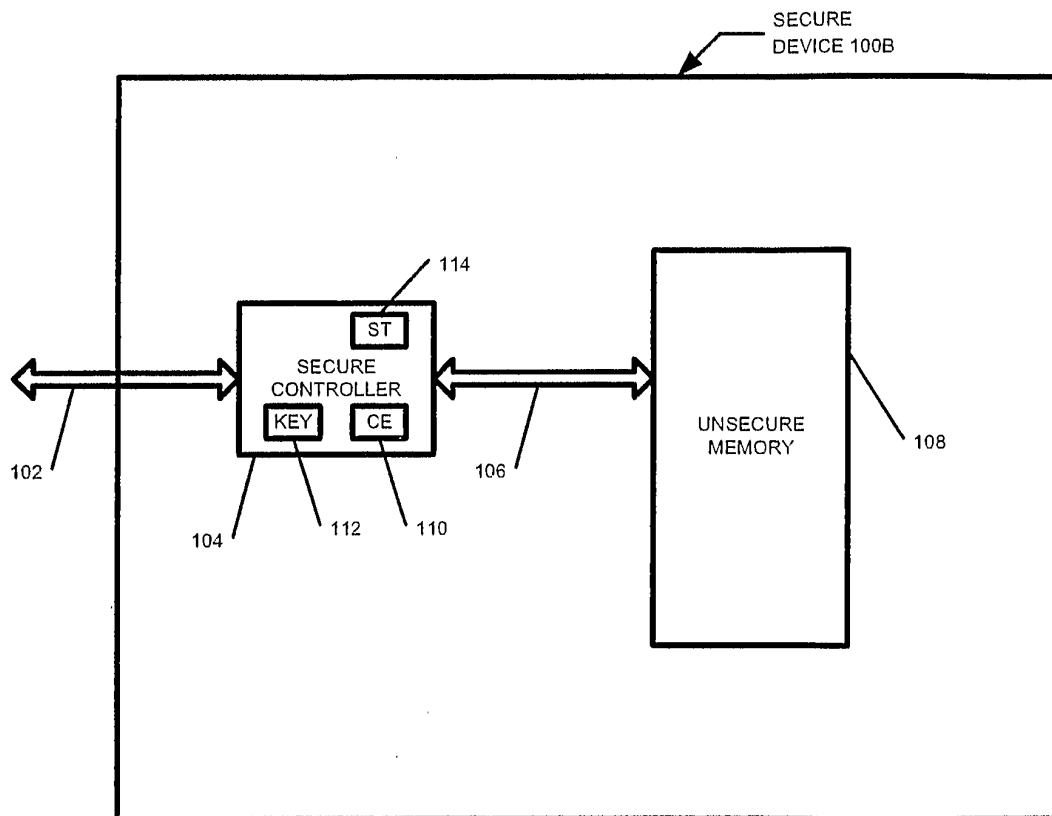


FIG. 1B

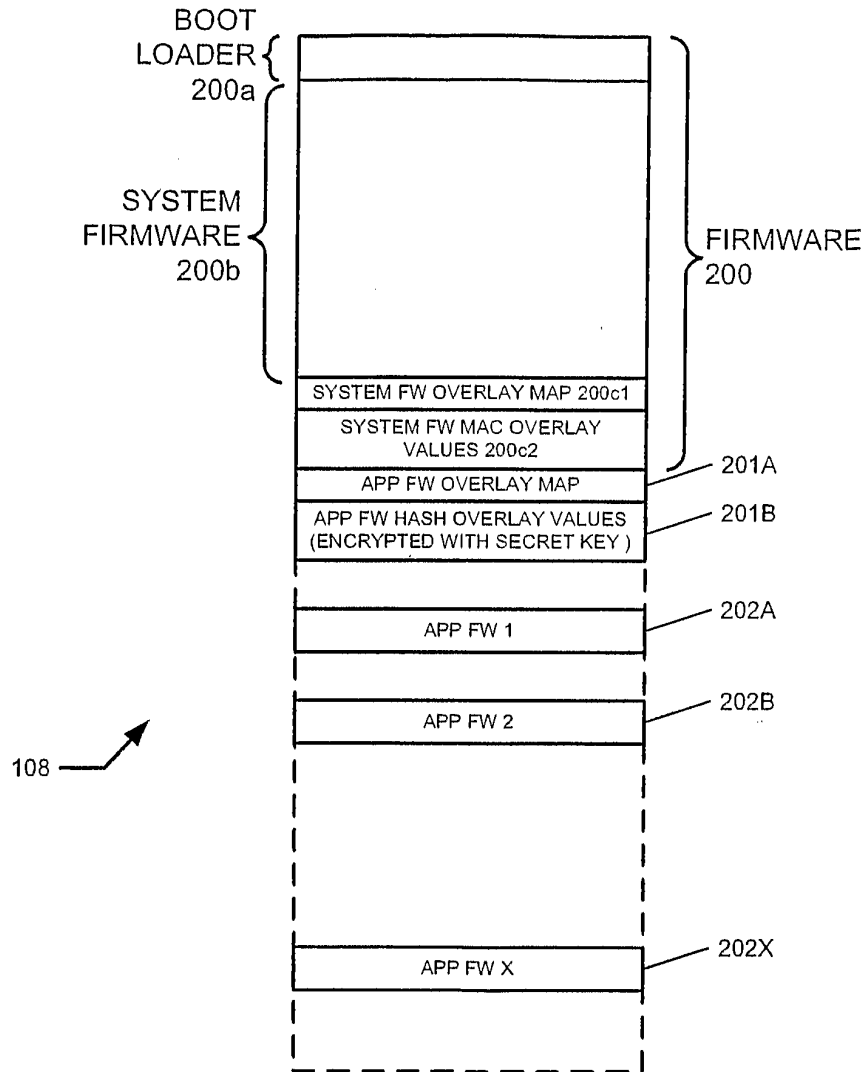
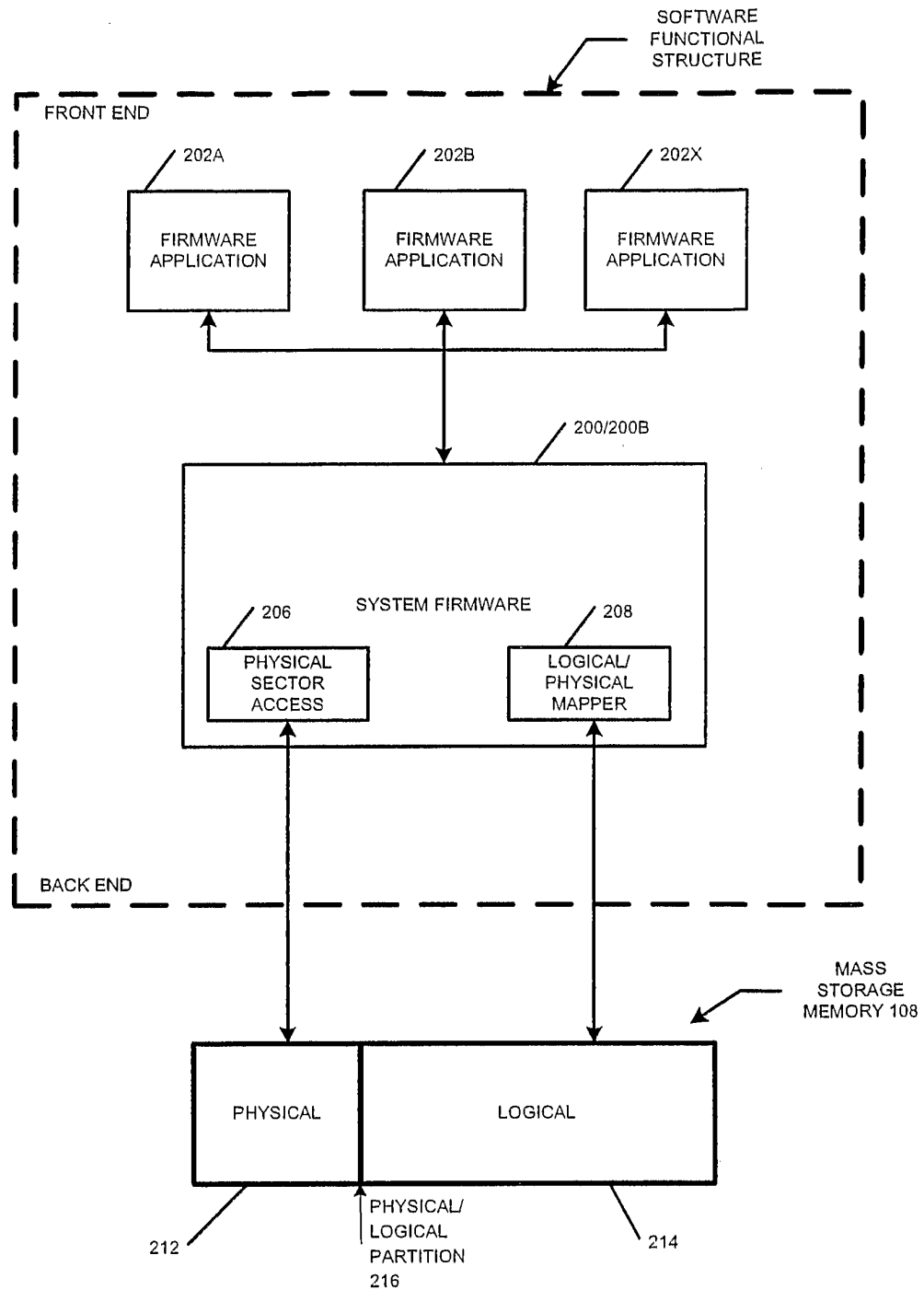


FIG. 2

FIG. 3



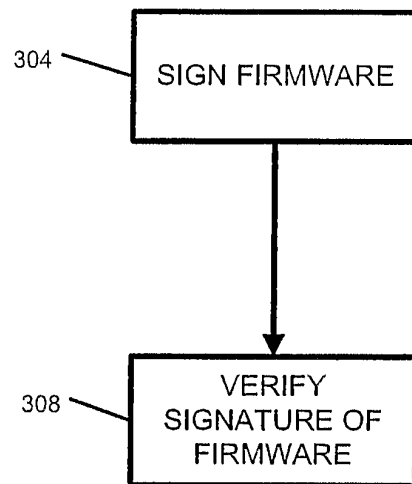


FIG. 4

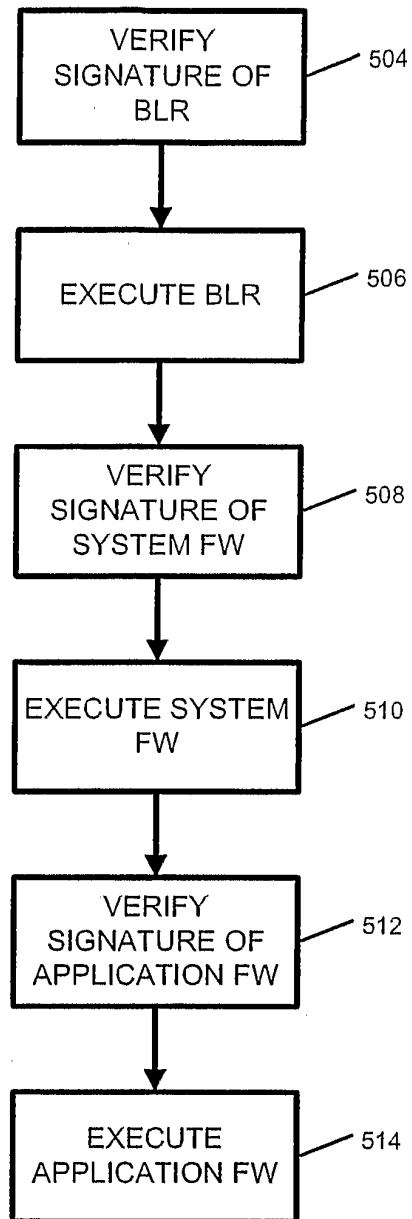


FIG. 5

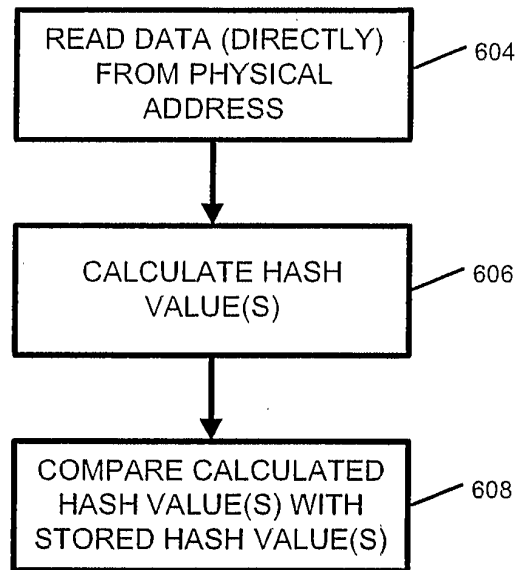


FIG. 6

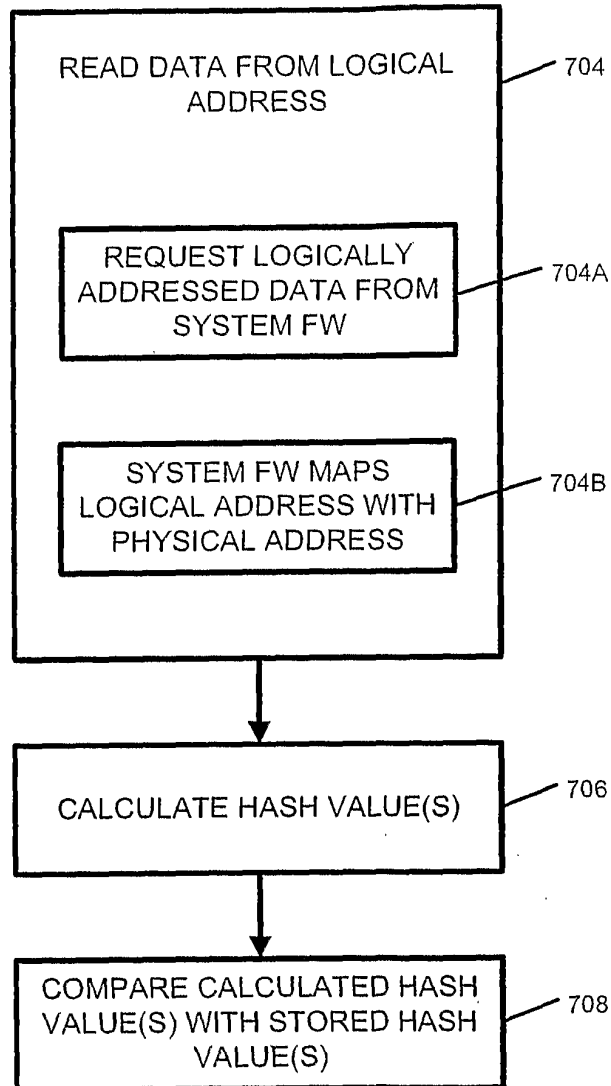


FIG. 7