NON-VOLATILE MEMORY (NVM) MANAGEMENT

TECHNICAL FIELD OF INVENTION

[001] The following relates to non-volatile memory (NVM) management.

BACKGROUND

[002] Non-volatile memory (NVM) such as PCM, STT-MRAM, memristor, etc. is not only scalable in-terms of memory density, but is also less power consuming and persistent. NVM may be perceived in two different ways. Firstly, as a direct way to achieve persistence to in-memory data structures accessible directly to programmers. Secondly, as a byte addressable storage class memory, accessible through a file system interface allowing discretionary access control, mandatory access control, etc.

[003] Currently, the most widely used form of primary storage today is a volatile form of random access memory (e.g. dynamic random access memory). Traditionally, NVM is relatively slower and has less capacity compared to volatile memory and therefore has not been suitable for use as primary storage. Recent technological advancements have given rise to faster and larger capacity NVM.

[004] With the promise of possibly large storage capacity, and accessibility akin to dynamic random access memory (DRAM), it is important to consider how NVM sits in the memory hierarchy. It may be placed in between DRAM and hard disk drives (HDDs), or placed overlapping primary and secondary memory, or possibly be placed replacing both DRAM and HDDs for certain applications.

[005] Also, with persisting data, there is a need to protect unwanted access, malicious or accidental, of information owned by different applications. Hence, security becomes a major consideration when designing logical architectures for NVMs.

[006] In a traditional memory management architecture 100 (e.g. see Figure 1), there is an assumption that object data is temporary. Furthermore, pages are protected but not objects as a single unit. As a result, the traditional memory management architecture 100 is not ideal for NVM because NVM requires named-memory objects and paging is not necessary.

[007] A need therefore exists to provide non-volatile memory (NVM) management methods and/or a NVM that seek to address at least some of the above-mentioned problems.

SUMMARY

[008] According to an aspect of the present invention, there is provided a method of non-volatile memory (NVM) management, comprising: creating an object within the NVM for storing data; and implementing an access control mechanism for the object to provide access control for the stored data.

[009] In an embodiment, the method may further comprise: (i) dividing a logical architecture of the NVM into: an object region comprising the object; a domain region comprising an object descriptor; and an owner region comprising a domain descriptor, and (ii) generating a token comprising: a pointer to the domain descriptor or object descriptor, and access control information to provide access control for the stored data.

[010] In an embodiment, the domain region may further comprise one or more domains, each domain serving as a memory protection boundary of a collection of one or more objects.

[011] In an embodiment, the object descriptor may comprise: an object name, an object size and/or an object structure.

[012] In an embodiment, the domain descriptor may comprise information related to the ownership of the domain.

[013] In an embodiment, the method may further comprise creating a new domain.

[014] In an embodiment, the new domain may be created by an application program that is meant to access the stored data.

[015] In an embodiment, the method may further comprise assigning the object with an address for the pointer to be addressed to the object.

[016] In an embodiment, the method may further comprise maintaining a list of generated tokens held by an application program that is meant to access the stored data.

[017] In an embodiment, the method may further comprise encrypting the stored data.

[018] In an embodiment, each object or domain may have a unique name.

[019] In an embodiment, the tokens may be generated based on the name of the object or domain, physical address of the object or domain, cipher information and/or the access control information.

[020] In an embodiment, the method may further comprise initializing the NVM by: creating a first object for storing key-values, wherein the key comprises an

application program identity and the values comprise the list of generated tokens held by the respective application program; and creating a second object for storing keys used for encrypting the data and/or descriptors.

[021] In an embodiment, the method may further comprise sharing the object across a plurality of application programs by generating a token comprising access control information allowing access across the plurality of application programs.

[022] According to a second aspect of the present invention, there is provided a non-volatile memory (NVM) comprising an object for storing data, wherein an access control mechanism is implemented for the object to provide access control for the stored data.

[023] In an embodiment, the NVM may be divided into: an object region comprising the object; a domain region comprising an object descriptor; and an owner region comprising a domain descriptor, wherein a token is used to provide access control for the stored data, the token comprising: a pointer to the domain descriptor or object descriptor, and access control information.

[024] In an embodiment, the domain region may further comprise one or more domains, each domain serving as a memory protection boundary of a collection of one or more objects.

[025] In an embodiment, the object descriptor may comprise an object name, an object size and/or an object structure; and wherein the domain descriptor may comprise information related to the ownership of the domain.

BRIEF DESCRIPTION OF THE DRAWINGS

[026] Embodiments of the invention will be better understood and readily apparent to one of ordinary skill in the art from the following written description, by way of example only, and in conjunction with the drawings, in which:

[027] Figure 1 is a schematic showing an example of prior art traditional memory management architecture;

[028] Figure 2 is a schematic showing a secure NVM architecture according to an embodiment of the invention;

[029] Figure 3 is a schematic showing a secure NVM architecture according to an embodiment of the invention;

[030] Figure 4 is a flowchart illustrating a method of non-volatile memory (NVM) management and access control for NVM data according to an embodiment of the invention;

[031] Figure 5 is a flowchart illustrating a method of initializing a Capability Manager according to an embodiment of the invention;

[032] Figure 6 is a flowchart illustrating a method of initializing a Key Manager according to an embodiment of the invention;

[033] Figure 7 is a flowchart illustrating a method of creating a new Owner according to an embodiment of the invention;

[034] Figure 8 is a flowchart illustrating a method of deleting an existing Owner according to an embodiment of the invention;

[035] Figure 9 is a flowchart illustrating a method of creating a new domain according to an embodiment of the invention;

[036] Figure 10 is a flowchart illustrating a method of getting a domain according to an embodiment of the invention;

[037] Figure 11 is a flowchart illustrating a method of deleting a domain according to an embodiment of the invention;

[038] Figure 12 is a flowchart illustrating a method of creating a new object according to an embodiment of the invention;

[039] Figure 13 is a flowchart illustrating a method of getting an object according to an embodiment of the invention;

[040] Figure 14 is a flowchart illustrating a method of deleting an object according to an embodiment of the invention;

[041] Figure 15 is a flowchart illustrating a method of reading object content according to an embodiment of the invention;

[042] Figure 16 is a flowchart illustrating a method of writing object content according to an embodiment of the invention;

[043] Figure 17 is a flowchart illustrating a method of generating new capability tokens according to an embodiment of the invention;

[044] Figure 18 is a flowchart illustrating a method of assigning capability tokens to an application capability list according to an embodiment of the invention;

[045] Figure 19 is a flowchart illustrating a method of revoking capability to an application according to an embodiment of the invention;

[046] Figure 20 is a flowchart illustrating a method of non-volatile memory (NVM) management according to an embodiment of the invention; and

[047] Figures 21a and 21b are schematics illustrating an example use of an embodiment of the invention.

DETAILED DESCRIPTION

[048] Embodiments of the present invention will be described, by way of example only, with reference to the drawings. Like reference numerals and characters in the drawings refer to like elements or equivalents.

[049] Some portions of the description which follows are explicitly or implicitly presented in terms of algorithms and functional or symbolic representations of operations on data within a computer memory. These algorithmic descriptions and functional or symbolic representations are the means used by those skilled in the data processing arts to convey most effectively the substance of their work to others skilled in the art. An algorithm is here, and generally, conceived to be a self-consistent sequence of steps leading to a desired result. The steps are those requiring physical manipulations of physical quantities, such as electrical, magnetic or optical signals capable of being stored, transferred, combined, compared, and otherwise manipulated.

[050] Unless specifically stated otherwise, and as apparent from the following, it will be appreciated that throughout the present specification, discussions utilizing terms such as "selecting", "scanning", "calculating", "determining", "replacing", "generating", "initializing", "outputting", or the like, refer to the action and processes of a computer system, or similar electronic device, that manipulates and transforms data represented as physical quantities within the computer system into other data similarly represented as physical quantities within the computer system or other information storage, transmission or display devices.

[051] The present specification also discloses apparatus for performing the operations of the methods disclosed herein. Such apparatus may be specially constructed for the required purposes, or may comprise a general purpose computer or other device selectively activated or reconfigured by a computer program stored in the computer. The algorithms and displays presented herein are not inherently related to any particular computer or other apparatus. Various general purpose machines may be used with programs in accordance with the teachings herein. Alternatively, the construction of more specialized apparatus to perform the required method steps may

be appropriate. The structure of a conventional general purpose computer will appear from the description below.

[052] In addition, the present specification also implicitly discloses a computer program, in that it would be apparent to the person skilled in the art that the individual steps of the method described herein may be put into effect by computer code. The computer program is not intended to be limited to any particular programming language and implementation thereof. It will be appreciated that a variety of programming languages and coding thereof may be used to implement the teachings of the disclosure contained herein. Moreover, the computer program is not intended to be limited to any particular control flow. There are many other variants of the computer program, which can use different control flows without departing from the spirit or scope of the invention.

[053] Furthermore, one or more of the steps of the computer program may be performed in parallel rather than sequentially. Such a computer program may be stored on any computer readable medium. The computer readable medium may include storage devices such as magnetic or optical disks, memory chips, or other storage devices suitable for interfacing with a general purpose computer. The computer readable medium may also include a hard-wired medium such as exemplified in the Internet system, or wireless medium such as exemplified in the GSM, GPRS, 3G or 4G mobile telephone systems. The computer program when loaded and executed on such a general-purpose computer effectively results in an apparatus that implements the steps of the preferred method.

[054] The invention may also be implemented as hardware modules. More particular, in the hardware sense, a module is a functional hardware unit designed for use with other components or modules. For example, a module may be implemented using discrete electronic components, or it can form a portion of an entire electronic circuit such as an Application Specific Integrated Circuit (ASIC). Numerous other possibilities exist. Those skilled in the art will appreciate that the system can also be implemented as a combination of hardware and software modules.

[055] Embodiments of the present invention relate to non-volatile memory (NVM) management and access control for NVM data. In particular, embodiments of the present invention are directed to NVM architectures for secure computing based on Secure Object Stores (SOS).

[056] In an embodiment of the present invention, there is provided method(s) of organizing data that is created and stored in NVM, and method(s) of providing access

control and memory protection boundaries for data that is created and stored in NVM. In an exemplary implementation, NVM (that is sitting on the memory bus) is considered to be primary as well as secondary memory. In this manner, an application programmer may be able to create objects within the non-volatile region and make use of such objects to store application data, file data, etc. while at the same time consider them as named entities upon which access control mechanisms can be implemented. In this exemplary implementation, the logical architecture of the NVM is divided into three zones (or regions): an object zone, a domain zone and an owner zone.

[057] The object zone is the main data repository containing named memory objects created by, and accessible to programmers. Optionally, encryption / decryption may be done on read / write. In the foregoing description, "memory object" or "object" refers to a location in the NVM 201 capable of storing data, and "object data" refers to the data stored at that location.

[058] In the domain zone, each object is contained within a domain. Domains are the protection boundaries of objects, similar to directories in traditional file systems. In other words, domains are containers for objects which are used to provide memory protection boundaries to processes during runtime and also act as access control boundaries. Metadata related to domains (e.g. domain names, objects under domains, etc.), access control information, object-domain relationship, etc. may be stored. Domain names are preferably unique.

[059] In the owner zone, metadata related to ownership of domains, access control information, etc. are stored. In other words, the owner zone contains a list of object owners inside an NVM module. Owners of domains can be both system users and application programs. If the owners are system users who own memory domains, all memory domains created by applications are by default owned by the system user under which the application is run. If the owners are application programs, the application programs that create domains / objects by default are provided with access capabilities.

[060] Using the above described organization and management of NVM data, embodiments of the invention provide multiple technical advantages. Firstly, application programmers can be provided with a clean and consistent interface for securely creating, writing and accessing data.

[061] Secondly, users can apply security mechanisms such as discretionary access control, capability based access control, etc. Capability based access control is advantageous because capability distribution is completely transparent, the system does

not force any particular protection structure upon users, and it is possible to create restricted views of objects and revoke access. In this manner, only authorized users / applications access the persistent data after a system reboots. Domains allow administrators / system users / applications to implement explicit access control mechanisms. Objects or all objects under a domain can be encrypted by developing relevant APIs / system calls.

[062] Thirdly, processes storing application data on NVM can ensure memory protection during execution through domain boundaries.

[063] Fourthly, there is portability due to the ease of sharing data / objects created on NVM. This is because objects provide sufficient abstraction to allow any mechanism to be used to store and access them. Furthermore, capabilities can be easily shared across applications and platforms to allow object and domain sharing.

[064] Finally, there is scalability as object stores using capability tokens and a namespace allow easy expansion of a local store to global / distributed stores. Capability tokens can be easily expanded to encode compute / data node identifiers along with others such as object / domain name, physical address, and access rights. Such tokens can have a global reference, which can be used for distributed in-memory data stores.

[065] Embodiments of the invention allow two design goals to be achieved: a) the provision of a memory organization that allows the use of any NVM management methodology; and b) the provision of access control / memory protection capabilities for individual applications while maintaining object ownership with users.

[066] A secure NVM storage system according to an embodiment of the invention makes use of capability based access control for objects / domains. Capability tokens for each domain / object encodes an object reference (virtual address) and access control information. The advantage of using capability based systems for NVM is that: a) it allows persistent objects to be referred through tokens (similar to a virtual address) in programs, which provides memory protection boundaries to processes accessing them; b) it allows users to view persistent memory uniformly across applications, so that data can be effectively shared by sharing capability tokens; c) persisting capability tokens allow applications to access data that it stored on NVM directly, across application and system restarts; and d) it encodes access control information.

[067] As such, embodiments of the invention, which are implemented based on object stores, are able to: (i) provide interfaces that make programming with persistent

data easier, (ii) provide high level abstractions (which makes it relatively easier to share across applications), (iii) provide a common interface on which objects can be shared across different platforms, and (iv) allow security and privacy mechanisms to be applied at object granularity.

[068] Figure 2 is a schematic showing a secure NVM architecture according to an embodiment of the invention. In this embodiment, the NVM 201 acts as both primary and secondary memory. The NVM 201 has three key zones / regions: owner 202, domain 204 and object 206.

[069] The owner zone 202 stores a list of owners with unique names. Under each owner, there is a list of optionally encrypted domain structures owned by it. The encrypted domain structures include, but are not limited to: domain names, physical location (address) of the domain, and cipher information (if any) for the domain.

[070] The domain zone 204 stores a list of domains created by various applications and their metadata. The metadata for each domain include, but are not limited to: object name, object size, and optionally encrypted object structures. The object structures contain the physical location (address) of the object and cipher information (if any) for the object.

[071] The object zone 206 is the main data repository. The repository optionally encrypts the objects on read / write. The start address of each block is based on memory block boundaries.

[072] The Operating System Memory Management Module 208 and the processor memory management unit (MMU) 210 together provide for the management of the three zones 202 / 204 / 206.

[073] Objects provide abstraction to allow any mechanism to be used to store and access them.

[074] For instance:

Objects can be stored as byte addressable data blocks with pointers addressing them;

Objects storage can be implemented using any persistent programming environment (e.g. Mnemosyne); and/or

Objects can be stored using journaling based systems like CDDS to provide transactional properties such as consistency, durability, and atomicity.

[075] Security for the three zones 202 / 204 / 206 is provided using a local node security gateway module 212, which is part of the operating system kernel. Figure 3 is a

schematic showing a secure NVM architecture that is similar to the one in Figure 2, but with a particular emphasis on the local node security gateway module 212 and three zones 202 / 204 / 206.

[076] The local node security gateway module 212 comprises four parts / sub-modules:

[077] Capability Manager 214: The Capability Manager 214 maintains a list of capabilities for specific processes. In other words, the Capability Manager 214 maintains a capability list (C-List) for every application for a specific owner. The C-List for an application is a list of capability tokens held by the application. The Capability Manager 214 is responsible for managing access control in the owner 202, domain 204 and object 206 zones of the NVM as well as integrating other parts such as a Capability Generator 216 and a Key Manager 218 to generate tokens, store tokens and authenticate tokens. The Capability Manager 214 can request the Key Manager 218 to create a key for a specific domain or object identifier. Capability tokens and the application capability list are stored under a capability object created by the Capability Manager 214. The capability object is maintained under a capability domain.

[078] A capability object is a key-value store that maintains the capability list for every application of an owner. The key may be derived from the combination of owner and application identifiers. The value is a list structure of capabilities that the application has access to. A capability is an encrypted token of the following: physical address of a domain or an object metadata, and access information for the domain or object.

[079] Capability Generator 216: The Capability Generator 216 generates unique tokens based on object / domain name, physical location addresses, cipher information and access rights. The Capability Generator 216 also maps capabilities to offset from start address of the domain zone 204 or object zone 206. The Capability Translator 217 decodes capability tokens to obtain the physical offsets, domain/object names, access rights, etc.

[080] Cipher Engine 220: The Cipher Engine 220 comprises an Encryption Engine 322 (see Figure 3). The Encryption Engine 322 provides encryption/decryption for domain capabilities / metadata, object capabilities / metadata, and objects. The Cipher Engine 220 further comprises cipher modules such as a Random Number Generator 324 and a Message Digest Engine 326 (see Figure 3).

[081] Key Manager 218: The Key Manager 218 generates keys for the Encryption Engine 322. The Key Manager 218 also acts as a repository (i.e. database)

of keys for symmetric encryption algorithms (e.g. AES-256) and manages the lifecycle of keys. In other words, the Key Manager 218 is the key lifecycle manager for the Local Node Security Gateway 212. All keys are further stored under the Key Manager 218 owned domain on the NVM.

[082] The Local Node Security Gateway 212, together with the organization of NVM into three distinct zones (i.e. object zone 206, domain zone 204 and owner zone 202), support object based storage as well as allow security mechanisms such as access control and encryption to be implemented.

[083] In summary, the owner zone 202 contains a list of domain descriptors/metadata, the domain zone 204 contains a list of object descriptors/metadata, and the objects are the containers of the actual data. Capabilities are encrypted tokens that contain a pointer to a domain or an object descriptor and the access rights to the domain or object. Domains (which are identified by the domain descriptors) are the primary memory protection boundary of the object collection owned by the domain. A domain comprises a collection of object descriptors, which is the primary memory protection boundary of the object.

[084] Figure 4 is a flowchart 400 illustrating a method of non-volatile memory (NVM) management and access control for NVM data according to an embodiment of the invention. At step 402, the NVM is initialized. During system initialization, for e.g. during installation of an operating system, the Capability Manager 214 and Key Manager 218 of the Local Node Security Gateway 212 create their corresponding domains and objects. The Capability Manager 214 creates a capability domain under which it creates a capability object. The capability object can be used as a key-value or an index store containing an application ID as key and capability lists as values. The application ID is a unique number assigned to each application installed on the system. The application ID contains a capability list (C-List). The C-List is a data-structure containing all active capability tokens for the application. The capability object is a secure container that is used to persist access control information for installed applications.

[085] Similar to Capability Manager 214, the Key Manager 218 also creates a key domain and key object(s) on the NVM upon initialization (i.e. step 402). The key manager uses these key object(s) as repository for keys used for encrypting metadata, data or both.

[086] Figure 5 is a flowchart 500 illustrating a method of initializing the Capability Manager 214 according to an embodiment of the invention. At step 502, a

Capability Manager process and Root owner check if there is a Root owner in the owner zone / region 202 of the NVM. If there is not, at step 504, a Root Owner entry is created in the owner region 202 of the NVM.

[087] If there is, at step 506, the Capability Manager process checks if there is a Capability domain in the Domain list of the Root owner in the owner region 202 of the NVM. If there is no Capability domain or continuing from step 504, at step 508, a Capability domain entry is created in the Domain list of the Root Owner in the owner region 202 of the NVM and a Capability domain entry is created in the domain zone / region 204 of the NVM.

[088] If there is a Capability domain, at step 510, the Capability Manager process checks if there is a Capability object in the Object list of the Capability domain of the Root owner in the domain region 204 of the NVM. If there is no Capability object or continuing from step 508, at step 512, a Capability object entry is created in the Object list of the Capability domain of the Root owner in the domain region 204 of the NVM to reserve the memory region in the object zone / region 206 of the NVM.

[089] If there is a Capability object or continuing from step 512, at step 514, a "success" flag is returned to the Capability Manager 214. The Capability object is used by the Capability Manager 214 as a Capability-List (C-List) Key-Value Store.

[090] Figure 6 is a flowchart 600 illustrating a method of initializing the Key Manager 218 according to an embodiment of the invention. At step 602, the Key Manager process and Root owner check if there is a Root owner in the owner region 202 of the NVM. If there is no Root owner, at step 604, a Root Owner entry is created in the owner region 202 of the NVM.

[091] If there is a Root owner, at step 606, the Key Manager process checks if there is a Key Manager domain in the Domain list of the Root owner in the owner region 202 of the NVM. If there is no Key Manager domain or continuing from step 604, at step 608, a Key Manager domain entry is created in the Domain list of the Root Owner in the owner region 202 of the NVM and a Key Manager domain entry is created in the domain region 204 of the NVM.

[092] If there is a Key Manager domain, at step 610, the Key Manager process checks if there is a Key Manager object in the Object list of the Key Manager domain of the Root owner in the domain region 204 of the NVM. If there is no Key Manager object or continuing from step 608, at step 612, a Key Manager object entry is created in the

Object list of the Key Manager domain of the Root owner in the domain region 204 of the NVM to reserve the memory region in the object region 206 of the NVM.

[093] If there is a Key Manager object or continuing from step 612, at step 614, a "success" flag is returned to the Key Manager 218. The Key Manager object is used by the Key Manager 218 as a Key-Value Store for encryption keys.

[094] The Key Manager object is a simple key-value store of key-identifiers (usually derived values) and encryption keys (usually randomly generated values). The key-identifiers are saved in the cipher information of each Domain and Object metadata/descriptors. The encryption keys for the capability list entries are also stored. The key-ID's can be derived from the owner-application-other_data and the keys can be randomly generated.

[095] Once the capability repository and key repository are set up, the applications can now create / access and delete domains. Under each domain, the applications can create / access / write and delete objects.

[096] Turning back to Figure 4, at step 404, a new Owner is created (in a Privileged Mode). Figure 7 is a flowchart 700 illustrating a method of creating a new Owner according to an embodiment of the invention. At step 702, the Root user initiates a Create Owner process through suitable administrative tools and passes the Owner ID/name. Also, the Capability Manager 214 scans the owner region 202 to check if the owner already exists. If the owner exists, at step 704, the Capability Manager 214 returns an "error" flag. If the owner does not exist, at step 706, the Capability Manager 214 creates a new owner entry in the owner region 202 of the NVM.

[097] Thereafter, at step 708, the Capability Manager 214 returns a "successful operation" flag to the Root user.

[098] Besides creating a new Owner, an existing Owner may be deleted. Figure 8 is a flowchart 800 illustrating a method of deleting an existing Owner according to an embodiment of the invention. At step 802, the Root user initiates a Delete Owner process through suitable administrative tools and passes the Owner ID/name. Also, the Capability Manager 214 scans the owner region 202 to check if the owner already exists. If the owner does not exist, at step 804, the Capability Manager 214 returns an "error" flag. If the owner exists, at step 806, the Capability Manager 214 scans each entry in the domain list of the owner in the owner region 202 and deletes each entry in the object list of each domain in the domain region 204 and its corresponding capabilities in the capability object.

[099] Thereafter, at step 808, the Capability Manager 214 deletes each entry in the domain list of the owner in the owner region 202, its corresponding data structures in the domain region 204, and its corresponding capabilities in the capability object. Subsequently, at step 810, the Capability Manager 214 returns a "successful operation" flag to the Root user.

[0100] Turning back to Figure 4 again, at step 406, the application creates a new domain. This step includes updating an owner domain list, creating entries in a domain region 204 and creating entries in the application's C-List.

[0101] Figure 9 is a flowchart 900 illustrating a method of creating a new domain according to an embodiment of the invention. At step 902, the application initiates a Create Domain process and passes the Owner ID, Application ID and Domain Name to be created. Next, at step 904, the Capability Manager 214 scans the Domain list of the Owner to check if the domain is already present in the Owner's Domain List in the owner region 202 of the NVM. If it is not present, at step 906, the Capability Manager 214 creates an entry in the Domain list of the Owner in the owner region 202 of the NVM. The Capability Manager 214 also creates a new entry in the domain region 204 to identify the new domain.

[0102] Thereafter, at step 910, the Capability Generator 216 creates a full access capability for the application. The Capability Manager 214 stores the capability in the capability object. Subsequently, at step 912, the Capability Manager 214 returns the Capability Token to the application.

[0103] On the other hand, if at step 904, it is found that the Domain is already present, then at step 908, the Capability Manager 214 scans the Capability object of the Capability domain and checks if a capability for the Application ID is already present. If it is present, at step 912, the Capability Manager 214 returns the Capability Token to the application. Otherwise, at step 914, the Capability Manager 214 returns an "error" flag to the application, indicating that the application has no access to the existing capability.

[0104] After creating a new domain, a domain can be fetched and deleted.

[0105] Figure 10 is a flowchart 1000 illustrating a method of getting a domain according to an embodiment of the invention. At step 1002, the application initiates a Get Domain process and passes the Owner ID, Application ID and Domain Name to be fetched. Next, at step 1004, the Capability Manager 214 scans the Domain list of the Owner to check if the domain is already present in the Owner's Domain List in the owner region 202 of the NVM.

[0106] If it is not present, at step 1006, the Capability Manager 214 returns an "error" flag to the application. If it is present, at step 1008, the Capability Manager 214 scans the capability object of the capability domain and checks if a capability for the Application ID is already present. If it is present, at step 1010, the Capability Manager 214 returns the Capability Token to the application. Otherwise, at step 1012, the Capability Manager 214 returns an "error" flag to the application, indicating that the application has no access to the existing capability.

[0107] Figure 11 is a flowchart 1100 illustrating a method of deleting a domain according to an embodiment of the invention. At step 1102, an Application or Owner (using special tools) initiates a Delete Domain process and passes the Owner ID, Application ID, and Domain Capability to be deleted. Next, at step 1104, the Capability Manager 214 scans the Domain list of the Owner to verify if the domain is present. If the domain is not present, at step 1106, the Capability Manager 214 returns an "error" flag to the Application or Owner.

[0108] If the domain is present (and the deletion process is Application initiated), at step 1108, the Capability Manager 214 scans the capability object of the capability domain to verify if the application has a Delete Capability for the domain. If the application does not have a Delete Capability for the domain, the Capability Manager 214 returns an "error" flag to the Application at step 1106. If the application has a Delete Capability for the Domain, at step 1110, the Capability Manager 214 deletes the object entries of the domain in the domain region 204 of the NVM. All the capabilities corresponding to these object entries in the capability object of the capability domain are deleted.

[0109] If the domain is present (and the deletion process is Owner initiated), then at step 1110, the Capability Manager 214 deletes the object entries of the domain in the domain region 204 of the NVM. All the capabilities corresponding to these object entries in the capability object of the capability domain are deleted.

[0110] In any case, after step 1110, at step 1112, the Capability Manager 214 deletes the domain entry in the Owner's Domain list in the owner region 202 of the NVM. The domain entry in the domain region 204 of the NVM is also deleted. All the capabilities corresponding to the domain in the capability object of the capability domain are deleted. Subsequently, at step 1114, the Capability Manager 214 returns a "success" flag to the Owner or Application.

[0111] Turning back to Figure 4 again, at step 408, the application creates a new object. This step includes updating a domain object list, creating entries in an object region and creating entries in the application's C-List.

[0112] Figure 12 is a flowchart 1200 illustrating a method of creating a new object according to an embodiment of the invention. At step 1202, the application initiates a Create Object process and passes the Application ID, Domain Capability and Object Name to be created. At step 1204, the Capability Manager 214 decodes the Domain Capability Token and verifies if it has a Write capability for the domain. The Capability Manager 214 also verifies if the application has the Domain Capability on the application's Capability List in the capability object of the capability domain. If not, at step 1206, the Capability Manager 214 returns an "error" flag to the application.

[0113] If the application has the Domain Capability, at step 1208, the Capability Manager 214 checks if the Object Name is available in the Object List of the domain in the domain region 204 of the NVM.

[0114] If the Object Name is not available, at step 1206, the Capability Manager 214 returns an "error" flag to the application. If the Object Name is available, at step 1210, the Capability Manager 214 creates a new Object entry in the object list of the domain in the domain region 204 of the NVM.

[0115] Thereafter, at step 1212, the Capability Generator 216 creates a new Object capability. Also, the Capability Manager 214 updates the application's Capability List in the capability object of the capability domain. Subsequently, at step 1214, the Capability Manager 214 returns capability to the application.

[0116] After creating a new object, the object can be fetched and deleted.

[0117] Figure 13 is a flowchart 1300 illustrating a method of getting an object according to an embodiment of the invention. At step 1302, the application initiates a Get Object process and passes the Application ID, Domain Capability and Object Name. Thereafter, at step 1304, the Capability Manager 214 decodes the Domain Capability token and verifies if it has a Read capability for the domain. The Capability Manager 214 also verifies if the application has the Domain Capability on the application's Capability List in the capability object of the capability domain.

[0118] If not, at step 1306, the Capability Manager 214 returns an "error" flag to the application. If yes, at step 1308, the Capability Manager 214 checks if the Object Name is available in the Object List of the domain in the domain region 204 of the NVM.

[0119] If the Object Name is not available, at step 1306, the Capability Manager 214 returns an "error" flag to the application. If the Object Name is available, at step 1310, the Capability Manager 214 fetches the Capability Token from the application's Capability List if available in the capability object of the capability domain.

[0120] If the Capability Token is not available, at step 1306, the Capability Manager 214 returns an "error" flag to the application. If the Capability Token is available, at step 1312, the Capability Manager 214 returns capability to the application.

[0121] Figure 14 is a flowchart 1400 illustrating a method of deleting an object according to an embodiment of the invention. At step 1402, the application initiates a Delete Object process and passes the Application ID and Object Capability token. Thereafter, at step 1404, the Capability Manager 214 decodes the Object Capability token and verifies if a Delete capability is present. The Capability Manager 214 also checks if the Object Capability is present in the application's Object Capability list. If the Delete capability is not present or the Object Capability is not present, at step 1406, the Capability Manager 214 returns an "error" flag to the application.

[0122] If the Delete capability is present and the Object Capability is present, at step 1408, the Capability Manager 214 deletes the Object entry in the Object List of the domain in the domain region 204 of the NVM.

[0123] Thereafter, at step 1410, the Capability Manager 214 deletes the Object Capability in the application's Capability List in the capability object of the capability domain. Subsequently, at step 1412, the Capability Manager 214 clears the contents of the object in the object region 206 of the NVM. Next, at step 1414, the Capability Manager 214 returns a "success" flag to the application.

[0124] Figure 15 is a flowchart 1500 illustrating a method of reading object content according to an embodiment of the invention. At step 1502, the application initiates a Read Object process and passes the Application ID, Object Capability token, and Object Buffer.

[0125] Thereafter, at step 1504, the Capability Manager 214 checks if the Object Capability is available in the application's Capability List. If the Object Capability is not available, at step 1506, the Capability Manager 214 returns an "error" flag to the application. If the Object Capability is available, at step 1508, the Capability Manager 214 decodes the Object Capability token and verifies if it has a Read Capability. If it does not have a Read Capability, at step 1506, the Capability Manager 214 returns an "error" flag to the application.

[0126] If it has a Read Capability, at step 1510, the Read Object process writes the object from the object region 206 of the NVM to the object Buffer of the application. Thereafter, at step 1512, the Capability Manager 214 returns a "success" flag to the application.

[0127] Figure 16 is a flowchart 1600 illustrating a method of writing object content according to an embodiment of the invention. At step 1602, the application initiates a Write Object process and passes the Application ID, Object Capability token, and Object Buffer. Thereafter, at step 1604, the Capability Manager 214 checks if the Object Capability is available in the application's Capability List. If the Object Capability is not available, at step 1606, the Capability Manager 214 returns an "error" flag to the application. If the Object Capability is available, at step 1608, the Capability Manager 214 decodes the Object Capability token and verifies if it has a Write capability.

[0128] If it does not have a Write capability, at step 1606, the Capability Manager 214 returns an "error" flag to the application. If it has a Write Capability, at step 1610, the Write Object process writes the contents of the Object Buffer of the application to the object region 206 of the NVM. Thereafter, at step 1612, the Capability Manager 214 returns a "success" flag to the application.

[0129] In various implementations, the application that creates a domain or an object initially owns the full access rights to that domain. The access rights include "Read", "Write", "Delete" and "Share". Table 1 provides details on each access right / privilege in the context of domain and object.

[0130] Turning back to Figure 4, at step 410, the application shares the object. Apart from creating / accessing objects, it is also possible to share objects across different applications. This can be done using the get, put and revoke capability shown in Figures 17 to 19.

[0131] In this way, an application wanting to share a specific object or the entire domain gets a new capability token for the object along with the access privilege. The capability manager 214 first checks if the requesting application has a share privilege on the domain / object for which it is requesting. Once authorized, the capability manager 214 requests the capability generator 216 to create a new capability token and the token is returned back to the application. The application can now share only the token across to other application(s). A Put capability assigns a capability token to an application ID (applications C-List) while a Revoke capability removes the capability ID from the application C-List.

[0132] Figure 17 is a flowchart 1700 illustrating a method of generating new capability tokens according to an embodiment of the invention. At step 1702, the application initiates a Get Capability process and passes the Application ID, the Object or Domain Capability, and the Access Privilege that it wants that Capability to have. Next, at step 1704, the Capability Manager 214 decodes the Object or Domain Capability and verifies if the application has Share access rights. The Capability Manager 214 also checks if the Capability token is available on the application's capability list.

[0133] If the application does not have Share access rights or the Capability token is not available, at step 1706, the Capability Manager 214 returns an "error" flag to the application. If the application has Share access rights and the Capability token is available, at step 1708, the Capability Generator 216 generates a new Capability token based on the access rights requested by the application and returns that token to the application.

[0134] Figure 18 is a flowchart 1800 illustrating a method of assigning capability tokens to an application capability list according to an embodiment of the invention. At step 1802, the application initiates a Put Capability process and passes the Application ID, the application's Object or Domain Capability, the Capability token for Sharing, and the Target Application ID to share the Capability token to. Next, at step 1804, the Capability Manager 214 decodes the Object or Domain Capability and verifies if the application has Share access rights. The Capability Manager 214 also checks if the Capability token is available on the application's capability list. If the application does not have Share access rights or the Capability token is not available, at step 1806, the Capability Manager 214 returns an "error" flag to the application.

[0135] If the application has Share access rights and the Capability token is available, at step 1808, the Capability Generator 216 checks if the Target Application ID is valid. If it is not valid, at step 1806, the Capability Manager 214 returns an "error" flag to the application. If it is valid, at step 1810, the Capability Manager 214 updates the target application's capability list in the capability object of the capability domain. Next, at step 1812, the Capability Manager 214 returns a "success" flag to the application.

[0136] Figure 19 is a flowchart 1900 illustrating a method of revoking capability to an application according to an embodiment of the invention. At step 1902, the application initiates a Delete Capability process and passes the Application ID, the application's Object or Domain Capability, and the Target Application ID to delete the

Capability token from. Next, at step 1904, the Capability Manager 214 decodes the Object or Domain Capability and verifies if the application has Share access rights. The Capability Manager 214 also checks if the Capability token is available on the application's capability list. If the application does not have Share access rights or the Capability token is not available, at step 1906, the Capability Manager 214 returns an "error" flag to the application. If the application has Share access rights and the Capability token is available, at step 1908, the Capability Manager 214 deletes the capability from the target application's capability list in the capability object of the capability domain. Next, at step 1910, the Capability Manager 214 returns a "success" flag to the application.

[0137] Methods of NVM management have been described above. An exemplary method of NVM management may involve the following steps. With reference to Figure 20, which is a flowchart illustrating an exemplary method of NVM management, a first step (step 2002) involves creating an object within the NVM for storing data. A second step (step 2004) involves implementing an access control mechanism (as described above) for the object in order to provide access control and security for the stored data.

[0138] It will be appreciated by a person skilled in the art that numerous variations and/or modifications may be made to the present invention as shown in the specific embodiments without departing from the spirit or scope of the invention as broadly described. The present embodiments are, therefore, to be considered in all respects to be illustrative and not restrictive.

[0139] Figures 21a and 21b are schematics illustrating an example use of an embodiment of the invention. In particular, Figure 21a shows the implementation of access control on memory objects using memcached. As known by persons skilled in the art, memcached is a general-purpose distributed memory caching system. In this example, a number of web applications are using memcached on its back-end architecture. When implementing an embodiment of the invention on memcached, there is proper access control such that only the trusted web application, i.e. the one that holds the capability token, can access a specific area in the memcached. If there are malwares accessing the memcached, it is contained to its own domain, and cannot access the trusted web application's domain objects.

[0140] Figure 21b shows the process flow when using an embodiment of the invention with memcached. In this example, getting the domain capability for a web

application is done only once during the first access request. Subsequent requests can make use of cached capabilities and/or cached access decisions. As the capability stays on the object store for future checking of access control, system overhead is kept to a minimum.

[0141] Another example use of an embodiment of the invention is where a hypervisor can manage the memory (NVM). As known by persons skilled in the art, a hypervisor creates and runs virtual machines so that multiple operating systems can share a single hardware host. In this example, the hypervisor assigns owners as the subscribers to the system that require virtual machines. Every virtual machine (guest OS) that is assigned to the owners can have their own domains, which is usually tied to the applications that run on each guest OS. The owners can have sub-owners (which is a simple owner that shares its domain capabilities to the parent owner) for each guest operating system. In this manner, embodiments of the invention allow data isolation in the NVM for hypervisors such that the domains owned by each guest OS are not visible on other guest OSes.

CLAIMS

1. A method of non-volatile memory (NVM) management, comprising:
creating an object within the NVM for storing data; and
implementing an access control mechanism for the object to provide access control for the stored data.

2. The method as claimed in claim 1, further comprising:
dividing a logical architecture of the NVM into:
an object region comprising the object;
a domain region comprising an object descriptor; and
an owner region comprising a domain descriptor, and
generating a token comprising:
a pointer to the domain descriptor or object descriptor, and
access control information to provide access control for the stored data.

3. The method as claimed in claim 2, wherein the domain region further comprises one or more domains, each domain serving as a memory protection boundary of a collection of one or more objects.

4. The method as claimed in claim 2 or 3, wherein the object descriptor comprises: an object name, an object size and/or an object structure.

5. The method as claimed in any one of claims 2 to 4, wherein the domain descriptor comprises information related to the ownership of the domain.

6. The method as claimed in claim 3, further comprising creating a new domain.

7. The method as claimed in claim 6, wherein the new domain is created by an application program that is meant to access the stored data.

8. The method as claimed in any one of claims 2 to 7, further comprising assigning the object with an address for the pointer to be addressed to the object.

9. The method as claimed in any one of claims 2 to 8, further comprising maintaining a list of generated tokens held by an application program that is meant to access the stored data.

10. The method as claimed in any one of the preceding claims, further comprising encrypting the stored data.

11. The method as claimed in any one of claims 3 to 10, wherein each object or domain has a unique name.

12. The method as claimed in claim 11, wherein the tokens are generated based on the name of the object or domain, physical address of the object or domain, cipher information and/or the access control information.

13. The method as claimed in any one of claims 9 to 12, further comprising initializing the NVM by:

creating a first object for storing key-values, wherein the key comprises an application program identity and the values comprise the list of generated tokens held by the respective application program; and

creating a second object for storing keys used for encrypting the data and/or descriptors.

14. The method as claimed in any one of claims 2 to 13, further comprising sharing the object across a plurality of application programs by generating a token comprising access control information allowing access across the plurality of application programs.

15. A non-volatile memory (NVM) comprising an object for storing data, wherein an access control mechanism is implemented for the object to provide access control for the stored data.

16. The NVM as claimed in claim 15, wherein the NVM is divided into:

an object region comprising the object;

a domain region comprising an object descriptor; and

an owner region comprising a domain descriptor,

wherein a token is used to provide access control for the stored data, the token comprising:

a pointer to the domain descriptor or object descriptor, and

access control information

17. The NVM as claimed in claim 16, wherein the domain region further comprises one or more domains, each domain serving as a memory protection boundary of a collection of one or more objects.

18. The NVM as claimed in claim 16 or 17, wherein the object descriptor comprises an object name, an object size and/or an object structure; and wherein the domain descriptor comprises information related to the ownership of the domain.