

# Non-Volatile Memory and Disks: Avenues for Policy Architectures

Kevin Butler, Stephen McLaughlin, and Patrick McDaniel

## ABSTRACT

As computing models change, so do the demands on storage. Distributed and virtualized systems introduce new vulnerabilities, assumptions, and performance requirements on disks. However, traditional storage systems have very limited capacity to implement needed “advanced storage” features such as integrity and data isolation. This is largely due to the simple interfaces and limited computing resources provided by commodity hard-drives. A new generation of storage devices affords better opportunities to meet these new models, but little is known about how to exploit them. In this paper, we show that the recently introduced fast-access non-volatile RAM-enhanced hybrid (HHD) disk architectures can be used to implement a range of valuable storage-security services. We specifically discuss the use of these new architectures to provide data integrity, capability-based access control, and labeled information flow at the disk access layer. In this, we introduce systems that place a security perimeter at the disk interface—and deal with the parent operating system only as a largely untrusted entity.

## 1. INTRODUCTION

Computing models are changing at previously unseen rates. More than just faster processors, larger disks, and more bandwidth, the *way* in which computing is performed is changing. We find our data spread hither and yon because of increasingly malleable and convenient storage, i.e., SAN, NAS, distributed filesystems, web-based services, etc. Moreover, the nature of the relationship between the computers and storage is changing; virtualization divorces the hardware from the user and through services like *Internet suspend/resume* [27], storage from the computer. This is prompting a revolution in computing, where we are becoming less tied to physical location, and are able to reach the lofty vision of “computing when and where you want it”.

Of course, such a new paradigm leads to an enormous number of additional requirements on the computing infrastructure. This is particularly true of storage systems. Un-

trusted operating system may run on hardware. Mutually untrusted entities may need to share the same storage device. New environments require that the requests made to the storage be regulated in new ways. Unfortunately, these requirements are impossible to meet in current storage architectures without trusted intermediaries [8].

Partly in response to emerging requirements, the storage industry has begun to manufacture devices with significant architectural enhancements. New *hybrid hard drives* (HHD) provide a small but fast non-volatile memory in addition to the slower, spinning disk storage. Such architectures have thus far only been used to improve performance (by using the NVRAM as cache) and reducing power usage (by keeping the drive in hibernation when requests can be served by the cache). This disks are available to OEM system developers today.

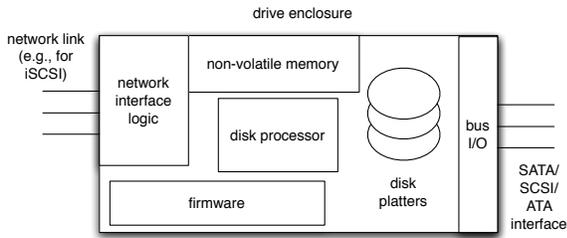
In this paper, we postulate that new storage architecture features can be used to implement long sought after security services at the disk layer. We envision using the fast NVRAM as a repository for meta-data. When used in conjunction with the added processing power, comprehensive security policy can be enforced by the disk on a per request basis. Such services build a security perimeter at the edge of the disk, and permit more secure realization of the new computing models. For example, these services will allow a disk to be used by mutually untrusted systems, or in environments with heightened security requirements.

This new disk model engenders a new security model: the threat model is different for this architecture than for regular disks. We add services that allow the disk to protect itself against adversarial operating systems, and ultimately make the disk itself more autonomous. For example, the disk can support primitives to allow information flow control while requiring the operating system to attest itself to a certain hardware and software level. This new layer of security and enforcement on the part of the disk ensures a stronger trust relationship between system and storage.

The remainder of this paper explores the ways these new disk architectures can be exploited to achieve a range of security goals. We begin in the following section by discussing the specifics of the new disk architectures and the ways in which they diverge from traditional systems. We then review posit three demonstrative applications, *a*) an integrity preserving disk, *b*) a disk that enforces access rights via capabilities, and *c*) a disk that implements a lattice-based information flow policy over the stored content. We conclude by considering a number of related works and describe our current efforts to realize the new storage model.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

Copyright 2002 ACM X-XXXXX-XX-X/XX/XX ...\$5.00.



**Figure 1: A proposed disk architecture for storing security metadata. The disk proposed has the ability to communicate over a network or through traditional disk interfaces (e.g., SATA, SCSI). Non-volatile memory is used for metadata storage, while the firmware may be burned in or configurable, depending on requirements.**

## 2. ARCHITECTURE

Disk access is a slow process. The amount of time required to retrieve information from a disk is orders of magnitude more than that required to retrieve the same information from the processor cache or main memory. Because of the vast amounts of data that can be stored on disks, it would be beneficial to store ancillary metadata related to the stored data on the disk as well, to provide information about the data’s structure, content, or semantics. This information may be substantially smaller than the data it represents. However, storing this data on the disk subjects its retrieval to the same long access times as the primary stored information. It is thus desirable to provide an additional storage medium that provides fast access to metadata, in addition to the slower bulk access provided by the mechanical disk.

Some manufacturers have already considered the solution of using non-volatile memory as an enhanced caching mechanism for the hard disk. In particular, Samsung and Seagate have released prototype “hybrid hard drive” (HHD) that may contain 256 MB of flash memory in addition to the traditional hard disk platters.<sup>1</sup> These drives are marketed to work with the Microsoft Windows Vista operating system, which includes “ReadyDrive”, an OS component allowing use of a hard drive’s flash memory as an extended read/write cache. While this memory can be used for performance, we propose an architecture that uses this memory, or an additional flash memory component on the drive, for storing metadata associated with disk blocks. This architecture is shown in further detail in Figure 1.

While some previous proposals for storage systems have advocated the use of metadata (e.g. NASD [19]), they employ the use of a metadata server between the client and the disk storage. These servers arbitrate policy and store all information, acting as gatekeepers to the data itself. By contrast, in our proposal, metadata is encapsulated within the drive enclosure, providing a sealed solution. If the drive manufacturer is concerned with physical security issues, either the non-volatile flash memory or the entire drive can be designed in a tamper-proof manner with mechanisms similar to those employed by the IBM 4758 secure co-processor [15]. Additionally, in contrast to using the non-volatile memory in

<sup>1</sup>Samsung recently released the MH80 series of hybrid hard drives, containing as much as 256 MB of flash along with a hard drive of up to 160 GB [16].

the drive as a disk cache accessible by the operating system, we are able to create scenarios where the operating system is untrusted that will still enforce data security.

By leveraging non-volatile memory for security metadata, the cost of accessing this data will be minimal compared to that of accessing information from the disk itself. For example, the seek time of a high speed server drive such as the Hitachi Ultrastar HUS15K300 73GB hard drive is 3.4 ms with an average latency of 2.0 ms [24]. By contrast, flash memory is very fast to access; Park et al. [35] show that the access time for a 4 KB page of NAND memory is 156  $\mu$ s for reads, 652  $\mu$ s for writes, and 2 ms to erase the page. By pipelining the operations of the disk such that it seeks and retrieves or writes data while authenticated encryption computations and accesses to nonvolatile memory are occurring, the costs of these operations may be completely masked. Performing operations such as integrity checking and providing confidentiality on disk necessitates processing by the drive, as shown in the figure. Disks have become increasingly able to perform processing within the drive unit; many drives are SMART-enabled [43], allowing them to monitor a variety of disk attributes; these require some degree of processing power from the disk, as do services such as automatic sector remapping. Additionally, storage hardware employing object-based storage (OSD) [44] paradigms, including Panasas Storage Blades [34] and forthcoming OSD drives from Seagate [40], contain significant processing ability to collate blocks into objects and manage attributes. Potential implementations could include burned-in policy through an ASIC or the ability to provide flexible services through EEPROM firmware or FPGAs that may be reconfigurable for different operations. Sufficiently powerful processors could allow the disk to act as a fully autonomous agent capable of acting as its own iSCSI server, for example, if there is an onboard network interface card.

We next examine how to leverage this new architecture by exploring a spectrum of security concerns, and show how applications using this architecture can solve these problems.

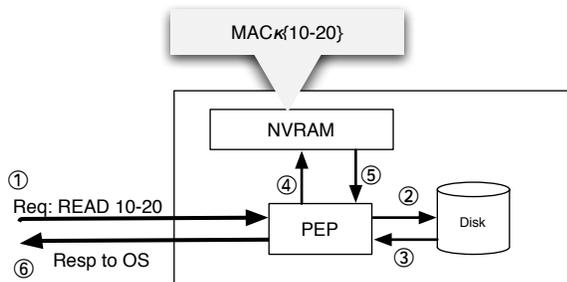
## 3. APPLICATIONS

We now consider methods of using the flash-assisted disk architecture for secure metadata. Our proposed applications are shown in increasing complexity from integrity protection (using flash memory to store MACs for disk blocks) to capability-based access control, which requires more involved management and storage layout considerations. We also consider how to implement information flow using labels, and consider requirements for policy implementation and enforcement.

### 3.1 Support for Authenticated Encryption

The IEEE P1619.1 standard [25] defines modes of operation for *authenticated encryption*. Authenticated encryption allows for both the confidentiality and integrity of data in storage systems. It does this through modes of operation, or just ‘modes’, which for a given plaintext and key pair produce a ciphertext for confidentiality, and a Message Authentication Code (MAC), a keyed hash, for integrity. Each mode also requires an Initialization Vector (IV), which must be unique for every plaintext/ciphertext pair.

Because the modes of authenticated encryption we investigate use Counter (CTR) mode to encipher plaintext, the resulting ciphertext is the same length. However, due to the



**Figure 2: Performing authenticated encryption using security metadata.** When a client makes a read request for blocks 10-20 (step 1), the policy enforcement point, or PEP (usually the processor on the disk) forwards a request to the disk platters, which read the blocks (steps 2,3). The non-volatile memory is consulted to determine the MAC for the integrity set (step 4); in this case, the set covers blocks 10-20. The results is returned to the PEP (step 5), and if the MAC is valid, the data is returned to the OS (step 6).

inclusion of the MAC and IV to each ciphertext, these modes are not length preserving. For this reason, the standard suggests use with devices which support length-expansion, such as tape drives. Such length expansion is problematic for disk drives, as it requires that any extra data beyond the length of a disk sector be stored elsewhere on the disk. This leads to extra disk seeks and therefore, higher access latencies to read or write each sector. In order to support authenticated encryption in devices which do not have a variable length base unit of storage, such as disk drives, we propose using the NVRAM in the aforementioned architecture as a store for the MACs and IVs which must accompany each ciphertext. We now examine the role of the NVRAM architecture in the process of authenticated encryption for two modes, Counter mode with Cipher Block Chaining (CCM) [13] and Galois Counter Mode (GCM) [14].

CCM mode works as follows. When a subject with a symmetric key writes a plaintext record to the disk, an IV is generated and used to compute a MAC from the plaintext. Both the IV and MAC are stored in NVRAM. Next, the plaintext is enciphered and stored on the disk. When a user with the same key reads the same record, the ciphertext record is deciphered and the resulting plaintext is used with the corresponding IV from NVRAM to compute a MAC, which is compared against the MAC stored in NVRAM. If the two are equal, the integrity of the data has been verified. GCM mode uses a similar procedure, except that it computes MACs from ciphertext as opposed to plaintext. Having seen how the NVRAM architecture is used in authenticated encryption, we now consider an important parameter of these modes which is left variable by IEEE 1619.1, the length of a plaintext/ciphertext record.

The length of the MAC according to IEEE P1619.1 is 128 bits and that of the IV is 96 bits. For this reason, the length of an associated unit of plaintext and therefore ciphertext is an important detail of any implementation of authenticated encryption. If a single disk sector is used as a unit of ciphertext, the space required for storing MACs and IVs is approximately 5.4% of the size of the entire disk

Parameter / Granularity	File	Page	Sector
required NVRAM space	Low	Medium to High	Medium to High
required processing power	High	Low (constant time)	Low (constant time)
coupling with FS	Very High	Simple with VM	None
complexity of enforcement mechanism	High	Simple	Simple
representation of capability token	Per File/File Descriptor	Per Page Stored in page table	Single, per-subject, Possibly a cryptographic key

**Figure 3: Comparison of capability granularities.**

Subject	Capability Token	Set Bitmap
S1	1e45e77a4c923	R R R W W ...
S2	3ab4d7492731c	RW RW RW RW RW ...
		⋮
		⋮
SN	3f971639c9283f	0 0 0 0 0 ...

**Figure 4: An example of how per subject capability tokens are represented with an associated capability bitmap.** S1 has read access to the first three integrity sets and write access to the next two. Similarly, S2 has read and write access to the first five integrity sets, and SN has no access to them.

assuming 512 byte sectors. For a 1TB system, 54GB of NVRAM would be required. To mitigate this large spacial cost, we consider using *integrity sets*, fixed size groups of adjacent sectors, for which a single MAC is calculated and stored. They are used as follows. When a subject writes one or more blocks in an integrity set, authenticated encryption is performed on the entire set, and a single MAC and IV are stored for the entire set. When a subject reads one or more blocks in a set, authenticated decryption is performed on the whole set. The necessary blocks are extracted from the ciphertext, and a MAC is calculated and compared against the one stored in NVRAM.

By controlling the size of integrity sets, one can control the amount of space needed in NVRAM for storing MACs and IVs. Of course, this savings in space is not without a cost in time. We have shown that the computational costs of performing the integrity functions in CCM and GCM increases linearly in the number of sectors per integrity set [6]. This implies that the cost of computing a MAC for a set of  $n$  sectors is equal to the cost of computing  $n$  MACs, one for each sector in the set, plus a constant setup time.

Another advantage of using integrity sets arises from the way modern operating systems handle block-level requests. When a block-level request arrives at the I/O scheduling layer, requests for adjacent disk blocks are merged together to reduce the number of requests sent to the disk and therefore disk seeks. This in turn also minimizes the number of MAC calculations as the size of an integrity set in sectors approaches that of the mean number of sectors per request.

### 3.2 Capability Based Access Control

The approach we now examine to mitigate the loss of confidentiality due to the theft of a hard disk, is to implement capability based mandatory access control within the disk itself. This implies that the disk must maintain a protection state consisting of per object capabilities for each subject and, an enforcement mechanism to mediate all access

according to them. In the event that a disk is stolen, an attacker should not be able to acquire capabilities or successfully perform any block requests to the disk. Such a capability based protection state will require two data structures.

1. An unforgeable capability token which can be assigned to a subject based on credentials
2. An associated descriptor containing per object capabilities for a subject

We propose that both of these data structures be maintained in NVRAM to ensure the secure storage and fast availability of the protection state to the enforcement mechanism, which we assume to be part of the disk’s firmware. We do not aim in this section to define semantics such as a subject’s initial capabilities with an object upon creation, as these are a matter of policy. We are presenting a plausible infrastructure for implementing such policies.

### 3.2.1 Capability Representation

Previous approaches to capability based systems have represented capabilities at various levels of granularity within the OS such as segments of memory [12] and nodes and pages [41]. The granularity of capabilities in the disk has an impact on the amount of space required in NVRAM, the amount of processing power required in the disk’s controller or processor, the complexity of the enforcement mechanism, the degree of coupling with the File System (FS), and the representation of the capability token. We examine these factors for three levels of granularity, files, pages, and sectors. The summary of these parameters for the three granularities can be seen in Figure 3. It is important that regardless of granularity, no block request made without the appropriate capability will be fulfilled.

**Files.** In order for capabilities to be represented at the file level, the disk must receive enough information from the FS to be able to verify that a given capability token is sufficient to access all of the sectors in a file. This implies a tight coupling between FS and disk. One such architecture for which capability based access control has been explored is that of Type-Safe Disks [42]. Maintaining capabilities at the file level requires a tight coupling between disk and file system, which we cannot assume trust in. It also typically requires more processor overhead than finer grained, more autonomous methods, due to the maintaining of data structures which represent file and directory relationships. It is however advantageous in that it requires less space in NVRAM for capabilities than page or sector level implementations which must maintain capabilities for the entire disk, as they lack knowledge of block allocation. File level capabilities also offer a familiar representation of capability tokens in the form of a file descriptor which can be returned by the `open()` system call.

**Pages.** Most modern microprocessor architectures supply page tables which map each process’s virtual address space to pages of physical memory [36]. Pages can be paged in and out of main memory into secondary storage, providing processes with address spaces larger than physical memory. Because of this mapping from physical memory pages to those stored on the disk, it could be convenient to leverage the page table data structures to maintain per page capabilities which could be used to control not only how processes access

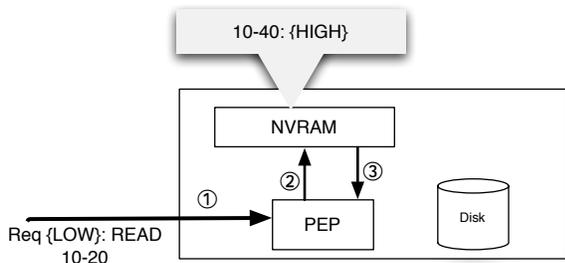
the pages in memory, but how the OS may read and write these pages to disk on behalf of processes. This could also be convenient as most OSes describe block level requests in terms of pages. For example, the Linux `bio` structure, the basic unit of IO requests to block devices, is composed of structures which map to pages of physical memory [11]. All an OS need do is include the correct subject credentials in these requests to obtain per page capabilities. These capabilities could then be stored in the page table descriptors for each processes and retrieved when their corresponding pages must be paged in or out.

Page granularity capabilities will require more space on average than file level, as the capabilities for the whole disk are maintained for each subject regardless of the number of files. However the amount of required space is static and once allotted will not change unless more subjects are added. The only coupling between the disk and the FS/VM subsystems is that needed to ensure that the VM pages always align with those on the disk. Only a small processor overhead is needed to initially distribute the per page capabilities. Once the subject has obtained the capability, the OS need only forward it to the disk when reading or writing a page on its behalf.

**Sectors.** Representing per sector read and write capabilities as a bitmap or capabilities list for each subject is not practical due to the limited capacity of NVRAM. Instead, we propose maintaining capabilities at the granularity of the integrity sets mentioned in section 3.1. This differs from the proposed page granularity in that it does not depend on the VM subsystem to align pages with the disk. Assuming a large enough set size, capabilities could be maintained in static, per subject bitmaps without exhausting the space in NVRAM. Such a bitmap would map two bits, enough to express read, write or read and write capabilities, to each integrity set as seen in Figure 4. A subject would obtain a single capability token from the disk which would be used for subsequent disk accesses to map to that subject’s capability bitmap stored in NVRAM.

There are several advantages to this method above the other two. First, because the access control is being done at the same granularity as the authenticated encryption, a subject’s symmetric key could double as their access token. This allows the same infrastructure used for key management to be implicitly used for capability management. This is advantageous as there has been much exploration in the area of key management [21] [9] [46]. Second, provided a sophisticated enough on-disk processor, this could also allow access control decisions and authenticated encryption to be done concurrently. Such a pairing of authenticated encryption and access control provides the means to not only protect confidentiality and verify integrity, but to proactively protect integrity by denying unauthorized writes. Third, it requires little processing power. A constant time mapping from the requested sectors in a block request to the corresponding entries in a subject’s bitmap is all that is needed to decide whether the request will be granted.

This method will require more space than the file granularity capabilities, and either more or less than those at page granularity depending on the size of each integrity set compared to the system page size. Due to the static size of the per subject bitmaps, once the necessary amount of space is allotted in NVRAM, it will not increase until the addition of more subjects to the disk.



**Figure 5: An example of enforcing information flow with disk metadata.** A request from a client labeled **LOW** to read blocks 10-20 is made (step 1). The policy enforcement point, or PEP (usually the processor) consults the non-volatile memory and finds that blocks 10-40 are labeled **HIGH**. When this information is returned to the PEP, it denies the request and no disk access is made.

### 3.2.2 OS Constructions

To reduce trust in the OS, the capability token should originate from within the disk. All subjects must provide credentials to the disk to obtain a token, and should not be able to obtain a token by any other means. As subjects exist primarily at the OS level, we examine a typical usage scenario to show the necessary OS level additions to accommodate the distribution of capability tokens by the disk.

A user logs in to a system using a program such as login which executes a shell and makes a new filesystem (VFS) call to send the user’s credentials and process ID (PID) of that shell to the disk containing the current working directory. Once the disk receives the credentials, it maps them to a capability token or creates a new token which it returns to the driver. The driver adds this token to the Process Control Block (PCB) of the shell who’s PID was specified in the VFS call. Any child processes of the shell will inherit the token into their PCBs as well. Whenever a subject makes a VFS system call such as `open()` or `write()`, the capability token is retrieved from the initiating subject’s PCB at the File System level and included in any block-level requests. The token is extracted from the request by the disk’s driver, and included in the request sent to the disk.

Note that in the above scenario, it is possible that the disk creates a new token for each login session. If a process continues execution after the user has logged out, due to the use of `nohup` or a similar method, it could be given its own token which maps to the user’s credentials, as part of the `nohup` command. Also, if no trusted path for the user’s credentials is assumed, they could be stored along with the user’s ID (UID) on an external device such as a smart card which is recognizable by the disk, and the UID could be used by the disk to distribute a capability token to that user’s processes.

For the purpose of capability creation, modification and revocation, a single meta-capability will be required per disk. This meta-capability could be used with a privileged utility to write special blocks to the disk containing user credentials and capabilities. The owner of the meta-capability is the disk’s administrator.

## 3.3 Preserving Information Flow

Providing mechanisms that preserve information flow is a complex task. While previous applications have dealt with tokens that may be modified but evaluated with relative ease, the challenges of fully implementing lattices for information flow are commensurately greater. While operating systems such as SELinux, and a small number of applications, such as JPMail [22] support information flow measures, little has been done to protect information flows at the storage layer. We first consider policies and models that preserve information flow, then discuss how to represent policy, how to authenticate security levels with the environment outside the disk, and how metadata is handled.

A well-studied and popular means of controlling information flow is through multi-level security (MLS). In the canonical model for MLS that preserves data confidentiality, as proposed by Bell and La Padula [2], subjects and data observe the *simple* property, which states that no subject can read to a level higher than it is authorized for, and the *\**-property, which states that no information may be written at a lower level than the subject is authorized for. There are numerous mechanisms that may be enforced for information flow preservation, such as the integrity-preserving model proposed by Biba [3], separation of duties as proposed by Clark and Wilson [10], and the Chinese Wall model [5], among many others. Depending on what characteristics of the information is considered most important to protect, the policy will be chosen accordingly by system architects.

Policy is represented as a series of labels, which identify characteristics of both users and data. Figure 5 shows an example of a disk enforcing an MLS scheme. A user making a request to the disk has been labeled **LOW** by the operating system and is attempting to read a set of blocks that has previously been labeled **HIGH**. Because of the simple security property, a **LOW** user cannot read **HIGH** data, so the access is denied at the policy enforcement point – in this case, the processor in the disk mediating the operation. The type of policy enforced by the disk may be modified within the firmware. Alternately, in a high-assurance environment, the manufacturer may burn in custom firmware to the drive such that a specific model must be followed. This approach also has the advantage that labels may be defined *a priori*; with knowledge of these and their semantics, reconciliation of semantics between user labels and those understood by the disk may be easier, as systems can ensure label consistency. However, predefining the label space may also limit the flexibility and granularity of expressible policy in the system.

Regardless of whether labels are predefined or if they can be configurable by the firmware, the drive must be able to understand their semantics. In the previous example of a **HIGH/MEDIUM/LOW** set of labels, policy enforcement mandates understanding a notion of ordering and comparison: semantics must be in place to determine that because **HIGH** is a “greater” level than **MEDIUM**, a user with level **MEDIUM** may not be allowed to access data labeled **HIGH**. Existing research by Hicks et al. has considered policy compliance by examining the intersection of labels in security-typed languages and those derived from an operating system [23]. In particular, the SIESTA tool created by Hicks et al. handles the establishment of connections between the Jif security-typed language and the SELinux operating system. With some modification, a similar tool may be used to provide compliance between operating system labels and

those deployed by the storage device. Alternatively, consider a high-assurance system where the entire system enforces information flow throughout and all of the hardware and software to be run is known. In this case, many if not all system parameters may be determined in advance, such that for a given system configuration and policy definition, a hash of the system state may be computed and burned into the drive’s firmware prior to its installation in the system. Then, a system possessing a trusted platform module (TPM) may provide a trusted attestation of the state to the hard disk, which will be able to ascertain the policy as being correct and the system as being trusted to preserve information flow.

## 4. DISCUSSION

In order to leverage our proposed security architecture with existing disks, some changes will be required. In particular, the drive’s firmware will need to be rewritten in order to use the available flash memory for storing security metadata rather than as a cache. Additionally, as previously mentioned, some processing power must be present within the disk to allow for policy evaluation; these can be specific ASICs or general processor designs that may be upgraded, depending on the desired implementation.

Some modifications may also be necessary from the operating system in order to use these mechanisms. Because access control decisions may be made in the disk, it may return an access denial message to the OS. The hardware driver must be modified to recognize these messages and propagate them up to the OS; the message to be returned to users will be implementation-specific. Drivers must also be cognizant of metadata sent to the disk layer by the operating system, such as labels. There may, however, be support within the OS structures to set this data; for example, the `bio` structure in Linux, which packages I/O requests to the block layer, has fields that may contain user-defined information, such that metadata may be passed from the OS to the hardware level. The `ioctl` control commands may also be used to directly push information to the disk, though cleaner interfaces would be preferable.

## 5. RELATED WORK

Storage security can be considered an offshoot of file system security, first considered formally by in Blaze’s CFS [4], which encrypted directories with a secret key and stored them as a file in the system. Extensions to CFS include CryptFS [47], which added a stackable kernel module on which the file system resides, and TCFS [7], which encrypts file data and names. Additionally, SFS [29] and SFS-RO [18] consider untrusted server operation and provide mutual authentication mechanisms between servers and users. More information on these and other early secure file system proposals can be found in Reidel et al.’s evaluation of storage security solutions [39].

Since that time, increased attention has been placed on securing data at the block, rather than file, level. Notably, the NASD project [19] hashed and encrypted blocks and groups of blocks called objects, storing the metadata on a server. Similarly, SNAD [31] uses hashing and keyed hashes extensively, calculating an either a digital signature or HMAC value over blocks. The latter scheme is similar in execution to ours, but relies on the client to perform the cryptographic

operations and store HMACs, rather than the disk, and does not take into account the ability to amortize costs and storage over multiple blocks. SCARED [38] provides data integrity but not at the block layer, so operations cannot be performed by the disk. SNARE [48] shares some similarities to NASD and SNAD but relies on capabilities and is best suited for a remote storage system. None of these schemes consider authenticated encryption using modes specified by the IEEE P1619.1 standard.

The Venti storage system [37] is an archival system that relies on write-once access. Their system considers block level performance, and attempts to optimize storage by seeking opportunities to compress blocks before adding them to the archive. Our proposal measures performance characteristics over multiple blocks but does not consider coalescing multiple writes to the same block because our solution is not strictly archival in nature; our goal is to preserve performance while providing integrity over a variety of potential workloads. Vilayannur et al. [45] also considered optimizing performance through tuning of parameters, but make characterizations and modifications to parallel file systems. By contrast, our approach works at the block layer and is thus largely independent of the file system running above it.

Oprea et al. [33] consider an on-disk model for protecting block integrity using calculated entropy. Their assumption is of an untrusted disk where the client performs all calculations and leverage the block’s entropy to make decisions whether to hash the blocks or not. This scheme yields large reductions in required storage but requires clients to retrieve their own integrity information. This concept has also been incorporated into cryptographic file systems [32], where file in conjunction with the construction of a Merkle hash tree [30] to represent the file contents; this approach of building hash trees for integrity has also been considered in systems such as Cepheus [17], Farsite [1], SUNDR [28], and Plutus [26], while SiRiUS [20] has hash tree validation at the file level. For a given file, only blocks considered high entropy are hashed, as blocks with low integrity are considered already structured; it is assumed that only if the block is tampered with will its entropy increase as it becomes corrupt, such that the entropy measurement takes the place of any hashing. This approach requires an amount of trusted storage accessible for each file. Our proposals differ in that we are concerned solely with blocks at the disk level, and all of our constructions and operations are meant to be able to be implemented within the disk itself. We also explicitly measure disk performance through detailed simulation to determine whether extra reads and writes at the block level contribute to degradation of overall throughput.

## 6. CONCLUSION

We have proposed a new storage architecture based on the emergence of hybrid hard disks, by using the non-volatile memory included in these devices to store security metadata. This allows provisioning of services such as integrity and confidentiality protection, capability-based access control, and information flow preservation within the drive itself, and ultimately allows for a higher level of assurance between systems and storage. Further work in this area will include detailed simulations of the operation of hybrid drives and understanding performance tradeoffs and implementation challenges when these services are deployed.

## 7. REFERENCES

- [1] A. Adya, W. J. Bolosky, M. Castro, G. Cermak, R. Chaiken, J. R. Douceur, J. Howell, J. R. Lorch, M. Theimer, and R. P. Wattenhofer. FARSITE: Federated, Available, and Reliable Storage for an Incompletely Trusted Environment. In *Proceedings of the 5th USENIX Symposium on Operating Systems Design and Implementation (OSDI 2002)*, Boston, MA, Dec. 2002.
- [2] D. Bell and L. LaPadula. Secure Computer Systems: Mathematical Foundations and Model. Technical Report M74-244, MITRE Corporation, Bedford, MA, 1973.
- [3] K. Biba. Integrity Considerations for Secure Computer Systems. Technical Report MTR-3153, MITRE Corporation, Bedford, MA, Apr. 1977.
- [4] M. Blaze. A Cryptographic File System for UNIX. In *Proceedings of the 1st ACM Conference on Computer and Communications Security (CCS'93)*, Fairfax, VA, USA, Nov. 1993.
- [5] D. F. C. Brewer and M. J. Nash. The Chinese Wall Security Policy. In *Proceedings of the IEEE Symposium on Security and Privacy (S&P)*, Oakland, CA, USA, Apr. 1989.
- [6] K. Butler, S. McLaughlin, and P. McDaniel. High-Performance Disk Integrity through Block Chaining. Technical Report NAS-TR-0072-2007, Network and Security Research Center, Department of Computer Science and Engineering, Pennsylvania State University, University Park, PA, USA, June 2007.
- [7] G. Cattaneo, L. Cauogno, A. D. Sorbo, and P. Persiano. The design and implementation of a transparent cryptographic file system for UNIX. In *Proceedings of the 2001 USENIX Annual Technical Conference*, Boston, MA, USA, June 2001.
- [8] S. Chaitanya, K. Butler, P. McDaniel, and A. Sivasubramaniam. Design, Implementation and Evaluation of Security in iSCSI-based Network Storage Systems. In *Proceedings of 2nd International Workshop on Storage Security and Survivability (StorageSS 2006)*, Alexandria, Virginia, October 2006.
- [9] P. Chen, J. Garay, A. Herzberg, and H. Krawczyk. Design and Implementation of Modular Key Management Protocol and IP Secure Tunnel. In *Proceedings of 5th USENIX UNIX Security Symposium*, pages 41–54. USENIX Association, 1996. Salt Lake City, Utah.
- [10] D. D. Clark and D. R. Wilson. A Comparison of Commercial and Military Computer Security Policies. In *Proceedings of the IEEE Symposium on Security and Privacy (S&P)*, Oakland, CA, USA, Apr. 1987.
- [11] J. Corbet, A. Rubini, and G. Kroah-Hartman. *Linux Device Drivers*. O'Reilly, third edition, 2005.
- [12] J. B. Dennis and E. C. V. Horn. Programming semantics for multiprogrammed computations. *Commun. ACM*, 9(3):143–155, 1966.
- [13] M. Dworkin. Recommendation for Block Cipher Modes of Operation: The CCM Mode for Authentication and Confidentiality, May 2004. NIST Special Publication 800-38C.
- [14] M. Dworkin. Recommendation for Block Cipher Modes of Operation: The Galois/Counter Mode (GCM) for Confidentiality and Authentication, May 2006. NIST Special Publication 800-38D (DRAFT).
- [15] J. G. Dyer, M. Lindermann, R. Perez, R. Sailer, L. van Doorn, S. W. Smith, and S. Weingart. Building the IBM 4758 Secure Coprocessor. *IEEE Computer*, 39(10):57–66, Oct. 2001.
- [16] Engadget. Samsung's Hybrid Hard Drive (HHD) Released to OEMs. <http://www.engadget.com/2007/03/07/samsungs-hybrid-hard-drive-hhd-released-to-oems/>, Mar. 2007.
- [17] K. Fu. Group Sharing and Random Access in Cryptographic Storage File Systems. Master's thesis, MIT, June 1999.
- [18] K. Fu, M. F. Kaashoek, and D. Mazières. Fast and secure distributed read-only file system. *ACM Trans. Comput. Syst.*, 20(1):1–24, Feb. 2002.
- [19] G. A. Gibson, D. F. Nagle, K. Amiri, F. W. Chang, E. Feinberg, H. Gobioff, C. Lee, B. Ozceri, E. Riedel, and D. Rochberg. A case for network-attached secure disks. Technical Report CMU-CS-96-142, Carnegie Mellon University, Pittsburgh, PA, USA, Sept. 1996.
- [20] E.-J. Goh, H. Shacham, N. Modadugu, and D. Boneh. SIRiUS: Securing Remote Untrusted Storage. In *Proceedings of the 10th ISOC Symposium on Network and Distributed Systems (NDSS'03)*, San Diego, CA, USA, Feb. 2003.
- [21] H. Harney, A. Colegrove, E. Harder, U. Meth, and R. Fleischer. Group Secure Association Key Management Protocol (Draft). *Internet Engineering Task Force*, June 2000. [draft-harney-sparta-gsakmp-sec-02.txt](http://www.ietf.org/rfc/rfc4851.txt).
- [22] B. Hicks, K. Ahmadizadeh, and P. McDaniel. Understanding Practical Application Development in Security-Typed Languages. In *Proceedings of the 22nd Annual Computer Security Applications Conferences (ACSAC)*, Miami, FL, USA, Dec. 2006.
- [23] B. Hicks, S. J. R. Rodriguez, T. Jaeger, and P. McDaniel. From Trusted to Secure: Building and Executing Applications that Enforce System Security. In *Proceedings of the USENIX Annual Technical Conference*, San Jose, CA, USA, June 2007.
- [24] Hitachi. Hitachi Ultrastar 15K300 3.5-inch Enterprise Hard Disk Drives. [http://www.hitachigst.com/tech/techlib.nsf/techdocs/F0954C336AF5E24F862572C200563CB3/\\$file/Ultrastar.15K300.final\\_DS.pdf](http://www.hitachigst.com/tech/techlib.nsf/techdocs/F0954C336AF5E24F862572C200563CB3/$file/Ultrastar.15K300.final_DS.pdf), Apr. 2007.
- [25] IEEE. IEEE P1619.1/20 Draft Standard for Authenticated Encryption with Length Expansion for Storage Devices. <http://attachments.wetpaintserv.us/2Qjro\%24n0iiv7kYzoz4BTmw\%3D\%3D326044>, June 2007.
- [26] M. Kallahalla, E. Riedel, R. Swaminathan, Q. Wang, and K. Fu. Plutus: Scalable Secure File Sharing on Untrusted Storage. In *Proceedings of the 2nd USENIX Conference on File and Storage Technologies (FAST'03)*, San Francisco, CA, Apr. 2003.
- [27] M. Kozuch and M. Satyanarayanan. Internet suspend/resume. In *Proceedings of the Fourth IEEE Workshop on Mobile Computing Systems and Applications*, pages 40–46. IEEE Computer Society,

- 2002.
- [28] J. Li, M. Krohn, D. Mazières, and D. Shasha. Secure Untrusted Data Repository (SUNDR). In *Proceedings of the 6th USENIX Symposium on Operating Systems Design and Implementation (OSDI 2004)*, San Francisco, CA, Dec. 2004.
- [29] D. Mazières, M. Kaminsky, M. F. Kaashoek, and E. Witchel. Separating key management from file system security. In *Proceedings of the 17th ACM Symposium on Operating Systems Principles (SOSP'99)*, pages 124–139, Kiawah Island, SC, USA, Dec. 1999.
- [30] R. Merkle. Protocols for Public key Cryptosystems. In *Proceedings of the 1980 Symposium on Security and Privacy*, pages 122–133. IEEE, April 1980. Oakland, CA.
- [31] E. L. Miller, W. E. Freeman, D. D. E. Long, and B. C. Reed. Strong Security for Network-Attached Storage. In *Proceedings of the 1st USENIX Conference on File and Storage Technologies (FAST'02)*, Monterey, CA, USA, Jan. 2002.
- [32] A. Oprea and M. K. Reiter. Integrity Checking in Cryptographic File Systems with Constant Trusted Storage. In *Proceedings of the 16th USENIX Security Symposium*, Boston, MA, USA, Aug. 2007.
- [33] A. Oprea, M. K. Reiter, and K. Yang. Space-Efficient Block Storage Integrity. In *Proceedings of the 12th ISOC Symposium on Network and Distributed Systems Security (NDSS'05)*, San Diego, CA, USA, Feb. 2005.
- [34] Panasas. Panasas: System Hardware. <http://www.panasas.com/hardware.html>, 2007.
- [35] C. Park, J.-U. Kang, S.-Y. Park, and J.-S. Kim. Energy-Aware Demand Paging on NAND Flash-based Embedded Sources. In *Proceedings of the International Symposium on Low Power Electronics and Design (ISLPED)*, Newport Beach, CA, USA, Aug. 2004.
- [36] D. A. Patterson and J. L. Hennessy. *Computer Organization and Design*. Morgan Kaufmann, third edition, 2005.
- [37] S. Quinlan and S. Dorward. Venti: A New Approach to Archival Storage. In *Proceedings of the 1st USENIX Conference on File and Storage Technologies (FAST'02)*, Monterey, CA, USA, Jan. 2002.
- [38] B. C. Reed, M. A. Smith, and D. Diklic. Security Considerations When Designing a Distributed File System Using Object Storage Devices. In *Proceedings of the 1st IEEE Security in Storage Workshop (SISW'02)*, Greenbelt, MD, USA, Dec. 2002.
- [39] E. Riedel, M. Kallahalla, and R. Swaminathan. A Framework for Evaluating Storage System Security. In *Proceedings of the 1st USENIX Conference on File and Storage Technologies (FAST'02)*, Monterey, CA, USA, Jan. 2002.
- [40] Seagate. The Advantages of Object-Based Storage – Secure, Scalable, Dynamic Storage Devices. Seagate Research TP-536, Apr. 2005.
- [41] J. S. Shapiro, J. M. Smith, and D. J. Farber. EROS: a fast capability system. In *SOSP*, 1999.
- [42] G. Sivathanu, S. Sundararaman, and E. Zadok. Type-Safe Disks. In *OSDI*, 2006.
- [43] Small Form Factor Committee. Specification Draft for S.M.A.R.T. Applications Guide for the ATA and SCSI Interfaces. SFF-8055i, June 1996.
- [44] T10. Information tecnology - SCSI Object-Based Storage Device Commands (OSD). Project T10/1355D, Revision 10, July 2004.
- [45] M. Vilayannur, P. Nath, and A. Sivasubramaniam. Providing Tunable Consistency for a Parallel File Store. In *Proceedings of the 4th USENIX Conference on File and Storage Technologies (FAST'05)*, San Francisco, CA, USA, Dec. 2003.
- [46] D. M. Wallner, E. J. Harder, and R. C. Agee. Key Management for Multicast: Issues and Architectures. *Internet Engineering Task Force*, June 1999. RFC 2627.
- [47] E. Zadok, I. Badulescu, and A. Shender. Cryptfs: A Stackable Vnode Level Encryption File System. Technical Report CUCS-021-98, Columbia University, New York, NY, USA, 1988.
- [48] Y. Zhu and Y. Hu. SNARE: A Strong Security System for Network-Attached Storage. In *Proceedings of the 22nd International Symposium on Reliable Distributed Systems (SRDS'03)*, Florence, Italy, Oct. 2003.