Disk-Enabled Authenticated Encryption

Kevin Butler, Stephen McLaughlin, and Patrick McDaniel
Penn State University, University Park PA 16802
Email: {butler,smclaugh,mcdaniel}@cse.psu.edu

***Abstract*—Storage is increasingly becoming a vector for data compromise. Solutions for protecting on-disk data confidentiality and integrity to date have been limited in their effectiveness. Providing authenticated encryption, or simultaneous encryption with integrity information, is important to protect data at rest. In this paper, we propose that disks augmented with non-volatile storage (e.g., hybrid hard disks) and cryptographic processors (e.g., FDE drives) may provide a solution for authenticated encryption, storing security metadata within the drive itself to eliminate dependences on other parts of the system. We augment the DiskSim simulator with a flash simulator to evaluate the costs associated with managing operational overheads. These experiments show that proper tuning of system parameters can eliminate many of the costs associated with managing security metadata, with less than a 2% decrease in IOPS versus regular disks.**

I. Introduction

Storage has become a central conduit for data loss and compromise. Reports of data exposed through lost or stolen laptops and disks occur on an almost daily basis. As just one example, laptops containing the mental health histories of over 300,000 people, and the full names and social security numbers for almost 2,000 people, were stolen from the Pennsylvania Public Welfare Department [19].

While solutions such as full disk encryption (FDE) and volume encryption mitigate some of these issues, they do not facilitate the protection of data *integrity* on the disk. The attacker may modify the contents of the disk in arbitrary ways, e.g., by overwriting the encrypted blocks. Depending on the encryption algorithm in use, the attacker may be able to replace encrypted sectors with known ones (e.g., changing a sector containing a database field with a "yes" value to one with a "no"), or may simply overwrite the blocks of interest with random data to block their retrieval. As a result, even if a stolen laptop is recovered, the data could have been tampered with in undetectable ways.

A promising means of providing data integrity in addition to confidentiality is authenticated encryption (AE). AE schemes, however, are not *length-preserving*: validating the data integrity requires an authentication tag (i.e., HMAC) to be appended to the calculated value. The question of where disks should store this information has been, to this point, unanswered. Storing it in a set location on the disk could yield potentially expensive seek times, while storing extra data in a sector hidden to the user could require large amounts of additional on-disk metadata and more importantly, potentially expensive changes to platter manufacturing and servowriters to ensure that tracks are not misaligned. In addition, because certain AE schemes are parallelizable, there may be performance benefits to accessing tags in parallel with data.

In this paper, we describe how a disk-based solution can leverage non-volatile memory within the drive enclosure (as found in hybrid hard drives) in conjunction with an ASIC that supplies cryptographic functions (as with FDE drives) in order to provide authenticated encryption services. By storing the metadata within the drive itself, we alleviate the need for external metadata servers (required in proposals such as NASD [8] and Ceph [29]), which may be a source of compromise.

We model access costs and emulate workloads to evaluate the trade-offs between disk performance and flash memory storage. Because of the high storage overhead of storing an authentication tag with every disk sector, we collate multiple disk sectors into *integrity sets* over which a tag is calculated. In support of this evaluation, we develop a generic block driver used to drive live workloads to the DiskSim simulator. We augment DiskSim with a flash simulator that implements a detailed model of non-volatile memory. Our preliminary results show that security guarantees such as confidentiality and integrity can be implemented with surprisingly little overhead; our measurements indicate as little as 2% percent reduction in IOPS in a mail-server workload.

Section II describes the challenges of providing authenticated encryption. Section III describes how we simulate authenticated encryption operations. Section IV evaluates performance under a number of differing workloads. Section V examines related work, while Section VI concludes.

II. Authenticated Encryption

Storage devices may actively protect the integrity as well as confidentiality of data at rest through a method of encryption known as authenticated encryption (AE). Several algorithms are defined for AE, including Galois Counter Mode (GCM) [6] and Counter Mode with CBC-MAC (CCM) [5]. Both modes provide confidentiality through encryption and integrity through HMAC calculation, which are stored for future comparison, When a ciphertext record is read, an HMAC is calculated for it and compared to the one stored the last time that record was written, in order to verify the integrity of the record. For a given plaintext, each of these modes produces a ciphertext of the same length and an HMAC of a fixed length, typically 128 bits.

Keys for AE may be stored in a secure portion of the disk, as with FDE drives. Along with the key matter, disks enforcing authenticated encryption must also store the HMACs and initialization vectors used for AE. Storing this information external to the disk invites tampering from a malicious operating system, while keeping them on disk requires additional seeks or platter redesign. We instead propose that HMACs and IVs used for authenticated encryption be maintained within a separate store in the disk that can protect them from tampering and allow them to be accessed efficiently. Non-volatile RAM provides a good medium for such a store, allowing the separation of on-disk data from security metadata. NVRAM may be managed by the disk controller, isolating it from the operating system, and making it accessible in parallel with the disk platters.

The length of a unit of ciphertext is an important parameter in an authenticated encryption scheme as it determines total number of HMACs that need to be stored for a fully encrypted volume. 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 assuming 512 byte sectors. Using IEEE P1619.1 [12], a MAC contains 128 bits of output and requires 96 bit IV. Thus, a 1TB disk would require 54GB of NVRAM to store the MACs. To mitigate this cost, we aggregate disk regions using *integrity sets*–fixed size groups of adjacent sectors for which a single MAC is calculated and stored. 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 stored for the 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, we control the amount of space needed in NVRAM for storing MACs and IVs. We have shown that the costs of performing the integrity functions in CCM and GCM increase linearly in the number of sectors per integrity set [4], implying 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 how 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.

III. Augmented Disk Emulation

For our evaluation of the effects of integrity set size on performance and storage overhead, we built an emulation environment for our augmented disk. Our emulator replaces the disk drive in an otherwise fully functional commodity host system with the DiskSim [3] software based disk simulator and Kim's flash memory simulator described in [11]. The

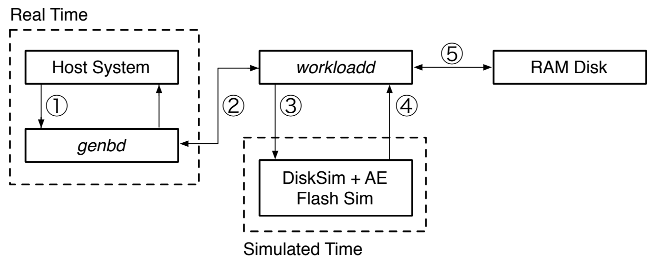


Fig. 1. An example of the disk emulator satisfying a block request. *genbd* recieves a block request at (1) and forwards it to *workloadd* in (2). *Workloadd* then passes the request to DiskSim and the flash simulator (3), which calculate the turnaround time for the request. The simulated completion time is returned to *workloadd* (4), which then waits for that amount of wall clock time before satisfying the request from the RAM disk (5).

emulator wraps DiskSim and the flash simulator, connecting it to both the host operating system and a backing store implemented in the host machine's RAM. This allows it to receive and satisfy block requests, thus maintaining the subtle interactions between the I/O behaviors of running applications and the storage device. Such dynamics cannot be captured by pure simulation and workload replay [27]. Storage emulation using this method has been shown to provide results that are comparable in accuracy to full implementation [10].

The flow of a block I/O request through the emulator is shown in figure 1. The two main components are *genbd*, a generic block device that interfaces with the host OS and *workloadd*, which marshals block requests to DiskSim and the flash simulator and satisfies them using a RAM disk. The key step in device emulation is the translation of the simulated completion time for a block request to a wall-clock completion time. This causes the host system to "feel" the delay from disk. In the case of write requests, the HMACs for any integrity sets containing modified blocks will have to be recalculated. Workloadd generates read requests for the blocks in these integrity sets and passes them to DiskSim. The completion times for these requests are added to the total wait time before satisfying the request from the RAM disk. We describe the specific additions made to DiskSim to simulate authenticated encryption and flash memory access in the following sections.

A. Modeling Augmented Disks

To simulate security operations and flash memory access, we placed a hook in the DiskSim disk controller code that is called once for each block request. This hook calculates the overhead created by these operations, and adds it to the total time for the disk access. We modeled these operations as follows.

- $T(f)$: The time to complete the function f in floating point milliseconds.
- $ISet(o, n)$: Returns the integrity sets containing the request with offset o and number of sectors n.
- $SpannedSets(o, n)$: The number of integrity sets containing all or part of the block request with offset o and number of sectors n.

- $R_D(s)$; $W_D(s)$: Reads or writes the contiguous set of sectors s to the disk. This operation is simulated by DiskSim.
- $R_F(o, n)$; $W_F(o, n)$: Reads or writes the block request with offset o and number of sectors n. This operation is performed by the flash simulator, as described in section III-B.
- $AE(n)$: Perform authenticated encryption on n contiguous disk sectors as described in Section II.
- $R_{AD}(o, n)$; $W_{AD}(o, n)$: Reads or writes the block request with offset o and number of sectors n to an augmented disk. This operation is a composition of the above functions.

We can thus model completion times with the following equations. The time for a read to the augmented disk is

$$\begin{aligned} T(R_{AD}(o,n)) &= T(R_D(ISet(o,n))) + T(AE(n)) \\ &\quad + T(R_F(o, SpannedSets(o,n))) \end{aligned}$$

The corresponding time for a write can be modeled as

$$\begin{aligned} T(W_{AD}(o,n)) &= T(W_D(o,n)) + T(AE(n)) \\ &\quad + T(W_F(o, SpannedSets(o,n))) \\ &\quad + T(R_{AD}(o,n)) \end{aligned}$$

Note that writes include the time for a read to the augmented disk over the set of specified blocks. When a write occurs, the HMAC of the integrity sets corresponding to the modified blocks must be recalculated and the new result stored to reflect the changes made.

B. Flash Memory Emulator

To support the experiments detailed in the following sections, we designed and implemented a simple flash memory disk emulator. Integrated into DiskSim [3], the resulting driver is comparable in behavior and operation to SanDisk's SSD Solid-State Drive and BiTMICRO's E-Disks [2], [25]. A flash memory based solid-state disk operates substantially similarly to a conventional block device/hard drive, except that the storage media is non-volatile memory.

Our emulator is composed of three software components: an I/O device driver, an address mapping module, and a flash core engine. All requests are queued by the I/O device driver and issued to the flash memory in order. We use the Flash Translation Layer (FTL) [16] for address mapping. The emulator supports *read*, *program* and *erase* operations corresponding to conventional disk I/O read and write operations with two-level mapping tables. Every request is serialized over a single memory channel (as in current flash drives). Interested readers are directed to the relevant literature for details on these operations [7], [20]. The emulated flash memory models a large block NAND flash memory containing 2 KB pages, each comprised of four 512-byte sectors. A block consists of 64 pages. Read and program operations are performed in units of pages, while the erase erase operation is performed units of blocks. We use Kang et al.'s [14] performance measurements to model each NVRAM operation, i.e., 0.027320 us per page read, 0.196370 us per page write, and 1.5 ms per block erase.

Each page must be erased before it is reused. Thus, a garbage collector is needed to select an appropriate block for erasure when no "fresh" page is available. When the garbage collector is called, a candidate block is selected based on the ratio of the number of invalid pages to valid pages. Consequently, the need for erasure on certain writes can induce significant variance in the write delay.

C. Experimental Setup

All tests described in the following section were performed on a 1.86 GHz Intel Core2 CPU with 1GB of RAM, running Ubuntu Linux with a 2.6.20-16-generic kernel. The filesystem used for the tests was ext2 with the kernel's anticipatory block I/O Scheduler. No actual disk was used, as all block requests were satisfied from the emulator's user-space RAM disk, which was allocated 512 MB of memory using `mlock()` to avoid paging. Note that the RAM disk is the backing store for the experiments and does not act as a cache for the simulated disk. It is important that the backing store being available in a fast memory so that the emulated augmented disk's performance is not impacted by the choice of backing storage device.

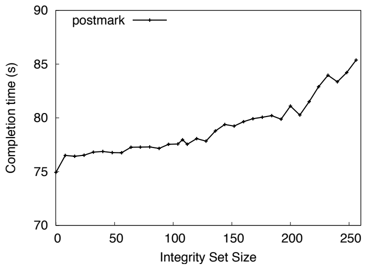
To simulate disk requests we used the DiskSim 3.0 simulation environment. and modeled the default Cheetah 4LP drive, a 4.5 GB, 10,000 RPM SCSI disk with a 512 MB cache. For authenticated encryption, we assume an AES-128 block cipher using completion times from ASICs [22]. The cost of authenticated encryption for a request of n blocks is the cost of $n \times 32$ encryptions, where 32 is the number of cipher blocks in one disk block. For all experiments, the integrity set size of 0 blocks represents the baseline configuration consisting of DiskSim running with no flash or encryption extensions.

We automated the benchmarking process using the Autopilot benchmarking suite [30], which we configured to run each test a minimum of 20 times, and to compute 95% confidence intervals for the mean elapsed, system and user times using the t-distribution. For consistent results, the buffer cache was cleared before each run.

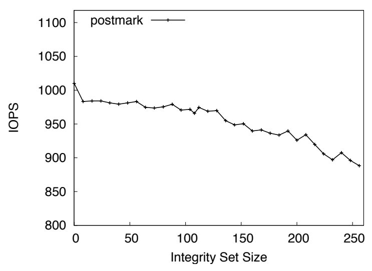
Two benchmarks were used, PostMark version 1.51 [15] and a simple in house benchmark. PostMark performs transactions on a set of many small files. Each transaction consists of a read or a write and either a creation or a deletion. We configured PostMark to perform 50,000 transactions on 20,000 files ranging in size from 500B to 20KB, using buffered I/O. Our in house benchmark does repeated sequential writes and reads of the entire disk, in order to test the effects of large sequential accesses on flash memory. We configured it to perform two write and read pairs on the entire disk.

IV. Preliminary Results

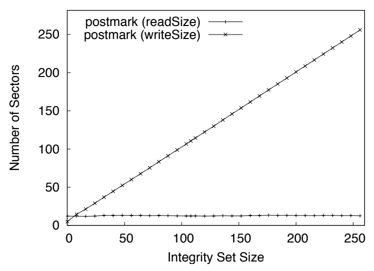
Supporting authenticated encryption requires managing more metadata than traditional disk services. Hence, an essential issue is cost; how much does the added functionality



(a) Average completion times for various integrity set sizes.

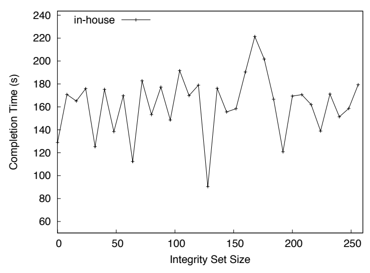


(b) Average throughput (in IOPS) for various integrity set sizes.

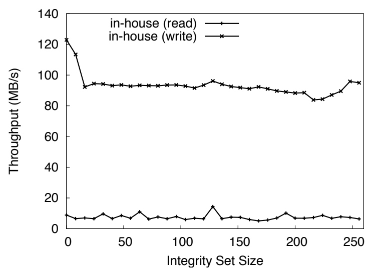


(c) Average block request size vs integrity set size

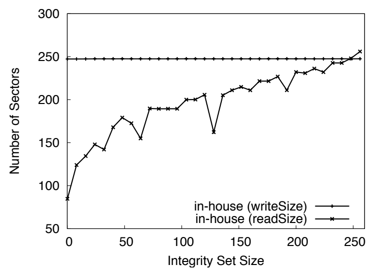
Fig. 2. Postmark benchmarks of completion time and throughput for various integrity set sizes. Figure (c) shows the relationship between block request size and integrity set size.



(a) Completion times for in-house benchmark.



(b) Read and write throughputs for in-house benchmark.



(c) Average block request size for in-house benchmark.

Fig. 3. In-house benchmark tests showing completion times, throughputs, and average request sizes.

from a disk augmented to provide authenticated encryption impede the normal functioning of storage?

The key trade-off parameter of the augmented disk is integrity disk set size. More specifically, the set size determines the trade-off between performance and flash memory requirements. In the following experiments, we observe the overheads caused by security operations and flash memory access, as well as disk level statistics which hint at possible methods for optimization. An integrity set size of zero is the baseline case in which no security functions are executed (and no flash memory is accessed).

A. *Small Random Access Workload*

The completion times for the PostMark tests are shown in figure 2(a). They increase with an average slope of 0.032 seconds overhead per additional sector in the integrity set size. Similarly, the IOPS for these test are seen in Figure 2(b). They decrease with integrity set size at an average rate of 0.4 IOPS per sector. In both graphs, there is a slope much steeper than the average between the points with set size zero and eight, due to the introduction of flash access to the baseline disk. The decrease in performance w.r.t. integrity set size is due to the steady increase in modified request sizes, seen in Figure 2(c).

In general, for workloads that perform small random transactions, disk performance is inversely proportional to integrity set size by a small constant. This is due to to increased read request sizes, which increase transfer time. Some of the effects of the increased sizes are absorbed by a reduction in the number of flash memory accesses.

B. *Large Contiguous Access Workload*

The completion times of the in house benchmarks are shown in figure 3(a). Unlike the random workload, in which performance degraded predictably with integrity set size, completion times oscillate with respect to set size. Similar but smaller oscillations are seen in the read and write throughputs as shown in figure 3(b). This is a similar phenomenon to the track-aligned extents explored by Schindler et al [26]. In our case, when the integrity set size reaches common block request sizes, e.g. 64 and 128 blocks, the read overhead incurred by HMAC recalculation is small or zero. Demonstrated in Figure 3(c), the same set sizes that have the shorter completion times and higher throughputs also have smaller average request sizes due to HMAC recalculation.

These results show that unlike the random workload, there is no approximately linear relationship between integrity set size and performance for a contiguous workload with large reads and writes. An important ramification is that performance is not proportional to disk size. This is because disk combined with available flash memory size dictates integrity set size. This is advantageous as the optimal integrity set size may be chosen for a random workload, and I/O system parameters may be tuned to align requests with the chosen set size. Thus, one initial recommendation for determining integrity set size is to determine the workload that the disk will primarily be exposed to. A suitable integrity set size for a system that is primarily used for small random accesses would be relatively small, e.g., 32, while for a mix of small and large accesses, a set size of 128 might be preferable.

Given that our preliminary experiments were done is a somewhat limited context of a 512 MB storage device, we wish to extrapolate results for larger disks to guide the future work described in the following section. Extending our data with a linear regression model, we found that optimal performance with a 4.5 GB SCSI driver requires 156 MB of flash memory, and a 1 TB drive requires 4.5 GB. Given the availability of SSDs with capacities of 64 and 128 GB, we believe these requirements to be well within reasonable limits.

V. Related Work

Securing data below the filesystem level has been an area of focus, particularly with the advent of network-attached disks that accept direct block reads and writes. Network-attached secure disks (NASD) [8] sought to replace the block model with variably-sized *objects* of variable size that provide greater semantics for data, and storing object metadata on servers. The metadata associated with these objects would then be stored on a metadata server. Similarly, SNAD [18] uses keyed hashes extensively, calculating 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 consider amortizing computation and storage overheads across multiple blocks. SCARED [24] provides data integrity but not at the block layer, so operations cannot be performed by the disk. SNARE [31] shares some similarities to NASD and SNAD but relies on capabilities and is best suited for a remote storage system. While our proposed capabilities-based application shares similarities with the block-based capabilities work by Aguilera et al. [1], that proposal relies on access control at a metadata server. By contrast, we consider enforcement directly within the disk itself. In addition, none of these schemes consider authenticated encryption using modes specified by the IEEE P1619.1 standard.

The Venti storage system [23] 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. [28] 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. [21] 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. Using hash constructions for integrity has been considered in other systems such as SUNDR [17], Plutus [13] and SiRiUS [9], but these proposals consider integrity either above the block layer (e.g., file-level validation in SiRiUS) or cannot be implemented within the disk itself.

VI. Conclusion & Future Work

We present a method of leveraing emerging componentry in disks, specifically cryptographic processors and non-volatile memory, to provide authenticated encryption. We developed an emulator to understand the performance characteristics of augmented disks, driving live workloads to Disksim and providing extensions to account for flash memory access as well as magnetic storage. Our evaluation shows that by tuning system parameters, much of the overhead of managing security metadata can be mitigated.

Our future work involves expansion of our simulation environment to use DiskSim 4.0 and the more modern disks that can be modeled, considering alternatives to NVRAM through on-disk metadata storage and the design challenges involved with this, and considering how filesystem support for extents may aid access to integrity metadata and overall performance. We also plan to consider reliability trade-offs particularly as they relate to NVRAM durability for writes versus expected disk lifetimes.

References

[1] M. K. Aguilera, M. Ji, M. Lillibridge, J. MacCormick, E. Oertli, D. Andersen, M. Burrows, T. Mann, and C. A. Thekkath. Block-Level Security for Network-Attached Disks. In *Proceedings of the 2nd USENIX Conference on File and Storage Technologies (FAST'03)*, San Francisco, CA, Apr. 2003.

[2] Bitmicro. http://www.bitmicro.com.

[3] J. S. Bucy and G. R. Ganger. The DiskSim Simulation Environment Version 3.0 Reference Manual. Technical Report CMU-CS-03-102, Carnegie Mellon University, Jan. 2003.

[4] K. Butler, S. McLaughlin, and P. McDaniel. High-Performance Disk Integrity through Block Chaining. Technical Report NAS-TR-0072-2007, Penn State NSRC, June 2007.

[5] M. Dworkin. Recommendation for Block Cipher Modes of Operation: The CCM Mode for Authentication and Confidentiality, July 2007. NIST Special Publication 800-38C.

[6] M. Dworkin. Recommendation for Block Cipher Modes of Operation: The Galois/Counter Mode (GCM) for Confidentiality and Authentication & GMAC, Nov. 2007. NIST Special Pub. 800-38D.

[7] E. Gal and S. Toledo. Algorithms and data structures for flash memories. *ACM Comp. Surveys*, 37(2):138–163, June 2005.

[8] G. A. Gibson, D. F. Nagle, K. Amiri, J. Butler, F. W. Chang, H. Gobioff, C. Hardin, E. Riedel, D. Rochberg, and J. Zelenka. A Cost-Effective, High-Bandwidth Storage Architecture. In *Proceedings of the 8th ACM Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS-VIII)*, San Jose, CA, USA, Oct. 1998.

[9] 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.

[10] J. L. Griffin, J. Schindler, S. W. Schlosser, J. S. Bucy, and G. R. Ganger. Timing-accurate storage emulation. In *Proceedings of the 1st USENIX Conference on File and Storage Technologies (FAST'02)*, pages 75–88, Monterey, CA, USA, Jan. 2002. USENIX Association.

[11] A. Gupta, Y. Kim, and B. Urgaonkar. DFTL: A Flash Translation Layer Employing Demand-based Selective Caching of Page-level Address Mappings. In *ASPLOS '09: Proceeding of the 14th international conference on Architectural support for programming languages and operating systems*, pages 229–240, Washington, DC, USA, Mar. 2009. ACM.

[12] 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.

[13] 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.

[14] J. Kang, J. Kim, C. Park, H. Park, and J. Lee. A Multi-Channel Architecture for High-Performance NAND Flash-based Storage System. *Journal of Systems Architecture*, 53(9):644–658, September 2007.

[15] J. Katcher. PostMark, A New Filesystem Benchmark. Technical Report TR3022, Network Appliance, 1997.

[16] J. Kim, J. Kim, S. Noh, S. Min, and Y. Cho. A Space-Efficient Flash Translation Layer for Compactflash Systems. *IEEE Trans. Consumer Elec.*, 48(2):366–375, 2002.

[17] 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.

[18] 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.

[19] J. Murphy. Computers stolen from welfare office. Harrisburg Patriot News, Sep. 11 2007. http://www.pennlive.com/midstate/patriotnews/article121468.ece.

[20] H. Niijima. Design of a Solid-State File Using Flash EEPROM. *IBM J. Research and Developement*, 39(5), 1995.

[21] 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.

[22] N. Pramstaller, S. Mangard, S. Dominikus, N. Pramstaller, S. Mangard, S. Dominikus, and J. Wolkerstorfer. Efficient AES Implementations on ASICs and FPGAs. In *Proceedings of the AES Conference*, 2004.

[23] 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.

[24] 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.

[25] Sandisk. http://www.sandisk.com.

[26] J. Schindler, J. L. Griffin, C. R. Lumb, and G. R. Ganger. Track-aligned extents: Matching access patterns to disk drive characteristics. In *Proceedings of the 1st USENIX Conference on File and Storage Technologies (FAST'02)*, Monterey, CA, USA, Jan. 2002.

[27] A. Traeger, E. Zadok, N. Joukov, and C. P. Wright. A nine year study of file system and storage benchmarking. *ACM Trans. Storage*, 4(2):1–56, 2008.

[28] 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.

[29] S. A. Weil, S. A. Brandt, E. L. Miller, D. D. E. Long, and C. Maltzahn. Ceph: A Scalable, High-Performance Distributed File System. In *Proceedings of the 7th USENIX Symposium on Operating Systems Design and Implementation (OSDI'06)*, Seattle, WA, Dec. 2006.

[30] C. P. Wright, N. Joukov, D. Kulkarni, Y. Miretskiy, and E. Zadok. Auto-pilot: A platform for system software benchmarking. In *USENIX Annual Technical Conference, FREENIX Track*, pages 175–188, 2005.

[31] 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.