2023 IEEE International Symposium on High-Performance Computer Architecture (HPCA) | 978-1-6654-7652-2/23/$31.00 ©2023 IEEE | DOI: 10.1109/HPCA56546.2023.10070924

D-Shield: Enabling Processor-side Encryption and Integrity Verification for Secure NVMe Drives

Md Hafizul Islam Chowdhuryy[1], Myoungsoo Jung[2], Fan Yao[1] and Amro Awad[3]
[1]University of Central Florida [2]KAIST [3]North Carolina State University
reyad@knights.ucf.edu, mj@camelab.org, fan.yao@ucf.edu, ajawad@ncsu.edu

***Abstract*—Ensuring the confidentiality and integrity of data stored in storage disks is essential to protect users' sensitive and private data. Recent developments of hardware-based attacks have motivated the need to secure storage data not only *at rest* but also *in transit*. Unfortunately, existing techniques such as software-based disk encryption and hardware-based self-encrypting disks fail to offer such comprehensive protection in today's adversarial settings. With the advances of NVMe SSDs promising ultra-low I/O latencies and high parallelism, architecting a storage subsystem that ensures the security of data storage in fast disks without adversely sacrificing their performance is critical.**

In this paper, we present D-Shield, a processor-side secure framework to holistically protect NVMe storage data confidentiality and integrity with low overheads. D-Shield integrates a novel *DMA Interception Engine* that allows the processor to perform security metadata maintenance and data protection without any modification to the NVMe protocol and NVMe disks. We further propose optimized D-Shield schemes that minimize decryption/re-encryption overheads for data transfer crossing security domains and utilize efficient in-memory caching of storage metadata to further boost system performance. We implement D-Shield prototypes and evaluate their efficacy using a set of synthetic and real-world benchmarks. Our results show that D-Shield can introduce up to 17× speedup for I/O intensive workloads compared to software-based protection schemes. For server-class database and graph applications, D-Shield achieves up to 96% higher throughput over software-based encryption and integrity checking mechanisms, while providing strong security guarantee against off-chip storage attacks. Meanwhile, D-Shield shows only 6% overhead on effective performance on real-world workloads and has modest in-storage metadata overhead and on-chip hardware cost.

I. Introduction

With all the challenges to secure a trustworthy supply chain of hardware devices, there is a clear trend for limiting the trust boundary to as minimum number of hardware components inside the system as possible [28], [45]. The issue is further exacerbated with the proliferation of cloud, and edge computing, where users have minimal or zero knowledge about the vendors of motherboards, memory modules, and storage devices, and thus can risk data breaches by hardware trojans, malicious devices, or even adversarial system maintainers and physical attackers. As a result, processor vendors rightfully start to support secure environments where the confidentiality and integrity of data are protected when leaving the processor chip. A major objective of such secure processors is to protect the integrity and confidentiality of data when written to off-chip storage.

While memory security has been a major focus of secure processors [28], [45], storage security is traditionally assumed in a classical context. In particular, the protection of storage data is typically implemented in software. Such approaches are acceptable for slow storage devices with access latencies in the scale of milliseconds [65] where the corresponding software overhead is only a marginal component. However, this is no longer the case for emerging ultra-low latency storage devices such as Intel's Optane Solid-State Drives (SSDs) [43]. These devices feature microseconds-range access latency and are expected to offer even faster I/Os with the unprecedented maturity of various NVM technologies [40], [59], [70]. In such fast storage devices, the overheads of security enforcement mechanisms in software can be significant: our investigation on real system shows that for a fast SSD device interfaced via state-of-the-art NVMe protocol, the overheads can be up to 4× (for software encryption) or up to 34× (for software encryption with integrity protection). On the other hand, disk vendors currently provide self-contained approaches for disk data protection. Commercial *self-encrypting disk* (SED) [3], [30], [50], [51] performs data encryption with specialized hardware components in the disk drive. Note that SED is designed to protect the security of the storage data at rest, while the data *in-transit* during I/O requests and responses are still in *plaintext*, making it vulnerable to attackers with physical access to the I/O bus. Furthermore, SEDs require users to trust the disk manufacturers while prior studies have found severe weaknesses in the design and implementation of security mechanisms in off-the-shelf self-encrypting disks [50].

We envision future servers incorporating both latency-optimized memories (DRAM or NVM) and capacity-optimized slightly-slower SSD drives. While several storage functionalities such as mirroring and error isolation of fast NVMe disks are currently integrated as hardware support inside processors (e.g., Intel's Volume Management Device [8]), there exists no processor-side support for fast storage security. Considering the limitations of existing data confidentiality and integrity protection schemes for fast storage devices, it is only natural to expect such processor-side hardware support to be architected to provide security mechanisms that can achieve desirable guarantees of data confidentiality and integrity.

While security primitives have been widely studied for main memories [56], [57], [64], several key challenges exist for designing processor-side storage protection due to the architecture and access protocols of NVMe disks. *Firstly*,

978-1-6654-7652-2/23/$31.00 ©2023 IEEE

unlike main memories where responses of read/write requests are expected with a fixed latency, NVMe protocol aims for high-level parallelism through *asynchronous operations* that involve complex interactions (e.g., command buffering and ring bell signaling) among main memory, OS kernel drivers and device-side NVMe controllers. It is critical to derive efficient and non-intrusive mechanisms that account for the additional encryption and integrity checking on the NVMe access path while managing transaction lifetimes. *Secondly*, with processor side security protection for both storage and main memories, data transfer *crossing boundaries* of these two protection domains can incur non-trivial performance overheads due to the repeated data encryption and decryption. Therefore, a synergistic design of architecture support that effectively coordinates secure storage and memory is necessary to optimize system performance. The final challenge involves designing efficient SSD security metadata storing/caching mechanisms to further alleviate runtime overhead due to the long path of storage metadata accesses.

In this paper, we propose *D-Shield*, a processor side architectural framework that enables strong security protection for NVMe disks. D-Shield leverages a novel design of encryption and integrity checking techniques tailored for NVMe storage devices. To address the aforementioned challenges, i) D-Shield integrates an NVMe protocol-aware *DMA Interception Engine* that allows the processor to perform storage data protection and security metadata maintenance transparently to the NVMe disks; ii) Built on top of the basic D-Shield scheme, we further propose D-Shield-Hyb that maintains disk logic blocks *transferred to main memory* as part of the *secure storage domain*, effectively eliminating repeated cryptographic operations for disk reads crossing the boundary of security domains. iii) We propose *D-Shield-Pro*, an optimized scheme that employs *in-memory caching for storage metadata* in NVMe SSDs to minimize the performance impact due to long latencies for secure metadata accesses. We build a prototype of the D-Shield design and evaluate its efficacy using I/O intensive benchmarks (i.e., FIO) and 10 real-world applications (including database and graph applications). The results show that D-Shield exhibits significant performance improvements compared to state-of-the-art software-based protection mechanisms (which are not fully secure), achieving up to 17× speedup (in terms of execution time) for the I/O intensive workloads and up to 96% throughput gain for real-world applications. Notably, D-Shield can retain *almost the same level of performance* (<6% impact) compared to the baseline without any storage protection, while offering complete protections against off-chip NVMe disk attacks. D-Shield is highly efficient with only 3.14% overhead for storing metadata in NVMe disks and modest hardware cost for on-chip logic. Our evaluations show that D-Shield has the promise of unleashing the high performance of NVMe disks with strong security guarantee at the same time. In summary, the key contributions of our work are:

- We highlight the need for processor-side security primitives to protect emerging high-speed storage systems. In particular, future secure processors should extend the security guarantees of confidentiality and integrity beyond memory to disk drives.
- We design D-Shield, the first architecture framework that efficiently manages security metadata via the NVMe protocol to enable comprehensive data protection for NVMe SSDs. D-Shield handles data protection and metadata maintenance through hardware without requiring any modification to the NVMe protocol or storage devices.
- We propose D-Shield-Hyb that extends memory blocks read from disks to the secure storage domain, removing redundant cryptographic operations as data blocks transit between the secure storage and secure memory.
- We further design an optimized framework–D-Shield-Pro–that minimizes the overhead of security enforcement for NVMe SSDs by utilizing effective caching mechanisms in main memory for performance-critical security metadata.
- We build prototypes of D-Shield framework and optimizations and evaluate their efficacy using representative disk I/O workloads. D-Shield offers strong security guarantee for storage data and maintains performance with modest hardware cost.

II. Background

A. Disk Encryption

Hardware-based Self-encrypting Disk (SED). Hardware-based SED performs encryption operations for I/O data blocks inside the storage disk. SED typically includes specialized encryption hardware in the storage controller [50], [53]. Generally, cryptographically secure AES is used for block encryption. While SED offers certain data protections without imposing additional overhead on the processor, it suffers from several major security issues. Firstly, since the encryption/decryption takes place in the storage device, data still communicate between the processor and the disk as plaintext. Adversaries with physical access to the device can snoop into the bus (e.g., SATA or PCIe) to intercept the transmitted data [26], [71]. Hence, SED itself fails to provide the same level of security expected as in secure memory sub-systems (e.g., secure NVMs [10], [20], [56]). Secondly, relying on various disk vendors to integrate reliable and quantifiable security mechanisms can be extremely risky. Recent works reveal that due to poorly-designed specifications and improper implementation, complete data recovery by attackers is possible for SEDs from multiple major vendors [50].

Software-based Disk Encryption. Several software-based storage system protection techniques are integrated into mainstream operating systems. Encryption-enabled file systems (e.g., Linux *ecryptfs* [37], and Windows *EFS* [1]) allow directory-level encryption. Block-layer encryption techniques such as *dm-crypt* [25] directly encrypt the entire block device. *dm-crypt* also offers integrity checking of read-only filesystems where the entire block device is verified at once. This approach is particularly time-consuming and thus is typically used only during device startup [6], [44]. *dm-verify* [6] uses a software

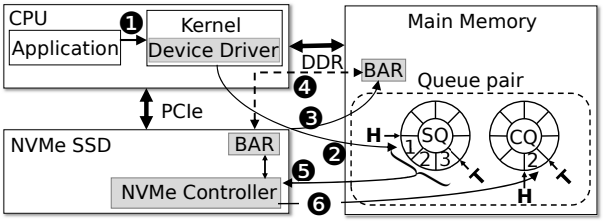


Fig. 1: High-level overview of NVMe disk I/O operations.

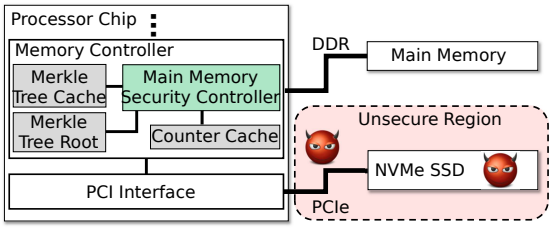


Fig. 2: Threat model showing the processor chip (i.e., trusted region) and the untrusted region (i.e., off-chip).

maintained Merkle tree structure to compute and validate hashes of read-only data blocks against pre-computed hashes. In contrast, *dm-integrity* keeps individual hashes for each data block during runtime, which allows verification for read/write system. However, it cannot detect physical attacks such as reordering the blocks within the same device due to the lack of a secure root of trust in the system. Finally, software-based schemes can have substantial overhead as the en/decryption is done in software via executing many kernel sub-routines across software layers [15], [63].

B. Memory Security

Traditional secure processor designs offer protection to the off-chip accesses between the CPU and the main memory with both encryption and integrity checking [10], [49], [56], [64]. These schemes are capable of protecting the memory sub-systems from various physical attacks such as bus snooping, bus spoofing, and data replay attacks [10], [31], [34], [56], [61] along with detecting memory faults [55], [67]. Typically, secure main memory employs the Galois counter-mode encryption to protect the confidentiality of the data using unique and non-repeatable encryption seeds (or counter) [64]. The secure processor stores encryption counters and maintains a Merkle tree (MT) where the root of the tree is securely kept on chip. The encrypted data and the corresponding counters are then used to compute a cryptographic hash, which is verified against a previously stored hash [56]. The hash of encrypted data and counter together will only match if both elements of the hash functions have not been tampered with. Since the MT protects the integrity of the counter, the hash verification also ensures the detection of the data integrity and prevents it from being replayed or compromised.

C. NVMe Storage System

Storage systems can use different interfaces to communicate with the host. PCIe is the most widely-adopted interface in consumer-grade storage devices since it provides sufficiently high bandwidth and low latency operation. NVMe protocol is designed to standardize the software interface optimized for PCIe SSDs [41]. It enables asynchronous high-speed communication for optimal latency, performance, and parallelism. A typical NVMe SSD access involves multiple components, including the device driver in the OS kernel, submission/completion queues in main memory, PCI interface in the on-chip I/O controller (i.e., IIO controller in Intel chips [5]), PCIe bus and the NVMe controller in the NVMe disk. When an NVMe SSD is attached to the system, the device driver creates *per-core* I/O queue pairs (e.g., a submission queue or SQ and completion queue or CQ pair) in the host memory, which are used to handle I/O requests. A memory-mapped *base address register* (BAR) containing *SQ tail doorbell* and *CQ head doorbell* is maintained and used to signal the NVMe disk about the status of SQ and CQ.

Figure 1 illustrates the interactions of the major system components during an NVMe I/O operation. Different from main memory that accesses data in the unit of 64B memory lines, data transactions in NVMe SSD are typically carried in 512B blocks, which are called *logic blocks* (LB). When software from the host initiates a block I/O request (❶), the kernel device driver first creates a new SQ entry with the logic block address (LBA) of the data in NVMe disk as well as a pointer to the physical-region page (PRP) list, which describes the main memory location where this data is stored (❷). The device driver then updates the *SQ tail doorbell* in BAR (❸), which will notify the NVMe controller about the most recent SQ entry location (❹). The NVMe controller then fetches the newly-added SQ commands (❺). Note that the NVMe controller can fetch multiple SQ commands at the same time. The NVMe controller later processes the SQ commands (out of order) and initiates the data transfer accordingly. Once the NVMe controller finishes processing an SQ command, it writes the corresponding CQ entry (❻) to the location addressed by the *CQ head doorbell*. The decoupling of I/O requests and responses allows the NVMe protocol to perform asynchronous operations with high parallelism. Finally, the NVMe controller sends a Message Signaled Interrupt (MSI) to notify the device driver about completing this I/O request, allowing the software to start processing the data.

III. Threat Model

Our threat model is based on prior works on secure processor architectures [10], [69], [72]. Specifically, we assume that the processor chip is the root of trust in the hardware stack. Any component outside of processor chip can be potentially compromised by adversaries. We assume that state-of-the-art protection techniques for the memory subsystems including counter-mode encryption and Merkle tree-based integrity verification are already employed for memory security [10], [20], [56]. We assume a strong attacker who can i) arbitrarily snoop traffics from system bus, ii) tamper with the data by

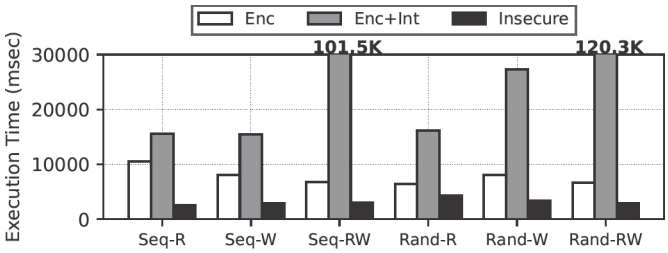


Fig. 3: Performance overhead with software-based disk protection schemes on the FIO benchmark.

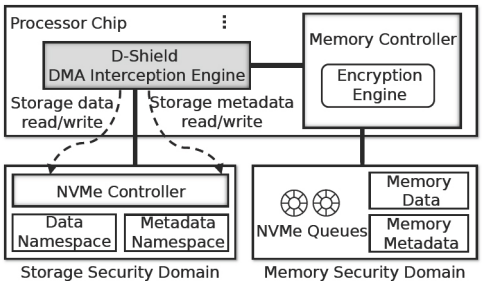


Fig. 4: An overview of D-Shield Framework.

either manipulating the bus state when packet is in transition or by manipulating data-at-rest, iii) replay the data by supplying earlier snapshots of data in the disk, and iv) physically attach the NVMe disk to a different system with the attempt to compromise disk data. Note that microarchitecture security such as timing channels [21], [22], [24], [66] is out of the scope of this work. Figure 2 illustrates the overview of the attack vectors.

IV. Motivation

To provide holistic security for data in the memory/storage hierarchy, it is necessary to protect the confidentiality and integrity of data outside of the processor chip both at rest as well as in transmission. In addition, the protection scheme has to be transparent to applications and ideally bring minimal negative performance impact to end users.

To investigate the overhead of software-based encryption from application level on NVMe SSD-based system, we run experiments to analyze the executions of the *flexible I/O* [11] benchmark under various workload patterns on an Intel Core i7-9700K system with a Samsung 970 EVO Plus V-NAND SSD [4]. We use an Ubuntu 22.04 system with Linux kernel v5.15.72 for this experiment. Figure 3 shows the execution times for *encryption-only* (dm-crypt) and *encryption with integrity verification* (i.e., dm-crypt+dm-integrity) as well as *insecure* default configuration. It is observed that the software-based encryption incurs substantial overhead (2.4× on average for encryption only and 16.1× for encryption with integrity protection) to complete the same amount of I/O transactions. As observed here, there is a considerable performance gap between the insecure baseline and dm-crypt/dm-integrity due to the additional computation and I/O resource usage for cryptographic operations and integrity checking in software execution path. More critically, even with such high overhead, software-based approaches still cannot provide complete disk security (See Section II-A). Based on these observations, we aim to design a processor-side framework for securing NVMe SSDs that guarantees the same level of protection as state-of-the-art secure memory scheme for the storage sub-systems, while minimizing its performance overhead.

V. D-Shield Framework

In this section, we present D-Shield, a framework that provides strong processor-side data protection guarantee for off-the-shelf NVMe disks. D-Shield employs state-of-the-art encryption and integrity checking mechanisms with the aid of security metadata (See Section II-B). Figure 4 illustrates an overview of D-Shield scheme. Essentially, D-Shield builds a new protection domain alongside secure memories with the processor serving as the root of trust. Storage security metadata is separately stored in a reserved region of the NVMe disks. D-Shield integrates a *DMA Interception Engine* between the on-chip I/O controller (that sends/receives the PCIe packets [2], [5]) and memory controller, which oversees the lifetime of I/O transactions and maintains storage metadata as data blocks are moved from/to the NVMe disks.

A. Challenges

While conceptually enabling secure storage in processors seems straightforward, we note that there exist several main designs and implementation challenges.

Firstly, the NVMe protocol is highly optimized to enable high parallelism and performance for device accesses. In NVMe, I/O requests are first buffered in main memory, then the NVMe controller fetches them and later executes the actual operation asynchronously. In other words, the NVMe controller is the one that executes the commands in any order it wants, which is unlike memory controller where the host controller observes the exact order of execution of commands. This decoupled control/data flow requires hardware support to identify and map DMA requests from the NVMe controller to the host memory (i.e., part of the data flow), with their corresponding block addresses only used in the control flow (NVMe commands queues). While it is possible to solve this issue via NVMe protocol modifications that either serialize the I/O transactions or install custom metadata-management primitives (e.g., using PCIe commands), such designs not only would incur non-trivial protocol design complexities but also can severely limit the I/O performance. Furthermore, modifications in NVMe protocols can break compatibility with current off-the-shelf hardware.

Additionally, unlike main memory accesses that are completely handled by hardware (i.e., memory controller), NVMe disk accesses feature intricate operational interactions between the host/device-side hardware and system software (See Section II-C). Specifically, the device driver is involved with the initiating and concluding of the I/O requests through accesses to designated memory-mapped queues. If such invocation is required for the accesses to the storage metadata (e.g.,

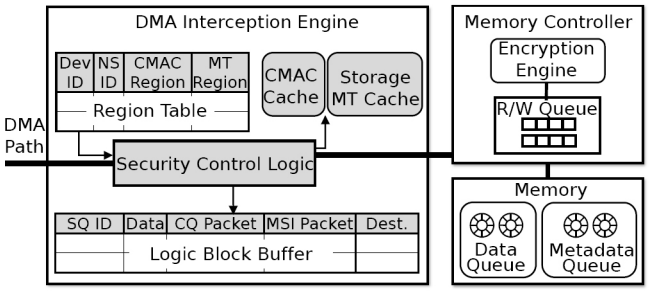


Fig. 5: Major hardware components in D-Shield framework (shaded blocks represent the added structures).

potentially spanning multiple blocks), it can introduce *non-trivial performance degradation due to excessive software intervention (i.e., processor interrupts)*. Therefore, an important consideration is to constrain the design to only *hardware changes* on-chip, and eliminate the required participation in software for storage metadata maintenance.

Finally, even with proper hardware support for metadata management, security storage mechanism can still suffer from considerable performance degradation if *numerous additional metadata accesses to SSDs are needed (especially for normal disk reads)*. Consequently, it is critical to design efficient metadata storage management and caching mechanism *tailored for* storage I/O characteristics to sustain the original performance advantage in today's ultra-low latency disks.

B. Basic D-Shield Design

We present the D-Shield design that addresses the aforementioned challenges arising from the complex NVMe I/O path involving multiple stages of software-hardware interactions. Specifically, D-Shield integrates a DMA interception engine that seamlessly enables processor encryption and integrity verification for storage data over the NVMe protocol.

1) D-Shield NVMe-aware Security Mechanism: D-Shield integrates a DMA interception engine (DIE) that captures in-flight I/O transactions between the host and NVMe disk. Figure 5 demonstrates the hardware structures in D-Shield and their interaction with the rest of the system. D-Shield augments the SSD read/write completion path with security metadata access and verification. Due to the asynchronous request/response in NVMe protocol (Section V-A), D-Shield needs to incorporate the following functionalities to efficiently synchronize with the I/O events: 1) recognize in-flight NVMe packets and map them to the corresponding I/O transactions; 2) determine storage metadata accesses and initiate encryption/decryption and integrity checking; 3) extend the lifetime of I/O transactions by delaying NVMe-related interrupts.

D-Shield dedicates reserved regions in the NVMe disks to store security metadata (more details in Section V-B2). As shown in Figure 5, it first introduces a *Region Table* to store the unique identifier of NVMe disks and the corresponding metadata region addresses. The Region Table serves as a bookkeeper of the logic block address ranges associated with data and different types of metadata, which are leveraged to determine subsequent encryption and data authentication operations. Specifically, Region Table stores the storage security metadata region addresses for each NVMe namespace attached to the system (identified by `Dev ID, NS ID` pair). To maintain the operations, D-Shield introduces a *Security Control Logic* (SCL) that coordinates out-of-order I/O operations and manages metadata read/write following the NVMe data path. SCL monitors incoming NVMe packets passing through the DMA path. Upon receiving a data read/write packet, SCL retrieves the logic block address of the I/O request and differentiates between NVMe data (i.e., data and metadata) and command packets with the help of the Region Table. SCL intercepts the logic block and stores it temporarily in a small *Logic Block Buffer (LBB)* for regular data. SCL then initiates access to storage metadata for encryption/decryption and integrity verification. To avoid maintaining a separate encryption engine in DIE, D-Shield *delegates the cryptographic operations to the corresponding memory controller*. Note that for NVMe disk reads, SCL also needs to intercept the MSI completion interrupt to the host. Once the security operations are completed and the logic block is sent to the main memory, the completion packets are forwarded to notify the device driver. Finally, D-Shield uses *metadata cache* in DIE to cache recently used metadata to speedup future metadata reuses.

To be non-intrusive with the NVMe protocol, D-Shield leverages the same submission/completion queue mechanism for *storage metadata access*. Note that current NVMe controllers inherently support multiple sets of SQ/CQ pairs for each namespace [54]. D-Shield uses *separate queues* for *storage metadata* I/O requests (illustrated as *Metadata Queue* in Figure 5). This metadata queue pair is never exposed to the device driver. Instead, D-Shield handles the initialization and completion of SSD reads/writes for storage metadata completely in hardware (i.e., through SCL). Such a design ensures a transparent embedding of the storage data security enforcement within the original data I/O transaction. With D-Shield, the device driver only handles data I/O transaction as it would originally, and the SSDs process I/Os without the need to distinguish data and metadata.

2) Efficient Security Metadata Storage in SSDs: State-of-the-art processor-side data protection involves three types of metadata, namely *MAC*, *counters* and *merkle tree hash* [10], [56], [57], [64]. The performance of metadata access can largely determine the performance of secure storage design. One straightforward way to enable fast storage metadata access is to completely store them in main memory. However, as disk storage is typically more than an order of magnitude larger than main memory, such approach is impractical due to the excessive memory storage overhead. Therefore, in D-Shield, storage metadata and memory metadata are stored separately in NVMe disks and memory devices respectively. We reserve a namespace to store the security metadata in SSDs. Originally, to create namespaces in NVMe disks, NVMe userspace tool for Linux includes a `nvme-create-ns` API [48]. We add a modified version of this API called `nvme-create-mdns`, to be used by the NVMe driver to create the metadata namespace

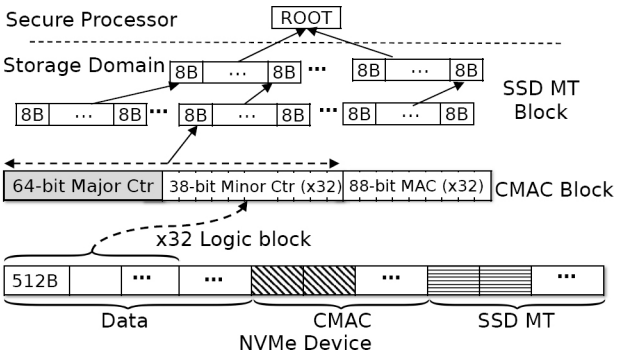


Fig. 6: Fused NVMe SSD counter/MAC block organization and logical Merkle Tree organization over the counters.

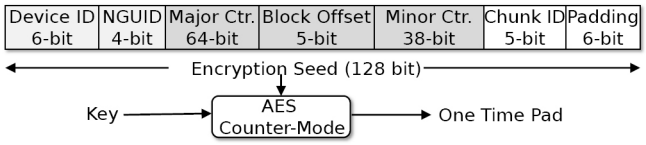
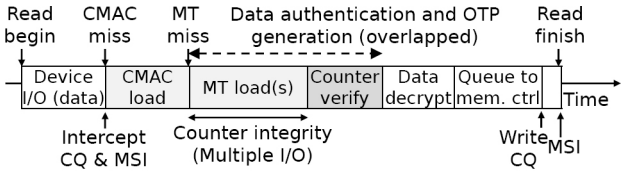


Fig. 7: *One Time Pad* (OTP) generation for logic block encryption/decryption in storage. Here, *Block Offset* is used to differentiate 32 data blocks sharing the same major counter.

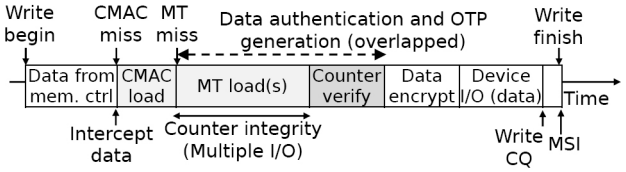
during device initialization.

In secure main memory architectures, MAC could be held in conjunction with ECC bits and loaded together with the memory line in DDR protocols [58]. However, such capability is not available in the NVMe protocol. As a result, a disk logic block read can potentially initiate storage accesses to all three metadata individually, which is extremely expensive given the complex path of each NVMe disk access. As a result, it is highly desirable to couple different types of metadata needed for *one data logic block* access into as few metadata blocks as possible. Since integrity verification using MT may require traversal of many layers (dependent on the highest level of on-chip cached node), it is infeasible to aggregate MT blocks. However, counter and MAC have a one-to-one mapping with the data block, as such a coupled storage organization form is possible. Based on this observation, D-Shield uses a *fused layout* of Counter and MAC (termed CMAC as illustrated in Figure 6) to store a group of counters (32) together with a group of MACs (32) corresponding to a set of storage data logic blocks into one single metadata block. This allows loading counter and MAC both to use a single read operation in case there is a miss in the D-Shield metadata cache.

We use split-counter based encryption [56], [64] to encrypt/decrypt data in the secure storage domain. As the handling of minor counter overflow in disk is expensive (requiring re-encryption of many logic blocks sharing the same major counter in the disk), D-Shield utilizes a large minor counter (38-bit), which can sustain about 30 days of *nonstop writes* exclusively to one single block (in contrast, typical secure main memory minor counters can overflow within minutes [64]). In addition, we use cryptographically-secure 88-bit MAC for each data



(a) DMA Interception Engine Events for Read operation



(b) DMA Interception Engine Events for Write operation

Fig. 8: Overview of NVMe SSD I/O operations in D-Shield (shaded blocks will be avoided if CMAC is cache hit).

block and a 64-bit major counter for the entire group of 32 data blocks. Figure 7 shows the encryption seed generation process in D-Shield which ensures the uniqueness of the seed for the same logic block address (LBA) across different NVMe disks. Note that although we use one minor counter for each 512B data block, the encryption seed is generated for each 128-bit chunk and hence it is possible to decrypt each *128-bit aligned* chunk individually (i.e., a 64B aligned chunk can be independently decrypted out of this 512B block).

D-Shield adopts the bonsai MT scheme [56] that uses MT over counter blocks to protect the integrity of the counter. Figure 6 illustrates the logical organization of MT over the counters. Note that the security of MT itself depends on the length of the produced hash [35]. In a naive brute-force method, a hash of n bit is said to be secure against hash collision for 2^n number of tries. Employing the birthday paradox [32], [62], a hash collision for the same length can be obtained with half the amount of tries (i.e., $2^{n/2}$ tries). D-Shield uses 8B hash over one unit (i.e., one major counter and 32 minor counters), which is resistant against 2^{32} trials. D-Shield uses GHASH based MAC calculated over each 64B part of encrypted logic block and the corresponding counter to verify the authenticity of data. Overall for one 512B block, D-Shield uses 88-bit MAC, which is secure against 2^{44} tries (similar to MT) [35].

C. Complete I/O Path of D-Shield

Figure 8 illustrates the overall NVMe read and write process augmented with data decryption and integrity checking in D-Shield. For a disk read operation, after the logic block is intercepted by the DMA Interception Engine (Figure 8a), it is first buffered into the logic block buffer (LBB). D-Shield also intercepts the CQ packet and MSI interrupt from the NVMe disk. Then it looks up the on-chip metadata cache in DIE to see if the corresponding CMAC is in cache, otherwise, it creates an SQ command in the *metadata queue pair* to load metadata blocks from disks. Note that the lookup and load of security metadata can be started as soon as D-Shield is aware of the LBA of the ongoing data read. Once all the metadata

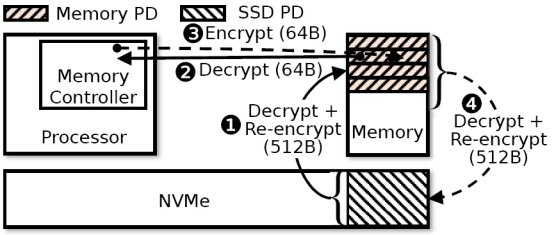


Fig. 9: Decryption and re-encryption (including integrity verification and authentication) operations when data crossing protection boundary in D-Shield.

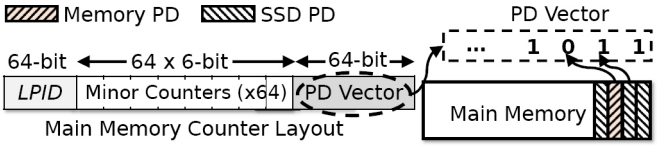


Fig. 10: Counter block layout of secure main memory augmented with 64-bit *protection domain (PD)* bit vector.

are returned to DMA Interception Engine, D-Shield verifies the integrity of the counter (if needed) and MAC of data and then finally requests the decryption. The decrypted 512 data block would be stored in the memory location indicated by PRP list by the memory controller. Finally, the secure NVMe data read is completed by forwarding the CQ packet and the MSI interrupt held in LBB. Figure 8b shows the NVMe write operation, which is similar to the read. For writes, the data from the memory controller is intercepted and buffered into LBB. Once all the metadata are ready, logic block encryption and updating the metadata are performed before completing the NVMe write. Note that for NVMe write, interception of the CQ packet and the MSI interrupt are not necessary as the logic block is already encrypted before it arrives in the disk.

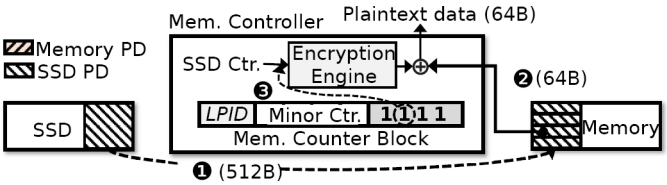
VI. D-Shield-Hyb: Optimizing Cross Domain Access Overhead

The basic D-Shield design provides processor hardware support to enable an individual protection domain for storage. While standalone protection for main memory and NVMe SSD can provide proper off-chip data security, the cost of moving data between protection domains (i.e., memory and storage) can be non-trivial. Specifically, when data from the main memory (e.g., as in-memory file cache) is written to the NVMe disk, the same memory location can be subsequently updated many times by the processor (e.g., as anonymous page), with each update changing its counter value to prevent counter reuse. Therefore, it is necessary to keep separate counters for each NVMe block, and such counters must have their integrity protected as well. In other words, when data is written to a specific storage unit (i.e., memory or storage), it must be encrypted using counter corresponding to its unique address to ensure the uniqueness of OTP. As a consequence, re-encryption of data is needed in traditional design as it transfers from one location to another (i.e., from memory to storage or vice versa) to ensure proper update of counters in its new location. However, re-encryption of data when crossing domains can introduce overhead due to additional utilization of the cryptographic engine and as well as the prolonged NVMe data path.

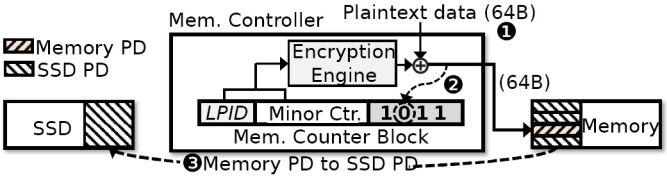
As illustrated in Figure 9, when an NVMe block is read (❶), the entire 512B first needs to be decrypted/integrity-checked using storage metadata and for memory then re-encrypted with memory metadata to be stored in main memory. Later, the processor might read (❷) or write (❸) one or more 64B memory line(s) and hence requires decryption and/or encryption, respectively. When the data block is persisted to SSD again (❹), the entire 512B has to be again decrypted from main memory and re-encrypted for SSD. As we can see, a significant portion of the operations including the entire ❶ operation and *potentially* a few of ❹ can be avoided with a synergistic secure storage-memory design. We further explore the processor's interaction of main memory and NVMe disks and present D-Shield-Hyb that extends the storage security domain for the transferred blocks in main memory to further improve system performance.

The main idea behind D-Shield-Hyb is that it is not necessary to re-encrypt data when it changes storage location (e.g., from storage to memory), but required if the actual data content changes (i.e., processor write updating the data). If the proper metadata for a protected data block can be located to decrypt and verify its integrity, the data block can be transferred between disk and memory without changing the protection domain. D-Shield-Hyb tracks the security domain for transferred data blocks and performs only one iteration of decryption/encryption using the owner domain's metadata.

D-Shield-Hyb requires the addition of a security domain tracking mechanism that bookkeeps the ownership of logic blocks in memory. Such tracking itself should not pose high overhead, otherwise, the improvements from reducing cryptographic operations can be diluted. Secure main memory scheme typically packs a 64-bit unique *logical page identifier (LPID)* as major counter and 64 minor counters (7-bit each) corresponding to one physical page (4KB) together in a counter block [10], [56], [57], [72]. D-Shield-Hyb makes use of this pre-existing structure to store a per-memory-block security domain vector (as illustrated in Figure 10). Specifically, we store a 64-bit *protection domain (PD) bit vector* to identify the corresponding security domain of each individual memory block in a physical page. This allows D-Shield-Hyb to store blocks belonging to storage domain in main memory and later determine the correct metadata when processor requests memory lines in that transferred logic block. If the corresponding PD vector bit of a 64B line is `SET` (i.e., `1`), the memory line belongs to the secure storage domain, otherwise the memory block belongs to secure memory domain. To accommodate the 64-bit PD vector while keeping the size of memory counter block unchanged, we use 6-bit minor counters for main memory metadata. Prior work [57] shows such nominal change in minor counter does not have a noticeable impact on the performance of common workload. Additionally, to identify the storage metadata corresponding



(a) Processor read requiring block from NVMe



(b) Processor write to NVMe

Fig. 11: D-Shield-Hyb operation (NVMe protocol specific events are not shown for simplification).

to a memory block belonging to the secure storage domain, D-Shield maintains a *page frame-to-logic block* mapping table to lookup the logic block address of the memory block and subsequently locate the required SSD metadata.

D-Shield-Hyb Read Operation. Figure 11a depicts data protection mechanism for processor read requiring block transfer from the SSD. When the NVMe controller services the read, the data is stored in main memory as part of the *storage domain* without performing any de/en-cryption or authentication (❶). The PD vector bits corresponding to the main memory PRP region blocks are set to indicate the protection domain of those memory blocks. Note that D-Shield-Hyb *do not* need to intercept and delay the completion queue packet and MSI interrupt packet anymore since here the secure NVMe data read operation is exactly the same as regular unsecured NVMe data read. When main memory services the processor read request of this data (❷), instead of directly using the secure main memory metadata, the proper security metadata is determined based on the corresponding PD vector bit (❸). In this case, since the PD vector bit is 1, the data block is authenticated using NVMe MAC and then decrypted using NVMe counter. Note that the CMAC read and corresponding MT reads are issued when the submission queue entry for the storage read passes the DMA interception engine.

D-Shield-Hyb Write Operation. Figure 11b illustrates D-Shield-Hyb operation during processor write. During the processor write (❶) under D-Shield-Hyb, the memory controller will encrypt the memory line using the memory metadata regardless of the corresponding PD vector bit value, and transfer the domain ownership to the main memory (reset PD bit ❸). This is because if that particular memory line belonged to the storage domain before, encrypting with storage's metadata will update the minor counter, which can create inconsistency for the sibling memory lines (mapped to the same 512B logic block) as they have not been written to yet. Note that while this newly-written 64B memory line now belongs to the memory domain, the rest of the 64B lines can still map to the storage

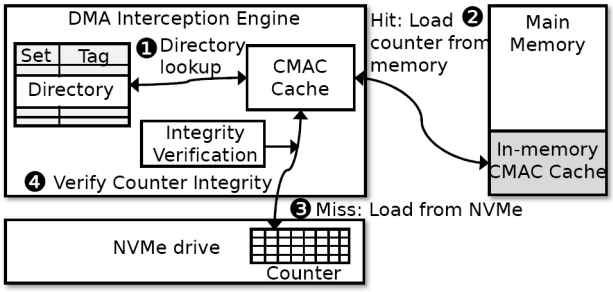


Fig. 12: Structures and interactions in D-Shield-pro.

domain. Finally, when the 512B logic block is written back to the NVMe disk, if all PD vector bits are **SET**, this logic block can be directly stored without requiring any cryptographic process as the content of the block has not changed. In contrast, if one (or more) PD vector bit is `CLEAR`, then the security domain shift for the entire logic block is required. This is done by decrypting each 64B chunk corresponding to PD vector bit and then encrypting using the storage metadata. This ensures that one minor counter protects the 512B block when stored on disk. Note that while theoretically security domain transition can be made at the finer granularity of 64B chunk, this requires an individual encryption counter for each 64B within a 512 block, which can increase the storage metadata overhead.

VII. D-Shield-Pro: In-memory Security Metadata Caching

Although typically on-chip storage metadata caches have a high hit ratio for workloads that exhibit data locality, workloads with more complex data access patterns may have non-trivial overhead due to metadata cache misses. We note that a miss in fused CMAC cache generally tends to be more expensive than an MT cache miss since a miss in CMAC block can potentially trigger multiple additional metadata block reads (i.e., MT blocks) for counter integrity verification, in addition to the actual CMAC block read itself. While the number of MT block loads required for integrity verification of a counter varies based on the state of the MT cache, these loads can be avoided if the counter is already on-chip (i.e., CMAC cache). To address this issue, we propose D-Shield-Pro, an optimized framework augmenting D-Shield-Hyb with in-memory CMAC block caching to increase the CMAC block hit ratio.

Figure 12 shows the overall design changes in D-Shield-Pro. We introduce a new device driver API called `nvme-create-fused-cache` that allocates a fixed region of physical memory space and stores the pointer to this region into a register in the DMA Interception Engine. This region is used by the DMA Interception Engine as a direct-mapped cache for NVMe CMAC blocks. Note that such API is only called once when a new NVMe disk is initialized. Additionally, D-Shield-Pro keeps a directory structure that records which fused blocks are currently in the memory cache. The on-chip directory allows quick lookup of in-memory cache without issuing any memory reads. Specifically, when a miss occurs

Hardware	Configurations
Processor	4-core, 3.0 GHz in-order, x86
L1 I/D-cache	Private, 64KB, 4-way
L2 cache	Shared, 16MB, 16-way
Main memory	DDR4 based 16GB
Cryptographic Engine	
Encryption/Hash operation (64B)	40 cycles [33]
DMA Interception Engine	
Metadata cache	256KB 8-way each
Hash operation (512B)	320 cycles
NVMe disk	
Capacity	512GB
Cell model	Z-NAND based MLC PCM [59]
Avg. random access latency(μS)	READ: 10.5, WRITE: 9

TABLE I: Parameters in D-Shield architecture.

Benchmark	Configurations
FIO	*# of threads:* 4; *# of I/O transactions:* 128-512K *Blocksize:* 4KB, *I/O space:* 8GB *I/O size:* 512MB, 1024MB, 2048MB
RocksDB & **MongoDB** & **PostgreNoSQL**	*# of threads:* 4; *# of db transactions:* 128K *Record size:* 1KB, *Database size:* 8GB *Total transaction size:* 128MB
Accumulo & **Redis**	*# of threads:* 4; *# of transactions:* 128K *Record size:* 1KB, *Key-value storage size:* 8GB *Total transaction size:* 128MB
Graph algorithms	*# of threads:* 4; *Dataset:* Twitter (24GB) *Vertices:* 2^5 uniform random

TABLE II: Configurations parameters for various workloads.

in the NVMe CMAC cache, D-Shield-Pro first checks the directory to see if that is cached in the main memory (❶). If so, the DMA Interception Engine loads the block from memory by issuing load requests (❷). Note that since we assume that the data in the main memory is protected, the integrity of the storage counters loaded from the main memory would be automatically verified using the security metadata from the main memory. As a result, additional integrity checking using storage metadata is not required. In contrast, if the block is not present in the in-memory cache, it is loaded from the NVMe disk (❸) and will be integrity-checked using the MT from the disk side (❹). A 5-bit Useful flag is used for each CMAC cache entry to aid in selecting eviction victim during eviction. This flag is incremented each time a corresponding SSD block is read and is decremented when a corresponding SSD block is written back or de-allocated. Upon eviction of a CMAC block in the NVMe on-chip cache, D-Shield-Pro populates it to the in-memory cache if the *Useful* flag is non-zero.

Although encryption counters and MACs are not required to be encrypted (same as secure main memory [10], [56], [64]), D-Shield-Pro still leverages the memory controller to encrypt the in-memory cached CMAC blocks. This is done to ensure that when the CMAC block is loaded from the in-memory cache, its integrity can be verified through main memory MT, instead of going through the more expensive verification via the storage security domain. In addition, for dirty CMAC block eviction, D-Shield-Pro also writes this block to the NVMe disk to keep the blocks persistent in the disk. By doing this, the blocks in the in-memory cache are always guaranteed to be consistent with that in the NVMe disk. Therefore, D-Shield-Pro can directly evict an in-memory block without the need for writing it back to the NVMe disk, which can be a costly procedure due to the indirection and complexity involved in memory to disk interactions. We reset the in-memory cache whenever an NVMe namespace is formatted. This can be completed by simply clearing out the on-chip directory.

VIII. Experimental Setup

We build the prototype of D-Shield and evaluate it using SimpleSSD [36] that integrates fine-grained NVMe SSD models on top of the gem5 simulator [13]. We model a shared cryptographic engine used by the memory controller and DIE. Requests to this engine are queued in the ingestion queue of the cryptographic engine and served when the engine is ready. Table I shows the system configurations for the SimpleSSD simulation. We boot a Ubuntu 18.04 system with Linux Kernel 4.9.92 compiled with *dm-crypt* and *cryptsetup* module support. To model secure main memory, we use the same memory controller and main memory configuration as in [10]. For D-Shield-Pro, by default, we use a 128MB in-memory cache.

Workloads. We choose three sets of I/O generations benchmarks with both synthetic and real-world applications: i) the Flexible I/O (`FIO`); ii) the Yahoo Cloud Serving Benchmark (`YCSB`) [27] on real-world server applications; iii) the GAP Benchmark Suite (`GAPBS`) [12] for graph processing. FIO generates disk access workloads including sequential and random workloads for various read and write combinations: *sequential read* (`Seq-R`), *sequential write* (`Seq-W`), *sequential read/write* (`Seq-WR`), *random read* (`Rand-R`), *random write* (`Rand-W`) and *random read/write* (`Rand-RW`). We configure YCSB with five representative applications including RocksDB [17] (NoSQL database), MongoDB [14] (document-store system), PostgreNoSQL [52] (commercial document storage database), Accumulo [46] (unstructured document store system), and Redis [18] (popular key-value database). We choose three workload patterns provided by YCSB based on real-world server traces, including *read/write mixed* with 50% split (`RWm`), *read heavy* with 95% reads and 5% writes (`Rh`), *read only* (`Ro`). Finally, we use graph algorithms provided by GAPBS on Twitter follow network graph [47] as a representative combined compute and I/O bound workload. This graph (24GB) is considerably larger than the system memory. For the graph workloads, on-demand paging is used by default so that only the portion of the graph accessed is loaded during runtime. We instrument each benchmark to perform 10K I/O operations to warmup the metadata caches before collecting statistics. Table II summarizes the benchmarks and workload configurations.

IX. Evaluation

A. Performance Evaluation

To quantify the efficacy of our proposed framework, we additionally set up three baseline configurations: i) *Insecure*, the default NVMe storage system without security mechanisms deployed; ii) *Enc*, the dm-crypt-based block-level encryption for storage system; and iii) *Enc+Int*, dm-crypt with dm-integrity for NVMe disk encryption and integrity checking at block-device granularity. Note that as discussed in Section II-A,

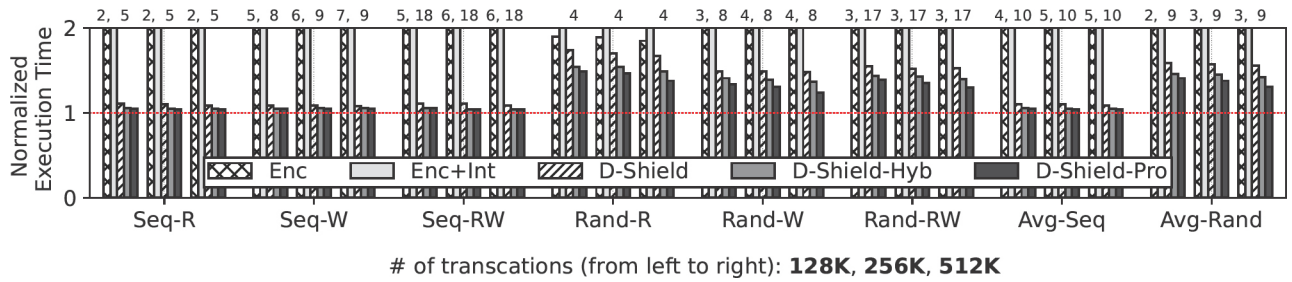


Fig. 13: Runtime (normalized to *Insecure*) of FIO benchmark running on 4 threads for different numbers of I/O transactions.

Enc+Int does not provide full data protection (as compared to the hardware-based mechanism in D-Shield). All configurations have integrated state-of-the-art secure main memory for confidentiality and integrity protection of main memory [10].

Execution Times on I/O Intensive Workloads. Due to the I/O-intensive nature of FIO workloads, the runtime of each workload can represent the raw performance of the underlying storage system. As a result, we first analyze the execution times on FIO using all schemes compared to the *Insecure* baseline. Figure 13 shows the execution times for FIO under various access patterns across different numbers of transactions. As we can see, under the *sequential I/O pattern*, all three D-Shield schemes perform significantly better than the software-based protection schemes. In particular, considering 128*K* transactions, D-Shield demonstrates on average 2.6× and 9× speedup compared to *Enc* and *Enc+Int*, respectively. Interestingly, we observe under the read-only access pattern (i.e., *Seq-R*), the software-based encryption-only scheme (*Enc*) also shows comparatively lower overhead. This is because i) the default configuration of Enc does not require any metadata for decryption, hence the decryption process does not require additional I/O; and ii) the benchmark continuously performs data reads one after another, hence by the time one block read is serviced by the NVMe disk, the decryption operation of the prior block is already finished, causing no bottleneck in operation. More importantly, D-Shield-Pro can retain almost the same performance (i.e., 95%) as the baseline with no disk protection. We note that for sequential operations, the NVMe disk metadata accesses exhibit very high locality, allowing D-Shield to secure NVMe disk data with minimal additional overhead from metadata maintenance.

With random access patterns, D-Shield has on average 49% overhead on random write (*Rand-W*) while showing 74% and 55% performance degradation for random read (*Rand-R*) and random read/write (*Rand-RW*) compared to *Insecure*. The underlying reason is that random disk I/O leads to non-trivial degradation of metadata caching on-chip, resulting in off-chip storage security metadata. For instance, we observe 57% NVMe CMAC cache miss in D-Shield for random workloads. Note that such a phenomenon applies to all security metadata based protection schemes. By reducing the redundant cryptographic operations with D-Shield-Hyb (Section VI), this execution time overhead is minimized to 46% (13% improvement compared to D-Shield). We further observe that the number of cryptographic operations (128-bit granularity) on D-Shield-Hyb is 2.3× lowered compared to regular D-Shield. Finally, D-Shield-Pro can further improve performance due to the utilization of NVMe CMAC caching in main memory. Specifically, we observe 41% runtime overhead among the random patterns (Rand-R, Rand-W, and Rand-RW) compared to the baseline. On the contrary, both *Enc* and *Enc+Int* introduce tremendous performance degradation (up to 18×) for FIO under both sequential and random disk I/Os.

We further characterize the impact of large transaction sizes on D-Shield. While the software based *Enc* and *Enc+Int* schemes mostly observe increased overhead as the transaction size becomes larger (this is likely due to the accumulative effect of computational bottleneck), that is not the case for D-Shield scheme. Specifically for the D-Shield-Pro, we observe that for random workloads the runtime overhead is reduced by 10% with the increase in transaction size from 128K to 512K. This is because the in-memory cache in D-Shield-Pro increases the effective metadata hit ratio with the increase in transactions. In summary, D-Shield-Pro demonstrates only 6% less effective bandwidth than *Insecure* on average while providing state-of-the-art data confidentiality and integrity guarantee.

Throughput on Sever-class Workloads. We evaluate the performance in terms of throughput for the YCSB-based database workloads (shown in Figure 14). We observe that D-Shield has consistently much higher throughput compared to software schemes in all workloads. Specifically, there is only 9% and 4% throughput degradation in D-Shield-Pro compared to 60% and 65% of *Enc+Int* for NoSQL (MongoDB and PostgreNoSQL) and storage engines (RocksDB and Accumulo) respectively. Also, D-Shield demonstrates minimal changes in performance advantage across different read/write patterns for each workload. This is because D-Shield typically does not directly perform additional I/O operations for writes as security metadata are mostly updated in the cache. Finally, for purely memory bound applications (i.e., Redis), we observe less performance advantage for D-Shield over *Enc* and *Enc-Int* schemes (i.e., 5% and 9%). This is because Redis exhibits less intensiveness in terms of disk I/Os as an in-memory key-store database. Overall, D-Shield schemes can maintain 89%, 93%, and 94% throughput of the performance of insecure baseline, and on the other hand, *Enc* and *Enc+Int* only achieve 71% and 47%, respectively.

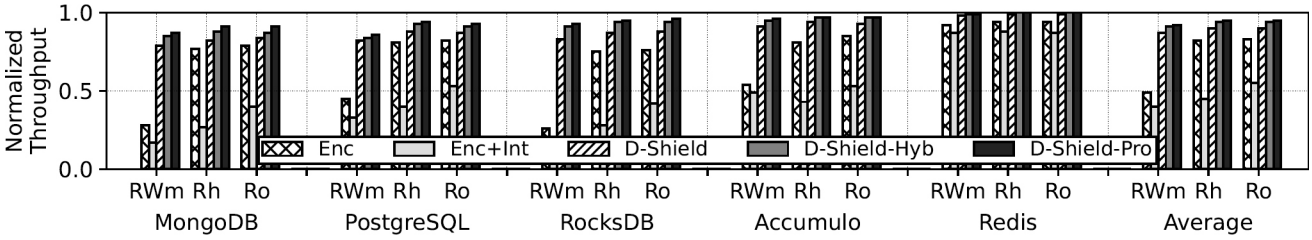


Fig. 14: Performance evaluation (throughput) of D-Shield variants on real-world server applications.

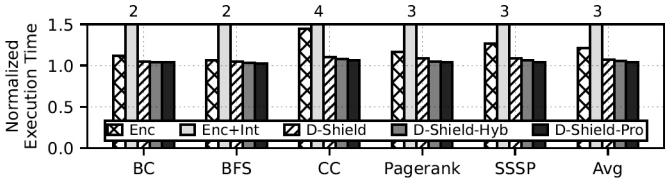


Fig. 15: Performance evaluation (runtime) of graph processing algorithms using *twitter* dataset.

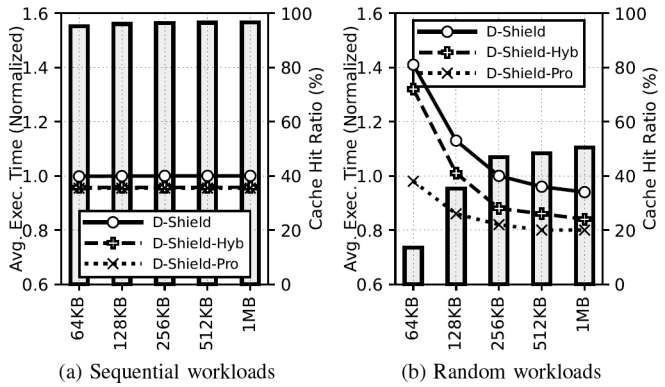


Fig. 16: Metadata cache sensitivity of D-Shield schemes. Bars showing corresponding CMAC cache hit ratio.

Combined Memory/Disk Intensive Workloads. We use graph processing applications to evaluate the impact of D-Shield on combined memory and disk I/O intensive workloads (Figure 15). We process Twitter follower dataset [47] using 5 popular graph algorithms from GAPBS [12]. The layer-wise traversal of *BC* and *BFS* results in higher metadata hit ratio compared to *Pagerank* and *SSSP*. On average, D-Shield schemes have 7%, 5%, and 4% overhead compared to the baseline, while *Enc* and *Enc+Int* have 21% and 213% overheads respectively. Note that *CC* is a directed graph algorithm that is more computationally intensive while sharing similar I/O intensities with other workloads, resulting in a higher overall slowdown in software schemes due to contention on the compute resources. Overall, we observe outstanding performance of D-Shield compared to the insecure baseline while offering complete on-chip storage security

Sensitivity to On-Chip Cache Size. We further study the sensitivity of D-Shield on on-chip metadata cache sizes. Figure 16 shows the execution time of FIO workloads (normalized to default D-Shield configuration) and the corresponding metadata cache hit ratios for different D-Shield schemes. We observe that performance of sequential workloads (Figure 16a) is only negligibly impacted as on-chip metadata cache size increases (i.e., metadata cache hit ratio stays around 96%). This is anticipated due to the high NVMe metadata block to data block coverage ratio (e.g., each counter block covers 32 data blocks in the SSD). For instance, a 256KB CMAC cache maps to 8MB of storage data, and Merkle tree cache with the same size cover hashes for 512MB of data. We observe real-world workloads typically are optimized with high spatial locality and can show good NVMe cache performance. On the other hand, FIO configured with more random patterns shows higher sensitivity (Figure 16b). Specifically, cache hit ratio increases from 13% to 50% with the increase in cache size. It also follows that such benefit of having a larger on-chip cache will increase under such non-deterministic access patterns with larger workload sizes. Finally, we observe that due to the existence of in-memory cache, D-Shield-Pro is less sensitive to on-chip metadata cache size compared to D-Shield, performing well with relatively smaller caches.

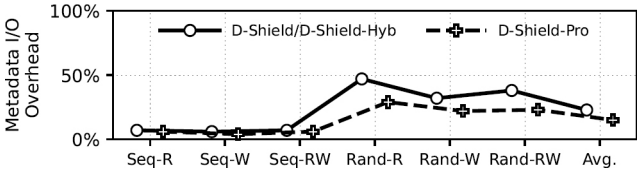


Fig. 17: Additional disk I/O for metadata maintenance.

B. Overhead Analysis of D-Shield

Impact of Metadata Management. We investigate the impact of NVMe security metadata management on the performance of our proposed D-Shield framework. At a high level, the overhead is contributed by additional I/O requests to each of the two types of metadata upon misses in on-chip caches. Figure 17 shows the overall percentage of additional disk I/Os due to the metadata accesses (normalized to all disk I/Os) for the FIO benchmark. Overall, D-Shield and D-Shield-Hyb introduces 7% and 39% of additional overhead for sequential and random workloads while D-Shield-Pro reduces the overhead to 5% and 24% respectively. We note that a considerable portion of these metadata-related I/Os (48% and 33% for Rand-W and Rand-RW respectively) are for metadata block writes, which are not in the critical path of NVMe disk accesses. In general,

real-world workloads exhibit certain degree of data locality for storage I/Os. As one storage metadata block covers many consecutive NVMe data blocks, we observe good metadata hit performance in representative I/O workloads such as the server database applications ($> 85\%$). Additionally, our fused CMAC design further reduces the total number of metadata misses by aggregating storage counters and MAC metadata. To store the counters, MT nodes, and MACs, D-Shield only requires $<3.14\%$ of the storage of NVMe SSDs (i.e., 3.12% for CMAC block, $<0.06\%$ for MT). This amount of metadata is insignificant for typical SSD use cases.

On-Chip Hardware Overhead of D-Shield. To evaluate the logic overhead of D-Shield, we synthesize the proposed logic components for *LBB* and *SCL*. Note that logic overhead for the small region table is trivial. We use Synopsis Design Compiler with the 45nm FreePDK standard cell library [60]. These two require $0.23mm^2$ and $0.08mm^2$ on-chip area respectively, which is negligible compared to the overall CPU die area. Note that D-Shield-Hyb and D-Shield-Pro only need SCL, and LBB is not required. In addition, all D-Shield schemes require two 256KB on-chip metadata cache and D-Shield-Pro requires an additional 128MB in-memory storage. Note that this adds overhead is minimal compared to the memory metadata that is already needed in the baseline with secure main memory (i.e., >250MB for memory counters alone). The region table takes only a minimal 552B storage to support up to 32 individual SSDs. Overall, the hardware design for D-Shield operational logic incurs a very small area on-chip, which is minimal compared to the size of typical processor die [7].

X. Discussion

Similar to secure memories, loss of metadata in storage (i.e., in case of system crash) may result in inconsistency of data blocks. Note that modern processors have integrated *enhanced asynchronous DRAM refresh* (eADR), which is a platform feature ensuring the whole system is powered on until the volatile caches on-chip are drained [42]. As our D-Shield design including the DMA Interception Engine is *on-chip*, it can be drained through extension to the eADR SMI routine execution. Specifically, instead of persisting storage metadata to SSDs that can involve additional software intervention, the cached on-chip storage metadata can be persisted to the main memory in case of a system crash or power loss. During subsequent system boot-up, the storage metadata can be restored from the main memory. In our design, D-Shield metadata only occupies less than 1M on-chip space. We envision that such an extension is unlikely to incur significantly higher hardware costs. Finally, we note that crash consistency for security metadata [10], [68], [72] and other forms of metadata [23] is an active research area that deserves separate studies, we leave the investigation of detailed and more advanced crash consistency schemes for SSDs as future work.

XI. Related work

There are several recent efforts from both academia and device manufacturers to provide security for data-at-rest using full-disk encryption (FDE). One such approach is StrongBox [29] which proposes the use of low-latency stream ciphers (e.g., ChaCha) instead of block ciphers such as AES. This can potentially reduce the computational overhead of software-based FDE. While FDE should be able to protect the confidentiality of the data at rest and in transmission, the integrity of the data is not protected [16] making it vulnerable to data replay attacks. Although there exist several *device mappers* targeting integrity verification, they either do not work on read/write enabled system [6] or fail to provide proper security guarantee against data tampering [19]. Additionally, FDE stores information about the encryption key in the main memory and hence it is vulnerable to cold-boot attacks where the attacker can retrieve the cryptographic key information for an unlocked disk and then retrieve the content of the encrypted file-system using that recovered key information [38].

Self-encrypting disks (SEDs) can not provide the same level of protection as FED since they are fundamentally only able to protect the data at rest. The data transmitted over the bus is unprotected. In addition to that, a recent study has found many inconsistencies in the implementation of SED in 60% of the consumer off-the-shelf SSDs which exposes serious weakness in the system, leading to full-data recovery [50]. Even when implemented properly, SED can only protect the confidentiality of the data at rest, but the integrity of the data can not be maintained. Finally, different from physical attack vector studies, a line of recent works have proposed obfuscating access patterns to SSDs to prevent side channel leakage [9], [39]. These approaches are orthogonal to our work and can be implemented in conjecture with our proposed design to provide protection against physical attacks.

XII. Conclusion

Recent attacks have motivated the need for data protection both at rest and in transmission. While software-based encryption has a prohibitive performance penalty, current hardware based implementations cannot provide necessary security guarantee. In this work, we explore the design space for potential processor-side secure storage design. Based on key design insights, we propose D-Shield, a high performing architectural scheme that provides state-of-the-art data protection for NVMe disks, without changes in NVMe protocol. D-Shield can achieve close to insecure design performance in sequential workloads while improving the performance over software schemes significantly.

Acknowledgement

This work is supported in part by National Science Foundation under grants CNS-2008339 and CNS-1908471. It was partially developed with funding from the Defense Advanced Research Projects Agency (DARPA) and the Office of Naval Research (ONR) when Amro Awad was at UCF. The views, opinions and/or findings expressed are those of the authors and should not be interpreted as representing the official views or policies of the Department of Defense or the U.S. Government. Approved for public release. Distribution is unlimited.

REFERENCES

[1] "File Encryption - Win32 apps | Microsoft Docs." [Online]. Available: https://docs.microsoft.com/en-us/windows/win32/fileio/file-encryption

[2] "Intel® Xeon Processor Scalable Family Technical Overview." [Online]. Available: https://www.intel.com/content/www/us/en/developer/articles/technical/xeon-processor-scalable-family-technical-overview.html

[3] "Samsung SED Security in Collaboration with Wave Systems." [Online]. Available: https://www.samsung.com/semiconductor/global.semi.static/Samsung_SSD_Security_Encryption_Brochure.pdf

[4] "Samsung V-NAND SSD 970 EVO Plus: 2021 Data Sheet." [Online]. Available: https://s3.ap-northeast-2.amazonaws.com/global.semi.static/Samsung_NVMe_SSD_970_EVO_Plus_Data_Sheet_Rev.3.0.pdf

[5] "Utilizing the Intel Xeon Processor Scalable Family IIO Performance Monitoring Events." [Online]. Available: https://www.intel.com/content/www/us/en/developer/articles/technical/utilizing-the-intel-xeon-processor-scalable-family-iio-performance-monitoring-events.html

[6] "Verified Boot | Android Open Source Project." [Online]. Available: https://source.android.com/security/verifiedboot

[7] "Intel® Core™ i7-930 Processor (8M Cache, 2.80 GHz, 4.80 GT/s Intel® QPI) Product Specifications," 2010. [Online]. Available: https://ark.intel.com/content/www/us/en/ark/products/41447/intel-core-i7-930-processor-8m-cache-2-80-ghz-4-80-gt-s-intel-qpi.html

[8] "Intel® Volume Management Device—SSD Hot Plug for the Data Center," 2021. [Online]. Available: https://www.intel.com/content/www/us/en/architecture-and-technology/intel-volume-management-device-overview.html

[9] A. Ahmad, K. Kim, M. I. Sarfaraz, and B. Lee, "OBLIVIATE: A Data Oblivious Filesystem for Intel SGX." in *IEEE NDSS*, 2018.

[10] A. Awad, M. Ye, Y. Solihin, L. Njilla, and K. A. Zubair, "Triad-nvm: Persistency for integrity-protected and encrypted non-volatile memories," in *IEEE ISCA*, 2019, pp. 104–115.

[11] J. Axboe, "Fio-flexible I/O tester." [Online]. Available: https://github.com/axboe/fio

[12] S. Beamer, K. Asanović, and D. Patterson, "The GAP Benchmark Suite," 2015. [Online]. Available: https://arxiv.org/abs/1508.03619

[13] N. Binkert, B. Beckmann, G. Black, S. K. Reinhardt, A. Saidi, A. Basu, J. Hestness, D. R. Hower, T. Krishna, S. Sardashti *et al.*, "The gem5 simulator," *ACM CAN*, pp. 1–7, 2011.

[14] A. Boicea, F. Radulescu, and L. I. Agapin, "MongoDB vs Oracle–database comparison," in *IEEE EIDWT*, 2012, pp. 330–335.

[15] M. Broz, "NVMe Protocol Impact on CPU Utilization," Aug 2015. [Online]. Available: https://global-uploads.webflow.com/5ab1342d0735aa53115fca62/5b47a9ee76d89c63215e55b4_NVME-Protocol_WP_080915.pdf

[16] M. Brož, M. Patočka, and V. Matyáš, "Practical cryptographic data integrity protection with full disk encryption," in *IFIP SEC*, 2018, pp. 79–93.

[17] Z. Cao, S. Dong, S. Vemuri, and D. H. Du, "Characterizing, modeling, and benchmarking rocksdb key-value workloads at facebook," in *USENIX FAST*, 2020, pp. 209–223.

[18] J. Carlson, *Redis in action*. Simon and Schuster, 2013.

[19] A. Chakraborti, B. Jain, J. Kasiak, T. Zhang, D. Porter, and R. Sion, "Dm-x: protecting volume-level integrity for cloud volumes and local block devices," in *ACM APSys*, 2017, pp. 1–7.

[20] S. Chhabra and Y. Solihin, "i-NVMM: A secure non-volatile main memory system with incremental encryption," in *IEEE ISCA*, 2011, pp. 177–188.

[21] M. H. I. Chowdhuryy, R. Ewetz, A. Awad, and F. Yao, "Seeds of seed: R-saw: New side channels exploiting read asymmetry in mlc phase change memories," in *IEEE SEED*, 2021, pp. 22–28.

[22] M. H. I. Chowdhuryy, H. Liu, and F. Yao, "Branchspec: Information leakage attacks exploiting speculative branch instruction executions," in *IEEE ICCD*, 2020, pp. 529–536.

[23] M. H. I. Chowdhuryy, M. R. H. Rashed, A. Awad, R. Ewetz, and F. Yao, "Ladder: Architecting content and location-aware writes for crossbar resistive memories," in *IEEE MICRO*, 2021, pp. 117–130.

[24] M. H. I. Chowdhuryy and F. Yao, "Leaking secrets through modern branch predictor in the speculative world," *IEEE TC*, 2021.

[25] K. D. Community, "dm-crypt - The Linux Kernel documentation." [Online]. Available: https://www.kernel.org/doc/html/latest/admin-guide/device-mapper/dm-crypt.html

[26] M. Conti, N. Dragoni, and V. Lesyk, "A survey of man in the middle attacks," *IEEE Communications Surveys and Tutorials*, pp. 2027–2051, 2016.

[27] B. F. Cooper, A. Silberstein, E. Tam, R. Ramakrishnan, and R. Sears, "Benchmarking cloud serving systems with YCSB," in *ACM SoCC*, 2010, pp. 143–154.

[28] V. Costan and S. Devadas, "Intel SGX Explained." *IACR Cryptol. ePrint Arch.*, pp. 1–118, 2016.

[29] B. Dickens III, H. S. Gunawi, A. J. Feldman, and H. Hoffmann, "Strongbox: Confidentiality, integrity, and performance using stream ciphers for full drive encryption," in *ACM ASPLOS*, 2018, pp. 708–721.

[30] I. Documentation, "About self-encrypting drives." [Online]. Available: https://www.ibm.com/docs/en/psfa/7.2.1?topic=administration-about-self-encrypting-drives

[31] R. Elbaz, D. Champagne, C. Gebotys, R. B. Lee, N. Potlapally, and L. Torres, "Hardware mechanisms for memory authentication: A survey of existing techniques and engines," *Springer Transactions on Computational Science IV*, pp. 1–22, 2009.

[32] P. Flajolet, D. Gardy, and L. Thimonier, "Birthday paradox, coupon collectors, caching algorithms and self-organizing search," *Discrete Applied Mathematics*, pp. 207–229, 1992.

[33] A. Freij, H. Zhou, and Y. Solihin, "Bonsai merkle forests: Efficiently achieving crash consistency in secure persistent memory," in *IEEE MICRO*, 2021, pp. 1227–1240.

[34] B. Gassend, G. E. Suh, D. Clarke, M. Van Dijk, and S. Devadas, "Caches and hash trees for efficient memory integrity verification," in *IEEE HPCA*, 2003, pp. 295–306.

[35] M. Girault and J. Stern, "On the length of cryptographic hash-values used in identification schemes," in *Springer Annual International Cryptology Conference*, 1994, pp. 202–215.

[36] D. Gouk, M. Kwon, J. Zhang, S. Koh, W. Choi, N. S. Kim, M. Kandemir, and M. Jung, "Amber*: Enabling precise full-system simulation with detailed modeling of all SSD resources," in *IEEE MICRO*, 2018, pp. 469–481.

[37] M. A. Halcrow, "eCryptfs: An enterprise-class encrypted filesystem for linux," in *Linux OLS*, 2005, pp. 201–218.

[38] J. A. Halderman, S. D. Schoen, N. Heninger, W. Clarkson, W. Paul, J. A. Calandrino, A. J. Feldman, J. Appelbaum, and E. W. Felten, "Lest we remember: cold-boot attacks on encryption keys," *USENIX Security*, pp. 91–98, 2009.

[39] D. Harnik, E. Tsfadia, D. Chen, and R. Kat, "Securing the storage data path with SGX enclaves," *arXiv preprint arXiv:1806.10883*, 2018.

[40] B. Harris and N. Altiparmak, "Ultra-Low Latency SSDs' Impact on Overall Energy Efficiency," in *USENIX HotStorage*, 2020.

[41] A. Huffman and D. Juenemann, "The nonvolatile memory transformation of client storage," *IEEE Computer Society Press*, pp. 38–44, 2013.

[42] Intel, "eADR: New Opportunities for Persistent Memory Applications." [Online]. Available: https://software.intel.com/content/www/us/en/develop/articles/eadr-new-opportunities-for-persistent-memory-applications.html/

[43] J. Izraelevitz, J. Yang, L. Zhang, J. Kim, X. Liu, A. Memaripour, Y. J. Soh, Z. Wang, Y. Xu, S. R. Dulloor *et al.*, "Basic performance measurements of the intel optane DC persistent memory module," *arXiv preprint arXiv:1903.05714*, 2019.

[44] U. Kanonov and A. Wool, "Secure containers in Android: the Samsung KNOX case study," in *ACM CCS SPSM*, 2016, pp. 3–12.

[45] D. Kaplan, J. Powell, and T. Woller, "AMD memory encryption," *White paper*, 2016.

[46] J. Kepner, W. Arcand, D. Bestor, B. Bergeron, C. Byun, V. Gadepally, M. Hubbell, P. Michaleas, J. Mullen, A. Prout *et al.*, "Achieving 100,000,000 database inserts per second using accumulo and d4m," in *IEEE HPEC*, 2014, pp. 1–6.

[47] H. Kwak, C. Lee, H. Park, and S. Moon, "What is twitter, a social network or a news media?" in *IEEE IW3C2*, 2010, pp. 591–600.

[48] Linux, "nvme-cli: NVM-Express user space tooling for Linux." [Online]. Available: https://github.com/linux-nvme/nvme-cli

[49] S. Liu, A. Kolli, J. Ren, and S. Khan, "Crash consistency in encrypted non-volatile main memory systems," in *IEEE HPCA*, 2018, pp. 310–323.

[50] C. Meijer and B. Van Gastel, "Self-encrypting deception: weaknesses in the encryption of solid state drives," in *IEEE Security and Privacy*, 2019, pp. 72–87.

[51] MICRON, "Data Security Features for SSDs," 2013. [Online]. Available: https://www.micron.com/-/media/client/global/documents/products/white-paper/self_encrypting_drives_white_paper.pdf

[52] B. Momjian, *PostgreSQL: introduction and concepts*. Addison-Wesley New York, 2001, vol. 192.
[53] T. Müller and F. C. Freiling, "A systematic assessment of the security of full disk encryption," *IEEE TDSC*, pp. 491–503, 2014.
[54] NVM Express, "NVMExpress Base Specification Revision 1.4," 2019. [Online]. Available: https://nvmexpress.org/wp-content/uploads/NVM-Express-1_4b-2020.09.21-Ratified.pdf
[55] A. S. Rakin, M. H. I. Chowdhuryy, F. Yao, and D. Fan, "Deepsteal: Advanced model extractions leveraging efficient weight stealing in memories," *IEEE Security and Privacy*, pp. 1157–1174, 2022.
[56] B. Rogers, S. Chhabra, M. Prvulovic, and Y. Solihin, "Using address independent seed encryption and bonsai merkle trees to make secure processors os-and performance-friendly," in *IEEE MICRO*, 2007, pp. 183–196.
[57] G. Saileshwar, P. J. Nair, P. Ramrakhyani, W. Elsasser, J. A. Joao, and M. K. Qureshi, "Morphable counters: Enabling compact integrity trees for low-overhead secure memories," in *IEEE MICRO*, 2018, pp. 416–427.
[58] G. Saileshwar, P. J. Nair, P. Ramrakhyani, W. Elsasser, and M. K. Qureshi, "Synergy: Rethinking secure-memory design for error-correcting memories," in *IEEE HPCA*, 2018, pp. 454–465.
[59] Samsung, "Ultra-Low Latency with Samsung Z-NAND SSD," Samsung Memory Solutions Lab, Tech. Rep., 2017.
[60] J. E. Stine, I. Castellanos, M. Wood, J. Henson, F. Love, W. R. Davis, P. D. Franzon, M. Bucher, S. Basavarajaiah, J. Oh, and R. Jenkal, "FreePDK: An Open-Source Variation-Aware Design Kit," in *IEEE MSE*, 2007, pp. 173–174.
[61] G. E. Suh, D. Clarke, B. Gasend, M. Van Dijk, and S. Devadas, "Efficient memory integrity verification and encryption for secure processors," in *IEEE MICRO*, 2003, pp. 339–350.
[62] K. Suzuki, D. Tonien, K. Kurosawa, and K. Toyota, "Birthday paradox for multi-collisions," in *Springer ICISC*, 2006, pp. 29–40.
[63] Q. Xu, H. Siyamwala, M. Ghosh, T. Suri, M. Awasthi, Z. Guz, A. Shayesteh, and V. Balakrishnan, "Performance analysis of NVMe SSDs and their implication on real world databases," in *ACM SYSTOR*, 2015, pp. 1–11.
[64] C. Yan, D. Englender, M. Prvulovic, B. Rogers, and Y. Solihin, "Improving cost, performance, and security of memory encryption and authentication," *IEEE ISCA*, pp. 179–190, 2006.
[65] Q. Yang and J. Ren, "I-CASH: Intelligently coupled array of SSD and HDD," in *IEEE HPCA*, 2011, pp. 278–289.
[66] F. Yao, M. Doroslovacki, and G. Venkataramani, "Are coherence protocol states vulnerable to information leakage?" in *IEEE HPCA*, 2018, pp. 168–179.
[67] F. Yao, A. S. Rakin, and D. Fan, "Deephammer: Depleting the intelligence of deep neural networks through targeted chain of bit flips," in *USENIX Security*, 2020, pp. 1463–1480.
[68] M. Ye, C. Hughes, and A. Awad, "Osiris: A Low-Cost Mechanism to Enable Restoration of Secure Non-Volatile Memories," in *IEEE MICRO*, 2018, pp. 403–415.
[69] V. Young, P. J. Nair, and M. K. Qureshi, "DEUCE: Write-efficient encryption for non-volatile memories," *ACM ASPLOS*, pp. 33–44, 2015.
[70] J. Zhang, M. Kwon, D. Gouk, S. Koh, C. Lee, M. Alian, M. Chun, M. T. Kandemir, N. S. Kim, J. Kim *et al.*, "FlashShare: Punching through server storage stack from kernel to firmware for ultra-low latency SSDs," in *USENIX Security*, 2018, pp. 477–492.
[71] Y. Zhu, Y. Cheng, H. Zhou, and Y. Lu, "Hermes Attack: Steal DNN Models with Lossless Inference Accuracy," in *USENIX Security*, 2021, pp. 1973–1988.
[72] K. A. Zubair and A. Awad, "Anubis: ultra-low overhead and recovery time for secure non-volatile memories," in *IEEE ISCA*, 2019, pp. 157–168.