NASA: NVM-Assisted Secure Deletion for Flash Memory

Weidong Zhu, *Student Member, IEEE*, and Kevin R. B. Butler, *Senior Member, IEEE*

***Abstract*—Secure deletion in flash-based storage is crucial for data security. However, existing secure deletion schemes for flash memory suffer from performance degradation and reliability issues and cannot provide secure deletion guarantees. Although emerging nonvolatile memory (NVM) allows in-place updates and provides high performance, it is unable to fully replace flash memory and, thus, cannot solve the secure deletion problem. In this article, we propose NVM-assisted secure deletion scheme for flash memory (NASA), a stale-free storage system that combines NVM and flash memory to provide immediate secure deletion without significant performance degradation in SSDs. *NASA* uses block erasure to provide secure deletion guarantees for flash memory and exploits NVM to conceal time-consuming erasure operations. We demonstrate that unless the unique characteristics of NVM are considered, schemes that merely implement existing approaches to secure deletion will end up with stale data replicas within their storage media. Moreover, we evaluate *NASA* with different real-world workloads and demonstrate that *NASA* increases the average latency by 0.01% compared to LRU and decreases 2.1% average latency over the FIFO caching policy. *NASA* is a novel storage system that provides strong secure deletion guarantees with high performance.**

***Index Terms*—Flash memory, nonvolatile memory (NVM), secure deletion.**

I. Introduction

THE "information explosion" has been an issue for decades [1] with no signs of abating. As the amount of stored data increases rapidly, managing and accessing this data in a secure fashion is crucial to assure user security and privacy [2], [3]. Secure data deletion is an important secure data management technology for individuals, enterprises, and governments. Moreover, secure deletion is required by various laws and regulations [4], [5]. For example, individuals could suffer from coercive attackers, such as border guards, who can examine their drives. Secure deletion of potentially controversial material thus provides personal protections. Governments may require secure deletion as well; for example, in 2018, a Belgian court ordered Facebook to delete all illegally collected data from Belgian citizens [6]. Data destruction is required by the U.S. Department of Defense [7] to ensure the erasure of hard drives and other storage devices. Therefore, reliably erasing data from storage is essential for secure data management, and many built-in storage techniques, such as ATA [8] and NVM Express (NVMe) [9], provide secure erase commands.

NAND flash is a popular storage medium for solid-state drives (SSDs) due to its high capacity and performance advantages over magnetic storage found on hard disk drives (HDDs). However, the physical characteristics of flash memory, which necessitate erasing a storage block before it can be rewritten, invalidate assumptions about in-place overwriting of data. Therefore, traditional software-based secure deletion methods, which are effective for HDDs, cannot provide secure deletion guarantees to flash memory.

Past secure deletion schemes remove stale data by leveraging physical features of flash memory [10], [11], [12] or using cryptography [13], [14]. *Scrubbing-based* approaches sanitize flash memory by zeroing individual flash pages (e.g., 4 or 16 kB) to remove data. *Erasure-based* secure deletion is implemented by moving valid data pages within a selected block to other free blocks and then erasing the block containing stale data. *Encryption-based* approaches rely on encrypting data with a cipher such as AES [15]; deletion then occurs by sanitizing the key, rendering the encrypted data unrecoverable. However, these strategies are problematic due to the following reasons.

1) *Security Concerns:* Encryption-based methods are susceptible to side-channel attacks [16], improper key management [17], and bad crypto implementation [18]. For scrubbing-based mechanisms, the physical properties of flash memory can allow the recovery of scrubbed data [19].
2) *Performance Overhead:* Erasure-based secure deletion incurs significant performance overhead due to time-consuming flash block erase operations (e.g., 2 ms). While encryption overheads can be alleviated by deploying hardware accelerators [20], the encryption function is within the critical I/O path; the encryption overhead is still non-negligible [21]. Scrubbing-based approaches also degrade the performance because of extra data page copies required within the flash memory.
3) *Reliability Concerns:* Scrubbing-based strategies decrease the reliability of the flash memory due to the increased bit-error rate (BER) from data disturbance [12], [22]. Moreover, the scrubbing number in a flash block is limited [11]. Thus, scrubbing-based methods need to exploit block erasure to sanitize data; however, the program/erase (P/E) cycles of a flash page are limited, and erasure-based methods shorten the lifetime of flash memory.

Manuscript received 4 August 2022; accepted 5 August 2022. Date of publication 10 August 2022; date of current version 24 October 2022. This work was supported by NSF under Grant CNS-1815883. This article was presented at the International Conference on Embedded Software (EMSOFT) 2022 and appeared as part of the ESWEEK-TCAD special issue. This article was recommended by Associate Editor A. K. Coskun. *(Corresponding author: Weidong Zhu.)*

The authors are with the Department of Computer and Information Science and Engineering, University of Florida, Gainesville, FL 32611 USA (e-mail: zwdong1994@gmail.com).

Digital Object Identifier 10.1109/TCAD.2022.3197514

Emerging nonvolatile memory (NVM), such as STT-MRAM [23], resistive random access memory (ReRAM) [24], PCM [25], and 3-D-XPoint [26], provide higher endurance and lower latency than NAND flash. As such, NVM brings opportunities for secure deletion because it supports in-place write. However, given the price and density consideration, NVM is primarily used for caching within SSDs, but not as a direct replacement for flash memory.

In this article, we present NVM-assisted secure deletion for flash memory (NASA), which integrates NVM technologies into flash-based SSDs to achieve secure deletion with low performance overhead. *NASA* deploys NVM as the cache for flash memory. In contrast to traditional caching systems, *NASA* accommodates secure deletion across different storage devices; we thus leverage the in-place update property of NVM and block erasure operation of flash memory to securely delete data. We consider properties of real-world workloads, including *recency* [27] (i.e., accessed data has a high probability of being used again in the next few I/O requests), *hotness* [28] (i.e., most I/O requests will access relatively few data blocks), and *spatial locality* [29] (i.e., if data is accessed, neighboring data will likely be accessed soon). We leverage these properties by proposing a *Hybrid Evictor* to evict data based on hotness and recency without losing spatial locality. Furthermore, we propose a new copyback method, called spatiality- and hotness-enabled copy back (SHOCK), to prefetch data pages based on the spatial locality and hotness properties when data overwriting occurs within a flash block. Our work makes the following contributions.

1) We develop *NASA*, the first scheme using NVM to assist the secure deletion in flash-based SSDs by integrating NVM into the flash translation layer (FTL). *NASA* exploits block erasure to provide stronger secure deletion guarantees and can effectively shield the time-consuming block erasure operation to provide higher performance than existing secure deletion schemes.
2) We design and implement an NVM emulator (NVMU) that can model the physical structure of an NVM device. Our performance evaluation illustrates NVMU can accurately emulate the latency of a real NVM device with 5.7% and 4.8% latency deviation on write and read.
3) We implement a prototype of *NASA* using NVMU and a flash-based SSD emulator. Our evaluation verifies the existence of stale data in the current cache systems. Compared with the caching policies (LRU and FIFO) without secure deletion, *NASA* increases 0.01% and decrease 2.1% average response latency, respectively. Moreover, we implement existing secure deletion methods as comparisons, and *NASA* achieves lower flash block erases that can improve flash memory reliability.

II. Background

A. Flash Memory Basics

A NAND flash floating-gate transistor is a metal–oxide semiconductor technology capable of trapping electrons between the gate and tunnel oxides. In Fig. 1(a), the electronic charge could be deployed from the substrate to the floating gate through the tunnel oxide. The voltage of a flash cell is determined by the amount of electronic charge in a floating gate. A

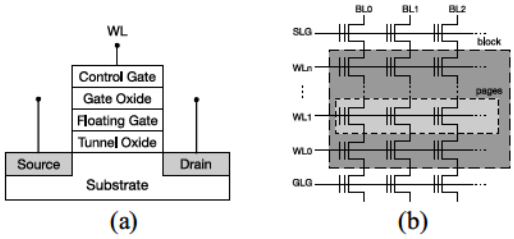

Fig. 1. NAND flash memory floating gate transistor and flash memory array. (a) Floating gate. (b) NAND flash block.

flash cell comprises multiple bits (e.g., 3 bits in TLC) based on the variance of floating gate voltage. In Fig. 1(b), multiple cells are grouped as a wordline (WL), which represents pages. The page number within a WL is determined by the number of bits within the flash cell. The source level gate (SLG) and ground-level gate (GLG) are deployed to clamp WLs, which group to a flash block. Flash operations can thus be performed by adjusting the voltages of the SLG and GLG.

Flash memory supports *program*, *read*, and *erase* operations. Program operations occur at page-level granularity by charging the flash cells within a selected WL to switch the flash cell bit state from 1 to 0. Read operations sense the voltage levels of flash cells within a WL to identify the bit state at page granularity. Erase operations reset the flash cell for serving the next program (i.e., erase-before-program). The erase operation removes charge from the floating gate by applying high voltage to the substrate. Since all flash cells in a block are grounded on the same substrate, erase works at block-level granularity and is time consuming. Furthermore, program/erase (P/E) cycles are limited (e.g., 3000 P/Es for MLC NAND flash memory) for a flash cell.

The scrub was proposed to sanitize the stale data [11]. Scrub requires no removal of charge in a flash cell, and it reprograms the data page from 1s to 0s. Thus, scrub incurs smaller latency overhead than erasure operation without increasing the lifetime of flash memory. However, scrub is less secure than erase operation as discussed in Section V-E.

B. Flash-Based SSDs

Flash-based SSDs provide high performance and energy efficiency compared to HDDs. To fully take advantage of the high bandwidth of SSDs, the NVMe protocol has been proposed to replace traditional storage interfaces (e.g., SATA). The design of most file systems is primarily still based on the block-level granularity of traditional hard drives. Thus, SSDs leverage an FTL to expose a block-based interface to the OS, enabling a clean abstraction of the SSD from the OS. The main functions of the FTL operate independently from the host OS.

Address Translation: It translates the logical block addresses (LBAs) to a physical page address (PPA) on the flash memory. Garbage collection (GC) reclaims the invalidated data, and it has the following operations: 1) select blocks that satisfy the predetermined reclaim threshold; 2) move valid pages within selected blocks to other free blocks; and 3) erase the selected blocks. Since blocks undergoing the GC process cannot provide service for incoming I/Os, the performance is generally greatly degraded. The limited program/erase cycles limit the lifetime of an SSD. The FTL mitigates this through *wear-leveling* (i.e., the technique to distribute writing on blocks

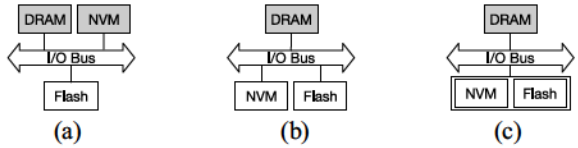

Fig. 2. Different storage hierarchies with NVM. (a) Main memory. (b) NVMe SSD. (c) Cache in SSD.

evenly) and *bad block management* functions, improving the reliability of the SSD.

C. Emerging NVM Technology

Although flash-based SSDs vastly outperform HDDs in performance, they still suffer performance instability because of the GC and endurance problems. Thus, byte-addressable NVM technology dramatically changes the state-of-the-art storage hierarchy. In this article, we use Intel Optane [30] as the NVM device because it is an available NVM device on the market. Intel Optane adopts 3-D-XPoint as the storage media, and it is a kind of PCM [25]. PCM exploits the physical property of chalcogenide [31] by observing that the difference in resistivity between the crystalline (low resistivity) phase and the amorphous (high resistivity) phase is very large. Therefore, the crystalline phase represents "0," and the amorphous phase defines "1" in the storage. Switching between physical phases is achieved by deploying different temperatures: 1) switching from crystalline to amorphous needs a high temperature to melt and then quench the chalcogenide rapidly by performing a high electrical pulse and 2) switching from amorphous phase to crystalline phase needs a lower (still much higher than room temperature) temperature by applying medium electric current for a period of time. Different from NAND flash, PCM supports bit-level in-place writes by directly switching chalcogenide material phases.

NVM is mainly deployed as a cache layer in the current I/O stack. Fig. 2(a) shows a practical architecture where NVM is located in the main memory system and shares the memory bus with DRAM. Since NVM has a much higher capacity than the DRAM and can provide an acceptable bandwidth for main memory, this architecture has been well researched [32], [33], [34]. NVM can also be deployed as a block device in the storage system to avoid using a DRAM DIMM slot, a scarce system resource. In Fig. 2(b), NVM can be used as an independent storage device. For example, Intel Optane SSD [35] is served as a block device and leverages NVMe for transferring data and has RAID-like internal structure [36]. In Fig. 2(c), NVM is incorporated into the flash-based SSD [37], [38], [39], [40], and we will discuss this in a greater detail in Section IX. Although NVM is expensive at the current moment, NVM could provide much higher endurance. For example, the cost of 1 GB of storage capacity of the Intel Optane 905P SSD is 5.6 times to Samsung 870 QVO SSD; however, the endurance of the Intel Optane 905P is 44.2 times higher than the Samsung 870 QVO.

III. Secure Deletion For Flash Memory

Secure deletion aims to securely sanitize data from a storage device to prevent an adversary from recovering the deleted data. However, traditional secure deletion methods for HDDs [41], [42], [43] fails to ensure data sanitization in the SSD because of the erase-before-program property of flash memory. Although the FTL prevents the invalidated data from leaking, adversaries can recover the "deleted" data if they can access the flash memory physically. Therefore, traditional secure deletion approaches are not effective if an adversary has physical access to the SSD.

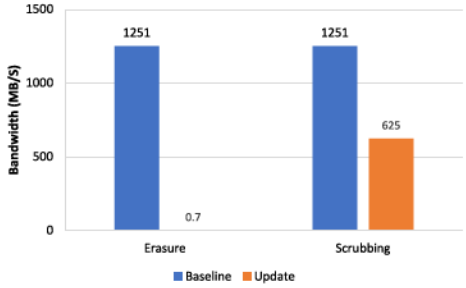

Fig. 3. Bandwidth of erasure- and scrubbing-based secure deletion schemes when a large number of updates are served by flash-based SSD.

A. Erasure-Based Secure Deletion

Erasure-based methods leverage erasure operation to sanitize stale data and migrate valid data pages within the selected block (including the updated data page) to other free blocks. Since discharging a flash cell can reset the bit state to 1 without data remanence issues, erasure-based methods could provide a strong security guarantee to the deleted data. However, erasure-based schemes significantly degrade the performance due to the time-consuming block erasure operation. In Fig. 3, we implement erasure-enabled and scrubbing-enabled SSDs using methodologies in Section VII. When we overwrite the data within a flash-based SSD, the bandwidth of erasure-based method will be decreased by 99.9% compared to a normal SSD. Therefore, the erase operation is impractical for use with secure deletion.

B. Scrubbing-Based Secure Deletion

After scrubbing, the "sanitized" data page could be recovered [19], leading to significant security issues. In Section II-A, electrons are trapped within the floating gate. However, the oxide layer will gradually decay over time, and the voltage of a flash cell will be changed due to leakage of electrons. When data pages are sanitized by scrubbing, flash cells must be reprogrammed from 1s to 0s, and the data 0s will not be charged for reprogramming. Due to the electron leakage, the voltage of data 0s will be different from the reprogrammed 0s. An attacker can thus recover scrubbed data by observing the voltage difference of flash cells.

Leveraging scrubbing for secure deletion is challenging for multiple-bits flash memory (i.e., MLC, TLC, and QLC). For example, QLC flash memory enables 4 bits to be stored within a flash cell, and one WL comprises four data pages. To scrub a data page, we need to move three pages within the selected WL to other free pages and then perform scrubbing on flash cells within the WL. Therefore, three additional read and three additional write operations are incurred that will significantly degrade the performance. Furthermore, reprogramming a data page will lead to data disturbance [12] to its adjacent pages

TABLE I
PARAMETERS OF THE FLASH MEMORY AND NVM USED FOR EVALUATION

Flash Parameter	Value	NVM Parameter	Value
Capacity	128GB	Capacity	300M
Page Size	4KB	Page Size	4KB
Pages Per Block	256	Pages Per Segment	256
Blocks Per Plane	4096	Chip Per Channel	2
Planes Per Chip	1	Channels	3
Chips Per Channel	8	Page Read	4us
Channels	4	Page Write	5us
Page Read	0.04ms		
Page Write	0.2ms		
Block Erase	2ms		

and increase the BER significantly. Since the BER must stay under a specific threshold for reliability, scrubbing in a block should be limited. For example, we could limit the scrubbing number to 16 for a block in a MLC NAND flash [11]. Once the scrubbing number exceeds 16 in that block, erase operations are necessary to securely delete the data. As shown in Fig. 3, scrubbing-based secure deletion will decrease I/O bandwidth by 50% compared to a normal flash-based SSD.

C. Encryption-Based Secure Deletion

For flash-based SSDs deployed with encryption-based secure deletion, I/O requests must be processed by an encryption module. Thus, the performance overhead from encryption cannot be circumvented. In addition, a hardware-backed cryptographic accelerator could be used to improve performance. For example, an AES hardware accelerator [44] provides encrypted throughput of 51.2 Gb/s. In a flash-based SSD with the parameters in Table I, encrypting a data page requires 0.6 us, and we assume the latency of software stack in the OS is 8 us, which is measured in Ubuntu 20.04.4; encryption thus increases flash read latency by 1.2% and flash write latency by 0.3%. However, when considering NVM as a cache within the flash-based SSD, the read and write latency will be increased to 5% and 4.6%, respectively. Therefore, the overhead from encryption cannot be neglected within the NVM-cached flash-based SSD. In addition, the keys used for deletion need to be stored within the flash memory. Erasure or scrubbing operations are still necessary to sanitize the key; encryption-based methods will thus suffer from the issues found in solutions that rely on erasure or scrubbing.

When encryption is performed within the OS [13], the key is vulnerable in the DRAM because the cold boot attack [16] allows the recovery of keys after power is cut. Furthermore, when encryption is performed within the SSD [14], the key should not leave the SSD controller; however, existing works demonstrate that maintaining the key in the SSD is not safe due to the specification issues [17], [45], bad design [17], [46], and improper implementation [17], [18].

IV. THREAT MODEL

We consider an adversary capable of recovering the deleted data from a storage system. The adversary can physically access the storage device, including processor, DRAM, NVM, and flash memory, to perform arbitrary operations (such as read, write, and other low-level memory operations) to any data (i.e., include valid and invalid data). Thus, the attacker can bypass the OS and FTL to directly access the storage cells in the raw NVM and flash chips. For the encrypted data storage, similar to a "peek-a-boo attacker" defined by Reardon *et al.* [13], we assume adversaries can the storage because they can coerce a user to surrender the encryption keys and passwords [47]. Moreover, the adversary has knowledge about how encryption is implemented and can perform side-channel attacks on the main memory (e.g., cold-boot attack [16]) to recover deleted keys in the DRAM. The protection of undeleted data is an active research [48], [49], [50]. We also consider an adversary, who with legal authority (e.g., a border guard), can compel users to reveal their existing data. The protection of nondeleted data is thus an orthogonal problem.

Since we only consider secure deletion in the storage layer, we assume normal users can delete data by overwriting the original logical block address in the OS using a built-in deletion command (e.g., TRIM [51]). Moreover, users cannot perform any extraordinary sanitization operation to force secure deletion before an attack happens (e.g., coercive attack) since we are unable to predict the attack time. We consider obsolete data to be securely deleted if the adversary cannot recover any deleted or stale data from the storage device.

V. NASA DESIGN

Existing secure deletion strategies cannot satisfy secure deletion requirements—performance and security—for flash-based storage systems. Therefore, we propose *NASA*, a secure deletion system that manages data across different storage media, that leverages the block erasure for sanitizing stale data in flash memory. To fully exploit the internal parallelism of flash memory and provide flexibility while managing data movement, *NASA* leverages NVM as a cache by modifying the FTL [39], as current persistent memory region (PMR)-assisted SSDs do not provide internal data migration between NVM and flash memory, forcing data to be transferred through the host I/O stack.

A. Secure Deletion in NASA

NASA fulfills a strong secure deletion guarantee while incurring trivial performance overhead by exploiting NVM. The NVM first serves incoming writes to storage. When an update or deletion operation occurs and the old data resides in the NVM, *NASA* directly sanitizes the old data by performing an in-place overwriting operation with new data or 0s in the NVM. Due to the secure bit switch in the NVM storage as discussed in Section V-E, the deleted data in the NVM will not be recovered. Otherwise, when old data exists in the flash memory, the new data will be written to the NVM, and *NASA* will leverage *SHOCK* for prefetching and erase operations to sanitize the stale data. Thus, *NASA* first locates the flash block, containing the stale data, and then copies valid data pages in the selected flash block to other storage (NVM or other flash blocks) and performs block erasure.

Since erase operations are time consuming and shortens the lifetime of flash memory, we seek to minimize updates in the flash memory. Thus, we propose *Hybrid Evictor* and *SHOCK*. *Hybrid Evictor* exploits the characteristics of recency

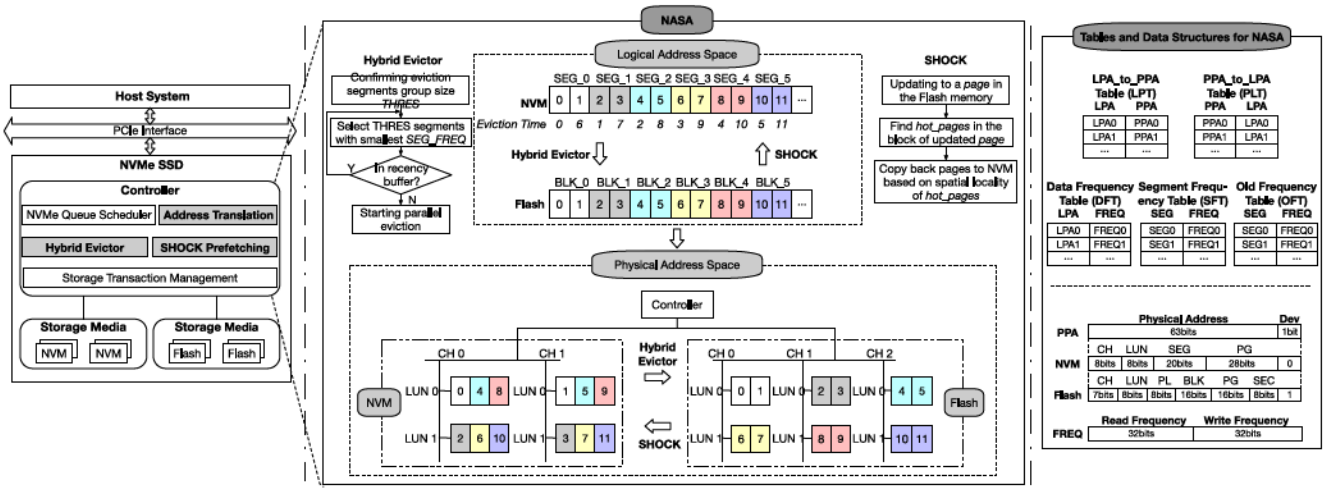

Fig. 4. Architecture overview and data structures of *NASA*.

Algorithm 1 Page Allocation

Input: *SEG_SET* = the segments set in the NVM
1: **for** each segment *SEG* in *Seg_SET* **do**
2: **if** free pages exist in *SEG* **then**
3: *Page* = A free page in the *SEG*
4: break
5: **end if**
6: **end for**
7: return *Page*

(i.e., temporal locality) and hotness in real-world workloads for eviction to increase the hit ratio to the NVM. *SHOCK* prefetches data from flash memory to NVM based on the hotness and spatial locality of the data.

B. Address Translation

Since the NVMe protocol processes data with memory block granularity (e.g., 4 kB), and the flash memory is accessed with page granularity (e.g., 4 kB), *NASA* accesses the data in the NVM with the page size (i.e., flash memory page). Furthermore, *NASA* divides the NVM storage space into segments (SEG), each of which comprises the same number of pages as within a flash memory block. As discussed in Section II-C, NVM-based SSD has a RAID-like structure similar to the flash-based SSD. To exploit the parallelism of NVM, as shown in Fig. 4, *NASA* evenly distributes pages within a SEG to the NVM chips across different channels. *NASA* writes data to segments sequentially to maintain spatial locality of data within the segment. To allocate a free NVM page, the allocation method should be lightweight to avoid performance degradation in searching the free page, and it can leverage the parallelism of the NVM. Thus, as shown in Algorithm 1, *NASA* will query the segment set *SEG_SET* in the NVM to get a segment SEG with free pages and then return a free page for serving the incoming write request.

In Fig. 4, *NASA* uses an logical page address (LPA) to PPA Table (LPT) and a PPA to LPA Table (PLT) to record the address reflection between the LPA in the OS and PPA in the storage. For the PPA, we reserve 1-bit *Dev* to indicate the storage media type and use 63 bits to represent the physical address on the storage device. When the *Dev* bit is 0, the PPA represents the physical address of the NVM. Otherwise, the PPA indicates a flash memory address. The NVM physical address is composed of channel number (*CH*), NVM die number on the channel (*LUN*), segment number in the NVM storage space (SEG), and page number in the NVM chip (*PG*). The physical address in the flash memory is similar to the NVM physical address except for the plane number within a flash die (*PL*), the data block number in a plane (*BLK*), and the sector number in a flash page (*SEC*). In addition, *NASA* will exploit the hotness for data eviction. As shown in Fig. 4, the data frequency table (DFT) is used to record the I/O access frequency of each data page (i.e., LPA), and segment frequency table (SFT) records the I/O access to the segment. In addition, we create an old frequency table (OFT) to record the segment access frequency in the last eviction process for hotness decaying purpose in Section V-C. To evaluate the hotness of data, *NASA* calculates the hotness using the read frequency (FREQ_R) and the write frequency (FREQ_W) with the following equation:

$$\text{HOTNESS} = \text{FREQ}_R * W_R + \text{FREQ}_W * W_W$$

where W_R is the read weight, and W_W is the write weight in a time period. *NASA* monitors the read and write percentage for every 1000 I/O requests. Thus, *NASA* allocates W_R and W_W with read and write percentages. Therefore, *NASA* can judge the hotness flexibly with different I/O access conditions. For example, a read-intensive workload has a higher W_R, and a write-intensive workload will have a larger W_W.

C. Hybrid Evictor

Hybrid Evictor will exploit the characteristics (i.e., recency and hotness) of real-world workloads to decrease data updating in flash memory. Since *NASA* leverages the spatial locality of the evicted data in the flash memory for *SHOCK* in Section V-D, *NASA* will select data with segment granularity and then flush them into the flash memory in parallel. Moreover, we devise a *recency buffer* to record the recency of the accessed NVM segments. *Recency buffer* is a list that is able to represent the accessed sequence to the segments; the head of *recency buffer* indicates the most recent accessed

Algorithm 2 Data Eviction With *Hybrid Evictor*

Input: *SEG_SET* = the segments set in the NVM
RECENCY_BUF = the *recency buffer*
Evic_SEGS = {}
for each segment *SEG* in *Seg_SET* **do**
  if *SEG* in *RECENCY_BUF* **then**
    Continue
  end if
  if the size of *Evic_SEGS* < *THRES* **then**
    Insert *SEG* into *Evic_SEGS*
    Continue
  end if
  if the *HOTNESS* of *SEG* < the smallest *HOTNESS* in *Evic_SEGS* **then**
    Insert *SEG* into *Evic_SEGS*
    Delete a segment with largest *HOTNESS* in the *Evic_SEGS*
  end if
end for
SEG_ID = 0
while *Evic_SEGS* is not empty **do**
  SEG = *Evic_SEGS*[*SEG_ID*]
  Evict a *page* from *SEG* to flash memory
  SEG_ID = (*SEG_ID* + 1) % *THRES*
  if All the *pages* in the *SEG* are evicted **then**
    Remove *SEG* from *Evic_SEGS*
  end if
end while

Algorithm 3 Data Prefetching With *SHOCK*

Input: *Page* = the updated data page in the flash memory
BLK = the flash block of the *Page*
HOT_PG_SET = {}
for each page *PG* in *BLK* **do**
  if the hotness value *HOTNESS* of *PG* > *HOT_THRES* **then**
    Insert *PG* into *HOT_PG_SET*
  end if
end for
for each page *T_PG* in *BLK* **do**
  if *T_PG* in *HOT_PG_SET* or (the LPA of *T_PG* - a LPA in the *HOT_PG_SET*) < *DIST* **then**
    Prefetch *T_PG* to NVM
  else
    Copy *T_PG* to other flash blocks
  end if
end for
Erase *BLK*

segment, and the tail means the segment that is not accessed with the longest time on the list. The incoming I/O requests will update the *recency buffer*. Thus, *NASA* will first query the *LPT* for the incoming I/O request. If the accessed data locate in the NVM, we extract the segment number (SEG) from its PPA and then update the *recency buffer* by inserting the SEG into the head of *recency buffer*. Otherwise, the accessed data does not reside in the NVM, and *NASA* allocates a new *page* using Algorithm 1 and updates the *recency buffer* with segment SEG of the *page*. In addition, the number of segments within the *recency buffer* is no more than the buffer threshold (12% of segments in the NVM based on our evaluation).

We select a limited number *THRES* of segments for eviction, and the *THRES* is determined by the following equation:

$$\text{THRES} = \text{FLASH\_CH} * \text{FLASH\_LUN}$$

where *FLASH_CH* represents the channels (CHs) number, and *FLASH_LUN* indicates the dies (LUNs) number in the flash memory. As shown in Algorithm 2, for each segment SEG in the *SEG_SET*, if SEG is in the *recency buffer*, *NASA* avoids the eviction of it and skip to the next segment. Otherwise, *NASA* leverages the hotness value *HOTNESS* of the SEG by searching the *SFT* to determine whether to select it for the eviction. If the *HOTNESS* of SEG is smaller than all the segments in the selected eviction segments set *Evic_SEGS*, *NASA* will insert the SEG into *Evic_SEGS* and then remove the segment with the largest *HOTNESS* when the segment number exceeds the *THRES*. Once the eviction segments selection finishes, *Hybrid Evictor* will flush the data in the *Evic_SEGS* in parallel as shown in Fig. 4, and the data pages within a segment could be stored within a flash block. Since *NASA* writes the new data into the NVM segment sequentially in Section V-B, the data's spatial locality is maintained within a segment, and the spatial locality could be ensured within a flash block.

We need a hotness decay system to degrade the *HOTNESS* because data pages with high *HOTNESS* can be read or written with a high frequency in a period of time but will not be accessed in the future. Since the high *HOTNESS* data pages in flash memory will not affect eviction, *NASA* only decays the *HOTNESS* of the data in NVM. When eviction happens, *NASA* queries all the segments with *DFT* and *SFT*. If the *HOTNESS* value of the current segment in *DFT* is not changed a lot (no more than 10) compared with the *HOTNESS* in *OFT*, we deem this segment is not accessed frequently for a long time. Therefore, *NASA* will halve the read and write frequency of this segment and all of the data pages in the segment. In addition, *NASA* also updates the *OFT* with the new frequencies of the decayed segment for the subsequent eviction.

D. SHOCK Prefetching

NASA leverages erasure operation to dispose of the old data within the flash memory. To alleviate the negative impact of erasure operation, *SHOCK* prefetches data pages to the NVM. As shown in Algorithm 3, *SHOCK* will first query the data page *PG* within *BLK*, and then insert the *PG* into a hot data page set *HOT_PG_SET* if the hotness value *HOTNESS* of *PG* is larger than the hotness threshold *HOT_THRES* (we set it to 10 based on empirical observation of workloads). Then, *NASA* traverses the pages within *BLK* again for spatial locality judgment using *HOT_PG_SET*. If the traversed page *T_PG* is in the *HOT_PG_SET*, *NASA* prefetches the *T_PG* to NVM. The spatial locality is still maintained for data pages with small address stride [52]. If the LPA distance between the *T_PG* and a page in *HOT_PG_SET* is smaller than the threshold *DIST* (2 * page number of a flash block), *NASA* deems *T_PG* has a good spatial locality with the hot pages and copies it to the NVM. Otherwise, the unprefetched data pages in *BLK* will be copied to another flash block. Finally, *NASA* leverages block erasure operation to sanitize the *BLK*.

E. Security Analysis

This section illustrates how *NASA* can provide strong secure deletion guarantees in presence of a strong adversary defined in Section IV.

Secure Deletion in NVM: To demonstrate secure deletion within NVM, we must address the following two concerns: 1) whether the NVM can perform in-place overwrite (i.e., update or delete) when it receives requests from OS and 2) whether the in-place overwrite can guarantee the old data is unrecoverable. The current NVM-augmented SSDs (i.e.,

Intel Optane SSD) adopt an LBA-based mapping policy [36] to process the incoming I/O request. Therefore, overwritten requests are processed in place. For the NVM media (i.e., 3-D-XPoint), the in-place overwrite means switching between crystalline phase and amorphous phase as we discussed in Section II-C. Moreover, the required temperature for switching to the amorphous phase is much higher than the temperature to the crystalline. For example, GeSbTe [53], which is one kind of chalcogenide glass, needs 600 °C to hit the amorphous phase, but it only needs to maintain the temperature between 100 °C and 150 °C for a period of time to hit crystalline phase. The phase change requires a specific electrical current to generate a suitable temperature, and the overwrite can explicitly change the physical phase of the NVM material. Additionally, since the conductivity in the crystalline phase is stable without the evidence of changing for indicating the stale data [25], we overwrite the deleted data with 0s (i.e., crystalline phase) in the NVM. Therefore, the in-place overwrite in the NVM can guarantee the nonrecoverability of obsolete data.

Considering Reardon *et al.*'s [54] taxonomy of secure deletion adversaries, this approach provides security against the unbounded coercive adversary, who can arbitrarily access the physical media after the data is securely deleted. Note that the "peek-a-boo" attacker defined by Reardon *et al.* [13] is not considered in this setting as we are concerned with post-delete access to data, not an adversary who is allowed to obtain the content of the storage media prior to its compromise. However, it may be possible to apply an approach such as DNEFS [13] to our scheme by storing keys in NVM; we consider this an interesting extension for future work.

Secure Deletion in NAND *Flash:* In the NAND flash, there are two ways to delete the stale data: 1) scrubbing and 2) block erasure. The scrubbing strategy reprograms the obsolete page by charging the remaining 1s NAND cell. However, the charged floating gate will leak continuously and make unreprogramed bits voltage lower than reprogramed bits voltage. Therefore, the scrubbed data can be recovered by exploiting this property that scrubbed flash cells have strong 0s, and unscrubbed flash cells have weak 0s. Although Hasan and Ray [19] proposed partial page programming to make scrubbed data unrecoverable by performing different scrubbing time based on the creation time of the page, the prediction of the scrubbing time is very challenging. Since every write to the flash memory will make the oxide layer of the NAND flash cell to degrade slightly, the NAND flash cell's charge is more likely to be leaked with the increase of writes. They cannot accurately predict scrubbing time. Thus, scrubbing cannot guarantee data secure deletion. On the contrary, block erasure is performed by removing the charge in the flash cell. So, electrical leakage will not be a problem for block erasure.

Latency of Secure Deletion: The deletion latency of a secure deletion approach is important to understand the window of opportunity an adversary has to access data between the time the deletion operation commences and the time in which data is removed from the storage medium. Reardon *et al.* [54] observed that some secure deletion schemes have an indeterminate period between the commencement of the secure delete operation and the erasure of storage medium due to factors such as waiting to batch data. In our approach, the secure deletion operation is immediate: as data is moved to NVM, the stale data block is erased. At worst, given an erase block comprising 256 pages as shown in Table I, this 256-page block would be erased, and 255 (excluding the stale data page) pages would be written to NVM, incurring read overhead from reading the flash memory and write overhead from the NVM. Our calculations, based on the empirical parameters in the Table I, bound this operation at 13.28 ms. Moreover, this is not a time in which an attacker can profitably gain access to data in an intermediate deletion state. Consider the case where the adversary is able to cut the power to the device immediately after a secure deletion request. SSDs contain capacitors that ensure even in the event of sudden power loss, operations can be completed; historically, all secondary storage devices demonstrate similar mechanisms (e.g., magnetic hard drives allow the safe parking of drive heads in the event of power loss). We anticipate that the implementation of these devices will allow for the safe conclusion of read and write activity before the drive loses power.

A caveat to this guarantee is considering the case where explicit secure deletion of the entire drive is requested to occur instantaneously. In the case of magnetic storage, one could perform a whole-drive overwrite or an operation such as degaussing the drive, and the EM properties of such an operation would destroy the drive data. Such an approach would not work with *NASA* because of the reliance on block erasure by the drive for secure deletion. However, such operations can occur in the background, which is the primary use case that we are concerned about, and full block erasure over the entire device can occur efficiently and without significant performance degradation.

VI. Implementation

There is currently no NVM-based SSD emulator available. Thus, to demonstrate *NASA*'s properties, we leverage FEMU [55], a QEMU-based NVMe SSD emulator with an accurate delay emulation module and scalability for supporting high bandwidth of I/O requests. *NASA* incorporates an NVM-based simulator that we develop as an additional module for FEMU, and link the modules together to model an NVM-based SSD accurately.

Based on our evaluation, FEMU could provide bandwidth with 7 GB/s at the maximum that greatly outperforms the existing NVM-based SSD. For example, the read and write bandwidths of Intel Optane SSD 905P [35] are 2.6 and 2.2 GB. For the incoming I/O request, FEMU will generate a latency and then emulates a real delay using delay emulation module. Therefore, to build a NVM-based SSD using FEMU, we need to devise a latency simulation method to determine the access latency to the NVM. Since NVM is parallelized by channel, each channel will be blocked until the read or write is finished, we therefore set variables to represent the next available time on each channel. If a new I/O request comes to a channel where the next available time of it is $T_{\text{AvailOfChannel}}$ and the current time T_{Current} is larger than $T_{\text{AvailOfChannel}}$, the simulated access latency is $T_{\text{Transfer}} + T_{\text{I/O}}$ and we will update the next available time of this channel by $T_{\text{AvailOfChannel}} = T_{\text{Current}} + T_{\text{Transfer}} + T_{\text{I/O}}$, where T_{Transfer} represents the transferring time on the channel and $T_{\text{I/O}}$ represents the read or write latency to a page on NVM. Otherwise, if T_{Current} is less than $T_{\text{AvailOfChannel}}$, the simulated access latency will be

$T_{\text{AvailOfChannel}} - T_{\text{Current}} + T_{\text{Transfer}} + T_{\text{I/O}}$. Thus, the simulated access latency will be sent to the delay emulation module for emulating the real latency. In addition, the calculations of indicators, such as frequencies, are included in the simulator and, thus, their overhead will be incorporated into the emulated latency.

VII. Evaluation

1) *Goals:* We seek to answer the following research questions in our evaluation: *(RQ0)* Is NVMU accurate for demonstrating the efficiency of *NASA*? *(RQ1)* Does stale data exist in with regard to current caching systems? *(RQ2)* Is *NASA* efficient (i.e., performance and cache hit ratio)? *(RQ3)* Does *NASA* affects the lifetime of flash memory? For *RQ0*, in Section VII-A, we test the latency of NVMU to compare with a real NVM-based SSD. To answer *RQ1*, we perform experiments with multiple caching policies across different real-world workloads in Section VII-B. For *RQ2* and *RQ3*, we compare *NASA* against existing caching systems and secure deletion schemes in Section VII-C.

2) *Experimental Setup:* We conduct experiments with an Intel Xeon E3-1245 v5 @ 3.50-Hz 8-core processor with 64-GB DRAM. Ubuntu 20.04.4 with kernel 5.13.4 is deployed as the host OS. We allocate a 50-GB QCOW2 image file for the guest and install Ubuntu 18.04 along with kernel 4.15.0. The emulated SSD consists of 128-GB flash memory and 300-MB NVM using parameters shown in Table I. We allocate 4-GB DRAM to the guest with four vCPUs.

3) *Comparison Selection and Workloads:* To demonstrate the accuracy of NVMU, we compare the latency of NVMU with Intel Optane SSD 905P [35]. We reimplement the existing caching policies (i.e., FIFO [56], [57] and LRU [58]) without secure deletion to compare with *NASA*. Moreover, we compare *NASA* with existing secure deletion methods by combining immediate block erasure (i.e., LRU-IE and FIFO-IE) and scrubbing (i.e., LRU-Scrub and FIFO-Scrub) strategies with LRU and FIFO. When updates or deletions occur within a specific data block, the block erasure scheme copies all valid pages to other free blocks, then performs an erasure operation to purge the stale data. For scrubbing, we emulate its latency by rewriting the updated flash page. When the scrubbing number of a flash block is over 16, we use the same method of erasure scheme to sanitize this block as we discussed in Section III-B. For the update in cache (i.e., NVM), we overwrite the stale data by performing an in-place write. To demonstrate the efficiency of *SHOCK*, we remove the *SHOCK* from NASA as a comparison *NASA-NoSHOCK* and use block erasure for sanitizing the stale data in the flash memory. We use the flexible I/O tester (FIO) benchmark [59] and real-world enterprise workloads from microsoft research (MSR) [60] and virtual desktop infrastructure (Systor [61]) to evaluate NVMU and *NASA*.

4) *SSD Warmup:* To testify the efficiency of a cache system, we need to warm up the storage device to fill up the cache before running the experiment. Therefore, we first use FIO with a sequential write workload to ensure the cache is filled up. Then, for each trace workload, we run one million traces as shown in Table II for warmup before starting the formal experiments. In addition, the ratio of warmup traces number to the formal experiment traces is 10:1.

TABLE II
Characteristics of Our Evaluation Workloads

	Name	Write Ratio	Warmup Traces Number
FIO	SeqWrite	100%	-
	RandWrite	100%	-
MSR	hm_0	73.7%	1 million
	prxy_0	96.9%	1 million
	rsrch_0	90.7%	1 million
	wdev_0	79.9%	1 million
	mds_0	88.1%	1 million
Systor	LUN0	36.9%	1 million
	LUN1	18.8%	1 million
	LUN2	25.1%	1 million
	LUN3	25.4%	1 million

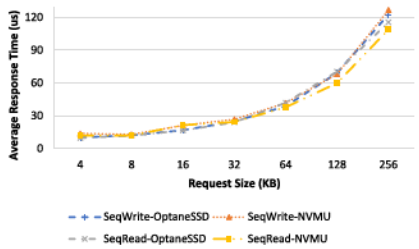

Fig. 5. Average latency comparison between NVMU and Intel Optane SSD.

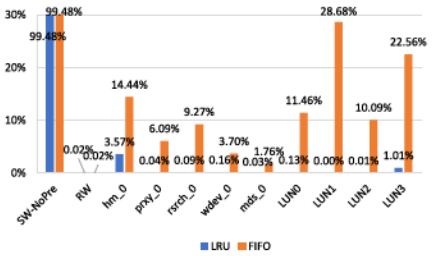

Fig. 6. Stale data percentage to all the writes in FIFO and LRU with different workloads.

A. NVMU Accuracy Testing

We use FIO to test the latency of NVMU and compare it with Intel Optane SSD 905P. The latency of Intel Optane SSD is evaluated from a real device using the FIO benchmark. As shown in Fig. 5, the average read and write request latency is lower than the Intel Optane SSD by 4.8% and 5.7%, respectively. Since NVMU's latency is lower than physical devices; we can accurately model their latency by adding delays to I/O operations. Since the channel number is configurable in NVMU, we can define the parallelism of the NVM device. Since more channels allow for better parallelism, NVMU can exhibit higher bandwidth with more channels. We select three channels for the NVM based on our experiment. NVMU exhibits accurate write and read bandwidths with 2.3 and 2.7 GB/s that are close to the bandwidths (2.2 GB/s for write and 2.6 GB/s for read) of Intel Optane 905p SSD.

B. Stale Data Testing

We test FIFO and LRU caching strategies with MSR and Systor traces to demonstrate the stale data generation in caching systems. Before we calculate the stale data percentage, we perform the aforementioned warmup process, then run the formal workload experiment. In Fig. 6, stale data are

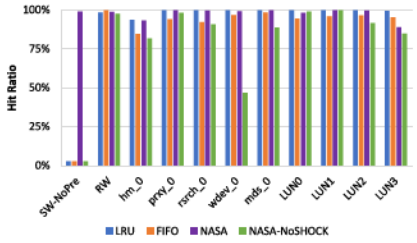

Fig. 7. Hit ratio of I/O requests processed by NVM to all I/Os in FIFO, LRU, and *NASA* with different workloads.

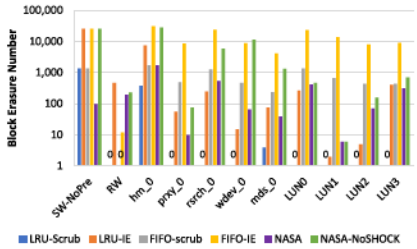

Fig. 8. Block erasure number of LRU-Scrub, LRU-IE, FIFO-Scrub, FIFO-IE, and with different workloads.

inevitably generated in both FIFO and LRU within all workloads. The average percentage of stale data to all written data under LRU and FIFO caching schemes are 9.5% and 18.9%, respectively. The stale data generation is determined by the access pattern of workloads and caching policies.

To testify the stale data generation if no preemption exists in the cache, we use a sequential write benchmark of FIO to write 400-MB data into the SSD and then use the sequential write benchmark to update 100-MB data in the flash memory (i.e., *SW-NoPre*). LRU and FIFO exhibit the same stale data percentage because no cache hit happens for serving the incoming request. In addition, random write (RW) generates RW requests, and there is no specific I/O access pattern can be leveraged by caching policy. Thus, LRU and FIFO are showing similar stale data generation rates. For real-world workloads (MSR and Systor), LRU has a lower stale data percentage than FIFO. Since LRU exploits the recency of real-world workloads, it can decrease the data eviction from NVM to flash memory. As shown in Fig. 7, the average NVM hit ratio of LRU and FIFO schemes is 99.2% and 94.3%, respectively, in the MSR and Systor workloads. LRU method will generate much fewer stale data than FIFO. In addition, although a good caching algorithm can decrease the generation of stale data, Since the stale data generation in the secondary storage (i.e., flash memory) is inevitable, a secure deletion strategy needs to be deployed to the current storage hierarchy.

C. Performance and Block Erasure Testing

For the performance, we compare our method with existing caching policies (FIFO and LRU) and sanitization-enabled caching strategies (i.e., LRU-Scrub, LRU-IE, FIFO-Scrub, and FIFO-IE). In addition, we run each experiment three times.

Fig. 9 shows *NASA* will increase the average response time over LRU by 0.01% and decrease the response time over FIFO by 2.1% on average. In the workload SW-NoPre, *NASA* decreases the average response time over LRU and FIFO by 93.6% and 93.5%, respectively. Since *NASA* leverages *SHOCK*

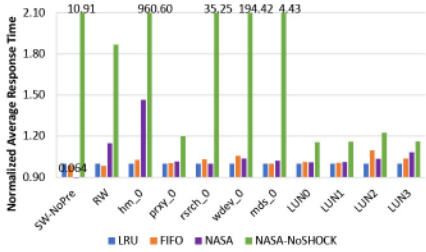

Fig. 9. Normalized average response time of FIFO, LRU, and *NASA* with different workloads.

to prefetch the unpreempted data to the cache, *NASA* can achieve a high hit ratio (99.1%) with few block erasure (100) in SW-NoPre. In addition,*NASA* achieves a high hit ratio because it exploits the workloads' characteristics (i.e., *recency*, *hotness* and *spatial locality*) to decrease eviction. In Fig. 7, *NASA* achieves 97.9% average NVM hit ratio that is higher than 90.4% of LRU and higher than 86.5% of FIFO. Due to the high hit ratio, *NASA* can conceal the low-speed access to flash memory and incurs trivial response time overhead. At worst, in the workload *hm_0*, the average response time of *NASA* increases by 47% and 42% compared to LRU and FIFO, respectively. *NASA* incurs nontrivial performance degradation in this case because the hit ratio (93.4%) of *NASA* is not high enough to conceal the overhead of block erasure operations (1738) in flash memory.

NASA greatly improves the performance compared with existing secure deletion methods. In Fig. 10, *NASA* decreases average response time over LRU-IE, FIFO-IE, and FIFO-Scrub by 74.1%, 93.8%, and 5.2%, respectively. Moreover, *NASA* only increases 0.1% average response time over LRU-Scrub. Erasure-based secure deletion schemes generate tremendous erasure operations in the flash memory. In Fig. 8, LRU-IE and FIFO-IE leads to 3152 and 14279 block erases on average, respectively. Since *NASA* efficiently manages data within the NVM and decreases block erasure in flash memory, *NASA* generates much fewer block erases (only 317). Therefore, *NASA* is able to decrease significant I/O access latency over erasure-based secure deletion strategies. Moreover, scrubbing-based methods still need to exploit block erasure to sanitize data when the scrubbing number in a flash block exceeds the scrubbing threshold, as we discussed in Section III-B. LRU-Scrub and FIFO-Scrub can lead to block erasure operations in flash memory, and they are 161 and 776 on average, respectively. *NASA* leads more block erasure than LRU-IE in workloads of rsrch_0, wdev_0, LUN0, LUN1, and LUN2. *NASA* generates 217.8 block erases on average in these workloads, whereas LRU-IE incurs 107.8 block erases. Moreover, *NASA* only causes 66, 6, and 71 block erases in wdev_0, LUN1, and LUN2, respectively. Since their block erasure numbers are small, they can only impose limited negative effects to the performance and lifetime of the storage device. Therefore, *NASA* achieves a longer lifetime than other erasure-based secure deletion strategies and scrubbing-based methods with inefficient caching policy.

D. SHOCK Testing

Since data eviction is essential for a caching system, we only evaluate the effectiveness of *SHOCK*. As shown in Figs. 8–10,

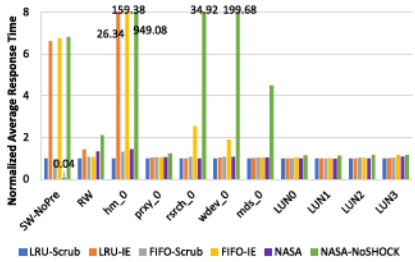

Fig. 10. Normalized average response time of LRU-Scrub, LRU-IE, FIFO-Scrub, FIFO-IE, and *NASA* with different workloads.

NASA-NoSHOCK significantly degrades the performance and lifetime of flash memory. *NASA-NoSHOCK* increases average response time and block erasure over *NASA* by 109× and 19×, respectively. *SHOCK* is an critical part of *NASA* because it can efficiently leverage the spatial locality to decrease block erasure in the flash memory. Moreover, *Hybrid Eviction* can retain the spatial locality of data in the flash memory with a high hit ratio. Although the hit ratio might decrease in some workloads, *NASA* incurs fewer block erasure. Therefore, *NASA* avoids significant performance and storage lifetime degradation.

VIII. Discussion

This section primarily discusses the practicality of *NASA*.

NVM Technologies: Since the recoverability of deleted data in the NVM depends on the NVM material, we should consider the physical characteristics of NVM before developing the secure deletion method. ReRAM [24] has a resistance switching property that the resistance of a ReRAM cell could be changed by deploying the voltage with specific magnitude, polarity and duration. Thus, the resistance of a ReRAM cell has a high-resistance state (HRS) and low-resistance state (LRS) to represents 0 and 1, respectively. Since the LRS and HRS are decided by the different physical status of the stable material, adversaries are unlikely to uncover the overwritten data. For example, a SiO_x ReRAM device has no resistance degradation under the room temperature [62]. Assuming a deletion operation overwrites all the 0s to 1s, the resistance of newly writing 1s will be the same as the old 1s. Therefore, adversaries cannot differentiate the stale data from the overwritten data.

Magnetoresistive random access memory (MRAM) [63] leverages the magnetic tunnel junction (MTJ) to carry the data. An MTJ consists of two ferromagnetic layers (free and pinned layer) and a tunnel barrier. Free layer has various magnetization directions, which decide the resistance of MTJ. If the magnetization directions of the free and pinned layers are parallel, the MTJ has a stable low resistance indicating a 0. Otherwise, the MTJ has a stable high resistance that represents 1. Although the MRAM has durable resistance, not all the MRAM methods can ensure secure deletion. For example, in the STT-MRAM, the write current can lead to a time-dependent degradation [64] of the MTJ, indicating the possibility of deleted data recovery. On the contrary, SOT-MRAM incurs no resistance degradation [64] and, thus, it can provide a secure deletion guarantee.

Caching Policy: Different caching policies incur different performance and reliability impacts on flash-based storage. Since the cache (i.e., NVM) has a higher performance than the secondary storage device (i.e., flash memory), more I/O requests served by the cache will indicate the system has a higher performance. In addition, the erasure operation is time consuming, and it can shorten the lifetime of the flash memory. Therefore, when we design an ad hoc cache system while supporting secure deletion, we should achieve the following targets: a high hit ratio of the cache and few erasure operations in flash memory. To ensure a high hit ratio, *NASA* exploits the hotness and recency properties of real-world workloads to decrease the eviction of those most possibly accessed data. Moreover, *NASA* leverages the spatial locality of the data within the flash memory to prefetch data pages for potential access in the future. Therefore, *NASA* could achieve aforementioned the two targets and provide better performance and reliability to the user with a strong secure deletion guarantee.

Extensibility: NASA could use other NVMs than Intel Optane memory if secure deletion within those memories can be assured. Moreover, *NASA* can also be deployed to other caching architectures as discussed in Fig. 2(a) and (b). Therefore, the OS should be able to operate the raw flash memory and have the following functions: single block erasure, page read, and page write. To achieve this, it would be possible to implement the secure deletion scheme on memory technology device (MTD [65]) and manage physical pages and blocks by using unsorted block images (UBI [66]). Moreover, OpenChannel SSD [67] is an alternative since it opens the internal structure of SSD and allows the user to control the physical data within the flash memory directly.

IX. Related Work

Reardon *et al.* [54] surveyed secure deletion and its implications when implemented at different layers of the storage hierarchy across different media, including flash memory; however, this survey did not consider the emerging NVM. The state-of-the-art secure deletion methods for flash-based SSDs are as follows:

Secure Deletion With Inherent Flash Operations: Erasure and scrubbing operations are exploited for sanitizing deleted data from the flash memory. GC leverages erasure to reclaim the invalidated (i.e., deleted) data periodically [68], [69]. However, GC can only be triggered by specific conditions, such as insufficient free space for serving new I/O requests and, thus, cannot assure secure deletion since attacks can happen before GC occurs. In addition, a large number of erasure operations could act to degrade performance significantly. Evanesco [47] assumes that retrieving raw data within 3-D NAND flash memory is difficult due to the complexity of its architecture. Therefore, two new flash commands are devised to prevent data access to the unsanitized flash block before the erasure operation. However, existing work [70], [71] demonstrates that 3-D flash memory can be profiled with high resolution to distinguish flash cells. Therefore, Evanesco cannot provide a secure deletion guarantee in our threat model.

Wei *et al.* [11] proposed an efficient secure deletion scheme by exploiting scrubbing technology without expensive block erasure. Since scrubbing could result in disturbance and bit errors, Wang *et al.* [12] combined block erasure and scrubbing to minimize secure deletion overhead. Hasan and Ray [19] demonstrated that scrubbed data could be partially recovered due to the analog property of NAND flash cells.

Secure Deletion With Encryption: This method encrypts all the data and then performs deletion by sanitizing the key.

Some works [13], [72], [73] deploy secure deletion within the filesystem. DNEFS [13] deploys secure deletion in the filesystem, while providing fine-grained data access, wear-leveling, and efficiency. Since the keys of data are still stored within the flash memory, the deleted data are vulnerable in DNEFS if an attacker accesses the raw data to retrieve the keys. Swanson and Wei [14] proposed SAFE combining the erasure operation with encryption to sanitize the obsolete data within the flash memory. However, SAFE is a global sanitization method that deletes all the data in the drive. Therefore, it cannot provide real-time secure data deletion.

SSDs With NVM: We further discuss the NVM-assisted SSDs to demonstrate the difference between NASA and other SSDs. PMR [40] exposes the internal NVM of the NVMe SSD to the host with byte accessing granularity. 2B-SSD [38] furthers PMR by designing two independent block- and byte-I/O paths and allowing internal data transfer between the NVM and flash memory. Lee *et al.* [39] considered the high capacity of NVM and integrate the NVM into the flash-based SSD as a cache. Then, they devise a cooperative data management to avoid data copies during GC. Tarihi *et al.* [52] analyzed the characteristic of real I/O workloads and proposed a hybrid DRAM-PCM SSD cache architecture to ensure flash memory's energy efficiency, performance, and lifetime. Although existing NVM-assisted SSDs explore to improve the performance and reliability of the storage, they cannot securely delete stale data from storage medium with trivial overhead.

X. Conclusion

Securely deleting data from storage media is a significant aspect of data security, but the current secure deletion schemes cannot provide both secure deletion guarantee and performance. In this article, we presented *NASA* to solve the secure deletion problem in the flash memory by leveraging the emerging NVM and block erasure operation of flash memory. Moreover, we experimentally demonstrated the existence of stale data in the current caching architecture. *NASA* assures no data remanence through stale data remaining within the storage device. Our evaluation showed that *NASA* can assure secure data deletion while incurring trivial performance degradation and a few flash block erasure.

References

[1] H. F. Korth and A. Silberschatz, "Database research faces the information explosion," *Commun. ACM*, vol. 40, no. 2, pp. 139–142, 1997.

[2] A. Biryukov and D. Khovratovich, "Egalitarian computing," in *Proc. 25th USENIX Security Symp. (USENIX Security)*, 2016, pp. 315–326.

[3] L. Vargas *et al.*, "Mitigating risk while complying with data retention laws," in *Proc. ACM SIGSAC Conf. Comput. Commun. Security (CCS)*, 2018, pp. 2011–2027.

[4] "Health insurance portability and accountability act of 1996 (HIPAA)." 2022. [Online]. Available: https://www.cdc.gov/phlp/publications/topic/hipaa.html

[5] "How to secure with the PCI data security standard." Accessed: 2020. [Online]. Available: https://www.pcisecuritystandards.org/pci_security/how

[6] "Facebook ordered to stop collecting user data by Belgian court." 2018. [Online]. Available: https://www.theguardian.com/technology/2018/feb/16/facebook-ordered-stop-collecting-user-data-fines-belgian-court

[7] "DoD 5220.22-M—The secure wiping standard to get rid of data." 2022. [Online]. Available: https://www.bitraser.com/blog/dod-wiping-the-secure-wiping-standard-to-get-rid-of-data/

[8] "ATA secure erase." 2013. [Online]. Available: https://industrial.apacer.com/en-ww/Technology/ATA-Secure-Erase#:~:text=A%20Secu%20Comma%20era%20is,a%20cann%20%20permanent%20retrieved

[9] "NVMe secure erase." 2022. [Online]. Available: https://industrial.apacer.com/en-ww/Technology/NVMe-Secure-Erase#: ~:text=NV%20Secu%20Era%20%20an,era%20a%20us%20da%20areas

[10] S. Diesburg, C. Meyers, M. Stanovich, A.-I. A. Wang, and G. Kuenning, "TrueErase: Leveraging an auxiliary data path for per-file secure deletion," *ACM Trans. Storage*, vol. 12, no. 4, pp. 1–37, 2016.

[11] M. Wei, L. M. Grupp, F. E. Spada, and S. Swanson, "Reliably erasing data from flash-based solid state drives," in *Proc. 9th USENIX Conf. File Storage Technol. (FAST)*, 2011, p. 8.

[12] W.-C. Wang, C.-C. Ho, Y.-H. Chang, T.-W. Kuo, and P.-H. Lin, "Scrubbing-aware secure deletion for 3-D NAND flash," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 37, no. 11, pp. 2790–2801, Nov. 2018.

[13] J. Reardon, S. Capkun, and D. Basin, "Data node encrypted file system: Efficient secure deletion for flash memory," in *Proc. 21st USENIX Security Symp. (USENIX Security)*, 2012, pp. 333–348.

[14] S. Swanson and M. Wei, "Safe: Fast, verifiable sanitization for SSDs," Dept. Comput. Sci. Eng., Univ. California, San Diego, CA, USA, Rep. TR-cs2011-0963, Jan. 2010.

[15] J. Daemen and V. Rijmen, *The Design of Rijndael: AES—The Advanced Encryption Standard* (Information Security and Cryptography). Berlin, Germany: Springer, 2002.

[16] J. A. Halderman *et al.*, "Lest we remember: Cold-boot attacks on encryption keys," *Commun. ACM*, vol. 52, no. 5, pp. 91–98, 2009.

[17] C. Meijer and B. van Gastel, "Self-encrypting deception: Weaknesses in the encryption of solid state drives," in *Proc. 41st IEEE Symp. Security Privacy (S P)*, 2019, pp. 72–87.

[18] T. Ristenpart and S. Yilek, "When good randomness goes bad: Virtual machine reset vulnerabilities and hedging deployed cryptography," in *Proc. 17th Netw. Distrib. Syst. Security Symp. (NDSS)*, 2010, pp. 1–18.

[19] M. M. Hasan and B. Ray, "Data recovery from 'Scrubbed' NAND flash storage: Need for analog sanitization," in *Proc. 29th USENIX Security Symp. (USENIX Security)*, 2020, pp. 1399–1408.

[20] X. Zhang, H. Li, S. Yang, and S. Han, "On a high-performance and balanced method of hardware implementation for AES," in *Proc. IEEE 7th Int. Conf. Softw. Security Rel. Companion*, 2013, pp. 16–20.

[21] J. Lee, K. Ganesh, H.-J. Lee, and Y. Kim, "FeSSD: A fast encrypted SSD employing on-chip access-control memory," *IEEE Comput. Archit. Lett.*, vol. 16, no. 2, pp. 115–118, Jul.–Dec. 2017.

[22] Y. Cai, O. Mutlu, E. F. Haratsch, and K. Mai, "Program interference in MLC NAND flash memory: Characterization, modeling, and mitigation," in *Proc. IEEE 31st Int. Conf. Comput. Des. (ICCD)*, 2013, pp. 123–130.

[23] D. Apalkov *et al.*, "Spin-transfer torque magnetic random access memory (STT-MRAM)," *ACM J. Emerg. Technol. Comput. Syst.*, vol. 9, no. 2, pp. 1–35, 2013.

[24] H. Akinaga and H. Shima, "Resistive random access memory (ReRAM) based on metal oxides," *Proc. IEEE*, vol. 98, no. 12, pp. 2237–2251, Dec. 2010.

[25] H.-S. P. Wong *et al.*, "Phase change memory," *Proc. IEEE*, vol. 98, no. 12, pp. 2201–2227, Dec. 2010.

[26] "3D XPoint: A breakthrough in non-volatile memory technology." Accessed: 2022. [Online]. Available: https://www.intel.com/content/www/us/en/architecture-and-technology/intel-micron-3d-xpoint-webcast.html

[27] A. Dan and D. Towsley, "An approximate analysis of the LRU and FIFO buffer replacement schemes," in *Proc. ACM SIGMETRICS Conf. Meas. Model. Comput. Syst.*, 1990, pp. 143–152.

[28] C. Min, K. Kim, H. Cho, S.-W. Lee, and Y. I. Eom, "SFS: Random write considered harmful in solid state drives," in *Proc. 10th USENIX Conf. File Storage Technol. (FAST)*, 2012, p. 12.

[29] S. Jiang, L. Zhang, X. Yuan, H. Hu, and Y. Chen, "S-FTL: An efficient address translation for flash memory by exploiting spatial locality," in *Proc. 27th Symp. Mass Storage Syst. Technol. (MSST)*, 2011, pp. 1–12.

[30] "Intel® Optane™ memory—Revolutionary memory." Accessed: 2022. [Online]. Available: https://www.intel.com/content/www/us/en/architecture-and-technology/optane-memory.html

[31] A. Zakery and S. Elliott, *Optical Nonlinearities in Chalcogenide Glasses and their Applications*. Berlin, Germany: Springer, 2007.

[32] J. Xu, J. Kim, A. Memaripour, and S. Swanson, "Finding and fixing performance pathologies in persistent memory software stacks," in *Proc. 24th Int. Conf. Archit. Support Program. Lang. Oper. Syst. (ASPLOS)*, 2019, pp. 427–439.

[33] Y. Kwon, H. Fingler, T. Hunt, S. Peter, E. Witchel, and T. Anderson, "Strata: A cross media file system," in *Proc. 26th Symp. Oper. Syst. Principles (SOSP)*, 2017, pp. 460–477.

[34] R. Dulong *et al.*, "NVCache: A plug-and-play NVMM-based I/O booster for legacy systems," in *Proc. 51st Annu. IEEE/IFIP Int. Conf. Depend. Syst. Netw. (DSN)*, 2021, pp. 186–198.

[35] "Intel® Optane™ SSD 9 series." Accessed: 2021. [Online]. Available: https://www.intel.com/content/www/us/en/products/memory-storage/solid-state-drives/consumer-ssds/optane-ssd-9-series.html

[36] K. Wu, A. Arpaci-Dusseau, and R. Arpaci-Dusseau, "Towards an unwritten contract of Intel Optane SSD," in *Proc. 11th USENIX Workshop Hot Topics Storage File Syst. (HotStorage)*, 2019, p. 3.

[37] "Intel Optane memory H10 with solid state storage." Accessed: 2021. [Online]. Available: https://www.intel.com/content/www/us/en/products/memory-storage/optane-memory/optane-memory-h10-solid-state-storage.html

[38] D. Bae *et al.*, "2B-SSD: The case for dual, byte- and block-addressable solid-state drives," in *Proc. ACM/IEEE 45th Annu. Int. Symp. Comput. Archit. (ISCA)*, 2018, pp. 425–438.

[39] E. Lee, J. Kim, H. Bahn, S. Lee, and S. H. Noh, "Reducing write amplification of flash storage through cooperative data management with NVM," *ACM Trans. Storage*, vol. 13, no. 2, pp. 1–13, 2017.

[40] C. Chadha. "NVMe SSD with persistent memory region." 2017. [Online]. Available: https://www.flashmemorysummit.com/English/Collaterals/Proceedings/2017/20170810_FM31_Chanda.pdf

[41] P. Gutmann, "Secure deletion of data from magnetic and solid-state memory," in *Proc. 6th USENIX Security Symp. (USENIX Security)*, 1996, p. 8.

[42] "Eraser." Accessed: 2022. [Online]. Available: https://eraser.heidi.ie/

[43] R. Geambasu, T. Kohno, A. A. Levy, and H. M. Levy, "Vanish: Increasing data privacy with self-destructing data," in *Proc. 18th USENIX Security Symp. (USENIX Security)*, 2009, pp. 299–316.

[44] Z. Jiang, H. Jin, G. E. Suh, and Z. Zhang, "Designing secure cryptographic accelerators with information flow enforcement: A case study on AES," in *Proc. 56th Annu. Des. Autom. Conf. (DAC)*, 2019, p. 59.

[45] *TCG Storage Security Subsystem Class: Opal Specification Version 2.01*, Trusted Comput. Group, Beaverton, OR, USA, 2015.

[46] "CVE-2018-12037." 2018. [Online]. Available: https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2018-12037

[47] M. Kim *et al.*, "Evanesco: Architectural support for efficient data sanitization in modern flash-based storage systems," in *Proc. 25th Int. Conf. Archit. Support Program. Lang. Oper. Syst. (ASPLOS)*, 2020, pp. 1311–1326.

[48] J. Huang, J. Xu, X. Xing, P. Liu, and M. K. Qureshi, "FlashGuard: Leveraging intrinsic flash properties to defend against encryption ransomware," in *Proc. 24th ACM Conf. Comput. Commun. Security (CCS)*, 2017, pp. 2231–2244.

[49] L. Zhao and M. Mannan, "TEE-aided write protection against privileged data tampering," in *Proc. Netw. Distrib. Syst. Security Symp. (NDSS)*, 2019, pp. 1–15.

[50] S. Baek, Y. Jung, D. Mohaisen, S. Lee, and D. Nyang, "SSD-assisted ransomware detection and data recovery techniques," *IEEE Trans. Comput.*, vol. 70, no. 10, pp. 1762–1776, Oct. 2021.

[51] "IBM spectrum scale with TRIM-supporting NVMe SSDs." 2022. [Online]. Available: https://www.ibm.com/docs/en/spectrum-scale/5.0.5?topic=devices-spectrum-scale-trim-supporting-nvme-ssds

[52] M. Tarihi, H. Asadi, A. Haghdoost, M. Arjomand, and H. Sarbazi-Azad, "A hybrid non-volatile cache design for solid-state drives using comprehensive I/O characterization," *IEEE Trans. Comput.*, vol. 65, no. 6, pp. 1678–1691, Jun. 2016.

[53] "GeSbTe." 2022. [Online]. Available: https://en.wikipedia.org/wiki/GeSbTe

[54] J. Reardon, D. A. Basin, and S. Capkun, "SoK: Secure data deletion," in *Proc. 34th IEEE Symp. Security Privacy (S&P)*, 2013, pp. 301–315.

[55] H. Li, M. Hao, M. H. Tong, S. Sundararaman, M. Bjørling, and H. S. Gunawi, "The CASE of FEMU: Cheap, accurate, scalable and extensible flash emulator," in *Proc. 16th USENIX Conf. File Storage Technol. (FAST)*, 2018, pp. 83–90.

[56] M. S. Bhaskaran, J. Xu, and S. Swanson, "Bankshot: Caching slow storage in fast non-volatile memory," *ACM SIGOPS Oper. Syst. Rev.*, vol. 48, no. 1, pp. 73–81, May 2014.

[57] Y. Oh, E. Lee, C. Hyun, J. Choi, D. Lee, and S. H. Noh, "Enabling cost-effective flash based caching with an array of commodity SSDs," in *Proc. 16th Annu. Middlew. Conf. (Middleware)*, 2015, pp. 63–74.

[58] H. Jung, H. Shim, S. Park, S. Kang, and J. Cha, "LRU-WSR: Integration of LRU and writes sequence reordering for flash memory," *IEEE Trans. Consum. Electron.*, vol. 54, no. 3, pp. 1215–1223, Aug. 2008.

[59] "Flexible I/O tester." Accessed: 2022. [Online]. Available: https://github.com/axboe/fio

[60] D. Narayanan, A. Donnelly, and A. Rowstron, "Write off-loading: Practical power management for enterprise storage," in *Proc. 6th USENIX Conf. File Storage Technol. (FAST)*, 2008, pp. 1–23.

[61] C. Lee, T. Kumano, T. Matsuki, H. Endo, N. Fukumoto, and M. Sugawara, "Understanding storage traffic characteristics on enterprise virtual desktop infrastructure," in *Proc. 10th ACM Int. Syst. Storage Conf. (SYSTOR)*, 2017, pp. 1–11.

[62] A. Mehonic *et al.*, "Intrinsic resistance switching in amorphous silicon oxide for high performance SiO_x ReRAM devices," *Microelectron. Eng.*, vol. 178, pp. 98–103, Jun. 2017.

[63] D. Apalkov, B. Dieny, and J. M. Slaughter, "Magnetoresistive random access memory," *Proc. IEEE*, vol. 104, no. 10, pp. 1796–1830, Oct. 2016.

[64] K. Garello, F. Yasin, and G. S. Kar, "Spin-orbit torque MRAM for ultra-fast embedded memories: From fundamentals to large scale technology integration," in *Proc. IEEE 11th Int. Memory Workshop (IMW)*, 2019, pp. 1–4.

[65] "Memory technology device (MTD) subsystem for Linux." Accessed: 2020. [Online]. Available: http://www.linux-mtd.infradead.org/index.html

[66] "UBI—Unsorted block images." Accessed: 2022. [Online]. Available: http://www.linux-mtd.infradead.org/doc/ubi.html

[67] M. Bjørling, J. González, and P. Bonnet, "LightNVM: The Linux open-channel SSD subsystem," in *Proc. 15th USENIX Conf. File Storage Technol. (FAST)*, 2017, pp. 359–373.

[68] S. Yan *et al.*, "Tiny-tail flash: Near-perfect elimination of garbage collection tail latencies in NAND SSDs," in *Proc. 15th USENIX Conf. File Storage Technol. (FAST)*, 2017, pp. 15–28.

[69] W. Choi, M. Jung, M. Kandemir, and C. Das, "Parallelizing garbage collection with I/O to improve flash resource utilization," in *Proc. 27th Int. Symp. High-Perform. Parallel Distrib. Comput.*, 2018, pp. 243–254.

[70] J. Hirota, K. Yamasue, and Y. Cho, "Profiling of carriers in a 3D flash memory cell with nanometer-level resolution using scanning nonlinear dielectric microscopy," *Microelectron. Rel.*, vol. 114, Nov. 2020, Art. no. 113774.

[71] M. Fan, R. Ranjit, A. Thurber, and D. Engelhard, "High resolution profiles of 3D NAND pillars using X-ray scattering metrology," in *Proc. Metrol. Inspect. Process Control Semicond. Manuf. XXXV*, 2021, Art. no. 116110S.

[72] J. Lee, J. Heo, Y. Cho, J. Hong, and S. Y. Shin, "Secure deletion for NAND flash file system," in *Proc. ACM Symp. Appl. Comput. (SAC)*, 2008, pp. 1710–1714.

[73] L. Yang, T. Wei, F. Zhang, and J. Ma, "SADUS: Secure data deletion in user space for mobile devices," *Comput. Security*, vol. 77, pp. 612–626, Aug. 2018.

Weidong Zhu (Student Member, IEEE) received the bachelor's degree from the Huazhong University of Science and Technology, Wuhan, China, in 2016, and the master's degree from Xiamen University, Xiamen, China, in 2019.

He is a Ph.D. Research Assistant with the Computer and Information Science and Engineering Department, University of Florida, Gainesville, FL, USA. His research focuses on system security, especially leveraging storage to handle security problem.

Kevin R. B. Butler (Senior Member, IEEE) received the Ph.D. degree in computer science and engineering from the Pennsylvania State University, State College, PA, USA, in 2010.

He is a Professor of Computer and Information Science and Engineering with the University of Florida, Gainesville, FL, USA, and the Director of the Florida Institute for Cybersecurity Research, Gainesville. His research focuses on establishing the trustworthiness of computer systems and embedded devices.

Prof. Butler is a Senior Member of ACM and IEEE.