Journal of Hardware and Systems Security (2018) 2:179–200
https://doi.org/10.1007/s41635-018-0034-5

A Survey of Techniques for Improving Security of Non-volatile Memories

Sparsh Mittal[1] · **Ahmed Izzat Alsalibi**[2]

Received: 18 December 2017 / Revised: 8 February 2018 / Accepted: 13 February 2018 / Published online: 2 March 2018

Abstract

Due to their high-density and near-zero leakage power consumption, non-volatile memories (NVMs) are promising candidates for designing future memory systems. However, compared to conventional memories, NVMs also face more severe security threats, e.g., the limited write endurance of NVMs makes them vulnerable to write attacks. Also, the non-volatility of NVMs allows the data to persist even after power-off, which can be accessed by a malicious agent. Further, encryption endangers NVM lifetime and performance by reducing the efficacy of redundant write avoidance techniques. In this paper, we present a survey of techniques for improving security of NVM-based memories by addressing the aforementioned challenges. We highlight the key ideas of the techniques along with their similarities and differences. This paper is expected to be useful for researchers and practitioners in the area of memory and system security.

Keywords Review · Counter mode encryption · Data shredding · Write attack · Side channel · Write endurance · Non-volatile memory · Phase change memory · Domain wall memory

1 Introduction

In the face of ever-increasing memory capacity demands and plateauing energy budgets, researchers have been forced to explore low-leakage, high-density alternatives of conventional memories. Non-volatile memories (NVMs) present as promising candidates for designing next-generation memory systems due to their high-density, near-zero leakage power consumption and non-volatile nature. However, the crucial limitations of NVMs are their limited write endurance (WRE) and high write energy/latency [1]. For example, the WRE of static/dynamic RAM (SRAM/DRAM) and phase change memory (PCM) are 10^{16} and 10^{9}, respectively. The multi-level cell (MLC) NVMs, which provide higher density and cost-efficiency than their single-level cell (SLC) counterparts, have even higher write energy/latency values [2].

✉ Sparsh Mittal
sparsh@iith.ac.in

Ahmed Izzat Alsalibi
asalibi@israa.edu.ps

[1] IIT Hyderabad, E-621, IIT Hyderabad, Khandi, Telangana, 502285, India

[2] Israa University, Gaza, Palestine

While these limitations degrade the performance/energy efficiency of NVMs, their impact on NVM security is even more severe. For example, the essential characteristic of encryption algorithms, namely the diffusion property, is at odd with the NVM write reduction schemes such as redundant write avoidance (refer to Section 2). Hence, encryption itself may degrade NVM lifetime and aggravate the write overhead. Further, an adversary can repeatedly write a memory cell to reach its WRE limit and thus, destroy the memory. The attack program can even be five to six lines of C code that can be executed with user privilege [3, 4]. Apart from malicious entities, even greedy users can launch such attack within the warranty period to obtain a new system. Such write attacks are especially a concern for systems providing 24/7 service since they provide large up-time for a write attack to succeed. Clearly, wide-scale commercial adoption of NVMs is possible only when the security vulnerabilities of NVMs are adequately addressed. Several recent techniques seek to address these challenges.

Contributions In this paper, we present a survey of techniques for improving security of NVMs. Specifically, we include the techniques proposed for mitigating stolen memory attacks, bus snooping attacks, and attacks which exploit limited WRE of NVMs and techniques reducing

overhead of data shredding (refer to Section 2.2). Figure 1 shows the organization of the paper. Section 2 presents a background on useful concepts and classifies the research works based on key parameters. Section 3 discusses techniques for mitigating write attacks. Sections 4 and 5 discuss techniques for reducing overhead of encryption and data shredding in NVMs, respectively. Section 6 reviews encryption techniques for domain wall memory (DWM). Section 7 concludes this paper with a discussion on future directions.

To limit the scope of the paper, we include only those works which are evaluated in context of NVM security, even though some security techniques proposed for conventional memories may also be applicable for NVMs. Since the evaluation platform and workloads used in different works vary, we include only their key ideas and do not generally show their quantitative results. Even though we discuss a technique under a single heading, many techniques span across the boundaries. This paper is expected to be useful for both newcomers and experts in the area of NVMs and memory security.

2 Background and Overview

In this section, we first discuss some terms which will be useful throughout this paper (Section 2.1). We then discuss the types of attacks considered in this survey (Section 2.2), the difference between direct and indirect encryption (Section 2.3), the motivation for using counter mode encryption (Section 2.4), and the avalanche effect in encryption (Section 2.5). We finally provide an overview of research works by organizing them in different categories (Section 2.6). We refer the reader to prior works for more details on the working mechanism of NVMs [5, 6], process variation [7], wear leveling [8], and encryption algorithms [9].

2.1 Useful Terms and Concepts

1. Redundant write avoidance scheme: Since NVM writes incur much more overhead than reads, by avoiding redundant writes to NVM, a large amount of energy can be saved. Specifically, for a write operation to NVM, the existing data can be read and compared with the incoming data, and then, only the changed bits are written. These and similar techniques are also termed as "data-comparison write" schemes.
2. Partial-write scheme: Conventionally, a writeback from LLC writes the entire block (e.g., 64B) to the main memory. However, since only few bits of a block are generally modified, this scheme incurs unnecessary write operations. The partial-write scheme works by logically dividing the cache block into multiple partial-blocks (called "words") and tracks dirtiness at the level of words. Then, only dirty (i.e., modified) words are written-back which reduces the number of write operations.

2.2 Types of Attacks

We now list the types of attacks discussed in this survey and compare their properties in Table 1.

1. Stolen memory attack: An adversary who has physical access to NVM can read its data. Although both volatile memories and NVMs are vulnerable to this attack, the vulnerability of NVMs is even higher since they retain data even after power-down.
2. Bus snooping attack: An adversary can detect data which is communicated over the off-chip network.

Fig. 1 Paper organization

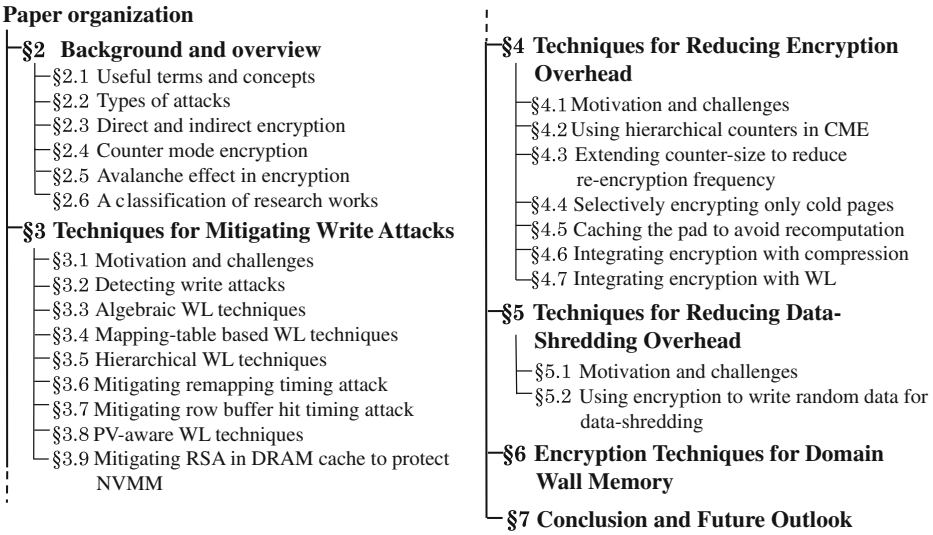

Table 1 A comparison of different attacks

Name	Only in NVMs	Memory destroyed	Data stolen	Mitigation
Stolen memory attack	No, but more severe in NVMs due to data-retention property	No	Yes	Encryption
Bus snooping attack	No	No	Yes	Encryption
Write attacks	Yes (due to limited WRE of NVMs)	Yes	No	WL and write reduction

3. Write attacks: The attacks which aim to destroy the NVM by repeatedly writing a memory cell to exceed its WRE limit are termed as write attacks. Since the WRE limit of volatile memories (e.g., SRAM/DRAM) is much higher than that of NVMs, they can be considered immune to write-attacks for all practical purposes. Thus, only endurance-limited memories (e.g., NVMs) are vulnerable to write-attacks.

As we show below, write attacks can be further classified based on the side-channel information that they exploit. These side-channels are formed due to certain characteristics of NVM and the main memory architecture.

1. Repeated address attack: This issues repeated writes to a physical address (PA) to make it fail. By changing the write-pattern, several variants of this attack can be created [10].
2. Repeated set attack: In a system with DRAM cache and non-volatile main memory (NVMM), a repeated set attack can be created by aggressively writing to addresses mapped to the same set of the DRAM cache [11, 12]. This may force DRAM cache to produce heavy writeback traffic to accelerate wear-out of NVM. Thus, repeated set attack in DRAM cache may manifest as repeated address attack in NVMM.
3. Remapping timing attack based on read/write and SET/RESET asymmetry: NVMs show write asymmetry whereby the write latency are much higher than the read latency [8, 13]. Also, the write latency of SET/RESET operations are different. For example, in PCM, latency of read/SET/RESET operations are 125/1000/125 ns respectively [8, 13, 14], and thus, the write/remap latency is different depending on the data value of written/remapped block. Since a remapping event stalls subsequent requests, by observing the delay in servicing of a request, a remapping event can be detected. Based on this, an adversary can write specific data values in a sequence and learn about the mapping decisions made by the wear-leveling (WL) technique [14]. This attack, termed as remapping timing attack (RTA), can fail NVM memory orders of magnitude faster than repeated address attack since it attacks a line/region accurately [14]. Also, RTA can circumvent WL techniques designed for mitigating repeated address attack (e.g., [3, 15]).
4. Row buffer hit timing attack: Main memory design uses a row buffer to exploit the access locality. However, this can provide a side channel from where the details of address remapping can be leaked. Specifically, by detecting the latencies of a row buffer hit and a row-buffer miss, information about LAs mapped to the same physical row can be inferred since they will lead to a row buffer hit. By tracking the LAs mapped to a given row, the LAs newly mapped to this row or no longer mapped to this row can be identified. Then, certain LAs can be chosen for attacking, so that a huge amount of write traffic is directed to a selected PA to make it fail. This attack is referred to as "row buffer hit time attack" [16], and it can also bypass the techniques proposed for mitigating repeated address attack.

2.3 Direct and Indirect Encryption

The encryption techniques can be divided into two types. In the "direct encryption", data is directly encrypted to ciphertext. By comparison, in the "indirect encryption", the encryption algorithm is first applied to obtain an intermediary data, called "pad" (or one-time pad, since it is used only once). Then, by XORing the pad with the data, encryption and decryption are achieved. Figure 2 shows the decryption operation in both direct and indirect encryption techniques. While direct encryption incurs lower cost, it incurs high latency since the decryption happens on critical path. By comparison, indirect encryption allows hiding decryption latency since it can be performed in parallel with the memory access. Hence, several recent works advocate use of indirect encryption to boost performance.

2.4 Counter Mode Encryption

In the encryption process, if all the cache lines use an identical key, an adversary can compare encrypted lines to identify the cache lines storing the same value and then launch a dictionary-based attack. To avoid this, the address of each cache line can be used along with the key for performing encryption. Thus, the "overall" key for each cache line becomes unique which thwarts "stolen memory attack". The limitation of this scheme is that successive writes to a cache line can still be monitored using the "bus snooping attack".

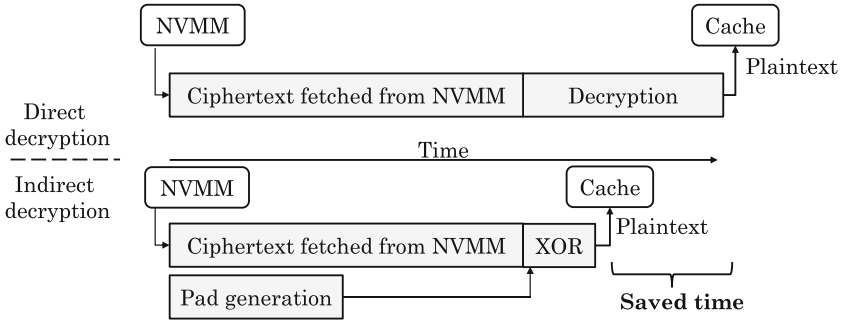

Fig. 2 The working and latency of decryption operation in direct and indirect encryption techniques

To strengthen the security even further, a per-line counter can be used along with key and line address for performing encryption. The counter value can be generated from a function which produces distinct values over a long period (e.g., [17]). In practice, the counter is simply incremented by one on each write. This ensures uniqueness of the overall key for every write to every line and thus, insulates the memory from both "stolen memory" and "bus snooping" attacks. Figure 3 summarizes the encryption and decryption operations in counter mode encryption (CME).

Counter Cache Since the counters are stored along with the ciphertext data in the NVMM, fetching the counter from NVMM delays the encryption process. To alleviate this overhead, the counters can be cached in a structure called the "counter cache" [18]. This allows overlapping encryption latency with the NVM access latency.

2.5 Avalanche Effect in Encryption

A good encryption algorithm has the property of diffusion whereby even a small change in plaintext changes a large number of bits in ciphertext. This is also referred to as avalanche effect. The diffusion property ensures that for two plaintexts with only minor difference, their ciphertexts have no relationship. Thus, even a minor change in plaintext or update of the counter in CME changes the ciphertext completely. Figure 4 shows an example of this, where a change in just one bit in plaintext leads to change in 65 bits in the ciphertext [19]. Avalanche effect reduces the effectiveness of bit-level and word-level redundant write avoidance techniques in NVMs and hence, leads to write-amplification. Hence, special techniques are required for using encryption in NVMs, as we discuss in Section 4.

2.6 A Classification of Research Works

Table 2 classifies the research works based on their objective and memory technology for which they are evaluated.

3 Techniques for Mitigating Write Attacks

The limited WRE of NVM makes them vulnerable to write attacks. In this section, we first highlight the factors involved in mitigating write attacks (Section 3.1) and then review techniques for detecting write attacks (Section 3.2), mitigating different types of write attacks using various WL techniques (Sections 3.3–3.7), accounting for process variation (PV) in WL techniques (Section 3.8), and thwarting repeated set attack in DRAM cache to protect NVMM (Section 3.9).

3.1 Motivation and Challenges

Protecting NVMs against write attacks presents several challenges.

1. Ineffectiveness of "delayed write" scheme: A naive policy to defer attacks is increasing the time to write to the same line. For this, the NVMM write queue

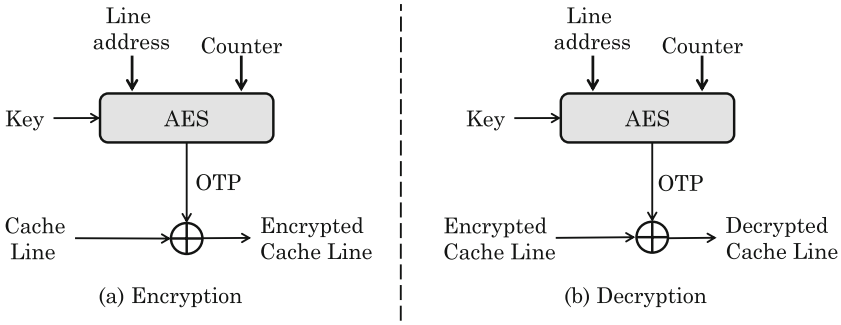

Fig. 3 In counter mode encryption, one time pad (OTP) is generated using the line address, counter, and key for both **a** encryption and **b** decryption

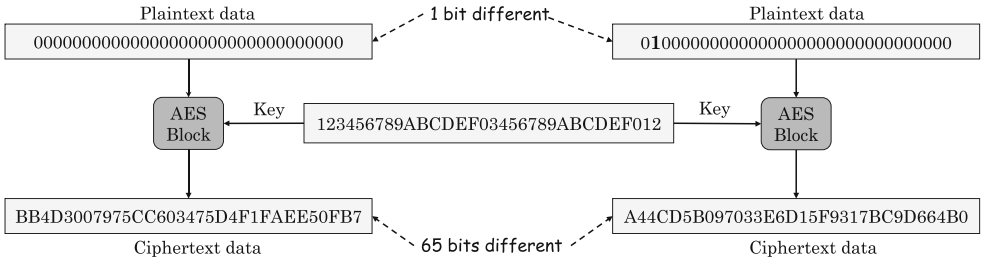

Fig. 4 An illustration of avalanche effect in encryption

can delay the writes till the queue occupancy reaches a fixed level (e.g., 1/4 of total capacity). This forces the adversary to write to multiple lines repeatedly. Since an attack is unlikely to continue for long, such an attack can be detected and thwarted, e.g., the OS may reclaim the page from a process which renders the attack ineffective. The limitation of this policy, however, is that it slows even benign writes (processes) and thus, leads to large performance loss.

2. Limitations of redundant write avoidance schemes: An adversary can circumvent bit-level and word-level redundant write avoidance schemes by writing inverted and shifted data, respectively.
3. Limitations of WL algorithms: In an NVM which does not utilize WL, the LA-to-PA mapping remains fixed which allows the adversary to easily launch repeated address attack. WL techniques are used to distribute the write traffic uniformly to avoid early wear-out of few cells [34]; however, the migration operations introduced by WL can also be leveraged to hide the actual location of a data item from an outsider. Thus, the WL techniques proposed for improving NVM *lifetime* can be rearchitected to improve NVM *security* also. Simple WL techniques remap the blocks in a systematic and predictable manner which can be inferred by the adversary. To strengthen the security even further, the remapping relationship in WL can be dynamically changed over time. This makes it difficult for an adversary to infer location of a PA inside the memory and thus, forces him to write to many cells which slows down the attack.

   As a tool for improving security, the key limitation of WL is that it only *redistributes* and *does not reduce* the total writes to memory. WL can delay the first failure of any cell but cannot delay the complete failure of the memory. Further, WL introduces additional writes which harms performance due to high write overhead of NVMs. Also, security-aware WL schemes (e.g., [3, 15, 31]) perform remapping at much higher rate than what is required for normal programs. Clearly, WL can provide only limited protection under a write attack, and hence, the techniques for reducing writes to NVMs (e.g., DRAM cache) are also required.
4. Tradeoffs of mapping schemes in WL techniques: In WL techniques, use of a redirection table to store LA to PA mapping allows implementing complex and dynamic mapping policies which can make it difficult for an attacker to infer the mapping. However, these tables also incur latency/area overheads. By comparison, use of algebraic mapping alleviates these latency/area costs. However, this mapping needs to be changed periodically to more effectively hide the mapping from an attacker. Also, in algebraic mapping, simply knowing the mapping formula allows an attacker to know the mapping of all the LAs, whereas in case of redirection table, an attacker cannot know all the remapping relationships by knowing one (or few) remapping relationship(s).

Table 2 A classification based on objective and memory technology evaluated

Classification	References
Objective/approach	
Energy	[20–24]
Wear leveling	[3, 10, 11, 14–16, 19, 21, 24–31]
Error correction	[19, 29]
Compression	[26]
Accounting for PV in WRE	[28]
Hierarchical WL	[14, 16, 24, 31]
Memory technology evaluated	
MLC/TLC ReRAM	[20]
Domain wall memory	[32, 33]
PCM	Nearly all others

Evidently, thwarting write attacks presents crucial challenges. Table 3 provides an overview of research works on mitigating write attacks.

3.2 Detecting Write Attacks

Accurate and timely detection of write attacks is the crucial first step in thwarting such attacks, since it allows triggering the security schemes and also adapting their aggressiveness.

Table 3 A classification of works on mitigating write attacks

Classification	References
Mapping strategies in WL schemes	
Algebraic mapping	[3, 10–12, 14–16, 21, 24–27, 29]
Both mapping table and algebraic mapping	[10, 30, 31]
Side channels for inferring remapping in WL	
NVMM row buffer hit time	[16]
Asymmetry in NVM read/SET/RESET time	[14]
WL granularity	
Intra-block WL	[26]
Inter-block/page WL	[3, 10, 11, 14–16, 19, 21, 24, 25, 27–31]
Other features	
Detecting write attack	[4, 10]
Use of Feistel network	[14, 15, 32]

Chhabra et al. [4] evaluate multiple metrics for identifying applications with anomalous behaviour in endurance-limited memories. An anomalous program may not be malicious, rather it is a program which lowers the lifetime of the system. The first metric "write per instruction" is expected to be high for an attacking code; however, this metric can be fooled by inserting `NOP` (no-operation) instructions. A second metric "write traffic distribution" measures intensity of writes to a few pages. This metric is based on the observation that an attack needs to repeatedly write to the same address in a short window of time, and hence, it needs to issue stores to the same address. However, an attacker can fool this metric by writing uniformly to all the lines of the page, although the attacker would have to now direct higher write traffic to destroy the page.

The third metric "writeback traffic per page" (WTPP) is defined by first assuming an ideal lifetime of PCM memory and then back-calculating the writeback traffic to each page to achieve ideal lifetime. Then, a program with higher than this WTPP is considered as an anomalous program. Any program with WTPP lower than this value will not be able to destroy the memory before its ideal lifetime and thus, such a program is not anomalous. A normal program with high WTPP will also be flagged as anomalous to trigger prevention mechanisms. Thus, they argue that WTPP is a complete metric for identifying programs with anomalous behaviour.

Qureshi et al. [10] note that while lowering the rate of WL leaves the system vulnerable to attacks, increasing the WL rate write overhead greatly, even for benign processes. Hence, they propose that the WL rate should be decided by the nature of memory access stream such that it is low/high for common/attacking programs, respectively. This is illustrated in Fig. 5.

They propose a scheme for detecting malicious write patterns. To find the attacking write patterns, they define "attack density" (AD) as the ratio of number of writes to the most heavily written line to the total writes in a time period. An attack that writes to the same line has AD = 1; however, given the multi-level cache hierarchy and large-sized caches in modern processors, repeatedly writing a memory line in a short time-period is difficult. For example, in common programs, the chances of two identical writebacks in last 1K writebacks are extremely small (less than 10^{-6}). Thus, to measure AD with low overhead, they use an attack detector which uses a buffer with few entries (e.g., 16) and inserts the write addresses to it with small probability (e.g., 1/256). This is nearly as effective as using a buffer with 1K entries. As for the replacement policy for this buffer, they show that use of "least recently used" (LRU) policy is ineffective since by writing to random lines for sufficient number of times, an attacked line can be evicted from the detector buffer. Hence, they use a frequency-based replacement policy which records the number of writes to each entry. In this policy, an entry with the lowest (possibly zero) number of writes, except a newly installed entry, is

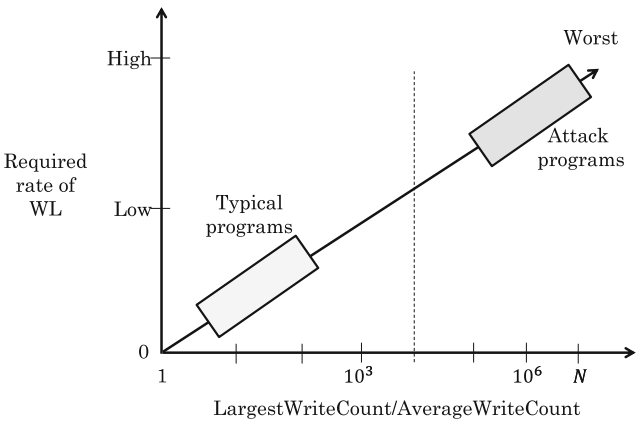

Fig. 5 Determining WL rate for normal and attack-like write streams [10]. *N* shows the number of lines in the memory. To succeed in a short time, an attack has to write to a few lines. Mitigating such attacks requires aggressive WL and vice versa

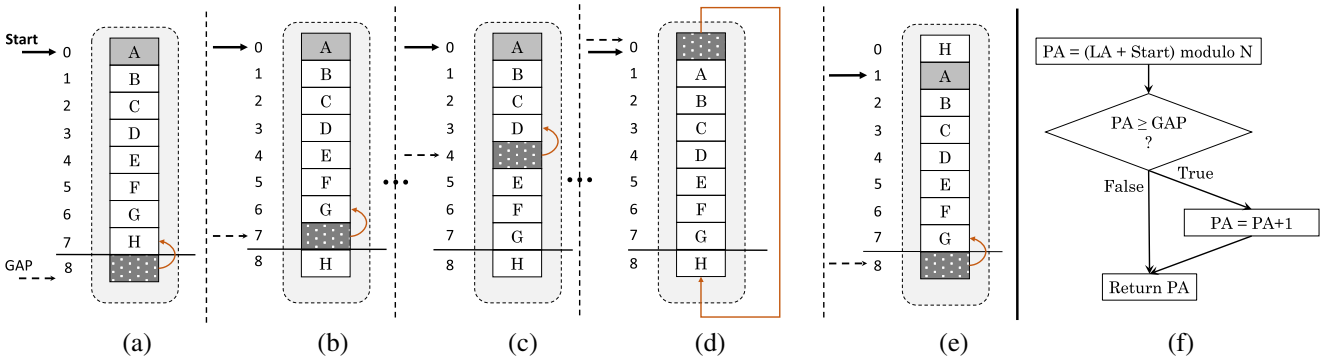

Fig. 6 **a**–**e** Start–Gap wear leveling on a memory containing eight lines [15] **f** Computation of PA from LA (N = number of memory lines)

chosen for eviction. Their technique estimates AD with reasonable accuracy and works well for different security-oriented WL schemes (e.g., [3, 15, 31]) and attack patterns. Also, it incurs small latency and storage overhead.

3.3 Algebraic WL Techniques

Qureshi et al. [15] present a WL technique which performs algebraic mapping between LAs and PAs and thus, avoids the need of write counters or mapping tables. Their technique works by periodically moving every line to its neighboring position. Their technique uses two registers (called start and gap) and a spare line. "Start" tracks the number of times all the lines in memory have been remapped and "Gap" records the number of relocated lines. In the beginning, "Start" points to location 0, whereas "Gap" always points to the position of spare line. After a fixed writes to memory, "Gap" is moved by one location by copying the data of [Gap-1] position to spare line and decreasing the value of "Gap" register. When "Gap" becomes zero, one remapping round is completed, and hence, "Start" is incremented by one and "Gap" points to the original location of spare line. Figure 6a–e shows one complete round of their WL scheme and Fig. 6f shows the computation of PA from the LA.

Although their technique improves NVM lifetime, its effectiveness is much lower than that of perfect WL. This is because their technique moves a line to its neighboring location only, and in typical workloads, hot lines occur close to each other. Hence, few frequently written lines limit the overall lifetime. To resolve this issue, they propose statically randomizing the logical address (LA) to intermediate address (IA) mapping and then applying their WL technique to obtain PAs. They experiment with two randomization strategies viz., Feistel network [9] and random invertible binary matrix. They show that combining start–gap technique with a randomization scheme increases the lifetime to close to that achievable by perfect WL. Also, the extra writes due to their WL technique are small. To tolerate repeated address attack, they extend their technique by dividing the memory into multiple regions. Each region uses separate "start" and "gap" registers and applies the WL technique independently. The size of region is chosen to ensure that a line shifts before it reaches its endurance limit. Their technique is effective in enhancing NVM memory security.

3.4 Mapping Table-Based WL Techniques

Seznec et al. [31] propose hiding PCM addresses from the adversary by remapping physical addresses and continuously changing this mapping in a random manner. They use a translation table which maintains one entry for each region with multiple contiguous memory blocks. This reduces the size of translation table compared to keeping one entry for each memory block. By also translating the displacement inside the region, a possible attack, which writes a specific block in all the regions, is avoided. The translation is performed as follows. Let R_{init} and D_{init} be random numbers generated at initialization time. Let T denote the translation

Table 4 Initial translation and their modification in the technique of Seznec et al. [31]

Initial translation of region address R	$T(R).address \oplus R \oplus R_{\text{init}}$
Initial translation of block D in region R	$(T(R).disp \oplus D \oplus D_{\text{init}})$
Modifying region address translation	$T(R).address \Leftarrow old(T(R').address) \oplus R' \oplus R$
	$T(R').address \Leftarrow old(T(R).address) \oplus R' \oplus R$
Modifying region displacement translation	$T(R).disp \Leftarrow old(T(R)).disp \oplus RAND$
	$T(R').disp \Leftarrow old(T(R')).disp \oplus RAND$

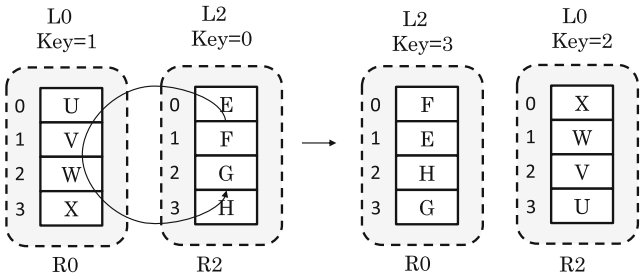

Fig. 7 An example of remapping in the technique of Seznec et al. [31]

table which is initialized with zero. Also, memory regions are numbered from 0 to $Q-1$. Then, the mapping of a region R in physical memory and displacement of a block D in region R are shown in the first two rows of Table 4. After a fixed number of writes, the physical-to-PCM address translation is updated by swapping the translation entries of two physical regions. For this, a random region R' is chosen and the "translation table" entries of R and R' are swapped as shown in the Table 4. RAND is a randomly generated number.

Figure 7 illustrates the working of the technique of Seznec et al. [31]. After a threshold number of writes to a region, all the lines of two physical regions ($R0$ and $R2$) are swapped in a single operation. Also, inside $R0$ and $R2$, the locations of lines are changed, e.g., lines U, V, W, and X in $R0$ are moved to the lines X, W, V, and U, respectively. They show that their technique can resist write attacks to PCM.

3.5 Hierarchical WL Techniques

In WL techniques, several parameters need to be carefully selected to achieve a fine balance between different optimization objectives. For instance, in region-level WL schemes (e.g., [15]), increasing the region size spreads the writes over a larger space. Further, if the remapping interval (measured in terms of write count) exceeds NVM WRE, an attacker can destroy an NVM block before the next remapping interval starts. However, use of short interval incurs high write overhead due to frequent remapping. Thus, with large region size, a shorter interval length is required to reduce performance penalty. To keep a balance between security and performance goals, some works use two-level region design where each region is divided in multiple sub-regions. Each sub-region performs internal remapping in an interval, and occasionally, remapping may be performed across different regions. Thus, each sub-region can perform frequent remapping due to smaller number of memory blocks inside every sub-region and use of two-levels increases the effective region size greatly. The limitation of the two-level remapping scheme is that for both reads and writes, a logical address needs to be translated twice to obtain the actual physical address. Also, it increases the complexity by virtue of requiring two remapping controllers.

Seong et al. [3] propose a technique to avoid data leakage from PCM by continuously changing the physical location of data. They use a "remapped memory address" (RMA) space for dissociating the memory address (MA) from its real data location. An access to an MA is redirected to the corresponding RMA. This mapping is dynamically changed for obfuscating the relationship between information leaked from side channels. Similar to protecting charge leakage in DRAM, this process protects data-leakage in PCM and hence, is referred to as "security refresh". They define a region (e.g., one bank) which has multiple memory blocks. A refresh operation happens after a fixed number of writes, called refresh interval. A refresh round refers to refreshing of all memory blocks in a region.

Figure 8 summarizes the working of their algorithm and it can be understood as follows. By XORing the MAs with a key, their corresponding RMAs are generated, e.g., if key = 4 and MA =0, then RMA = 4. On each security refresh, a "current refresh pointer" (CRP) points to the candidate MA to be refreshed. After every refresh, CRP is increased by one. When CRP reaches the first MA of a region, a new refresh round is started. Hence, a new key is randomly generated. One by one, the MAs pointed to by the CRP are refreshed and remapped, till all the MAs have been remapped which completes one refresh round. In a round, an MA is swapped only once. In any round, only the most recent two keys are stored and the previous keys are discarded.

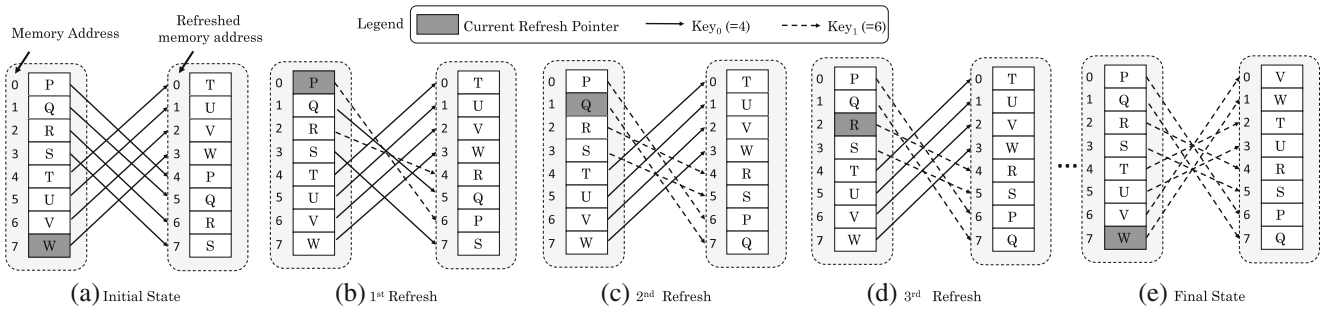

Fig. 8 An illustration of one full round of security refresh scheme [3] (K_0 and K_1 refer to the keys used in two rounds)

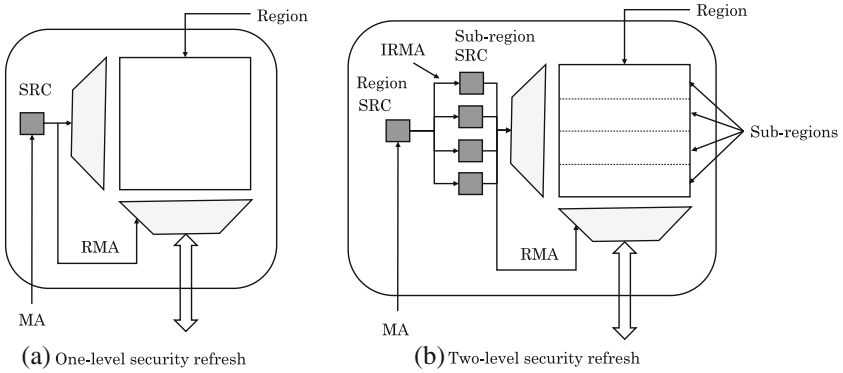

Fig. 9 A comparison of one-level and two-level security refresh in a bank (IRMA, intermediate remapped memory address; SRC, security refresh controller) [3]

To achieve a better tradeoff between performance and security, they further use a hierarchical two-level refresh scheme, where every region is divided into multiple sub-regions. Remapping is performed both within each sub-region and across different sub-regions. Figure 9 illustrates and compares the one- and two-level security refresh techniques. Even under worst-case attacks, their technique achieves large device lifetime with negligible performance penalty.

3.6 Mitigating Remapping Timing Attack

Huang et al. [14] present a WL scheme for protecting against RTA (refer to Section 2.2). Their technique works hierarchically as follows: at outer level, LA is mapped to IA using a dynamic multi-stage Feistel network [9]. They show that a static Feistel network, which uses fixed keys, is susceptible to RTA regardless of the number of stages. Hence, they use multiple randomly generated keys, which are changed in each remapping round, for avoiding address information leak via timing attack. The IA space is divided into sub-regions for averting repeated address attack and improving WL efficiency. At the inner level of each sub-region, Start–Gap WL [15] is used to transform IA into PA. In their technique, each bank is individually managed which avoids bank parallelism attack [3]. By changing the number of stages in Feistel network, a tradeoff can be achieved between security and overhead. They show the robustness of their against RTA, repeated address attack, and other types of write attacks.

3.7 Mitigating Row Buffer Hit Timing Attack

Mao et al. [16] propose a technique which hides PA information by changing the LA-to-PA mapping in a row. After the original swap of a WL scheme, another swap is immediately performed whereby two symmetric blocks are swapped in the row buffer. Two blocks whose PAs are i and $M - i - 1$ are considered symmetric, where M shows the number of blocks in the row ($M = 4$ in Fig. 10). Thus, their technique performs WL on both PA and LA.

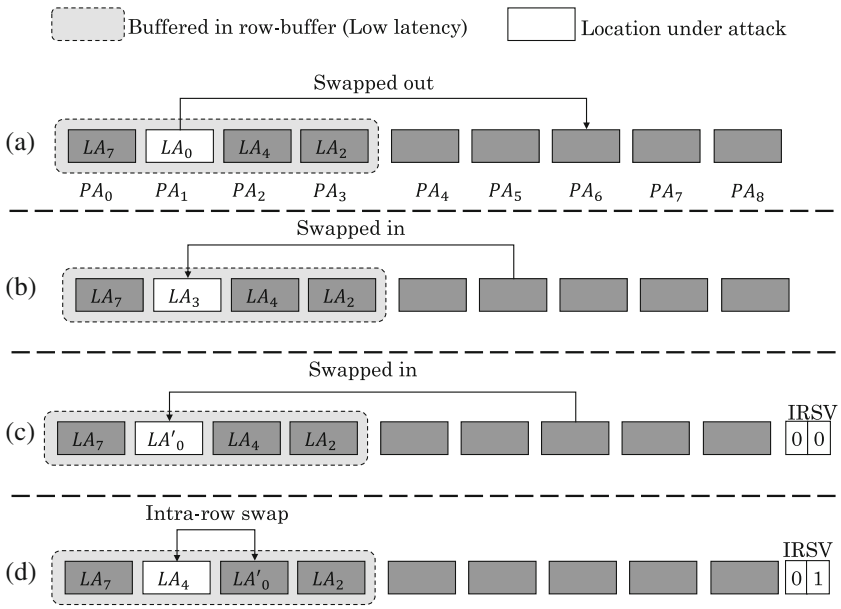

Fig. 10 **a**–**b** Row buffer hit timing attack. An adversary detects that the swapped out logical address (LA0) is no longer in the row b. Since LA3 is swapped in, adversary detects that it was swapped with LA0 and thus, it can repeatedly write LA3. **c**–**d**: technique of Mao et al. [16] (c) assumes that block at LA_0' is remapped into attacked row by the WL algorithm. **d** Right after this swap, an intra-row swap is performed between LA_0' and LA_4 which are at PA_1 and PA_2 locations, which are symmetric locations

Figure 10 (a–b) illustrate the row buffer hit timing attack, whereby the adversary can detect the LAs swapped in and out of the row and then choose certain LA to attack a specific PA location. Figure 10 (c–d) illustrate the technique of Mao et al. [16]. Figure 10 (c) shows the swap operation due to the underlying WL algorithm whereby the block at LA_0' is migrated to the attacked row. Then, as shown in Fig. 10 (d), right after the swap, an intra-row swap is performed between LA_0' and LA_4 which are at PA_1 and PA_2 locations, which are symmetric locations since 2 = 4-1-1. To record the intra-row swaps, an $M/2$-bit "intra-row swap vector" (IRSV) is used whose i^{th} bit shows whether the i^{th} block was swapped. Their technique insulates NVMM against row-buffer hit timing attacks, although it also incurs higher overhead due to two levels of wear-leveling operations.

3.8 PV-aware WL Techniques

Process variation (PV) leads to differences in the WRE of different NVM cells. In a PV-affected NVMM, conventional WL techniques may direct high amount of write traffic to cells/regions with low-WRE and hence, they may worsen the problem of WRE. To avoid this issue, some researchers propose PV-aware WL techniques.

Zhou et al. [30] propose a PV-aware WL technique which also ensures NVM security. They assume that the PCM memory has multiple domains. The endurance of lines in a domain are similar whereas that of lines in different domain may be different due to PV. Their WL technique logically partitions the memory into equi-sized regions. The region size is chosen to be smaller than the domain size while also keeping the total number of regions bounded. A region which receives a threshold (Z_r) number of writes is swapped with a randomly chosen region. The region with higher endurance also has higher probability of being selected for remapping. During remapping, the lines within a region are also swapped. Regions with higher endurance have higher value of Z_r and vice versa. They use a mapping table to record the mapping between LAs and PAs at the level of a region. Inside every region, algebraic remapping is used to ensure uniform write distribution. For example, a key and XOR operation is used for changing the address mapping, and the key used by a region is changed in every round. The algebraic mapping moves the lines one by one between two regions to reduce the impact of migrations on normal memory accesses. Their technique improves lifetime and security of a PV-affected NVMM.

Zhang et al. [28] note that PV-aware WL techniques have three phases: measurement, swap, and execution. In the measurement phase, hot/cold addresses are ascertained based on the write traffic. In the swap phase, hot/cold addreses are mapped to strong/weak cells, respectively and this mapping is used in the execution phase which is much longer. These WL techniques work on the assumption that the write-intensity distribution remains the same during measurement and execution phase. However, an adversary can easily cheat such a system. First, since a memory swap operation blocks all upcoming memory requests, it increases the memory access latency significantly; and by detecting this, the adversary can identify distinct phases of operation. Then, the adversary can reverse the distribution of write intensity in the execution phase compared to that in the measurement phase. Due to this, high write traffic can be directed to weak cells, which results in their quick failure.

They propose a technique to address this issue. Their technique works by pairing a strong page (say S) with a weak page (say W). If E_P denotes the endurance of a page P, for any write directed to page S or W, the write is performed on them with probability of $(E_S/(E_S+E_W))$ and $(E_W/(E_S+E_W))$, respectively, as shown in Fig. 11(a). This approach, termed "toss-up", does not require write traffic prediction. In case when the write is performed on a page different than that currently holding data, its original data will be lost. To avoid this, first, the original data is moved to the pair page and the incoming data is written to this page. This is shown in Fig. 11(b).

To reduce the number of swap operations, they propose two optimizations. First, the pages are sorted in order of their endurance and then the first page is paired with the last page and so on. Thus, pages with most distinct endurance values are paired. Second, "toss-up" is performed only after a fixed number of writes. Their technique improves PCM lifetime with minimal performance overhead.

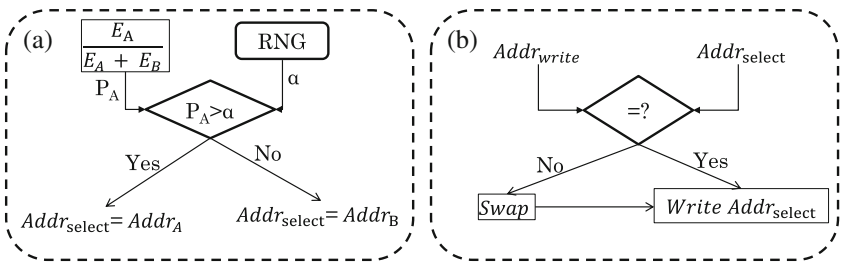

Fig. 11 Working of the technique of Zhang et al. [28]. **a** Deciding the physical page for writing the data (random number generator (RNG) generates a random number in the range [0,1]). **b** Deciding about the swap operation

3.9 Mitigating Repeated Set Attack in DRAM Cache to Protect NVMM

As discussed in Section 2.2, repeated set attack seeks to increase NVM traffic by increasing the number of writebacks from the DRAM cache. Thwarting this attack requires dissociating the address relationship between DRAM cache and PCM memory.

Wu et al. [11, 12] propose a technique to thwart repeated set attack. Their technique introduces an additional mapping layer between PA and the DRAM cache. It translates a PA into an IA which is used for cache addressing. However, on a cache miss, the original PA is used for PCM addressing. Thus, DRAM and PCM addressing are dissociated which renders the repeated set attack ineffective. The translation scheme between PA and IA should provide one-to-one mapping and distribute the writes to cache sets as evenly as possible. As such, their translation scheme swaps set-index bits with randomly-selected tag bits which may not be contiguous.

Figure 12 shows the circuit for swapping set bits with selected tag bits. The number of keys used are equal to the number of set bits and each key decides which bit of the tag will be swapped with a particular set bit. For example, in Fig. 12, the number of set bits are three, and hence, three keys are used. Thus, the association between DRAM and PCM is hidden from an adversary and the DRAM tries holding as many attacking writes as possible before issuing writebacks to PCM. They show that with careful design parameter choices, the number of writebacks due to a repeated set attack can be limited to no more than one entire cache flush.

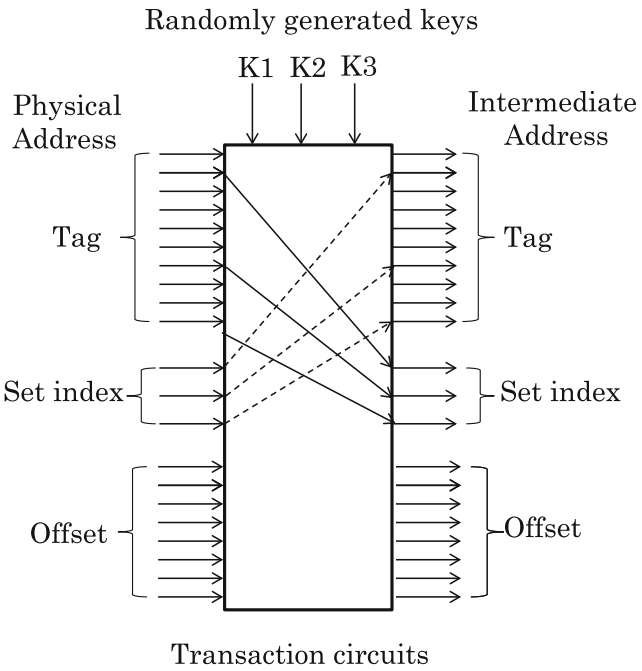

Fig. 12 The circuit for performing randomized cache address remapping using bitwise swap [11, 12]

They further propose changing the mapping function dynamically to hide it from the adversary. Since this also leads to flushing of DRAM, they partition both PCM and DRAM such that each partition is an independent set-associative design. The addresses in a PCM partition can be mapped only to the lines of its associated DRAM partition. When the number of DRAM cache flushes reach a threshold, it indicates possibility of an ongoing attack and hence, the mapping function between the partitions is changed. Their technique brings large reduction in DRAM cache miss rate and thus, improves PCM write traffic and lifetime.

4 Techniques for Reducing Encryption Overhead

Encryption incurs high overhead in NVMs due to their write agnostic nature and hence, intelligent techniques are required to lower encryption overhead in NVMs. In this section, we first discuss the tradeoffs (Section 4.1), and then discuss techniques which use hierarchical counters to avoid re-encryption (Section 4.2), extend the counter size to avoid counter overflow (Section 4.3), encrypt only cold page to lower the latency penalty of encryption (Section 4.4), cache the pad to avoid its recomputation (Section 4.5), and integrate encryption with compression (Section 4.6) and wear leveling (Section 4.7) to avoid encryption-induced write traffic.

4.1 Motivation and Challenges

Use of encryption in NVMs presents several challenges as we show below.

1. Latency overhead: Although encryption can be performed in the background, decryption happens on critical path since the plaintext is required to serve a read request. Due to this, only low-latency encryption techniques are suitable for NVMs.
2. Tradeoffs in counter size: In CME, a small counter leads to lower storage overhead and provides high hit rate of the counter cache. However, in case of high write traffic, a small counter can overflow soon which requires whole memory to be re-encrypted with a new key. This stalls the processor until the re-encryption is completed which incurs high overhead, due to the large capacity of NVMs and their high write latency/energy values. To avoid this issue, large-sized counter can be

used which reduces the frequency of re-encryption. This, however, leads to storage overhead.

3. Tradeoffs in granularity of encryption: On using an encryption counter for the entire cache line, the partial-write scheme becomes ineffective. This is because on each writeback, the counter is incremented which requires re-encryption of the entire cacheline. Hence, even if only one word is dirty, the entire cacheline needs to be written back and thus, the partial-write scheme cannot be applied. To resolve this issue, encryption can be performed at word level (128b) in a cache block (512b) [20] and at block level (512b) in a memory page (4096b) [35]. Using this, the advantage of redundant write avoidance using partial-write scheme can be still leveraged [19]. The limitation of this strategy is that it requires use of a separate encryption counter for each word.
4. Challenges in MLC memories: In MLC and tri-level cell (TLC) memories, a write operation requires precise control of the resistance value using multiple iterations of program-and-verify operations. However, due to this, programming to in-between states (10 and 01) consumes much higher (e.g., an order of magnitude) latency/energy than programming to terminal states (00 and 11) [2, 20]. Due to value locality, the plaintext is likely to contain long strings of zeros or ones [36]; however, since encryption increases data randomness, the ciphertext contains many 01 and 10 pairs. In other words, encryption increases the frequency of in-between states and thus, use of encryption increases the memory write energy significantly.
5. Limited scope: Encryption does not protect against write attacks and hence, if the adversary wants to merely wear out the memory but does not want to steal the data, encryption offers no protection. As such, some works combine encryption with WL [24–26] to provide more comprehensive protection at the cost of increased complexity.

Table 5 provides an overview of the encryption techniques reviewed in this survey.

4.2 Using Hierarchical Counters in CME

Several works use hierarchical encryption counters [19–21, 25] to retain the effectiveness of NVM write avoidance schemes and reduce the number of re-encryptions.

Kong et al. [19] propose a technique to reduce writes to an encrypted PCM-based main memory. They use indirect (i.e., pad-based) encryption approach to reduce the latency. They use both a global counter for the cache line and a local counter for each word [40]. Both the counters are combined for performing CME. On a writeback, the local counters of only dirty blocks are incremented and hence, only dirty blocks are re-encrypted. When a local counter overflows, all local counters of the cacheline are reset and global counter is incremented and thus, the whole cacheline is re-encrypted. For small word size, a change in even one word may trigger re-encryption of the whole cacheline whereas for large word-size, the effectiveness of partial-write scheme reduces.

They further discuss an adaptive error correction code (ECC) scheme which tracks the writes to NVM pages to detect its wear-out status and then, allocates ECC resources based on the wear-out status. They propose reusing encryption counters as write counters for guiding their adaptive ECC scheme. As for the interaction of ECC and encryption scheme, ECC can be computed from either plaintext or ciphertext. The former approach reduces write latency by allowing parallel operation of ECC computation

Table 5 A classification of encryption techniques

Classification	References
Reducing re-encryption overhead by	
Using hierarchical counters	[19–21, 25]
Avoiding re-encryption of	Unmodified [19–21] and zero words [20]
Extending counter size to reduce re-encryptions	[17, 29]
Performing encryption along with WL	[24, 25]
Complete or partial encryption	
Encryption of selected blocks/pages	[24, 26, 37]
Encryption of all blocks/pages	[19–21, 25, 29, 35, 38, 39]
Using hot/cold block information	
Encrypting only cold blocks	[37]
Caching the pad for a hot block	[22]
Performing more aggressive WL for hot blocks	[26]
Other features	
Interaction/integration of encryption with	ECC [19, 29], WL [19, 24–26]

Words modified during write	--	W1	W2	W3	--	W2, W4	W2	W4	--	W4	W7, W5, W6
Re-encrypted words	**ALL**	W1	W1, W2	W1, W2, W3	**ALL**	W2, W4	W2, W4	W2, W4	**ALL**	W4	W4, W7, W5, W6
Per-line counter	0	1	2	3	4	5	6	7	8	9	10
LCR	0	1	2	3	4	5	6	7	8	9	10
TCR	0	0	0	0	4	4	4	4	8	8	8

One interval

Fig. 13 Encryption operation in the technique of Young et al. [21]. The interval length is four and each line has eight words (W0 to W7)

and encryption. However, it increases the read latency since the ECC can be checked only after decryption. The latter approach increases write latency but lowers read latency. Of these, they use the latter approach since reads have higher criticality than the writes.

Young et al. [21] present a technique which avoids re-encryption of unmodified words during cache writeback. They use the latest value of the counter for encrypting the modified words and an older value of the counter for encrypting unmodified words. They use a parameter K such that the length of an interval is 2^K writes. Specifically, their technique uses two counters: a leading counter (LCR) which is the same as the line counter and a trailing counter (TCR) which is obtained by masking K least-significant bits of LCR. Thus, both LCR and TCR are virtual counters whose values are decided by the line counter. In an interval, LCR is incremented but TCR is not changed. For each word, a bit is kept to remember whether the word was modified in this interval. When LCR reaches TCR, the "modified" bits of all the words in a cacheline are reset. In the beginning of every interval, the values of TCR and LCR are equal and all words are re-encrypted as shown in Fig. 13. In an interval, any word that has been modified at least once from the beginning of that interval is re-encrypted using LCR, whereas unmodified words continue to use TCR. Compared to other techniques which use per-word counters/pads, their technique requires only two counter/pads for a line, although due to this, all words modified in an interval need to be re-encrypted whenever any word in that line is modified in that interval.

Figure 14 shows the decryption operation in their technique. They obtain two pads, one using the trailing counter (PAD-TCR) and one using the leading counter (PAD-LCR). For words whose "modified" bit is set or reset, the decryption is performed using PAD-LCR or PAD-TCR, respectively. Young et al. [21] further note that although their technique reduces bit writes by 2× compared to naive use of encryption, the improvement in lifetime is much smaller. This is because same words get encrypted repeatedly which leads to uneven distribution of writes to different parts of the line. To ensure uniform bit writes in a line, they integrate their technique with a WL scheme. This scheme works by periodically rotating the line similar to the idea of Start–Gap WL [15] and thus, achieves near-perfect WL. Overall, their combined technique increases the improvement in lifetime to 2×.

Swami et al. [20] note that in CME, during writeback of a cacheline to main memory, all the words are re-encrypted with a new OTP which incurs high overhead. They present techniques for avoiding re-encryption of unmodified and zero words and for reducing write patterns in ciphertext which have high programming energy in MLC/TLC resistive RAM (ReRAM). They use a local counter with each cache word which is incremented only when that word is modified during a writeback. Also, a global counter is used for the whole cache line. On a

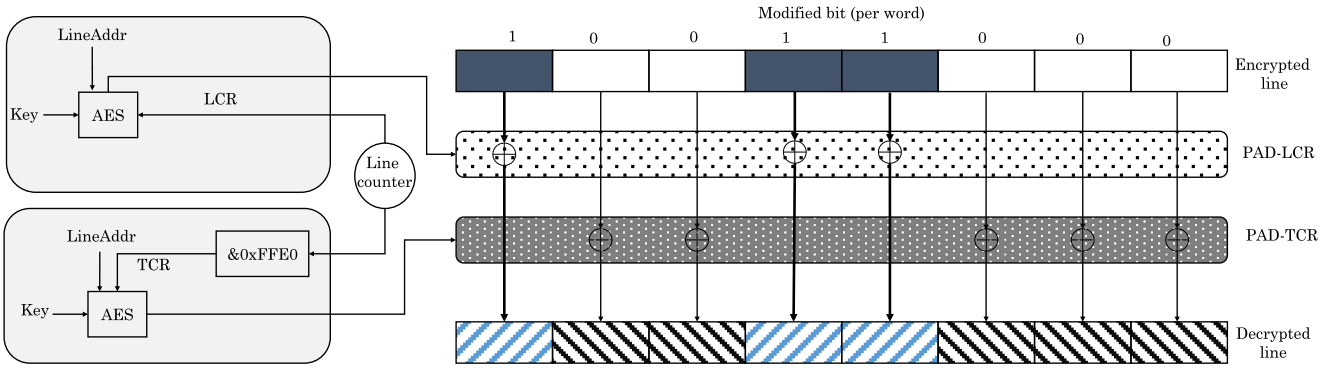

Fig. 14 Decryption operation in the technique of Young et al. [21]

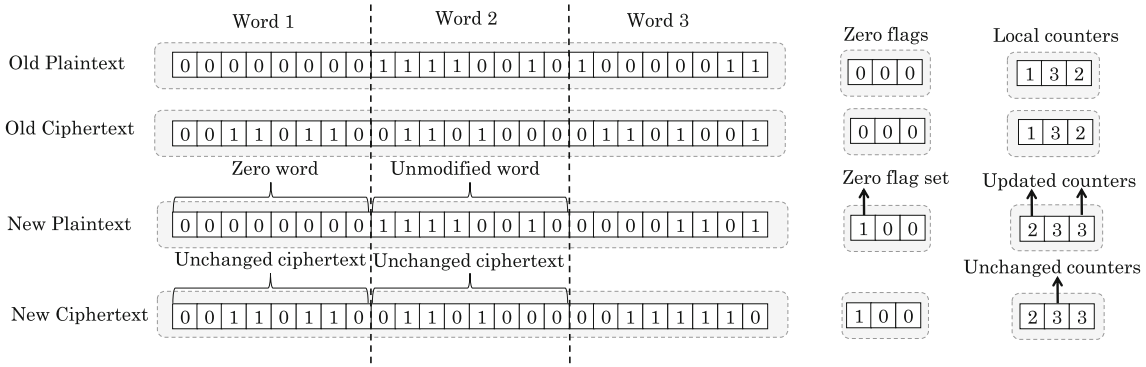

Fig. 15 Use of local counter and zero flag to avoid re-encryption of unmodified and zero words, respectively [20]. For word 1, the zero flag is set. The local counter for word 2 is not updated since the word was not modified. Hence, words 1 and 2 are not encrypted again

writeback, they perform word-level data comparison to find modified words. Then, every modified word is re-encrypted with an OTP obtained by using both the global counter and the updated local counter of the word. Unmodified words are not re-encrypted as shown in Fig. 15. Since most words remain unchanged during writeback, their technique reduces encryption overhead significantly. When local counter overflows, the OTP will be reused which can compromise security. To avoid this, on a local counter overflow, all the local counters are reset to 0, global counter is increased by 1, and the whole cache line is re-encrypted using the new global and local counters. They further note that a large fraction of plaintext data written to memory is zero. Hence, they use an all-zero flag which marks whether a word is all-zero and if so, the word is not re-encrypted, as illustrated in Fig. 15. The all-zero flag itself is written to memory in encrypted state.

They further note that encryption algorithms seek to make the correlation between the ciphertext and secret key very complex. Hence, while plaintext has several consecutive zeros/ones, ciphertext has many 01/10 pairs. However, writing 01/11 patterns to MLC NVM incurs much higher energy than 00/11 patterns and hence, writing ciphertext to memory consumes large amount of energy. To resolve this issue, in their technique, each word in the encrypted cache line is XORed with three masks in parallel: an all-zero mask, a mask with alternating 01 patterns (“0101...”) and a mask with half bits as 0, and remaining half

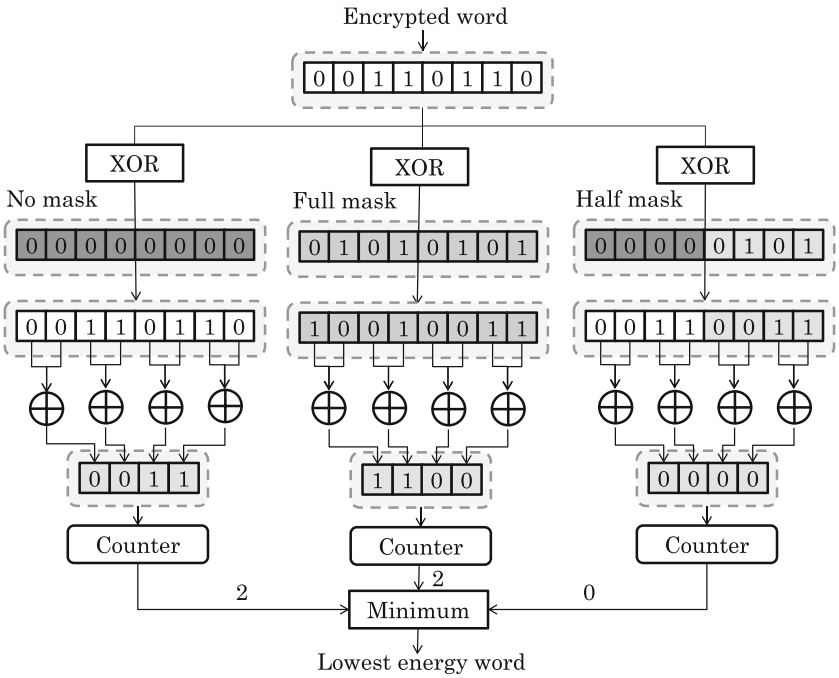

Fig. 16 Use of three masks while writing a word in an MLC/TLC ReRAM [20]. The mask which leads to the lowest number of intermediate states, i.e., least write energy, is selected

as 1 ("000..0011..111"). These masks are shown in Fig. 16. Then, from the three outputs, the one with lowest number of intermediate states is selected since writing this output will lead to lowest write energy, as illustrated in Fig. 16. The corresponding mask is recorded and used to get the original data during read operation. For both MLC and TLC ReRAM, their technique provides large reduction in write count and write energy.

4.3 Extending Counter Size to Reduce Re-encryption Frequency

Swami et al. [29] propose a technique which extends the size of encryption counters to reduce the frequency of counter overflow and subsequent re-encryption. They note that NVMs generally use error correcting pointer (ECP) for error-correction [41] and allocate ECPs uniformly to different cache lines; however, due to their varying error correction needs, ECPs remain unutilized. Hence, a large fraction of memory allocated for ECPs remains unused on a failed page and can be reused until it is required for error correction at the later stages of memory's life. On detecting a counter overflow on a memory write, their technique checks for availability of unused ECPs for the memory block where the cacheline is to be written. If so, one unused ECP is allocated to the overflowing counter which increases its size and helps in postponing the counter overflow by a large number of writes. When a cacheline with overflowing counter is mapped to an address with no left-over ECPs, full memory re-encryption is required.

Figure 17 illustrates the working of their technique. Initially, the "ECP-full" flag is empty (Fig. 17(a)), which indicates that an ECP is available. Their technique uses a 10-bit ECP to extend the counter-size from 16 bits to 26 bits, as shown in Fig. 17(b). To indicate that the counter has been extended, the "counter extension" flag is set and thus, the memory block now has only five ECPs. Their technique reduces re-encryption frequency and increases lifetime with acceptable overhead and no performance loss. It is noteworthy that Kong et al. [19] reuse encryption counters for ECC, whereas Swami et al. [29] reuse ECP counters for encryption.

4.4 Selectively Encrypting Only Cold Pages

Chhabra et al. [37] note that the working set of a program (pages frequently used by a program in a given interval) is much smaller than its resident set (all pages that belong to the program). They propose a technique which identifies the working set (i.e., hot pages) and encrypts the remaining pages in the resident set (i.e., cold pages). This allows keeping most of the memory encrypted at all times and leaves only a small fraction of memory that needs to be encrypted at system shutdown. The pages which are not accessed for a threshold period are considered cold since they are unlikely to be reused soon. Their technique encrypts different data at different times based on when it is predicted as hot/cold. Since cold pages are not frequently accessed, their encryption has minimal impact on overall performance.

Access to an encrypted page incurs decryption latency. On such a "misprediction", either the entire page or only the accessed block may be decrypted. The former approach avoids decryption overhead in future by proactively decrypting the entire page now whereas the latter approach avoids decryption overhead now but may incur it in future when other blocks in the page are accessed. To achieve a balance between these two approaches, an entire page may be decrypted after a fixed number of accesses to it. This ensures that the page has now become a part of the working set. Another strategy uses page address correlation to predict the page which will be accessed soon and then, decrypts the page proactively to overlap the decryption latency.

Figure 18 compares the level of security of different memories and encryption techniques. As shown in Fig. 18(a), on using DRAM memory with no encryption, the data lingers in DRAM for the duration of its retention period. After this time, the data in DRAM is no longer vulnerable to theft. On using NVM and performing encryption only on power-down (Fig. 18(b)), the vulnerability window increases greatly, since encrypting entire memory takes a large amount of time. By comparison, on using NVMM with incremental encryption (Fig. 18(c)), most part of memory remains in encrypted form. Hence, the vulnerability window is very small, approaching the retention period of

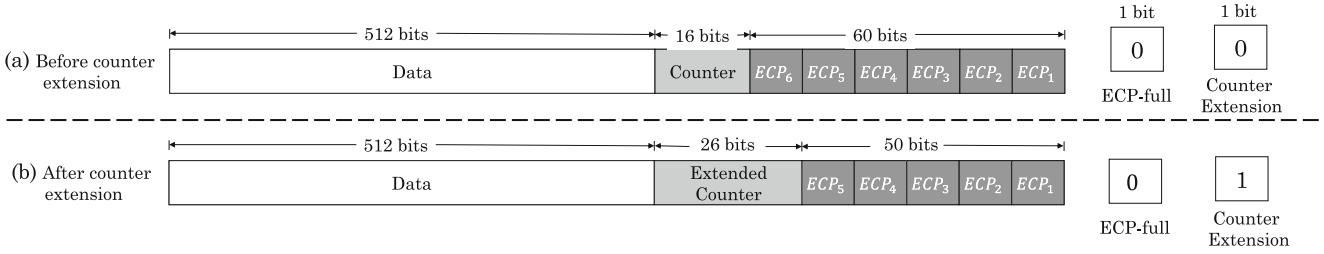

Fig. 17 Extending the counter size by using unused ECPs [29]

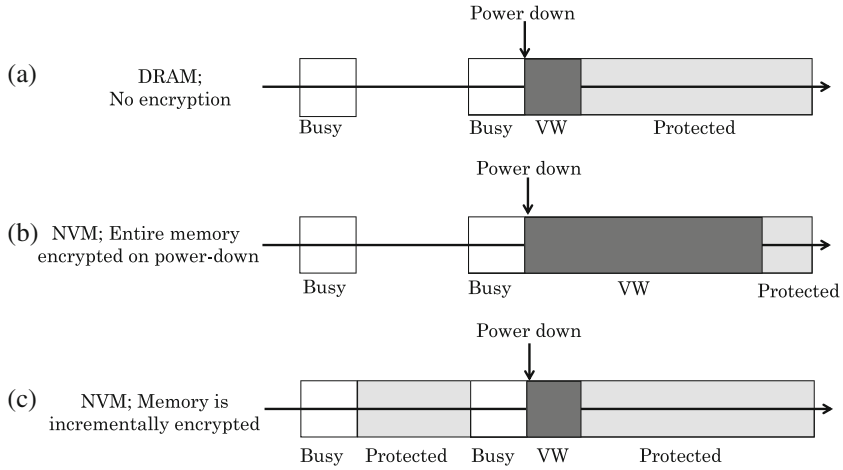

Fig. 18 A comparison of security strength in systems with **a** DRAM and no encryption **b** NVM with encryption of entire memory on power-down and **c** NVM with incremental encryption using the technique of [37] (VW, vulnerability window)

DRAM, and depends only on the working set size of the program.

Overall, their technique incurs lower latency overhead than a technique, which always keeps NVMM encrypted, and provides stronger security and smaller vulnerability period than a technique which encrypts whole NVMM at shutdown. The limitation of their technique is that it compromises NVM security since not all blocks are always kept encrypted, for example, since hot pages are not sent in encrypted form on the bus, their technique is vulnerable to bus snooping attack. Moreover, the motivating observation of their technique, namely, high decryption latency, holds true only for direct encryption techniques. By contrast, in indirect encryption techniques, the decryption latency is easily hidden, and hence, the entire memory can be kept encrypted without much performance penalty.

4.5 Caching the Pad to Avoid Recomputation

Luo et al. [22] propose an approach for improving the encryption/decryption performance which is especially useful in resource-constrained mobile systems. They first identify the hot data as that receiving high frequency of writes. Then, during encryption process, the pad generated for a hot data item is buffered in a DRAM buffer. On a read to last level cache (LLC), if the ciphertext corresponds to hot data, its pad is read from the buffer, which avoids recomputation. Otherwise, the pad is recomputed in parallel with fetching of data from NVMM which partially hides the decryption latency. Since a dynamic key ensures stronger security than a static key, the key for advanced encryption standard (AES) is generated using a random-number generator. Their technique decreases the energy consumption by reducing the need of pad recomputation.

4.6 Integrating Encryption with Compression

Jalili et al. [26] note that performing both compression and encryption achieves nearly the same level of security as encryption alone while reducing the bit-flip rate. Based on this, they propose using compression technique to offset the avalanche effect due to encryption. Their technique first performs compression, then encryption, and finally WL. For blocks with compressed size less than a threshold (λ), the entire block is encrypted. For blocks with compressed size higher than λ bytes, (1) a fixed fraction of randomly selected blocks are entirely encrypted and for remaining blocks, only λ bytes are encrypted. This is summarized in the left part of Fig. 19. λ is chosen to be equal to or larger than 16B which is the minimum size of AES block. Their selective-encryption technique encrypts higher fraction of blocks than the incremental encryption technique [37] (e.g., 99 vs. 87%). By controlling the avalanche effect, their technique allows exercising lifetime-security tradeoff. In summary, highly compressible blocks are fully encrypted and poorly compressible blocks are selectively encrypted in a non-deterministic manner to achieve acceptable level of security.

Jalili et al. [26] further note that always writing the compressed data from one end of the block leads to early failure of cells close to that end. To avoid this, they propose an intra-block WL technique which partitions a line into multiple (e.g., 4) segments. Then, based on the value of a rotating counter, the starting segment of the data is decided, as shown in Fig. 19. For blocks with higher write frequency (i.e., hot blocks), the starting segment rotates more quickly leading to uniform WL. Their technique reduces bit writes to NVMM and reduces performance penalty of encryption.

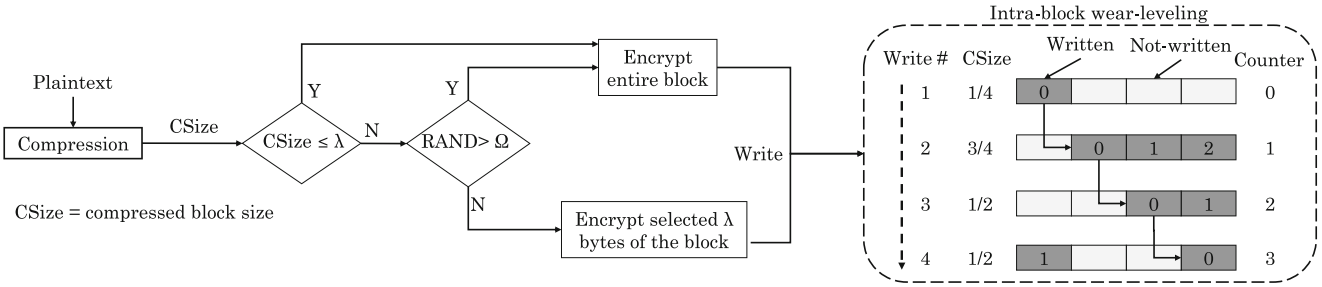

Fig. 19 Flow diagram of the technique of Jalili et al. [26]. λ and Ω are two thresholds, and RAND is a random number

4.7 Integrating Encryption with WL

We now review the works which propose to use encryption and WL in a synergistic manner to lower their overall overhead.

Huang et al. [25] propose a WL-aware CME technique for NVMs which uses remapping operation of WL to reset every line counter. This helps in reducing the size of line counter required for avoiding the counter overflow. The WL scheme periodically remaps all the memory lines in a region. For each line, they use a counter to track the writes to it after it was mapped the last time. This counter is incremented on each write to the line and is reset when the line is remapped by the WL algorithm. Further, for each region, a counter is used for recording the number of times the region (along with all of its lines) has been remapped. In Fig. 20, the per-line and per-region counters are shown as C_{line} and C_{region} respectively.

On each normal write to a line, its line counter is incremented. An overflow of a line counter resets all the line counters in the region, increments the region counter and triggers re-encryption of all the lines. When a line is to be remapped, it is first decrypted using old OTP which is produced from concatenation of C_{region}, C_{line}, and the line address of the remapped line (refer to Fig. 20). Then, the line is re-encrypted using a new OTP which is generated by using line address, reset (i.e., zero) line counter, and (region counter plus one). The new OTP is XORed with the plaintext and the new ciphertext of the line is written to its new physical location, as shown in Fig. 20. Further, if the remapped line was the last unremapped line in this remapping round, the C_{region} counter is incremented by 1.

In their technique, since the line counter is reset only during remapping, its size needs to be only large enough to record writes to most frequently written line of any region during a remapping period. Since remapping already leads to a write, re-encryption on resetting of the line counter does not lead to additional reads/writes. Their technique increases performance by reducing re-encryption-induced writes.

Liu et al. [24] propose improving security by both using both WL (i.e., changing the LA-to-PA mapping) and selective encryption. They perform WL at both page and block level. At both the levels, page/block are randomly chosen for swapping which provides another layer of security. A page not accessed for a fixed duration is considered cold and a page which gets more than a threshold number of writes is considered hot. For the page-level WL, their technique first encrypts the cold page and then swaps it with the hot page, as shown in Fig. 21. Since encryption is performed along with WL itself, extra write due to encryption is avoided. Write counters of swapped pages are reset. Block-level WL is performed using randomized Start–Gap WL algorithm [15], except that before being moved

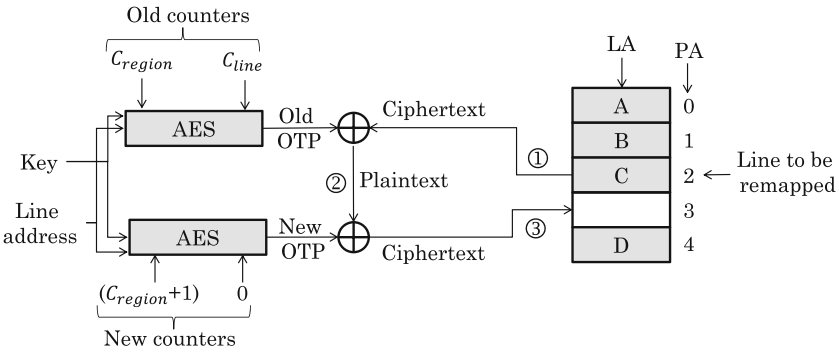

Fig. 20 Avoiding write overhead of encryption by performing it during WL [25]

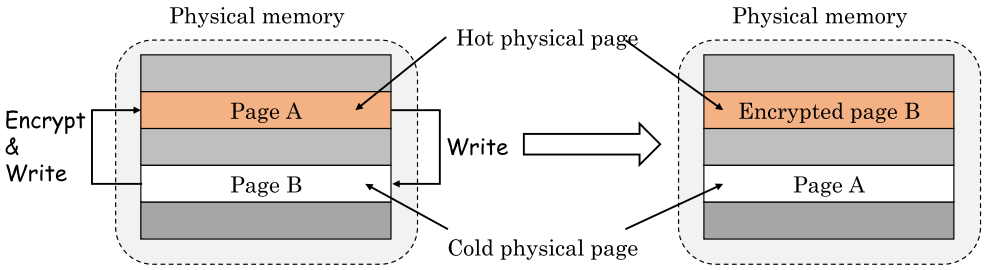

Fig. 21 Integration of WL and encryption in the technique of Liu et al. [24]. The cold page is encrypted and swapped with the hot page

to a gap location, an unencrypted block is first encrypted. Overall, to get the information about a data block, an adversary needs to find (1) the location of the block, (2) its encryption status (since a page has both encrypted and unencrypted blocks), and (3) encryption key. A block is always written as plaintext regardless of its prior encryption status, unlike in the technique of Chhabra et al. [37] where a data is written as ciphertext if the block was previously in encrypted state. Due to this, their technique requires smaller number of encryption/decryption operations compared to the technique of Chhabra et al. [37]. Liu et al. [24] also present variants of their technique that allow tradeoff between lifetime, security, and performance overhead.

5 Techniques for Reducing Data-Shredding Overhead

5.1 Motivation and Challenges

Data shredding is a strategy to destroy the contents of a physical page before allocating this page to another process. Data shredding avoids data leak between two processes or virtual machines. Generally, data shredding is achieved by writing zero on each cell of the page. It is frequently performed operation and hence, is responsible for a large fraction of writes to main memory.

Data shredding is generally achieved by writing zero on each cell of the page and hence, it contributes a large fraction of total memory writes and page fault-servicing latency. In case of server consolidation and virtualization where multiple processes or VMs share a system, a memory page may be shredded multiple times. For instance, hypervisor may perform shredding to avoid inter-virtual machine (VM) leaks and then, OS/kernel of the VM may perform shredding to avoid inter-process leaks. VMs and kernel programs may allocate large chunks of memory and data shredding for them incurs large overhead. Also, even if the entire allocated physical space is not used by the process, its shredding is still required. Evidently, data shredding, although important for avoiding data leak, is a costly operation. We now review techniques for reducing overhead of data shredding in NVM.

5.2 Using Encryption to Write Random Data for Data Shredding

Awad et al. [35] present a technique to avoid shredding-related writes in NVMM. They note that writing any random/unintelligible data to a page before its reuse has the same effect as zeroing it, since both ensure that no meaningful data can be read from the page. In an un-encrypted memory, writing random data provides no advantage over writing zero data; however, in an encrypted memory, changing the encryption key from say Key1 to Key2 ensures that decrypting the page leads to meaningless data. This allows initialization of a reused page with random data without any overhead. Due to the diffusion property of encryption, the new decrypted data has no correlation with the original data.

In their design, different processes use the same encryption key and due to this, modifying the key is infeasible. Instead, their technique modifies the initialization vector (IV) since in CME, it is IV which is encrypted to obtain an OTP. Then, data is encrypted and decrypted by performing XOR with the OTP. The IV is generated from a unique page ID, page offset (for distinguishing blocks of a page), a per-block local counter (for distinguishing multiple versions of the data), and a per-page global counter. As for modifying IV, the page ID and page offset are not good candidates for modification since they guarantee that the block's IV is spatially unique. As such, on initialization of a reused page, their technique increments its global counter and resets all the local counters to zero, which fulfills the purpose of modifying IV. This is shown in Fig. 22a.

The servicing of an LLC miss is done as shown in Fig. 22b. On the miss to a block, its block address is used to search the counter cache for reading its local counter. If the local counter is not zero, the block is fetched from NVMM by sending a request to the memory controller. The data obtained from NVMM is decrypted and returned to the LLC. However, if the local counter value is zero, a zero data is provided to LLC instead of providing random data. This ensures compatibility with software libraries, which expect pointer variables of a freshly allocated page to be zero. Since no access is made to memory in case when local counter value is zero, this approach reduces the page

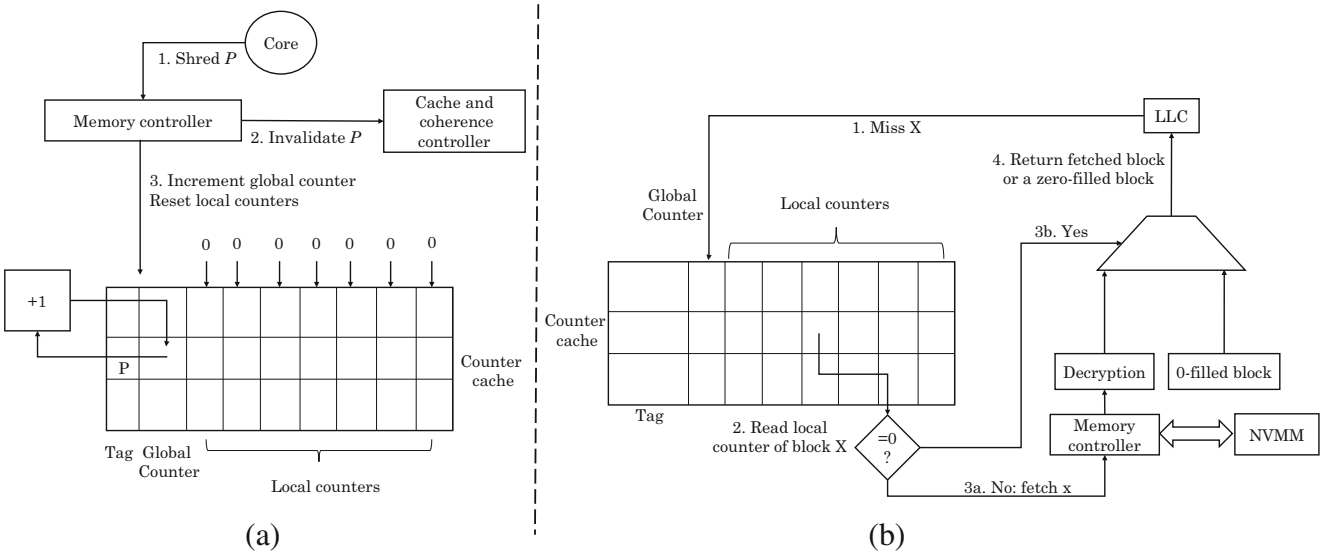

Fig. 22 **a** Shredding approach and **b** servicing of an LLC miss in the technique of Awad et al. [35]

re-encryption frequency and accesses to main memory. If the local counter overflows, it is reset to one and not zero. By avoiding shredding-induced writes and the subsequent reads from NVM, their technique improves performance and NVM lifetime significantly.

Haber et al. [17] claim that, on assuming that the attackers cannot observe the memory bus, the technique of Awad et al. [35] is less secure compared to the explicit zeroing scheme. Specifically, if an attacker gets the encryption key, he needs to generate only the IV to obtain the shredded data. Let us assume that the IV has 128 bits consisting of a cacheline address of 64 bits, a per-page global counter of 56 bits and a per-line local counter of 8 bits. Then, the adversary can obtain the global counter by decrementing the present global counter by 1. Then, the local counter can be guessed by trying all 2^8 combinations of 8b values. Thus, the adversary can get the data of the previous process.

They propose two techniques to thwart such attacks. The first technique uses a pseudo-random function (F), such as hash function and block cipher, for changing the manner in which the global counter of a to-be-shredded page is updated. Instead of merely increasing the global counter, its value J is replaced by $F(J)$. This makes it difficult to guess the previous value of the global counter since this would require computing $F^{-1}(J)$. The second technique reduces the cacheline address to 48 bits. The freed-up 16 bits are allocated to either the global counter or the local counter or are divided between them. Increasing the number of bits in local counter makes it difficult to test all the values of the local counter to launch an attack. Their two techniques can be used alone or together.

It is noteworthy that the claim of Haber et al. [17] holds only if the attacker cannot observe the memory bus. This is because, if the attacker can observe the memory bus, he can easily record the values before zeroing and in that case, explicit zeroing and the shredding by changing IV [35] provide similar security for shredding data in secure NVMs.

6 Encryption Techniques for Domain Wall Memory

The encryption techniques discussed in the previous section use a block cipher (e.g., AES) which achieves encryption by obfuscating the data *value* itself. Although the same technique can also be used in DWM, the unique characteristic of DWM enables achieving encryption also by obfuscating the *storage location* of the data value inside DWM. Table 6 compares these techniques. We now discuss a technique for encrypting data in DWM.

Table 6 Comparison of value-encryption and storage-location encryption [32] approaches

	Value-based encryption	Location-based encryption
Encryption	Use a block cipher (e.g., AES) to obfuscate the data *value* itself	Obfuscate the *storage location* of the data value inside DWM
Decryption	Decrypt the ciphertext	Perform shift operations to remove the randomness in storage location and re-align the stored data with access ports

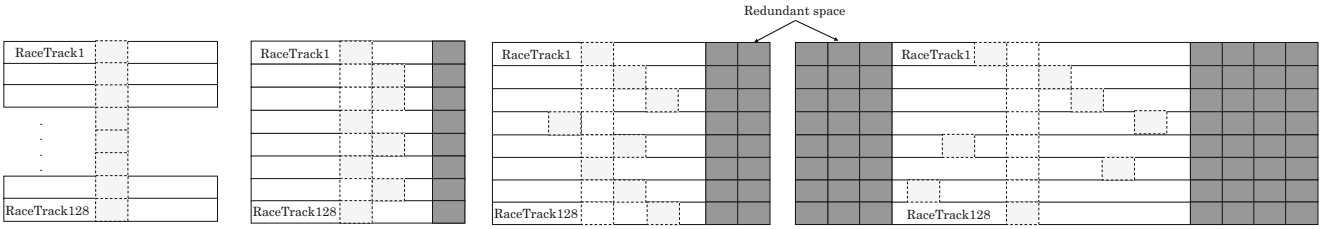

Fig. 23 **a** The baseline racetrack encryption using **b** 128-bit shift key (key width = 1), **c** 256-bit shift key (key width = 2), and **d** 384-bit shift key (key width = 3)

Zhang et al. [32] note that in DWM, data is stored in multiple racetrack tapes to allow parallel access to multiple bits of data, e.g., 128-bit data may be stored in 128 tapes [5]. In traditional DWM, all bits of a block are stored in the same position of each tape, whereas on applying variable shifts to these tapes, the bits can be hidden, similar to a "pin tumbler lock". Thus, they use the shift pattern as the "key" to perform data encryption. By using different key widths, different levels of security can be achieved. For example, on using a shift key of 256 bits and key width of 2, four shifts can be implemented: [–1, 0, 1, 2]. Similarly, on using a shift key of 384 bits and key width of 3, eight shifts can be implemented, ranging from –3 to 4.

On both sides of the racetrack, suitable amount of redundant space is maintained to save shifted-out bits, which is shown as the shaded space in Fig. 23. These redundant bits are initially filled with random data to prevent an adversary from inferring the shift key. A limitation of the shift-only encryption scheme is that certain data values (e.g., all 0 or all 1) can be easily decrypted since they remain the same despite shift operations. Hence, before performing the shift, a 4-stage Feistel network is used to change the data into random numbers. Overall, both encryption/decryption are performed with 4-stage Feistel network and shift operations using shift key. Since shift operation in different tapes can be concurrently performed in few cycles, the encryption process is completed with low latency and energy, while achieving higher security than the AES-128 (i.e., AES with 128-bit key) scheme.

7 Conclusion and Future Outlook

In this paper, we presented a survey of techniques for improving security of non-volatile memories. We classify the works based on key parameters to highlight their similarities and differences. We believe that the techniques reviewed here will also be useful for current and future lifetime-limited or write-agnostic memories such as Flash memory [42]. We conclude this paper with a brief mention of future research challenges.

The goal of security are often at odds with the goals of performance, energy/cost-efficiency, and user comfort [43]. For example, encryption and WL techniques incur large write overhead in NVMs which are already write-agnostic memories. Going forward, the researchers need to design holistic solutions which optimize performance and energy efficiency without compromising the security. For example, for error-tolerant applications, NVM writes can be performed in approximate manner [44] to mitigate write attacks and also lower the write overhead of encryption techniques.

Recent research has shown that at small feature sizes, PCM suffers from write disturbance errors (WDEs), whereby resetting a memory cell can disturb its nearby idle cell storing the value "0" [45]. This phenomenon can be exploited by an attacker to repeatedly write a cell to disturb neighboring cells and even gain kernel privilege similar to the case of row-hammer issue in DRAM [46]. The techniques for mitigating repeated write attacks such as WL aim to diffuse the writes targeted to a single cell. However, through WDEs, a particular cell can be attacked by writing to many of its neighbors. Also, crossing of the WRE limit due to repeated writes leads to hard errors (i.e, failure of the memory cell) in NVMs, whereas WDE leads to soft errors (i.e., flipping of the bit) [47]. Since flipping of the stored value is likely to get unnoticed more easily than the failure of the memory, WDE-induced attacks are more difficult to detect. Finally, a successful repeated write attack requires many writes to cross the WRE limit, whereas by exploiting WRE, a nearby value can be flipped even in few writes. Evidently, WDE-based attacks are more difficult to detect and thwart than WRE-based attacks. Thwarting these attacks will be crucial for ensuring complete security of next-generation memory systems.

We believe that addressing the challenges of NVM security requires synergistic solutions at all layers of abstractions. At the circuit level, design of PCM prototypes with high endurance can help in tolerating the write attacks. For example, a recent paper has shown an "ALD-based confined PCM cell" design [48] which shows endurance value of 2×10^{12}. This is much higher compared to the

$10^6 - 10^9$ value achieved by conventional commercial PCM designs. However, their proposed design also has much higher SET latency (e.g., 80ns compared to 8ns in conventional designs).

At microarchitectural level, low-overhead WL [49] and encryption solutions can help in increasing robustness to write attacks and stolen memory attacks. At system level, effective management of DRAM-PCM hybrid memory can provide better security and performance than a PCM-only memory. At application and OS level, runtime monitoring of applications to identify attacking programs and side channels is imperative. Finally, promoting awareness about importance of security to the end users will go a long way in moving towards the "security-first" design paradigm.

Since even a small "hole" (or security vulnerability) in the security can allow an adversary to gradually take complete control of the system, the goal of securing the system demands the designer to be always on vigil. This, in turn, includes proactively detecting any security vulnerability and finding techniques for thwarting any possible attacks. Clearly, the goal of ensuring NVM security is likely to keep the researchers fully challenged for more time to come.

Funding Information Support for this work was provided by the Science and Engineering Research Board (SERB), India, award number ECR/2017/000622.

References

1. Mittal S, Vetter JS, Li D (2015) A survey of architectural approaches for managing embedded DRAM and non-volatile on-chip caches. In: IEEE Transactions on parallel and distributed systems (TPDS)
2. Mittal S, Wang R, Vetter J (2017) DESTINY: a comprehensive tool with 3D and multi-level cell memory modeling capability. In: JLPEA
3. Seong NH, Woo DH, Lee H-HS (2010) Security refresh: prevent malicious wear-out and increase durability for phase-change memory with dynamically randomized address mapping. SIGARCH Comput Archit News 38(3):383–394
4. Chhabra S, Solihin Y (2010) Defining anomalous behavior for phase change memory. In: Workshop on the use of emerging storage and memory technologies, held in conjunction with HPCA
5. Mittal S (2016) A survey of techniques for architecting processor components using domain wall memory. ACM J Emerg Technol Comput Syst
6. Mittal S (2017) A survey of soft-error mitigation techniques for non-volatile memories. Computers 6:8
7. Mittal S (2016) A survey of architectural techniques for managing process variation. ACM Comput Surv 48(4):54,1–54,29
8. Mittal S, Vetter JS (2016) A survey of software techniques for using non-volatile memories for storage and main memory systems. IEEE Trans Parallel Distrib Syst 27(5):1537–1550
9. Menezes AJ, Van Oorschot PC, Vanstone SA (1996) Handbook of applied cryptography. CRC Press
10. Qureshi MK, Seznec A, Lastras LA, Franceschini MM (2011) Practical and secure pcm systems by online detection of malicious write streams. In: 2011 IEEE 17th International symposium on high performance computer architecture, pp 478–489
11. Wu G, Zhang H, Dong Y, Hu J (2012) CAR: securing PCM main memory system with cache address remapping. In: International conference on parallel and distributed systems, pp 628–635
12. Wu G, Gao J, Zhang H, Dong Y (2011) Improving pcm endurance with randomized address remapping in hybrid memory system. In: International conference on cluster computing, pp 503–507
13. Mittal S (2016) A survey of power management techniques for phase change memory. Int J Comput Aided Eng Technol (IJCAET) 8(4):424–444
14. Huang F, Feng D, Xia W, Zhou W, Zhang Y, Fu M, Jiang C, Zhou Y (2016) Security RBSG: protecting phase change memory with security-level adjustable dynamic mapping. In: International parallel and distributed processing symposium (IPDPS), pp 1081–1090
15. Qureshi MK, Karidis J, Franceschini M, Srinivasan V, Lastras L, Abali B (2009) Enhancing lifetime and security of pcm-based main memory with start-gap wear leveling. In: 2009 42nd Annual IEEE/ACM international symposium on microarchitecture (MICRO), pp 14–23
16. Mao H, Zhang X, Sun G, Shu J (2017) Protect non-volatile memory from wear-out attack based on timing difference of row buffer hit/miss. In: Design, automation test in Europe conference exhibition (DATE), pp 1623–1626
17. Haber S, Manadhata PK (2017) Improved security for non-volatile main memory
18. Chhabra S, Rogers B, Solihin Y, Prvulovic M (2009) Making secure processors os-and performance-friendly. ACM Trans Architect Code Optim (TACO) 5(4):16
19. Kong J, Zhou H (2010) Improving privacy and lifetime of pcm-based main memory. In: 2010 IEEE/IFIP International conference on dependable systems networks (DSN), pp 333–342
20. Swami S, Rakshit J, Mohanram K (2016) Secret: smartly encrypted energy efficient non-volatile memories. In: 2016 53nd ACM/EDAC/IEEE design automation conference (DAC), pp 1–6
21. Young V, Nair PJ, Qureshi MK (2015) Deuce: write-efficient encryption for non-volatile memories. SIGARCH Comput Archit News 43(1):33–44
22. Luo X, Liu D, Liangy L, Li Y, Zhong K, Long L (2015) Mobilock: an energy-aware encryption mechanism for nvram-based mobile devices. In: 2015 IEEE Non-volatile memory system and applications symposium (NVMSA), pp 1–6
23. Zhang X, Zhang C, Sun G, Di J, Zhang T (2013) An efficient run-time encryption scheme for non-volatile main memory. In: 2013 International conference on compilers, architecture and synthesis for embedded systems (CASES), pp 1–10
24. Liu C, Yang C (2015) Secure and durable (sedura): an integrated encryption and wear-leveling framework for pcm-based main memory. In: ACM SIGPLAN Notices, vol 50, no. 5.1. ACM, p 12
25. Huang F, Feng D, Hua Y, Zhou W (2017) A wear-leveling-aware counter mode for data encryption in non-volatile memories. In: Design, automation test in Europe conference exhibition (DATE), pp 910–913
26. Jalili M, Sarbazi-Azad H (2017) Endurance-aware security enhancement in non-volatile memories using compression and selective encryption. IEEE Trans Comput 66(7):1132–1144
27. Yu H, Du Y (2014) Increasing endurance and security of phase-change memory with multi-way wear-leveling. IEEE Trans Comput 63(5):1157–1168
28. Zhang X, Sun G (2017) Toss-up wear leveling: protecting phase-change memories from inconsistent write patterns. In: Design automation conference, pp 3:1–3:6
29. Swami S, Mohanram K (2017) Covert: counter overflow reduction for efficient encryption of non-volatlle memories. In: Design,

automation test in Europe conference exhibition (DATE), pp 906–909
30. Zhou W, Feng D, Hua Y, Liu J, Huang F, Zuo P (2016) Increasing lifetime and security of phase-change memory with endurance variation. In: 2016 IEEE 22nd International conference on parallel and distributed systems (ICPADS), pp 861–868
31. Seznec A (2010) A phase change memory as a secure main memory. IEEE Comput Archit Lett 9(1):5–8
32. Zhang H, Zhang C, Zhang X, Sun G, Shu J (2016) Pin tumbler lock: a shift based encryption mechanism for racetrack memory. In: Asia and South Pacific design automation conference (ASP-DAC), pp 354–359
33. Wang Y, Ni L, Chang C-H, Yu H (2016) DW-AES: a domain-wall nanowire-based AES for high throughput and energy-efficient data encryption in non-volatile memory. IEEE Trans Inf Forensics Secur 11(11):2426–2440
34. Mittal S, Vetter JS (2014) EqualChance: addressing intra-set write variation to increase lifetime of non-volatile caches. In: 2nd USENIX workshop on interactions of NVM/flash with operating systems and Workloads (INFLOW)
35. Awad A, Manadhata P, Haber S, Solihin Y, Horne W (2016) Silent shredder: zero-cost shredding for secure non-volatile main memory controllers. SIGOPS Oper Syst Rev 50(2):263–276
36. Mittal S (2017) A survey of value prediction techniques for leveraging value locality. Concurrency and computation: practice and experience
37. Chhabra S, Solihin Y (2011) i-NVMM: a secure non-volatile main memory system with incremental encryption. In: 2011 38th Annual international symposium on computer architecture (ISCA), pp 177–188
38. Enck W, Butler K, Richardson T, McDaniel P, Smith A (2008) Defending against attacks on main memory persistence. In: Computer security applications conference, 2008. ACSAC 2008. IEEE, pp 65–74
39. Hou F, He H (2015) Ultra simple way to encrypt non-volatile main memory. Secur Commun Netw 8(7):1155–1168
40. Yan C, Englender D, Prvulovic M, Rogers B, Solihin Y (2006) Improving cost, performance, and security of memory encryption and authentication. ACM SIGARCH Comput Architect News 34(2):179–190
41. Schechter S, Loh GH, Straus K, Burger D (2010) Use ECP, not ECC, for hard failures in resistive memories. In: International symposium on computer architecture (ISCA), pp 141–152
42. Alsalibi AI, Mittal S, Al-betar MA, Sumari PB (2018) A survey of techniques for architecting SLC/MLC/TLC hybrid flash memory based SSDs concurrency and computation practice and experience
43. Henson M, Taylor S (2014) Memory encryption: a survey of existing techniques. ACM Comput Surv (CSUR) 46(4):53
44. Mittal S (2016) A survey of techniques for approximate computing. ACM Comput Surv 48(4):62,1–62,33
45. Wang R, Mittal S, Zhang Y, Yang J (2017) Decongest: accelerating super-dense PCM under write disturbance by hot page remapping. IEEE Comput Architect Lett
46. Seaborn M Exploiting the dram rowhammer bug to gain kernel privileges. https://goo.gl/NMK6MM
47. Mittal S, Vetter J (2015) A survey of techniques for modeling and improving reliability of computing systems. IEEE Trans Parallel Distrib Syst
48. Kim W, BrightSky M, Masuda T, Sosa N, Kim S, Bruce R, Carta F, Fraczak G, Cheng H, Ray A et al (2016) ALD-based confined PCM with a metallic liner toward unlimited endurance. In: IEEE International electron devices meeting (IEDM), pp 4–2
49. Mittal S, Vetter JS (2016) EqualWrites: reducing intra-set write variations for enhancing lifetime of non-volatile caches. IEEE Trans VLSI Syst 24(1):103–114