Frontiers of Information Technology & Electronic Engineering
www.jzus.zju.edu.cn; engineering.cae.cn; www.springerlink.com
ISSN 2095-9184 (print); ISSN 2095-9230 (online)
E-mail: jzus@zju.edu.cn

Enhancing security of NVM-based main memory with dynamic Feistel network mapping*#

Fang-ting HUANG[1,2,3], Dan FENG[†‡1,2,3], Wen XIA[1,2,3], Wen ZHOU[1,2,3], Yu-cheng ZHANG[1,2,3], Min FU[1,2,3], Chun-tao JIANG[3], Yu-kun ZHOU[1,2,3]

[1]*Wuhan National Laboratory for Optoelectronics, Wuhan 430074, China*

[2]*MOE Key Laboratory of Information Storage System, Wuhan 430074, China*

[3]*School of Computer Science and Technology, Huazhong University of Science and Technology, Wuhan 430074, China*

[†]E-mail: dfeng@hust.edu.cn

Received Nov. 23, 2016; Revision accepted Mar. 23, 2017; Crosschecked July 8, 2018

Abstract: As promising alternatives in building future main memory systems, emerging non-volatile memory (NVM) technologies can increase memory capacity in a cost-effective and power-efficient way. However, NVM is facing security threats due to its limited write endurance: a malicious adversary can wear out the cells and cause the NVM system to fail quickly. To address this issue, several wear-leveling schemes have been proposed to evenly distribute write traffic in a security-aware manner. In this study, we present a new type of timing attack, remapping timing attack (RTA), based on information leakage from the remapping latency difference in NVM. Our analysis and experimental results show that RTA can cause three of the latest wear-leveling schemes (i.e., region-based start-gap, security refresh, and multi-way wear leveling) to lose their effectiveness in several days (even minutes), causing failure of NVM. To defend against such an attack, we further propose a novel wear-leveling scheme called the 'security region-based start-gap (security RBSG)', which is a two-stage strategy using a dynamic Feistel network to enhance the simple start-gap wear leveling with level-adjustable security assurance. The theoretical analysis and evaluation results show that the proposed security RBSG not only performs well when facing traditional malicious attacks, but also better defends against RTA.

Key words: Non-volatile memory (NVM); Endurance; Wear leveling; Timing attack
https://doi.org/10.1631/FITEE.1601652 **CLC number:** TP309; TP333

1 Introduction

Emerging non-volatile memory (NVM) technologies, such as phase change memory (PCM), spin-transfer torque RAM (STT-RAM), and resistive RAM (ReRAM), have become promising candidates for building the future main memory systems due to their advantages of high density, good scalability, and low power leakage (Mittal and Vetter, 2016). However, NVMs commonly have limited write endurance. For example, each PCM cell is projected to endure a maximum of about 10^7 to 10^8 writes (Freitas and Wilcke, 2008). This may be sufficient for a typical memory system. However, because of

‡ Corresponding author

* Project supported by the National High-Tech R & D Program (863) of China (Nos. 2015AA015301 and 2015AA016701), the National Natural Science Foundation of China (Nos. 61303046, 61472153, 61502190, and 61232004), the State Key Laboratory of Computer Architecture (No. CARCH201505), the Wuhan Applied Basic Research Project, China (No. 2015010101010004), and the Engineering Research Center of Data Storage Systems and Technology, Ministry of Education, China

A preliminary version was presented at the IEEE International Parallel and Distributed Processing Symposium, Chicago, USA, May 23–27, 2016

ORCID: Fang-ting HUANG, http://orcid.org/0000-0001-7887-8735

the non-uniform write traffic in real-world applications, some memory lines are written frequently, and fail much faster than others, causing the whole system to fail much earlier than its expected lifetime (Qureshi et al., 2009). To address this issue, 'wear-leveling' schemes are introduced to make the writes distribute evenly by remapping the frequently written lines to the seldom written lines.

In addition to the uneven writes issue coming from general applications, another severe problem should be tackled in NVM is malicious adversaries who might continuously write to a few physical lines and cause the whole NVM system to fail within a very short period of time. To deal with this security threat, many wear-leveling schemes have been researched. Region-based start-gap (RBSG) prevents a repeated address attack (RAA) by dividing the memory into several independent regions (Qureshi et al., 2009). Seznec (2009) proposed a hierarchical table-based secure wear-leveling scheme to resist a birthday paradox attack (BPA). Moreover, most of the lifetime-extending methods fail to prevent deliberately crafted malicious attacks and security refresh (SR) is proposed to further enhance security by dynamically remapping the logical address to the physical address using randomly generated keys (Woo et al., 2010). To achieve a longer lifetime, multi-way wear leveling (MWWL) leverages a table to track the mapping between logical and physical regions to accelerate the remapping of frequently written regions, and swaps regions by adopting security wear-leveling schemes such as start-gap and SR (Yu and Du, 2014).

In this study, we present a new type of timing attack against NVM wear-leveling schemes, the remapping timing attack (RTA). Note that wear-leveling schemes need to remap a logical address to a new location periodically and thus incur extra latency (called 'remapping latency'). The idea behind RTA is based on the observation that the remapping latency differs with different remapped data. On one hand, the latency incurred by remapping a line with data having all bits as '1' is much higher than that by remapping a line with data having all bits as '0' (Qureshi et al., 2012; Bishnoi et al., 2014; Mittal et al., 2015). On the other hand, this remapping difference is interfered by redundant write reduction technologies, which avoid writing unmodified bits to extend the NVM lifetime (Yang et al., 2007; Zhou et al., 2009). Therefore, it is possible to perform a carefully designed sequence of writes and infer specific mapping transformation of the wear-leveling schemes. To better illustrate this, we analyze the efficiency of RTA against three state-of-the-art wear-leveling schemes, RBSG, SR, and MWWL, which stand for algebraic mapping-based wear-leveling schemes with static keys, algebraic mapping-based wear-leveling schemes with dynamic keys, and hybrid wear-leveling schemes, respectively. Compared with RAA and BPA, the proposed RTA is able to make an NVM fail in much less time because it precisely targets a specified line or region.

To protect NVM against an RTA, we further propose a novel wear-leveling scheme called 'security region-based start-gap (security RBSG)'. Security RBSG employs a two-level strategy: (1) The outer-level wear leveling transforms the logical address (LA) to the intermediate address (IA) with a multi-stage Feistel network whose keys change dynamically to avoid address information leakage through a timing attack, ensuring security with adjustable levels (Menezes et al., 1996). Note that, by adding the number of stages in the dynamic Feistel network, the security is enhanced because the keys will change before the completion of an RTA key-detection procedure. (2) The inner-level wear leveling, after dividing the IA space into equally sized sub-regions, transforms the IA into the physical address (PA) by the simple start-gap scheme in each sub-region, which ensures that the write traffic is uniform with low overhead. Therefore, the proposed security RBSG provides NVM with both enhanced lifetime and security.

The contributions of this study are twofold:

1. We present a new type of timing attack, RTA, and show its efficiency in wearing out NVM devices that employ the latest wear-leveling approaches. To the best of our knowledge, this is the first study that reveals this kind of attack on NVM, which leverages the remapping latency difference of NVM. The experimental results show that RTA can make a PCM fail which adopts RBSG, SR, and MWWL (all with the default recommended configurations) 27 435, 322, and 127 719 times faster than an RAA, respectively. This increase allows an RTA to succeed within a reasonably short time (i.e., in several days or even minutes).

2. To resist RTA, we propose a security-level ad-

justable wear-leveling scheme called security RBSG that makes an efficient trade-off between security assurance and overhead. The theoretical analyses demonstrate that the proposed scheme is robust against an RTA. In addition, the lifetime and the performance impact of security RBSG are analyzed in the simulations.

Compared with the conference version (Huang et al., 2016), this study (1) extends the application scope of the RTA attack model and the security RBSG wear-leveling scheme to NVM devices including PCM, STTRAM, and ReRAM, (2) discusses another new source of the remapping latency difference that comes from the redundant write reduction technologies, which is crucial to the success of the RTA model, (3) adds a demonstration of the vulnerability of a fundamental hybrid scheme (i.e., MWWL) via an RTA in Section 4.4, which further proves not only the generality of RTA, but also the necessity of the proposed security RBSG, and (4) adds substantial experimental results and analysis in Section 6.

2 Background

2.1 Limited write endurance of NVMs

NVMs suffer from a limited write endurance problem because of their physical properties. PCM devices can endure only about 10^7 to 10^8 writes per cell (Freitas and Wilcke, 2008). For ReRAM, the write endurance bar is much improved, yet endures only 10^{10} to 10^{11} (Kim et al., 2011). For STT-RAM, the best endurance tested so far is less than 4×10^{12} writes (Huai, 2008). Once a cell has been written over its write endurance, it suffers from a 'stuck-at' hard fault. Moreover, the non-uniform memory write pattern makes the situation worse because the NVM lifetime is determined by a few of the most frequently written cells, which can dramatically reduce the lifetime of a large array of memory cells. Furthermore, malicious attackers can generate write streams that focus on a few physical lines and cause an NVM device to fail much faster than its expected lifetime.

2.2 Wear-leveling schemes

Wear leveling is an efficient way to extend the lifetime of NVM devices by distributing writes evenly throughout the entire memory space. Generally, wear-leveling schemes use a translation layer to map the accessed logical addresses to the physical locations where data are actually stored.

Intuitively, a table can be employed to enable address mapping and remapping, as used in the translation layer for the flash (Gal and Toledo, 2005). Table-based wear-leveling methods maintain an additional table to count the writes of each memory line and swap hot memory lines with cold ones (Zhou et al., 2009; Yun et al., 2012). Because table-based wear-leveling schemes incur great space and time overhead, algebraic mapping based wear-leveling schemes are proposed to calculate the address mapping using algebraic mapping instead of looking them up in a table (Qureshi et al., 2009; Woo et al., 2010). By continuously updating the parameters of the algebraic mapping functions periodically (every certain number of normal writes), the writes are distributed uniformly throughout the entire memory space. Furthermore, hybrid wear-leveling schemes use a table to record the region mapping, and swap two randomly chosen regions by algebraic schemes to achieve a better lifetime (Seznec, 2009; Yu and Du, 2014).

Wear-leveling schemes introduce extra write overhead during remappings, i.e., moving data from its old location to its new location. The customized 'remapping interval', defined as the number of normal writes before triggering a remapping, can impact the effectiveness of the wear-leveling schemes and the write overhead. A low remapping interval is more effective at distributing the writes uniformly across the memory space and more robust to malicious attacks, while it leads to a higher write overhead.

2.3 Security issues of wear-leveling

The limited write endurance of NVM not only affects lifetime reliability, but also incurs potential security threats. For example, a user who desires to obtain a product equipped with a brand new NVM within the warranty period might intentionally accelerate the wear-out of NVM. Both table-based and algebraic mapping based wear-leveling schemes suffer from security problems. For table-based methods, they are deterministic so that it is easy to guess and thus attack the location of the mapped line. For algebraic mapping based wear leveling, the vulnerability exists because the number of writes to cells is not consulted to generate the remappings. To better

understand the potential attacks, we roughly classify them into three categories:

1. Repeated address attack (RAA): RAA writes data to the same logical address repeatedly. Define the maximum number of writes to a line before that line is moved by a wear-leveling algorithm as the 'line vulnerability factor (LVF)' (Qureshi et al., 2011). Since a logical address is mapped to the same physical memory line within LVF writes, LVF should be less than the endurance, or an adversary can render a memory line unusable in one minute using RAA (Qureshi et al., 2009).

2. Birthday paradox attack (BPA): BPA randomly selects logical addresses and repeatedly writes to each one until it is remapped to another physical address. Surprisingly, after a small number of attempts, a memory cell is likely to be selected enough times and finally be worn out. To resist BPA, LVF should be dozen of times less than the endurance (Seznec, 2009).

3. Address inference attack (AIA): The attacker can compromise the operating system, and thereafter infer the logical addresses that will be subsequently mapped to the same physical location based on the knowledge of the wear-leveling scheme or the side-channel information gathered from the system (Woo et al., 2010). Writing to these logical addresses in sequence keeps writing the same target physical location and is able to make the NVM device fail in a short period.

As we can see, AIA is designed deliberately according to specific wear-leveling schemes, which is usually more efficient than RAA and BPA.

3 Remapping latency difference

The remappings of wear-leveling schemes incur extra latency because they are performed by either moving data from one line to a spare line or swapping data in two lines. Before elaborating on the new RTA timing attack, we introduce the observation that RTA is based on the fact that the remapping latency differs, depending on different remapping data.

3.1 Latency difference due to asymmetry in write time

Unlike charge-based memories such as dynamic random access memory (DRAM), NVMs store data in the form of change in two physical states represented by 0 and 1, respectively. The switching time of the states is different, leading to the 0/1 write asymmetry (Mittal et al., 2015). Assuming that the state of low resistance is used to present 0 and the state of high resistance is used to present 1, the time taken in the SET operation ($0 \rightarrow 1$) is several times larger than that taken in the RESET operation ($1 \rightarrow 0$) (Qureshi et al., 2012; Bishnoi et al., 2014); e.g., the latency of the SET operation is roughly eight times that of the RESET operation in PCM.

Generally, a memory line is the basic unit for memory requests. To write a memory line, after sending the write command and data, the controller is informed that all the bits have been written because it continually checks the status register (Micron Inc., 2011). Therefore, writing to a memory line is accomplished when all its bits have been written. Many studies indicate that the latency of writing a memory line with specific patterns differs (Qureshi et al., 2012; Li et al., 2016; Palangappa and Mohanram, 2016). Therefore, we could write artificial data to make the remapping latency different to gather side-channel information and detect 'potential information leakage'. Specifically, writing data whose bit values are all 0's (denoted as ALL-0) to a memory line is several times faster than writing data whose bit values are all 1's (denoted as ALL-1) to a memory line.

Fig. 1 shows the difference of remapping latency incurred by the asymmetric write time of NVM. In the case of moving a line to a spare line (Fig. 1a), a read and a RESET are needed to remap a line with data ALL-0 to the spare line, whose latency is 'relatively low', whereas to remap a line with data ALL-1, a read and a SET are needed, whose latency is 'relatively high'. In the case of swapping two lines, there are three remapping latency situations (Fig. 1b). If the data of the two swapping lines are both ALL-0, it takes two reads and two RESETs, and the latency is 'relatively low'. If the data of one line is ALL-0 and the other is ALL-1, the remapping latency is 'relatively medium' (two reads, a SET, and a RESET). When swapping two lines with data both being ALL-1, the remapping latency is 'relatively high' (two reads and two SETs). Noting that the extra remapping latency difference depends on the time associated with read, SET, and RESET, which is based on the type of NVM, the proposed timing

attack needs only the relativeness of remapping latency on the same NVM device.

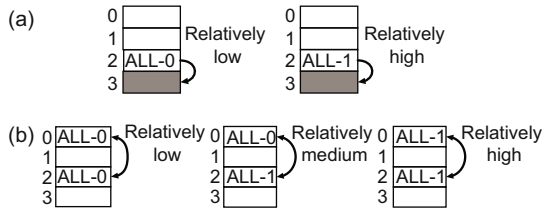

Fig. 1 Different remapping latency caused by asymmetric write time: (a) moving one line; (b) swapping two lines

3.2 Latency difference induced by redundant write reduction technologies

To solve the write endurance problem, data-comparison write schemes throttle unnecessary write traffic by preceding a write with a read (Yang et al., 2007; Zhou et al., 2009). The write traffic is reduced by about half because only the modified bits are written, and the extra read overhead is negligible because the latency and consumed energy of reads are much less than those of writes in NVMs. Therefore, write reduction technologies are being widely employed in NVM devices, e.g., a 58-nm PCM prototype (Chung et al., 2011).

Note that the remapping difference caused by the asymmetry in write time of NVM is altered by the redundant write reduction technologies: when writing ALL-0 to ALL-0 or writing ALL-1 to ALL-1, no write is actually performed and the latencies of these two cases are the same. However, these technologies introduce remapping latency in another manner. Fig. 2 shows the difference of remapping latency incurred by redundant write reduction technologies. Considering the Flip-N-write scheme (Cho and Lee, 2009), which further reduces the write traffic by flipping the written data if the number of modified bits is larger than half of the number of total bits, we flip half of the bits and use 0x00 and 0x01 as an example. In the case of moving a line to a spare line (Fig. 2a), if the remapped data is the same as the previously stored data, no write operation is performed and thus the remapping latency is equal to two-read latency, which is 'relatively low'. If there are bits that are different from the stored data, the remapping latency is equal to two-read latency plus a conventional write latency, which is 'relatively high'.

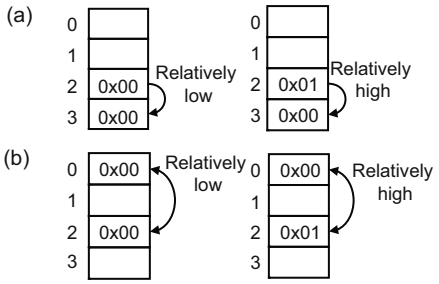

Fig. 2 Different remapping latency caused by redundant write reduction techonologies: (a) moving one line; (b) swapping two lines

In the case of swapping two lines (Fig. 2b), the latency is double that in the former case and shows similar characteristics.

In the rest of this study, we will clarify the proposed timing attack based on the remapping latency difference caused by the NVM's asymmetric write time, because it intrinsically exists in NVM devices. The proposed timing attack can also be performed in the case of redundant write reduction technologies with a little modification on constructing the artificial data and detecting the latency.

4 Vulnerability of prior wear-leveling schemes based on RTA

In this section, we first give a brief introduction to the remapping timing attack and explain it via the vulnerability of three state-of-the-art wear-leveling schemes in detail, i.e., region-based start-gap (Qureshi et al., 2009), security refresh (Woo et al., 2010), and multi-way wear leveling (Yu and Du, 2014). We assume that there are N memory lines in the memory space and the length of address is $\log_2 N$. Since the upper caches and buffers have been proved to be easily bypassed (Woo et al., 2010), we explain the attack models without considering any cache or buffer for simplicity.

4.1 Brief introduction to RTA

As in other attack models (Seznec, 2009; Woo et al., 2010), we assume that the operating system is compromised, in which only the malicious program is scheduled. RTA is able to detect the remapping latency, because any remapping halts other requests until it is completed (Woo et al., 2010), thereby incurring extra latency to the request that happens just after the remapping.

In RTA, the memory space is first filled with different artificial data (ALL-0 or ALL-1) according to different logical line address patterns. When a remapping is triggered, according to the remapping latency, RTA can infer which data (ALL-0 or ALL-1) is moved or swapped (Fig. 1). Because the data is associated with the logical line address, some remapping information is leaked. By repeatedly gathering the leaked information, RTA can obtain the essential information that is needed to perform malicious attacks. As we will explain, the essential information is the sequence of logical addresses that will be remapped subsequently for RBSG and the dynamic keys for SR and MWWL. Therefore, different procedures are designed to expose the vulnerability of the wear-leveling schemes.

4.2 Vulnerability of region-based start-gap

4.2.1 Region-based start-gap scheme

RBSG first translates LA to IA by a static randomizer to eliminate the spatial locality of the memory write traffic. The mapping from LAs to IAs is fixed once the system is booted. Then the IA space is divided into several continuous regions with equal size. Each region has an extra storage line called the 'gapline' that helps it evenly exhaust the memory space within a region by rotating each line one by one. Furthermore, for each region, there are two registers, where the 'start' register points to a memory line with the lowest intermediate address of the region and the 'gap' register points to the gapline.

An example of an RBSG remapping round is illustrated in Fig. 3. Initially, the gap register points to the extra gapline and the start register points to physical line 0 (Fig. 3a). Every certain number of writes to the region induces a remapping movement that copies (i.e., remapping) the content of the location $[(\text{gap}-1) \mod 4]$ to the location gap (Fig. 3b). A remapping round is accomplished when all the lines of the region have performed a remapping movement (Fig. 3c). With more writes, this remapping round continues (Fig. 3d).

4.2.2 RTA model to RBSG

Based on the remapping latency difference, we present the remapping timing attack model against RBSG. The whole memory space is assumed to be divided into R regions. The remapping interval is denoted as ψ; i.e., every ψ writes incur a remapping.

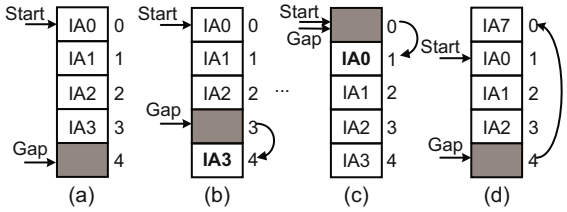

Fig. 3 An example of one RBSG remapping round: (a) initial state; (b) 1st remapping; (c) 4th remapping; (d) 5th remapping

The key idea of attacking RBSG is to find a sequence of logical addresses mapped to any sequence of physical addresses that are in the same region and sequential in the physical address space. Denote the dynamic remapping of LAs to PAs as f and the reverse remapping of PAs to LAs as f^{-1}. Given arbitrary logical address L_i, we would like to find another logical address L_{i-1} that is physically adjacent to L_i, i.e., $L_{i-1} = f^{-1}(f(L_i) - 1)$. It is easy to see that under RBSG L_i and L_{i-1} always point to two adjacent physical addresses, no matter how the remapping f changes over time. Assuming that now L_i is just remapped to a new physical address P, with another $\psi N/R$ writes to the region, all lines of the region will be moved by one line and P is pointed by L_{i-1}.

The goal is to wear out the physical address P, by writing to the logical addresses of the sequence $L_{i-n}, L_{i-n+1}, \cdots, L_i$ one by one after every $\psi N/R$ writes. We also note that, given the maximum endurance E of P, the length of the sequence must be at least $\lceil \frac{E}{\psi N/R} \rceil$ to wear out P. Given a logical address L_i, we now explain how to detect the sequence of $L_{i-n}, L_{i-n+1}, \cdots, L_{i-1}$ via RTA as follows:

1. We need to force the gapline to be adjacent to the memory line mapped by L_i, i.e., $\text{gap}+1 = f(L_i)$, by detecting the latency incurred by migrating the data of L_i.

Step 1: Write ALL-0 to all the logical addresses.

Step 2: Keep writing ALL-1 to L_i until the detected extra remapping latency is 'relatively high', which indicates that L_i is just remapped to a new physical address (Fig. 4a) (Recall that a remapping movement on the other lines incurs 'relatively low' latency by copying ALL-0 to the gapline).

Steps 1 and 2 need to be performed only once to determine the location to which gap points. After that, we can infer gap by counting the number of

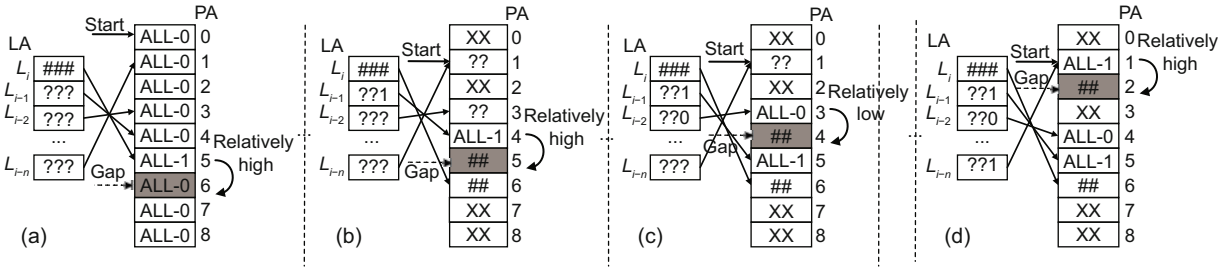

Fig. 4 An example of the detection of the logical address sequence $L_{i-n}, L_{i-n+1}, \cdots, L_{i-1}$

#: the value is known but omitted for simplicity; ?: the value is unknown and needs to be detected; X: the value is unknown and irrelevant to the detection

writes to the region.

2. Detect the j^{th} bit of $L_{i-n}, L_{i-n+1}, \cdots, L_{i-1}$.

Step 3: Write ALL-0 to the lines with the j^{th} bit of the logical address being 0; otherwise, write ALL-1. This step requires N writes, while only N/R writes occur to the region to which L_i belongs.

Step 4: Keep writing L_i with the content mentioned in step 3 to trigger the remapping of the region to which L_i belongs. Recalling that a logical address is remapped every $\psi N/R$ writes, with another $(\psi-1)N/R+\psi$ writes, L_{i-1} is remapped again. If the remapping latency is 'relatively low', the j^{th} bit of L_{i-1} is 0; otherwise, it is 1 (Fig. 4b). Repeat this step to deduce all the j^{th} bits of $L_{i-n}, L_{i-n+1}, \cdots, L_{i-1}$ as shown in Figs. 4c and 4d.

Step 5: Repeat steps 3 and 4 to detect all the bits of $L_{i-n}, L_{i-n+1}, \cdots, L_{i-1}$, which requires $(N + (\psi-1)N/R)\log_2 N$ writes.

The proposed RTA is more efficient than the existing attack models. Compared with the attack model on RBSG based on bank-level parallelism (Woo et al., 2010), which is invalid when a region is located inside a bank, RTA is valid under such circumstance. Compared with the attack model based on BPA (Seznec, 2009), which is impractical when decreasing the remapping interval by an online attack detector (Qureshi et al., 2011), decreasing the remapping interval instead accelerates RTA. As proposed in the original RBSG paper, the 'delayed write policy' ensures that the attackers have to write more extra lines besides the attacked line, and consequently it takes more time to make a memory line fail, whereas we note that RTA is still efficient.

4.3 Vulnerability of security refresh

4.3.1 Security refresh scheme

In SR (Woo et al., 2010), the PA of an LA is calculated by an XOR operation with a random key. SR dynamically remaps the memory lines, in which the candidate LA to be remapped is pointed by a register called the 'current refresh pointer (CRP)', which increases by 1 after each remapping. To facilitate address translation, two registers are employed: key_c indicates the key of the current round and key_p indicates the key of the previous round.

An example of one SR remapping round is shown in Fig. 5. Fig. 5a shows the initial state in which all the LAs have just been mapped to the PAs with key_p(0x10). Fig. 5b displays the first remapping of a new round with key_c (0x11). The new PA of LA (0x00) is 0x11, because LA $\oplus$ key_c=(0x00)$\oplus$(0x11)=(0x11). For SR, each logical address L has a paired LA denoted as paired(L), where $L \oplus \text{key}_\text{c} = \text{paired}(L) \oplus \text{key}_\text{p}$. We have paired($L$)$\oplus$ $\text{key}_\text{c} = L \oplus \text{key}_\text{p}$. That is, the new PA for L is actually the old PA for paired(L) and vice versa. Therefore, the remapping is performed by swapping the physical lines pointed by L and paired(L), i.e., line 0x10 and line 0x11 in the PA. For the second remapping (Fig. 5c), LA (0x01) has been remapped by the previous remapping of LA (0x00) and thus is not remapped again but simply increments the CRP. After several remapping movements, all the LAs have been remapped and the next remapping round starts (Fig. 5d).

Security refresh is further improved to a hierarchical, two-level security refresh scheme. The outer-level SR maps LAs to IAs, and the IA space is divided into several equally sized sub-regions as in RBSG. Each sub-region is managed by an inner-level SR

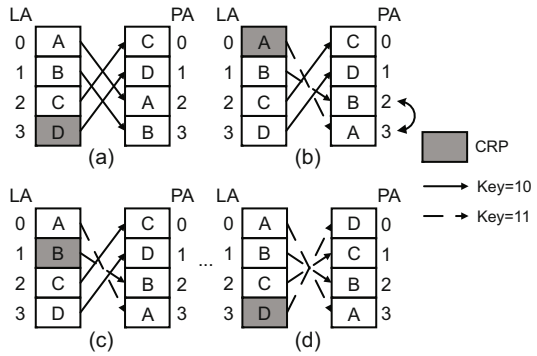

Fig. 5 An example of one SR remapping round: (a) initial state; (b) 1st remapping; (c) 2nd remapping; (d) 4th remapping

to translate IAs to PAs. Both levels apply the SR scheme, but are transparent to and independent of each other.

4.3.2 RTA to one-level SR

Recall that remapping L_i is just swapping the physical addresses and data of L_i and paired(L_i), where paired(L_i) = $L_i \oplus \text{key}_\text{c} \oplus \text{key}_\text{p}$. To wear out the physical line now pointed by L_i, we keep attacking L_i until it is remapped. By then the target physical line is pointed by paired(L_i), and we keep attacking paired(L_i) until the current remapping finishes (CRP reaches 0x00 again). In the new remapping round, we let $L_i \leftarrow$ paired(L_i), and keep attacking the new L_i and the corresponding $\text{paired}_\text{new}(L_i)$ (note that $\text{key}_\text{c} \oplus \text{key}_\text{p}$ is changed). This procedure continues until the target physical line fails.

To compute paired(L_i) with given L_i, we need the value of ($\text{key}_\text{c} \oplus \text{key}_\text{p}$), which is invariant in the current remapping round. At the beginning of the attack, we force the CRP to point to the logical address 0x00, by detecting the latency incurred by remapping logical address 0x00. The procedure is analogous to RTA model to RBSG. After that, the CRP can be calculated by counting the number of writes.

Now detect the j^{th} bit of ($\text{key}_\text{c} \oplus \text{key}_\text{p}$).

Step 1: Set all lines with the j^{th} bit of the logical address being 0 to ALL-0; otherwise, set to ALL-1. Because only the lines whose corresponding logical address on the j^{th} bit is different from that on the $(j-1)^{\text{th}}$ bit need to be rewritten, $N/2$ writes are needed.

Step 2: Keep writing ALL-0 to the logical address 0x00 to trigger remapping. Recall the different latency of remapping different data in Fig. 1b. If the extra remapping latency is 'relatively high' or 'relatively low', the j^{th} bit of ($\text{key}_\text{c} \oplus \text{key}_\text{p}$) is 0; otherwise, it is 1. To see this, recall that each remapping swaps two lines LA and paired(LA), where LA $\oplus$ paired(LA) = $\text{key}_\text{c} \oplus \text{key}_\text{p}$, and thus

$$\begin{aligned}&[\text{key}_\text{c} \oplus \text{key}_\text{p}]_j = [\text{LA} \oplus \text{paired(LA)}]_j\\ &= [\text{LA}]_j \oplus [\text{paired(LA)}]_j,\end{aligned}$$

where $[\cdot]_j$ is the j^{th} bit of a binary value. In the best case, we could detect the write delay incurred by a remapping movement just after step 1. In the worst case, another $N/2$ writes are needed to make a new remapping movement happen.

Repeat steps 1 and 2 to detect all bits of ($\text{key}_\text{c} \oplus \text{key}_\text{p}$). It takes at least $(N \log_2 N)/2$ writes and at most $N \log_2 N$ writes in total.

4.3.3 RTA to two-level SR

For two-level security refresh, to perform efficient and secure wear leveling, the address space of each sub-region is rather small, which makes wearing out all the lines of a subregion practical. Therefore, we keep track of the remapping of the target subregion and then attack the whole sub-region until one of its lines fails. Now we analyze the feasibility of this attack scheme. Because the first $\log_2 R$ bits of a logical address indicate the subregion to which it belongs, we need to detect the first $\log_2 R$ bits of the ($\text{key}_\text{c} \oplus \text{key}_\text{p}$) of the outer-level SR, which needs at least $(N \log_2 R)/2$ writes and at most $N \log_2 R$ writes. After that, we could attack the target subregion at least $N(\psi - \log_2 R)$ times for each outer-level remapping round. The detection time is much less than a remapping round, because $\log_2 R$ is usually 7 to 10, which is much smaller than ψ.

4.4 Vulnerability of multi-way wear leveling

4.4.1 Multi-way wear-leveling scheme

MWWL is a fundamental hybrid scheme that can be applied to existing wear-leveling algorithms, such as security refresh and start-gap (Yu and Du, 2014). The whole memory space is first divided into multiple fixed-size regions. To ensure security, each region is swapped with a randomly chosen region called its 'paired region' in each remapping round. The information of the paired region of each region

is tracked by a dedicated register. In Fig. 6, logical regions $\boldsymbol{L}_1$ and $\boldsymbol{L}_2$ are randomly chosen to be paired regions and are now mapped to physical regions $\boldsymbol{P}_1$ and $\boldsymbol{P}_2$ (here we use bold font to denote regions), respectively. Every certain number of writes to a region will trigger a remapping that swaps one line in the region with another line in its paired region by algebraic mapping based methods such as start-gap and security refresh. After all the lines of the paired regions have been swapped, $\boldsymbol{L}_1$ and $\boldsymbol{L}_2$ will be mapped to $\boldsymbol{P}_2$ and $\boldsymbol{P}_1$ respectively, which indicates that the current remapping round of $\boldsymbol{L}_1$ and $\boldsymbol{L}_2$ is finished and in the next remapping round they will randomly choose other new paired regions. Note that a region $\boldsymbol{A}$ may choose region $\boldsymbol{B}$ as its paired region which is currently paired with another region $\boldsymbol{C}$. In that case, the writes to region $\boldsymbol{A}$ will trigger and accelerate the swapping of region $\boldsymbol{B}$ and region $\boldsymbol{C}$ until the swapping finishes, so that region $\boldsymbol{B}$ could be swapped with region $\boldsymbol{A}$ from then on.

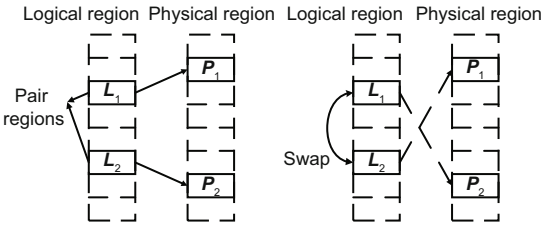

Fig. 6 An example of multi-way wear leveling

4.4.2 RTA model to MWWL

To keep attacking a physical region in MWWL, we need to learn about the logical region sequence $\boldsymbol{L}_{i-n}, \boldsymbol{L}_{i-n+1}, \cdots, \boldsymbol{L}_{i-1}$ that will be mapped to the same physical region in the following remapping rounds. For example, to attack $\boldsymbol{P}_1$ as shown in Fig. 6, we first write to $\boldsymbol{L}_1$ and then write to its paired region $\boldsymbol{L}_2$ in the next remapping round, and so on.

We show how to detect the paired region $\boldsymbol{L}_{i-1}$ of a given region $\boldsymbol{L}_i$ via RTA. First, similar to the former attack models, we make sure $\boldsymbol{L}_i$ is just remapped by detecting the latency caused by the swapping of $\boldsymbol{L}_i$ and $\boldsymbol{L}_{i-1}$.

Now detect the j^{th} bit of the address of $\boldsymbol{L}_{i-1}$, and by repeating steps 1–3 we can obtain all the bits of $\boldsymbol{L}_{i-1}$.

Step 1: For any region, we set all the lines that belong to the region whose j^{th} bit of the logical address is '0' to ALL-0, and to ALL-1 otherwise, which takes N writes and N/R writes occur in $\boldsymbol{L}_i$.

Step 2: Keep writing to $\boldsymbol{L}_i$ until an extra remapping latency is detected. If the latency is 'relatively medium', the j^{th} bit of $\boldsymbol{L}_{i-1}$ is different from $\boldsymbol{L}_i$; otherwise, it is the same.

Step 3: Write ALL-1 to the target region and ALL-0 to others. Note that a region may not enter a new remapping round after its current remapping round finishes, because its new paired region may be currently paired with another region. To ensure that the detection is performed at the beginning of a new remapping round, we write ALL-0 to the target region and ALL-0 to the other regions and keep writing on the target region. When the extra remapping latency is 'relatively high' or 'relatively low', it is indicated that the target region is remapped and we need to detect its next paired region.

Step 1 takes $N \log_2 R$ writes for each bit detection, which is the most time-consuming step. Fortunately, the write time can be dramatically reduced. Because we have confirmed the 0 to $(j-1)^{\text{th}}$ bits of $\boldsymbol{L}_{i-1}$ when detecting the j^{th} bit, only the regions that have the same value as these bits need to be written and detected, which means that only $N/2^j$ writes are needed. Therefore, it takes about $N + N/2 + \cdots + N/2^{\log_2 R - 1}$ writes for step 1 in total to detect all the bits of $\boldsymbol{L}_{i-1}$. Moreover, by a simple improvement that rewrites only the region whose old value is different from the new, we can reduce the write time of step 1 by half. That is, fewer than N writes are needed and about $(N \log_2 R)/(2R)$ writes occur to $\boldsymbol{L}_i$, which is far smaller than the number of writes of an $\boldsymbol{L}_i$ remapping round.

Further, we can target on one line of the region to accelerate the failure of the NVM memory. For start-gap, we keep writing the neighborhood logical line in a new remapping round because all the lines are rotated by one for each remapping round. For SR, as described in Section 4.3.3, it takes about $[N \log_2(N/R)]/R$ writes to detect which logical address in the target region is mapped to the target line. The efficiency of RTA model to MWWL is empirically studied in Section 6.1.3.

5 Design of security region-based start-gap

High security is critical in wear-leveling schemes for application of NVM as main memory, which must ensure that the lifetime of NVM-based memory is comparable to the ideal lifetime, even under worst-case scenarios, including resisting all the potential malicious attacks known to be feasible. In this section, we describe a novel wear-leveling scheme named 'security region-based start-gap', which is designed to resist malicious attacks, including the RTA presented earlier.

5.1 Overview of security RBSG

The proposed security-aware wear-leveling scheme is implemented in the memory controller and manages each bank separately to avoid bank parallelism attack. Fig. 7 shows the architecture of the security RBSG. Two-level dynamic mappings are adopted to do security-aware wear leveling; here, 'dynamic' indicates that the mapping changes every remapping round. The outer-level wear leveling is called 'security-level adjustable dynamic mapping', which employs a dynamic Feistel network mapping to transform LAs to IAs and is in charge of providing security assurance. The Feistel network is a widely used one-to-one mapping method in cryptography (Menezes et al., 1996). The details of dynamic Feistel network mapping will be explained in the following subsection. We also note that RBSG uses a static Feistel network to facilitate address-space randomization, which is different from the proposed dynamic Feistel network. To avoid a 'repeated address attack' and improve wear-leveling efficiency, the IA space is then divided into multiple, fixed-size sub-regions by the sequence of IAs. Because security is ensured by outer-level wear leveling, inner-level wear leveling just needs to ensure that the write traffic distributes uniformly in the normal situation with low overhead. The inner-level wear leveling adopts start-gap mapping in Section 4 for each sub-region separately to transform IAs to PAs (Qureshi et al., 2009). As described earlier, the start-gap mapping introduces very low overhead. Although the remapping start-gap rule is too simple to avoid address information leakage, security has been ensured by outer-level wear leveling.

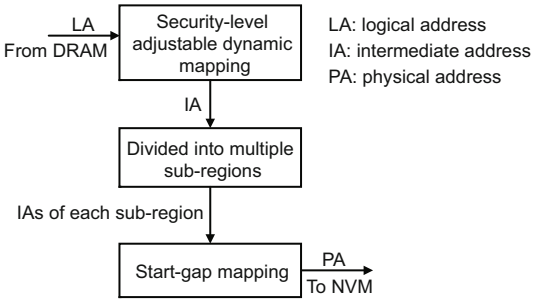

Fig. 7 Architecture of security RBSG

5.2 Security level adjustable dynamic mapping

The outer-level wear leveling uses a Feistel network to transform LAs to IAs by an array of random keys. A Feistel network is simple to implement, has been studied extensively, and is widely used in the construction of block ciphers. Before describing the proposed dynamic Feistel network in detail, we will first give a brief introduction to Feistel networks.

Fig. 8a shows the logic for the encryption of a multi-stage Feistel network. Each stage splits the B-bit input into two parts (L and R) and provides output that is also split into two parts (L' and R'). R' is equal to L. L' is provided by an XOR operation of R, the output of a round function F_1 on L, and some randomly generated key K. We choose the round function to be the cubing function of $(L \oplus K)$ in this case for ease of implementation. Thus, the transformation is given by

$$L' = R \oplus (L \oplus K)^3.$$

For the decryption of a Feistel network, one interesting thing is that the encryption and decryption operations are very similar, even identical in some cases, requiring only a reversal of the key schedule (Fig. 8b).

Recall that RBSG uses a three-stage Feistel network to randomize the address space with negligible storage and latency overhead, where theoretical work has shown that three stages can be sufficient to make a pseudo-random permutation of the address space. However, we have shown that a static Feistel network (whose secret keys are randomly generated and kept constant) is vulnerable to the presented RTA, no matter how many stages of the Feistel network are configured. In this work, we propose a dynamic Feistel network (DFN) whose secret keys change every remapping round and it aims to not only randomize

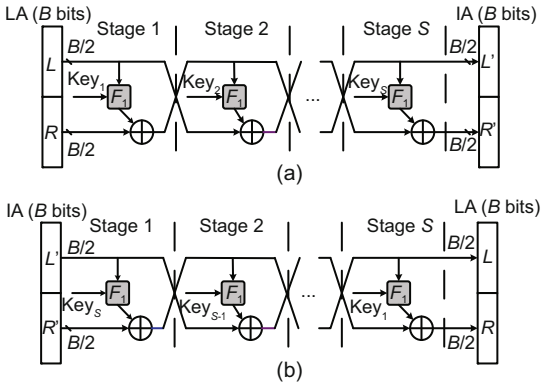

Fig. 8 Encryption (a) and decryption (b) of a multi-stage Feistel network

the address space, but provide security assurance.

We explain how the proposed DFN works using a complete remapping round on a memory system with eight lines and an extra spare line (Fig. 9). The transformation of IAs to PAs is transparent to the DFN, and the DFN illustration is presented as if it were operating on the physical lines directly.

To facilitate the remapping movement of the DFN, we need an extra spare line, two registers (namely, start and gap), and two register arrays K_c and K_p (the number of items is equal to the number of stages in the DFN). The start register tracks the old location of the line whose content has been moved to the extra spare line and the gap register keeps track of the location of the current spare line. The K_c array stores the secret keys used in the current remapping round and the K_p array stores the secret keys used in the previous remapping round. Denote the encryption and decryption of the DFN by $\mathrm{ENC}_{\mathrm{key}}(\cdot)$ and $\mathrm{DEC}_{\mathrm{key}}(\cdot)$, respectively, where key can be K_p or K_c.

In Fig. 9a, at the beginning of a remapping round, the transformation of LAs to IAs is all given by $\mathrm{ENC}_{K_p}(\cdot)$. A new secret key array K_c is then randomly generated. Line 8 is initially used as the extra spare line. The location of the spare line changes as the wear-leveling remapping goes on, which is always pointed by the gap register.

To start the remapping round, start first points to an un-remapped line (say, line 0) and moves the content of line 0 to the spare line (line 8), as shown in Fig. 9b. Registers gap and start now point to line 0. With every ψ writes to the memory, DFN will remap one line. Specifically, denote Loc as the logical address that points to the gap line under current secret keys, namely, $\mathrm{Loc} = \mathrm{DEC}_{K_c}(\mathrm{Gap})$. Move the data of Loc stored in line $\mathrm{ENC}_{K_p}(\mathrm{Loc})$ to the current gap line and mark the previous line $\mathrm{ENC}_{K_p}(\mathrm{Loc})$ as the new gap line by changing gap $\leftarrow \mathrm{ENC}_{K_p}(\mathrm{Loc})$. We can see that the transformation of Loc is now under the current secret keys K_c; that is, the remapping of Loc is completed in Fig. 9c with Loc $= D$ and $\mathrm{ENC}_{K_c}(\mathrm{Loc}) = 3$. Repeat the above operations to remap other lines, until the resulting $\mathrm{ENC}_{K_p}(\mathrm{Loc})$ is now equal to start (line 0). Although the old content of line 0 has been moved to line 8 at the beginning of the current remapping round, we need to move the content of line 8 to the gap line and let gap point to line 8 as shown in Fig. 9d. In the next remapping, let start point to an un-remapped line (say, line 1) and move the content of line start to line 8 just like the beginning step (Fig. 9e). The following remappings will be performed by moving the data of Loc to the current gap line until the $\mathrm{ENC}_{K_p}(\mathrm{Loc})$ is equal to start again (Fig. 9f).

The above remapping movement is formally depicted in Algorithm 1. It is not hard to see that, by repeating the above remappings, all the LAs will be remapped to new IAs with the current secret key array K_c (Fig. 9g). Therefore, at the end of the remapping round, the transformation of LAs to IAs will be given by $\mathrm{ENC}_{K_c}(\cdot)$, instead of $\mathrm{ENC}_{K_p}(\cdot)$. For the beginning of the next remapping round, K_c becomes K_p and K_c will be a new randomly generated secret key array.

The LA to be remapped is calculated during the running time. To facilitate address translation, each memory line needs an 'isRemap' bit to mark if it has been remapped. When a memory block is remapped, its corresponding isRemap bit is set. All isRemap bits are reset when a remapping round finishes. The address translation is rather simple with the use of extra isRemap bits. If the corresponding isRemap bit of each LA is set (meaning that LA has been remapped), use the current key array K_c to calculate its IA. If the corresponding isRemap bit of each LA is not set (meaning that the LA has not been remapped), use the previous key array K_p to calculate its IA. Noting that we move the content of the start line to the extra spare line for some remappings, if the resulting IA is equal to start and the corresponding isRemap bit is not set, let the IA be N if gap = start; otherwise, let it be start.

We can see how DFN can resist the potential

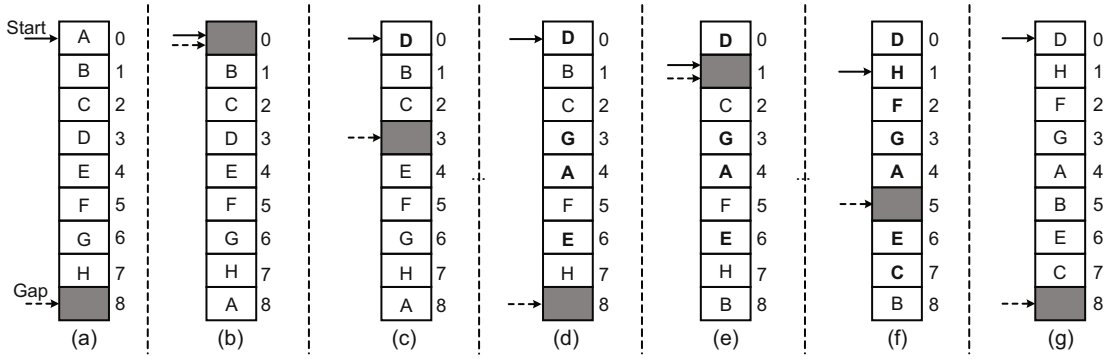

Fig. 9 An example of one complete dynamic Feistel network remapping round

(a) Initial state; (b) 1st remapping; (c) 2nd remapping; (d) 5th remapping; (e) 6th remapping; (f) 9th remapping; (g) 10th remapping. The line pointed by gap is indicated by a gray background, and the memory lines marked in bold indicate that the corresponding lines have been remapped in the current round

Algorithm 1 Remapping a line of DFN

Input: number of lines in memory N

if gap $= N$ **then**
  if all lines have been remapped **then**
    start $\leftarrow 0$
    reset all isRemap to False
    $K_\mathrm{p} \leftarrow K_\mathrm{c}$
    let K_c be equal to a new random key array
  else
    let start be equal to an un-remapped line
  end if
  [gap] $\leftarrow$ [start]
  gap $\leftarrow$ start
else
  Loc $\leftarrow \mathrm{DEC}_{K_\mathrm{c}}$(gap)
  if $\mathrm{ENC}_{K_\mathrm{p}}$(Loc) = start **then**
    [gap] $\leftarrow [N]$
    gap $\leftarrow N$
  else
    [gap] $\leftarrow$ [Loc]
    gap $\leftarrow$ Loc
    isRemap[$\mathrm{DEC}_{K_\mathrm{c}}$(gap)] $\leftarrow$ True
  end if
end if

remapping timing attack. Because the round function used in the DFN is much more complex than the simple XOR, it takes many more writes to detect a single bit than SR. To say the least, suppose one bit of the DFN's secure key can be detected via N/R writes, as in SR. To avoid the leakage of the secure key, the number of writes that are required to detect the key should be larger than that of a remapping round. For a 1 GB NVM bank with a 256-byte linesize, the length of the secure key in each stage is 22. Supposing the inner-level remapping interval is 64 and the outer-level remapping interval is 128, a 128-bit key array length will make the detection fail, which suggests a 6-stage DFN in this case.

The DFN is fixed on the circuit and the number of stages is determined beforehand. Adding stages of the Feistel network is an effective way to improve security with the cost of additional key storage and operation circuits of each stage. We can make a trade-off between security assurance and hardware/performance overhead by changing the number of DFN stages. For an NVM configuration, we can estimate the security level beforehand and use an appropriate number of DFN stages to prevent malicious attacks. How the number of stages affects the security will be discussed and detailed in Section 6.2.1.

6 Evaluation

In this section, we evaluate the efficiency of RTA against three state-of-the-art wear-leveling schemes (RBSG, two-level SR, and multi-way SR), and the efficiency of security RBSG. Without loss of generality, we use PCM as a representative example of NVM in the following evaluations. The other NVM technologies should have similar results.

In the evaluation, we assume the device read time as 125 ns and the write time as 1000 ns (Qureshi et al., 2012). The size of the block is the same as that of the last-level cache line (256 bytes). We also assume that the write endurance for a storage cell is 10^8. Wear leveling performs within a memory bank and the capacity of a bank is assumed to be 1 GB. Because the upper caches and buffers have

been proved to be easily bypassed (Woo et al., 2010), the lifetime evaluations use a simple memory model that ignores caches and buffers.

6.1 Efficiency of RTA

6.1.1 Against RBSG

For RBSG, to resist an RAA, the PCM bank should be divided into multiple regions and the number of lines in a region should be smaller than Endurance/ψ. Furthermore, to resist BPA, there must be no more than Endurance/(8ψ) lines in a region. Thus, we vary the number of regions from 32 to 128 and the remapping interval from 16 to 100, where 100 was recommended by Qureshi et al. (2009). We make several observations from the evaluation as shown in Fig. 10. First, the lifetime of RBSG under an RTA decreases as the number of regions increases, because the fewer lines in a region, the less time it takes to detect the address sequence. Second, decreasing the remapping interval, which is an efficient way to resist an RAA or a BPA (Qureshi et al., 2011), makes PCM fail faster under RTA. Third, with the recommended configuration (32 regions and remapping interval as 100), RTA makes the PCM fail in 478 s, which is 27 435 times faster than under RAA.

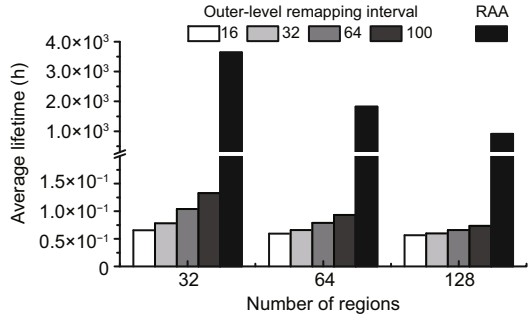

Fig. 10 The lifetime of RBSG under RTA and RAA

6.1.2 Against two-level SR

In this subsection, we evaluate RTA against two-level SR. The configuration suggested by two-level SR is 512 sub-regions, the inner-level remapping interval as 64, and the outer-level remapping interval as 128. To measure the sensitivity of the PCM lifetime under RTA according to the three parameters mentioned eariler, we vary the configurations as shown in Table 1. In Section 4.3, the key detection time varies from $(N \log_2 R)/2$ to $N \log_2 R$ with different outer-level wear-leveling keys. Therefore, for each configuration we run RTA five times, each time randomly generating a key, and the average lifetime is used for evaluation.

Table 1 Configuration of the subregion, inner-level refresh interval, and out-level refresh interval

Parameter	Value
Number of sub-regions	256, 512, 1024
Inner-level remapping interval	16, 32, 64, 128
Outer-level remapping interval	16, 32, 64, 128, 256

In Fig. 11, we draw the following observations. First, the lifetime of two-level SR under RTA decreases as the number of sub-regions increases, because there are fewer lines in the target subregion to be attacked. Second, the lifetime decreases as the outer-level remapping interval increases. Because the key detection time is uncorrelated to the outer-level remapping interval and the remapping round becomes longer with higher outer-level remapping interval, we could write the target sub-region more times in each remapping round. Third, with the recommended configuration, the lifetime of PCM is about 178.8 h. Above all, the results are consistent with our previous theoretical analysis.

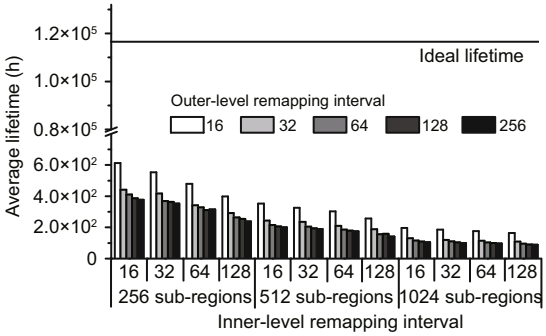

Fig. 11 Average lifetime of two-level SR under an RTA

To better evaluate the efficiency of RTA, an RAA is also evaluated for comparison. For two-level SR, an RAA has been proved to have the same effect as a BPA (Woo et al., 2010). In Fig. 12, the lifetime of two-level SR under RAA is about 105 months, which is 322 times longer than that under RTA.

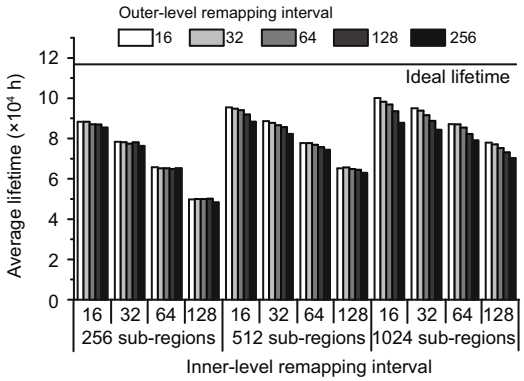

Fig. 12 Average lifetime of two-level SR under an RAA

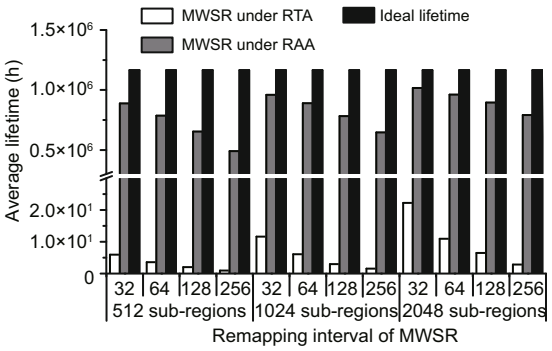

Fig. 13 Average lifetime of MWSR under RTA and RAA

6.1.3 Against MWSR

We use multi-way security refresh (MWSR) as an example to evaluate multi-way wear leveling, and other multi-way wear-leveling schemes should show similar results. MWSR incurs less write overhead and uses more regions to achieve a comparable lifetime with two-level SR. Thus, we use a number of regions that varies from 512 to 2048 and a remapping interval that varies from 32 to 256 in MWSR evaluation. The lifetime of MWSR under RTA is compared with the ideal lifetime and the lifetime under an RAA (Fig. 13). Note that, like SR, the lifetime of MWSR under an RAA is similar to that under a BPA. Under both RAA and RTA, the lifetime of MWSR increases as the region number increases and the remapping interval decreases. However, RTA is still efficient with a reasonably large number of regions and low remapping interval as our previous theoretical analysis shows. For example, with 2048 regions and the remapping interval as 32, the lifetime of MWSR under RTA is only about 2.2 h, which is 1/52 440 of the ideal lifetime. With 1024 regions and a remapping interval of 128, MWSR achieves 67% of the ideal lifetime under an RAA, but only 1/190 626 of the ideal lifetime under RTA. In summary, the experiment results demonstrate that MWSR performs well under an RAA and BPA but fails to protect PCM from RTA.

6.2 Effectiveness of security RBSG

6.2.1 Number of stages

We evaluate how the number of stages affects security in security RBSG. The number of stages can be configured on boot time to make a trade-off between security and overhead. More stages achieve better randomness and higher security, at the expense of more hardware and computational overhead. To decide the number of stages, two factors should be considered. One is security. In Section 5, $K \geq 6$ can avoid information leakage and ensure security when the outer-level remapping interval is not larger than 132. The second factor is lifetime. We vary the number of stages from 3 to 20 with the configuration suggested by two-level SR and compare the lifetime of security RBSG with that of two-level SR under an RAA and BPA and the ideal lifetime (Fig. 14). RBSG suggests that three stages can randomize normal workload writes, but it can achieve only about 20% of the ideal lifetime under an RAA. Under an RAA, the lifetime of security RBSG with seven stages is 67.2% of the ideal lifetime and a little more than the lifetime of two-level SR. Under a BPA, security RBSG with seven stages achieves 66.4% of the ideal lifetime. Note that BPA is insensitive to the number of stages because the attacked logical addresses are chosen randomly. To ensure both security and lifetime with low overhead, we choose the number of stages as 7 in the following evaluations.

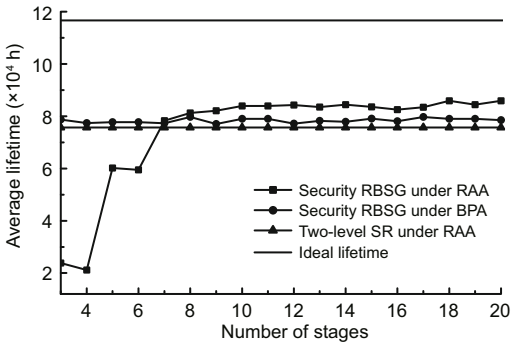

Fig. 14 Average lifetime of different DFN stages

6.2.2 Wear leveling

We evaluate the lifetime of security RBSG under an RAA with the same configurations in Table 1. From Fig. 15, we can see that the results show characteristics similar to two-level SR. Increasing the inter-level remapping interval and the number of regions could make the write traffic more uniform and thus achieve better lifetime, at the expense of incurring more overhead. The only difference is that lifetime increases as the outer-level remapping interval increases. This is because security RBSG adopts start-gap mapping for inner-level wear leveling, which makes writes from an RAA distribute subsequently within an outer-level remapping round. Security RBSG could achieve wear-leveling efficiency comparable to two-level SR, in the same time providing higher security. With the recommended configuration, security RBSG is able to endure more than 108 months.

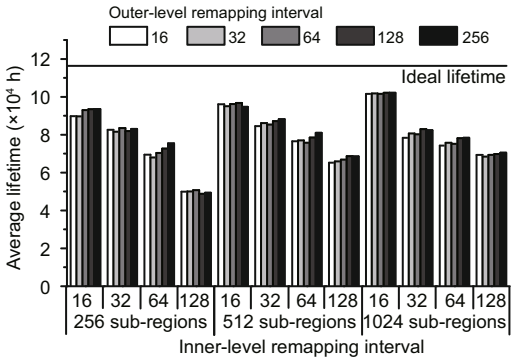

Fig. 15 Average lifetime of security RBSG under RAA

To evaluate the lifetime of security RBSG under normal workloads, we collect 100 million writes of four memory-intensive workloads of SPEC CPU2006 (Gove, 2007) and a multi-programmed workload consisting of the four workloads, and use the per-line profile to compute the lifetime. Limited by the huge simulation time, we run $4M \times 10^6$ writes (10^6 writes per line on average) and use the result to evaluate the lifetime with endurance of 10^8. Denoting the number of writes to the line with the maximum number of writes as $W_{\max}$, we define the normalized lifetime as

$$\frac{\text{The average number of writes per line}}{W_{\max}},$$

which equals (achieved lifetime)/(ideal lifetime) under endurance as $W_{\max}$. Normalized endurance closer to 100% indicates that the wear-leveling schemes can achieve a system lifetime closer to the ideal lifetime. In Fig. 16, the normalized lifetime increases as the total number of writes increases, which indicates that the writes are distributed more evenly. The normalized lifetime of leslie3d, milc, and multi-program are all above 86%, with $W_{\max}$ as 1.06×10^6, 1.11×10^6, and 1.16×10^6, respectively. Even for the worst case soplex, the normalized lifetime is 57.7% with $W_{\max}$ as 1.7×10^6. As the $W_{\max}$ increases to 10^8, which corresponds to the actual endurance of 10^8, the lifetime under normal workloads is expected to be nearly perfect.

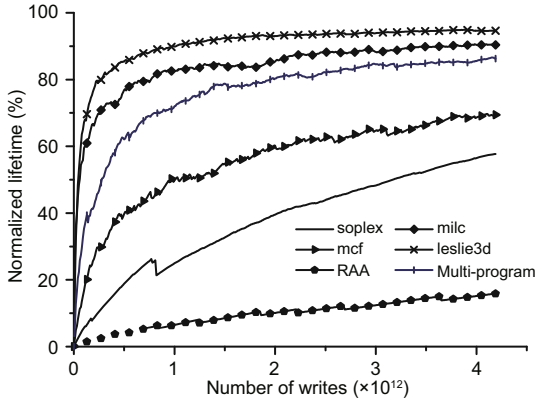

Fig. 16 Normalized lifetime of normal workloads and an RAA

6.2.3 Hardware overhead

In this subsection, we describe the hardware overhead of security RBSG. We first analyze the storage overhead. The outer-level of security RBSG needs $2B$ (i.e., $\log_2 N$) register bits for gap and start, and $\log_2 \psi$ register bits for the counter. For each stage, it requires B register bits to store K_c and K_p. Assuming that there are S stages, a total of $(S+2)B + \log_2 \psi$ register bits are needed. For the inner-level start-gap mapping, each sub-region requires $2\log_2(N/R)$ bits for gap and start, and $\log_2 \psi$ bits for the counter. Therefore, security RBSG requires a total of $(S+2)B + \log_2 \psi + [2\log_2(N/R) + \log_2 \psi]R$ register bits. For the recommended configuration, it costs about 2 KB of register for a 1 GB bank. The outer-level and each subregion of the inner-level

need an extra line, which is a total of $(S+1)\cdot 256$ bytes NVM. Each line needs an extra bit to represent whether it has been remapped, for a total cost of $\log_2 N$-bit (i.e., 0.5 MB) SRAM. We use the cubing function as the round function of the Feistel network. The cubing function can be considered as a multiplier after a squaring. Because the squaring circuit requires approximately $0.5B^2$ gates and the multiplier circuit requires approximately B^2 gates (Liddicoat and Flynn, 2000), each cubing circuit requires about $3B^2/8$ gates. Therefore, security RBSG of the S stage requires $3SB^2/8$ gates.

6.2.4 Performance impact

Finally, we evaluate the performance impact of the proposed security RBSG using the Gem5 simulator (Binkert et al., 2011) with 13 SPEC CPU2006 benchmarks (Gove, 2007). In the experiment platform, the system consists of an 8-core processor (3.2 GHz) with a private 32 KB L1 cache, a shared 256 KB L2 cache, and an 8 MB L3 DRAM cache. Because each stage of the DFN costs one cycle and an access to SRAM takes three to five cycles, we assume that the address translation of security RBSG requires 10 cycles, while the address translation latency of two-level SR is assumed to be one cycle. The address translation latency is negligible compared with the read/write latency. The inner-level and outer-level remapping intervals are assumed to be 64 and 128, respectively.

The IPC degradation of security RBSG and two-level SR compared against the baseline without any wear-leveling schemes is shown in Fig. 17. The IPC degradation of security RBSG shows characteristics similar to that of two-level SR, because the write overheads incurred by the two schemes are the same with the same remapping interval. The geometric mean of IPC degradation of security RBSG (2.4%) is slightly larger than that of two-level SR (2.1%) because of the extra address translation overhead. We also evaluate the average read latency (Fig. 18). The average read latencies of the baseline, two-level SR, and security RBSG for all workloads are almost the same. The geometric means of average read latency for the baseline, two-level SR, and security RBSG are 458, 474, and 476 μs, respectively, indicating that the extra read latency caused by security RBSG is negligible. In summary, security RBSG incurs very low performance degradation.

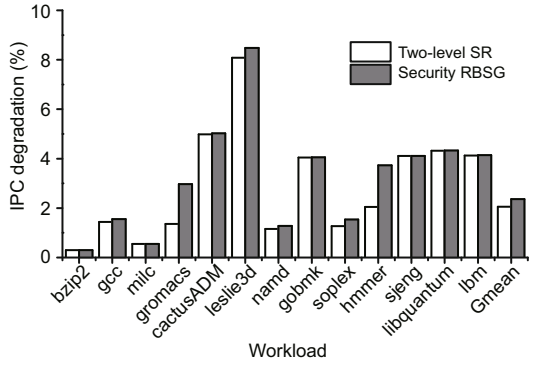

Fig. 17 IPC degradation of two-level SR and security RBSG

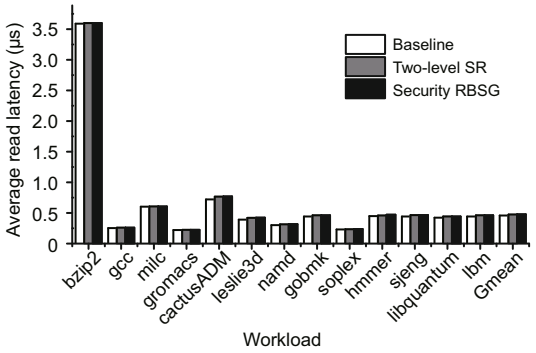

Fig. 18 Average read latency of the baseline, two-level SR, and security RBSG

7 Conclusions

In this study, we presented RTA, a new type of timing attack that is based on the remapping latency difference. RTA is shown to be more effective in attacking the NVMs with three state-of-the-art wear-leveling schemes (RBSG, SR, and MWWL). To resist RTA, we propose a novel wear-leveling scheme called security RBSG, which employs a two-level strategy and a dynamic Feistel network to enhance the simple start-gap wear leveling with level-adjustable security assurance. The theoretical analysis and the experimental evaluation demonstrate the feasibility of RTA in terms of making an NVM fail and the robustness of security RBSG in terms of resisting RAA, BPA, and RTA.

References

Binkert N, Beckmann B, Black G, et al., 2011. The gem5 simulator. *ACM SIGARCH Comput Archit News*, 39(2):1-7. https://doi.org/10.1145/2024716.2024718

Bishnoi R, Ebrahimi M, Oboril F, et al., 2014. Asynchronous asymmetrical write termination (AAWT) for a low power STT-MRAM. Design, Automation & Test in Europe Conf & Exhibition, p.1-6. https://doi.org/10.7873/DATE.2014.193

Cho S, Lee H, 2009. Flip-N-write: a simple deterministic technique to improve PRAM write performance, energy and endurance. Proc 42nd Annual IEEE/ACM Int Symp on Microarchitecture, p.347-357. https://doi.org/10.1145/1669112.1669157

Chung H, Jeong BH, Min BJ, et al., 2011. A 58nm 1.8V 1Gb PRAM with 6.4MB/s program BW. IEEE Int Solid-State Circuits Conf, p.500-502. https://doi.org/10.1109/ISSCC.2011.5746415

Freitas RF, Wilcke WW, 2008. Storage-class memory: the next storage system technology. *IBM J Res Dev*, 52(4-5):439-447. https://doi.org/10.1147/rd.524.0439

Gal E, Toledo S, 2005. Algorithms and data structures for flash memories. *ACM Comput Surv*, 37(2):138-163. https://doi.org/10.1145/1089733.1089735

Gove D, 2007. CPU2006 working set size. *ACM SIGARCH Comput Archit News*, 35(1):90-96. https://doi.org/10.1145/1241601.1241619

Huai Y, 2008. Spin-transfer torque MRAM (STT-MRAM): challenges and prospects. *AAPPS Bull*, 18(6):33-40.

Huang F, Feng D, Xia W, et al., 2016. Security RBSG: protecting phase change memory with security-level adjustable dynamic mapping. IEEE Int Parallel and Distributed Processing Symp, p.1081-1090. https://doi.org/10.1109/IPDPS.2016.22

Kim YB, Lee SR, Lee D, et al., 2011. Bi-layered RRAM with unlimited endurance and extremely uniform switching. Symp on VLSI Technology, p.52-53.

Li Z, Wang F, Hua Y, et al., 2016. Exploiting more parallelism from write operations on PCM. Design, Automation & Test in Europe Conf & Exhibition, p.768-773. https://doi.org/10.3850/9783981537079_0099

Liddicoat AA, Flynn MJ, 2000. Parallel square and cube computations. Conf Record of the 34th Asilomar Conf on Signals, Systems and Computers, p.1325-1329. https://doi.org/10.1109/ACSSC.2000.911207

Menezes AJ, van Oorschot PC, Vanstone SA, 1996. Handbook of Applied Cryptography. CRC Press, Boca Raton, USA, p.683.

Micron Inc., 2011. Micron 128Mb P8P Parallel PCM Data Sheet.

Mittal S, Vetter JS, 2016. A survey of software techniques for using non-volatile memories for storage and main memory systems. *IEEE Trans Parall Distr Syst*, 27(5):1537-1550. https://doi.org/10.1109/TPDS.2015.2442980

Mittal S, Vetter JS, Li D, 2015. A survey of architectural approaches for managing embedded dram and non-volatile on-chip caches. *IEEE Trans Parall Distr Syst*, 26(6):1524-1537. https://doi.org/10.1109/TPDS.2014.2324563

Palangappa PM, Mohanram K, 2016. CompEx: compression-expansion coding for energy, latency, and lifetime improvements in MLC/TLC NVM. IEEE Int Symp on High Performance Computer Architecture, p.90-101. https://doi.org/10.1109/HPCA.2016.7446056

Qureshi MK, Karidis J, Franceschini M, et al., 2009. Enhancing lifetime and security of PCM-based main memory with start-gap wear leveling. Proc 42nd Annual IEEE/ACM Int Symp on Microarchitecture, p.14-23. https://doi.org/10.1145/1669112.1669117

Qureshi MK, Seznec A, Lastras LA, et al., 2011. Practical and secure PCM systems by online detection of malicious write streams. IEEE 17th Int Symp on High Performance Computer Architecture, p.478-489. https://doi.org/10.1109/HPCA.2011.5749753

Qureshi MK, Franceschini MM, Jagmohan A, et al., 2012. PreSET: improving performance of phase change memories by exploiting asymmetry in write times. *ACM SIGARCH Comput Archit News*, 40(3):380-391. https://doi.org/10.1145/2366231.2337203

Seznec A, 2009. Towards Phase Change Memory as a Secure Main Memory. Technical Report, No. RR-7088, INRIA, Campus Universitaire de Beaulieu, Rennes.

Woo NH, Seongand DH, Lee HS, 2010. Security refresh: prevent malicious wear-out and increase durability for phase-change memory with dynamically randomized address mapping. *ACM SIGARCH Comput Archit News*, 38(3):383-394. https://doi.org/10.1145/1816038.1816014

Yang BD, Lee JE, Kim JS, et al., 2007. A low power phase-change random access memory using a data-comparison write scheme. IEEE Int Symp on Circuits and Systems, p.3014-3017. https://doi.org/10.1109/ISCAS.2007.377981

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. https://doi.org/10.1109/TC.2012.292

Yun J, Lee S, Yoo S, 2012. Bloom filter-based dynamic wear leveling for phase-change RAM. Design, Automation & Test in Europe Conf & Exhibition, p.1513-1518. https://doi.org/10.1109/DATE.2012.6176713

Zhou P, Zhao B, Yang J, et al., 2009. A durable and energy efficient main memory using phase change memory technology. *ACM SIGARCH Comput Archit News*, p.14-23. https://doi.org/10.1145/1555815.1555759