2016 IEEE International Parallel and Distributed Processing Symposium

Security RBSG: Protecting Phase Change Memory with Security-Level Adjustable Dynamic Mapping

Fangting Huang, Dan Feng*, Wen Xia, Wen Zhou, Yucheng Zhang, Min Fu, Chuntao Jiang, Yukun Zhou
Wuhan National Laboratory for Optoelectronics
School of Computer, Huazhong University of Science and Technology, Wuhan, China
*Corresponding author: dfeng@hust.edu.cn

***Abstract*—As an emerging memory technology to build the future main memory systems, Phase Change Memory (PCM) can increase memory capacity in a cost-effective and power-efficient way. However, PCM is facing security threats for its limited write endurance: a malicious adversary could wear out the cells and cause the whole PCM system to fail within a short period of time. To address this issue, several wear-leveling schemes have been proposed to evenly distribute write traffic in a security-aware manner. In this work, we present a new type of timing attack named Remapping Timing Attack (RTA), based on the *asymmetry in write time* of PCM. Our analysis and experimental results show that the new revealed RTA can make two state-of-the-art wear-leveling schemes (Region Based Start-Gap and Security Refresh) lose effectiveness, failing PCM with these two techniques in several days (even minutes). In order to defend such attack, we further propose a novel wear-leveling scheme called Security Region Based Start-Gap (Security RBSG), which employs a two-stage strategy and uses 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 is the most robust wear-leveling methodology so far, which not only better defends the new RTA, but also performs well on the traditional malicious attacks, i.e., Repeated Address Attack and Birthday Paradox Attack.**

I. Introduction

There has been increasing demand on the main memory system to retain the working set of all the concurrently executing instruction streams. As Dynamic Random Access Memory (DRAM) has increasing energy and scalability issues [1], [2], Phase Change Memory (PCM) becomes a promising candidate for building the future main memory systems due to its advantages of high density, good scalability and zero leakage power [3]. According to the physical properties of PCM, each PCM cell is projected to endure a maximum of about 10^7 to 10^8 writes [4], which may be sufficient for a typical memory system. However, because of the non-uniform write traffic of real world applications, some memory lines written heavily could fail much faster than the others, causing the whole system to fail much earlier than its expected lifetime [5]. This significantly hinders PCM from being widely used as part of the main memory. To address this issue, *wear-leveling* schemes are introduced to make the writes distribute evenly by remapping the heavily written lines to the less ones.

Besides the uneven writes issue coming from general applications, another severe problem should be tackled down for PCM: some malicious adversaries might continuously write to a few physical lines and cause the whole PCM system to fail within a very short period of time. To deal with such security threat, many wear-leveling schemes have been researched. Qureshi et al. [5] first point out the security threat that an adversary can run malicious code to wear out a few lines of PCM memory. They propose Region Based Start-Gap (RBSG) wear-leveling scheme that divides the memory into several independent regions to prevent Repeated Address Attack (RAA). Further, Seznec et al. [6] find that RBSG is vulnerable to Birthday Paradox Attack (BPA) and propose a hierarchical table-based secure wear-leveling scheme. Seong et al. [7] demonstrate that most of the lifetime extending methods fail to prevent deliberately-crafted malicious attacks and propose Security Refresh (SR) wear-leveling scheme to further enhance security by dynamically remapping the logical address to the physical address using a randomly generated key.

In this paper, we present a new type of timing attack against PCM wear-leveling schemes by leveraging the phenomenon of *asymmetry in write time* of PCM, which is named Remapping Timing Attack (RTA). The idea behind RTA is based on two important facts: 1) wear-leveling schemes need to remap a logical address to a new location periodically and thus incur extra latency. 2) the write latency in PCM devices is asymmetric, i.e., the latency of SET operation (writing the bit '1') is about 8X longer than that of RESET operation (writing the bit '0') [8]. As a result, the latency incurred by remapping a line with data having all bits '1' is much higher than that by remapping a line with data having all bits '0'. Based on this observation, 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 two state-of-the-art wear-leveling schemes RBSG [5] and SR [7]. Compared with RAA, RTA is able to fail PCM in much less time since it precisely targets on a specified line or region.

To protect PCM against 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 Logical Address (LA) to Intermediate Address (IA) with a multi-stage Feistel Network [9] whose keys are changing dynamically to avoid address information leakage through timing attack, ensuring security with adjustable levels. Note that, by adding the number of Feistel Network stages, its keys will change before the key detection completes by RTA; 2) the inner-level wear leveling, after dividing the IA space to equal-sized sub-regions, transforms IA to Physical Address (PA) by the simple Start-Gap algorithm in each sub-region, which ensures the write traffic uniform with low overhead. Thus the proposed Security RBSG provides PCM with both enhanced lifetime and security.

The contributions of this paper are twofold:

1) We present a new type of timing attack RTA and demonstrate that RTA can wear out PCM devices within a short time when employing the latest wear-leveling approaches (i.e., RBSG and SR). To the best of our knowledge, this is the first study that reveals such kind of attack, which leverages the *asymmetry in write time* of PCM to maliciously attack PCM. The experimental results show that a PCM (with the default recommended configurations) adopting RBSG and SR schemes can fail 27435 and 322 times faster under RTA respectively, compared with that under RAA.
2) We propose a security-level adjustable wear-leveling scheme called Security RBSG to resist RTA and make an efficient trade-off between security assurance and overhead. The theoretical analysis and evaluation demonstrate that the proposed Security RBSG is more robust than the state-of-the-art wear-leveling schemes when facing RAA, BPA, and especially RTA.

The rest of this paper is organized as follows. In section II, we give the background and motivation. In section III, we describe the attack models of RTA against the latest RBSG and SR wear-leveling schemes and study their feasibility. In Section IV, we present the solution, i.e., the Security RBSG. We provide the evaluation results in Section V and conclude this paper in Section VI.

II. Background and Motivation

Phase Change Memory (PCM) is an emerging non-volatile memory (NVM) technology, which employs electrical resistivity to store data. PCM offers both good scalability and low power consumption for the massively used memories.

Unfortunately, the PCM devices can endure only about 10^7 to 10^8 writes per cell. Once a cell has been written more than the number of the write endurance, it would result in a stuck-at hard fault. Moreover, the non-uniform memory write pattern makes the situation worse because the PCM lifetime is determined by a few frequently written cells, which can dramatically reduce the endurance of a large array of memory cells. Wear leveling is a well-known, low-level mechanism to even the uniform writes from real world applications. Moreover, malicious attackers could generate write streams that focus on a few physical lines and make a PCM device fail much faster than its expected lifetime. Therefore the security-aware wear-leveling schemes (e.g., [5], [7]) have been proposed to address the durability and security issues of a PCM system simultaneously.

A. Wear-Leveling Schemes

Wear leveling is an efficient way to extend the lifetime of PCM 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, such as that used in translation layer for the flash [10]. The table based wear-leveling methods maintain an additional table to track the write counts of each memory line and swap hot memory lines with cold ones[11], [12], [13]. As table based wear-leveling incurs 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 the table[5], [7], [14]. By continuously updating the parameters of the algebraic mapping functions every certain number of normal writes, the writes are made uniform throughout the entire memory space.

However, extra write overhead is introduced during the remapping of memory, i.e, moving data from its old location to the new one. The parameter *Remapping Interval*, defined as the number of normal writes before triggering a remapping, can impact on the effectiveness of the wear-leveling algorithm as well as 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 higher write overhead (for practical purposes, the write overhead of wear-leveling algorithms is expected to be no more than 1% [5]).

B. Security Issues of Wear-Leveling

The limited write endurance of PCM not only affects lifetime reliability, but also incurs potential security threats. For example, a user who desires to get a product equipped with brand new PCM within the warranty period might intentionally accelerate the wear-out of PCM. Note that, both table based and algebraic mapping based wear-leveling schemes suffer from security problem. For table based methods, they are deterministic in nature so that the location of the mapped line can be guessed easily, and thus can be attacked easily. For algebraic mapping based wear-leveling, the vulnerability exists because the numbers of writes to cells are 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, which is a very efficient attack method for the baseline system without any wear leveling. Define the maximum number of writes to a line before that line gets moved by a wear-leveling algorithm as the Line Vulnerability Factor (LVF) [15]. 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 by RAA [5].
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, the LVF should be dozen times less than the endurance [6].
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 [7].

As we can see, RAA and BPA are implemented without any knowledge of the mapping between logical and physical addresses, while AIA is designed deliberately according to specific wear-leveling schemes, which could be more efficient. Though some security-aware wear-leveling schemes (e.g., [5],

[7]) have been proposed, we demonstrate that the vulnerability of these schemes exists by exploiting a new timing attack i.e., RTA in this paper.

C. Asymmetry in Write Time

Before elaborating on the new timing attack RTA, we firstly introduce the fact that RTA is based on: the write latency differs depending on different written data. In PCM, the state of a PCM device is changed by heating with different pulses. The duration of the pulse to SET the cell (i.e., writing the bit '1'), is typically about several times longer than that of the RESET (i.e., writing the bit '0') pulse [8], [16]. For READ operation, the data stored in the cell is retrieved by sensing the device resistance with a very low-power pulse. The read & write characteristics of PCM is depicted in Fig. 1. In this work, we assume that the latency of SET operation is 1000ns and the latency of RESET operation (as well as the READ operation) is 125ns as previous study[8].

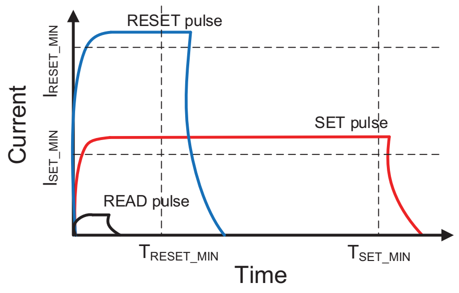

Fig. 1. PCM read & write characteristics.

Write to a memory line accomplishes when all bits of it have been written, which means that the duration of a write is determined by the worst-case write time of all the cells. Moreover, given that a memory line contains hundreds of bits, it is highly likely, when writing, that both RESET and SET transitions will occur, thus the write latency of different data is usually equal, i.e., the SET time (1000ns). While on the other hand, we could write artificial data to make the remapping latency different to gather side-channel information and detect *potential information leakage*. Specifically, remapping data whose value of bits are all '0' (denoted as ALL-0) to a memory line, is several times faster than remapping data whose value of bits are all '1' (denoted as ALL-1) to a memory line.

III. Vulnerability of Prior Wear-Leveling Schemes Based on Remapping Timing Attack

In this section, we describe the vulnerability of two state-of-the-art wear-leveling schemes (Region Based Start-Gap [5] and Security Refresh [7]) based on the proposed timing attack RTA in detail. RTA is able to detect the remapping latency since remapping halts other requests until it is completed thus incurs extra latency to the request which happens just following the remapping. We first introduce the basic ideas of the two schemes and then present how they are still vulnerable to RTA. In the following of this section, we assume that the operating system is compromised as other attacks [6], [7], in which only the malicious program is scheduled. Since the upper caches and buffers have been proved to be easily bypassed [7], we explain the attack models without considering any caches or buffers for simplicity.

A. Brief Introduction of Region Based Start-Gap

Region Based Start-Gap (RBSG) [5] first translates the Logical Address (LA) to the Intermediate Address (IA) by a static randomizer such as a *Feistel Network* [9] or a *Random Invertible Binary Matrix*, which eliminates the spatial locality of the memory write traffic. Note that 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 *GapLine*) that facilitates it to evenly wear out the entire 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 empty GapLine. The mapping of IA to the Physical Address (PA) of each region is managed independently by individual *Start* and *Gap* registers.

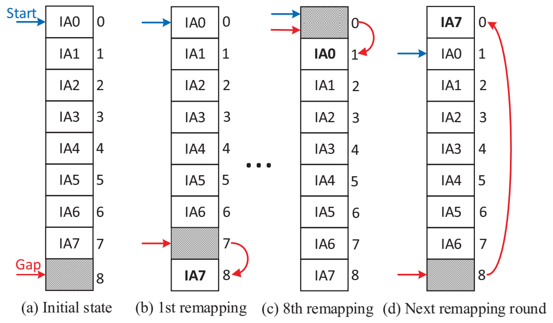

Fig. 2. An example of one Start-Gap remapping round.

To show how the algebraic mapping works, Fig. 2 illustrates an example of RBSG remapping round. Initially, the *Gap* register points to the extra GapLine and the *Start* register points to $PA0$ (Fig. 2(a)). Every certain number (which is fixed and manually configured) of writes to the region induces a remapping movement that copies (i.e., remapping) the content of the location $Gap - 1 \pmod 8$ to the location Gap (Fig. 2(b)). A remapping round accomplishes when all the lines of the region have performed a remapping movement (Fig. 2(c)). With more writes, such remapping round continues (Fig. 2(d)).

B. Remapping Timing Attack Model to RBSG

Based on the asymmetry in write time of PCM, we present the remapping timing attack model against RBSG. Assuming that there are N memory lines in the memory space and the whole memory space is divided into R regions. The remapping interval is denoted as ψ, i.e., every ψ writes incurs a remapping. The key idea of attacking RBSG is to find a sequence of logical addresses that mapped to *any* sequence of physical addresses which are in the same region and sequential in the physical address space, which will be further discussed in detail. 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} which 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. Assume that now L_i is just remapped to a new physical address (denoted

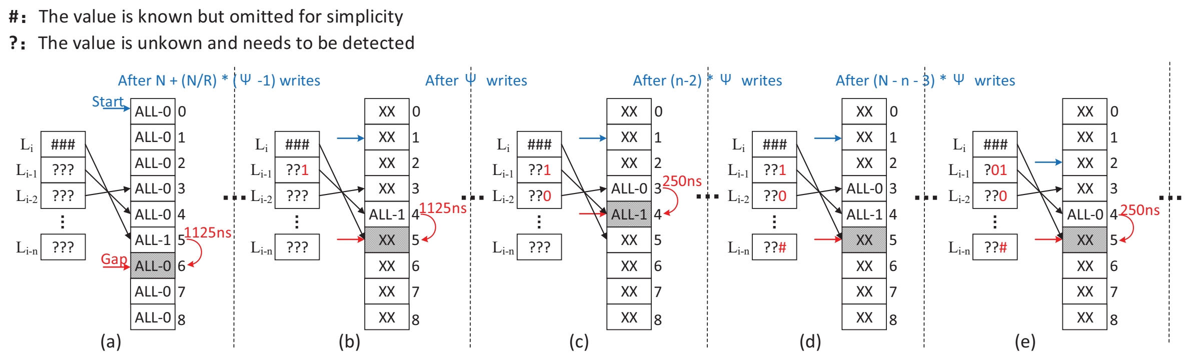
#: The value is known but omitted for simplicity
?: The value is unkown and needs to be detected
After N + (N/R) * (Ψ -1) writes
After Ψ writes
After (n-2) * Ψ writes
After (N - n - 3) * Ψ writes
Start
Gap
1125ns
250ns
(a)
(b)
(c)
(d)
(e)

Fig. 3. An example of the detection of the logical address sequence $L_{i-n}, \ldots, L_{i-1}$.

by P_n). With another $(N/R) \times \psi$ writes to the region, all lines of the region will be moved by one line and P_n is pointed by L_{i-1}. The goal is to wear out the physical address P_n, by writing to the logical addresses of the sequence $L_{i-n}, \ldots, L_i$ one by one after every $(N/R) \times \psi$ writes. We also note that, given the maximum endurance E of P_n, the length of the sequence n must be at least $\lceil \frac{E}{(N/R)\psi} \rceil$ to wear out P_n. Given the logical address L_i, we now explain how we detect the sequence of $L_{i-n}, \ldots, L_{i-1}$ via RTA as follows:

Firstly, we need to force the GapLine to be adjacent to the memory line mapped by L_i, i.e., $Gap + 1 = f(L_i)$, by detecting the latency incurred by migrating the data of L_i.

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

Step 2: Write ALL-1 to L_i. Therefore the data of all the physical lines are set to ALL-0 except L_i whose data is ALL-1.

Step 3: Keep writing ALL-1 to L_i until the detected extra remapping latency is about 1125ns, which indicates that L_i is just remapped to a new physical address as shown in Fig. 3(a). Recall that a remapping movement on the other lines requires just 250ns (time of a read and a RESET write) by copying ALL-0 to the GapLine while requiring 1125ns (time of a read and a SET write) on L_i by copying ALL-1 to the GapLine, as shown in Fig. 4(a).

Step 1 to Step 3 need performing only once to determine the location which Gap points to. After that, we can calculate the location of Gap by counting the number of writes.

Now detect the j^{th} bit of $L_{i-n}, \ldots, L_{i-1}$ (from Step 4 to Step 6).

Step 4: 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 fall into the region which L_i belongs to.

Step 5: Keep writing L_i with the content mentioned in Step 4 to increase the wear-leveling write counts of the region which L_i belongs to. Recalling that a logical address is remapped every $(N/R) \times \psi$ writes, with another $(N/R) \times (\psi - 1)$ writes perform, L_i is just remapped again. Keeping writing L_i with the same content ψ times, L_{i-1} is remapped at this moment. If the remapping time is about 250ns (time of a read and a RESET write), the j^{th} bit of L_{i-1} is 0, otherwise is 1 (Fig. 3 (b)). Repeat this step to deduce all the j^{th} bit of $L_{i-n}, \ldots, L_{i-1}$ as shown in Fig. 3(c).

Step 6: Repeat Step 4 and Step 5 to detect all the bits of $L_{i-n}, \ldots, L_{i-1}$, which requires $(N + (\psi - 1) \times N/R) \times \log_2 N$ writes.

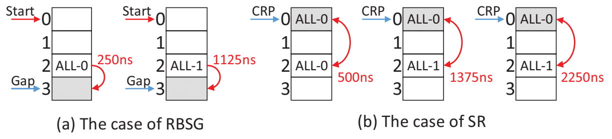

Fig. 4. Remapping latency of RBSG and SR.

As we can see, the proposed RTA is different from existing attack models, making it more dangerous. Compared with the attack model to RBSG based on bank-level parallelism [7], which is invalid when a region is located inside a bank, RTA is still valid under such circumstance. Moreover, compared with the attack model based on BPA [6] that is unpractical when increasing the rate of wear leveling by an online attack detector presented in [15], increasing the rate of wear leveling instead accelerates RTA. As proposed in the original RBSG paper [5], the *Delayed Write Policy* ensures that the attackers have to write more extra lines besides the line attacked, and consequently it takes more time to fail one memory time while we note that RTA is still efficient.

C. Brief Introduction of Security Refresh

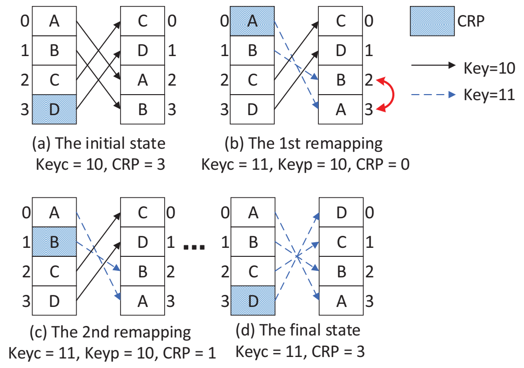

Fig. 5. An example of one Security Refresh remapping round.

Another wear-leveling method Security Refresh (SR) [7] dynamically remaps the memory lines according to a randomly generated key. Similar to RBSG, SR remaps one logical address every certain number of writes. The remapped PA of a LA is calculated by XORing with a random key. The candidate LA to be remapped is pointed by a register called Current Refresh Pointer (*CRP*). The CRP 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.

Fig. 5 depicts an example of one SR remapping round. Fig. 5(a) shows the initial state in which all the LAs have been mapped to the PAs with $key(10)$.

Fig. 5(b) displays the first remapping of the new round with $key(11)$. The new location of $LA0$ is 3, since $LA0(00)$ XOR $key(11)$ = $PA3(11)$. Hence, the content of location 2 should be moved to location 3 and the content of location 3 should be evicted to somewhere else. Fortunately, SR has an interesting pairwise property which makes dynamical remapping possible. Each LA has a pair denoted as $LApair$. We have LA XOR key_c = $LApair$ XOR key_p and $LApair$ XOR key_c = LA XOR key_p. That is, the new location for LA is actually the old location for $LApair$ and vice versa. Therefore, the remapping could perform by swapping line 2 and line 3.

For the second remapping shown in Fig. 5(c), the $LA1$ has been remapped by the previous remapping of $LA0$ and thus is not remapped again but simply increment the CRP. After several remapping movements, all LAs have been remapped and the next remapping round starts, as shown in Fig. 5(d).

Security Refresh is further improved to a hierarchical, two-level Security Refresh scheme. The outer-level SR remaps LAs to IAs, and then the IA space is divided into several equally sized sub-regions as that in RBSG. Each sub-region is managed by a inner-level SR to translate IAs to PAs. Both levels apply the SR scheme, but are transparent and independent to each other.

D. Remapping Timing Attack Model to One-level SR

We first present the attack model to one-level SR and then extend the model to attack two-level SR. Denote a logical address L_{i+1} = $paired(L_i)$ if L_{i+1} = L_i XOR key_c XOR key_p, where remapping L_i is just swapping the data of L_i and L_{i+1}. To compute L_{i+1} given L_i, we show how to compute $(key_c\ XOR\ key_p)$ via timing attack.

Firstly, to keep $(key_c\ XOR\ key_p)$ unchanged for a long time, we first force the CRP to point to the logical address 0x00, by detecting the latency incurred by remapping logical address 0x00.

Step 1: Write ALL-0 to all logical addresses. Therefore the data of all the physical lines are set to ALL-0;

Step 2: Keep writing ALL-1 to logical address 0x00 until detecting the remapping latency is $1350ns$. At that time, the logical address 0x00 is just remapped and the CRP is pointing to 0x01. From now on, $(key_c\ XOR\ key_p)$ keeps unchanged before another $(N/R-1)\times\psi$ writes.

The detection of CRP just performs once since the CRP position could be calculated by counting the number of writes.

Now detect the j^{th} bit of $(key_c\ XOR\ key_p)$ (from Step 3 to Step 5).

Step 3: Set all lines with the j^{th} bit of the logical address being '0' to ALL-0, otherwise set to ALL-1. Since only the lines whose corresponding logical address on j^{th} bit is different from $(j-1)^{th}$ bit need to be rewritten, it needs $N/2$ writes.

Step 4: Keep writing ALL-0 to the logical address 0x00 to increase the write counts of wear leveling. If the write latency is larger than the normal latency of writing ALL-0 ($125ns$), it suggests a remapping just happened. Recall the different latency of remapping different data as Fig. 4(a) shows. If the extra remapping latency is approximate to 500ns or 2250ns, the j^{th} bit of $(key_c\ XOR\ key_p)$ is 0 since the j^{th} bit of L_i is the same as L_{i+1}, otherwise is 1. To see this, recall that each remapping swaps two lines LA and $paired(LA)$. We have $LA\ XOR\ paired(LA) = key_c\ XOR\ key_p$ and thus

$$\begin{aligned}[key_c\ XOR\ key_p]_j &= [LA\ XOR\ paired(LA)]_j\\ &= [LA]_j\ XOR\ [paired(LA)]_j\end{aligned}$$

where $[\cdot]_j$ means 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 3. While for the worst case, another $N/2$ writes is needed to make a new remapping movement happen.

Step 5: Repeat Step 3 and Step 4 to detect all bits of $(key_c\ XOR\ key_p)$. It takes at least $(N/2)\times log_2(N)$ writes and at most $N\times log_2(N)$ writes as the length of address is $log_2(N)$.

From now on, we show how to wear out the physical line now pointed by L_i. The malicious process keep attacking L_i until CRP is larger than min(L_i, $paired(L_i)$), which indicates that the remapping just happens over L_i, and by then attacking $paired(L_i)$, since the target physical location is pointed by $paired(L_i)$ now. This attack continues until CRP reaches 0x00 again, which indicates the beginning of a new remapping rounds. We then detect the new $(key_c\ XOR\ key_p)$ and after that attack $paired\_new(paired\_old(L_i))$. Such procedure continues until the physical line fails.

E. Remapping Timing Attack Model to Two-level SR

For two-level Security Refresh, to keep track of the logical addresses remapped to the target physical line, we must get the knowledge of both keys of inner-level and out-level wear leveling every remapping round. It takes totally at most $(N/R)\times(2\times log_2N + log_2(N/R))$ writes to a sub-region to detect the keys of both levels with the steps mentioned above, which is usually more than the number of writes of a remapping round.

On the other hand, 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 sub-region practically. We keep track of the remapping of the target sub-region and then attack the whole sub-region until it fails. Now we analyze the feasibility of this attack method. Since the highest log_2R bits of a logical address indicates which sub-region it belongs to, the new logical address space remapped to the target sub-region could be calculate by XORing the high log_2R bits of previous logical address space of the sub-region and the $(key_c\ XOR\ key_p)$ of the outer-level SR, which takes at least $(N/2)\times log_2(R)$ writes and at most $N\times log_2(R)$ writes to be detected. After that, we could attack the target sub-region at least $N\times(\psi - log_2(R))$ times for each outer-level remapping round. The detection time is much less than a remapping round, since $2\times log_2(R)$ is usually 14 to 20, which is much smaller than ψ.

Those wear-leveling schemes which divide memory space into many sub-regions by the address sequence and perform wear leveling for each sub-region independently, suffer from the above detection attack as well. For example, for Multi-Way SR [14], it takes at most $(2N/R)\times log_2(R)$ writes to detect the remapping of the target sub-region and we can wear out the sub-region $(2N/R) \times (\psi - log_2(R))$ times before a new remapping round starts.

IV. Design of Security Region Based Start-Gap

So far, we have presented the vulnerability of prior PCM wear-leveling schemes via a completely new type of timing attack by exploiting the asymmetry in write time. Existing wear-leveling schemes, including the security-aware wear-leveling scheme Security Refresh, suffer from the deliberately crafted malicious attacks and fail to prevent address information leakage from PCM accesses. Wear-leveling schemes with high security level must ensure PCM lifetime comparable with the ideal lifetime even under the worst-case scenarios (including resisting all the potential malicious attacks known to be feasible, e.g., the RTA presented in this paper), which is critical for the application of PCM as main memory.

In this section, we propose a novel wear-leveling scheme named *Security Region Based Start-Gap (Security RBSG)* to address the vulnerability of malicious attacks, especially RTA presented above. Similar to RBSG, the proposed Security RBSG addresses PCM wear leveling with the Start-Gap mapping scheme. Furthermore, Security-Level Adjustable Dynamic Mapping is proposed to avoid useful information leakage through timing attack, by dynamically changing a Feistel Network mapping function, which enhances RBSG with adjustable security level.

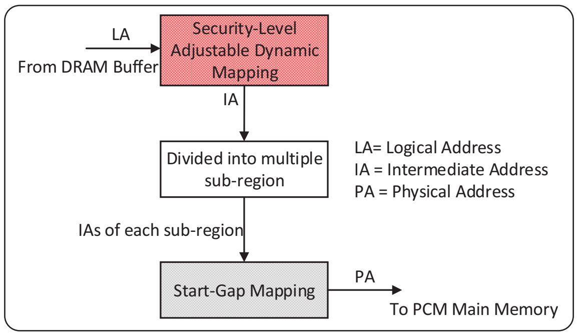

Fig. 6. Architecture of Security RBSG.

A. 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. 6 shows the architecture of the Security Region Based Start-Gap. Two-level dynamic mappings are adopted to do security-aware wear leveling, where the word *dynamic* here 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 LA to IA and is in charge of providing security assurance. Feistel Network [9] is a widely used one to one mapping method in cryptography. The details of dynamic Feistel Network mapping will be explained in the following subsection. We also note that the RBSG utilize a *static* Feistel network to facilitate address-space randomization, which is different from the proposed *dynamic* Feistel Network. To avoid Repeated Address Attack (RAA) and improve wear-leveling efficiency, IA space is then divided into multiple, fixed-size sub-regions by the sequence of IAs. Since the security has been ensured by the outer-level wear leveling, the 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 the Start-Gap mapping [5] (as described in Section III) for each sub-region separately to transform IA to PA. As described earlier, the Start-Gap mapping is with low overhead. Though its remapping rule is too simple to avoid address information leakage, the security has been ensured by the outer-level wear leveling.

B. Security Level Adjustable Dynamic Mapping

The outer-level wear leveling uses Feistel Network to transform LA to IA by an array of random keys. Feistel Network is simple to implement and has been studied extensively, which 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 Network.

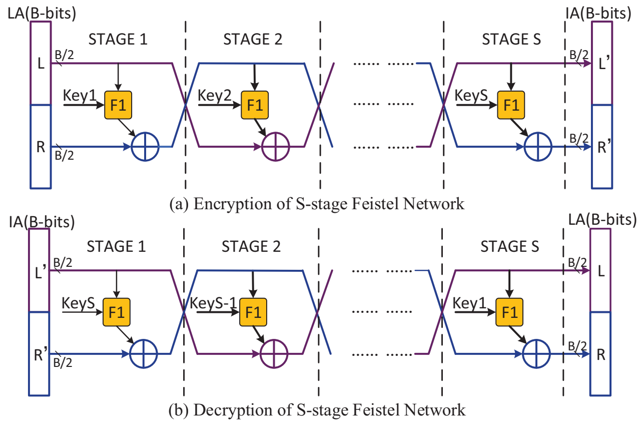

Fig. 7. Encryption and decryption of a Multi-stage Feistel Network.

Fig. 7(a) 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 which is split into two as well (L' and R'). R' is equal to L. L' is provided by an XOR operation of R, the output of a Round function (F1) on L, and some randomly chosen key K. We choose the Round Function to be the cubing function of (L XOR K) in this work for easy of implementation. Thus the transformation is given by:

$$L' = R \text{ XOR } (L \text{ XOR } K)^3$$

For the decryption of 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, as shown in Fig. 7(b).

Recall that RBSG uses a three-stage Feistel Network to randomize the address space with negligible storage / latency overhead, where theoretical work has shown that 3 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 is configured. In this work, we propose a *dynamic Feistel Network* (DFN, whose secret keys change every remapping round), which aims to not only

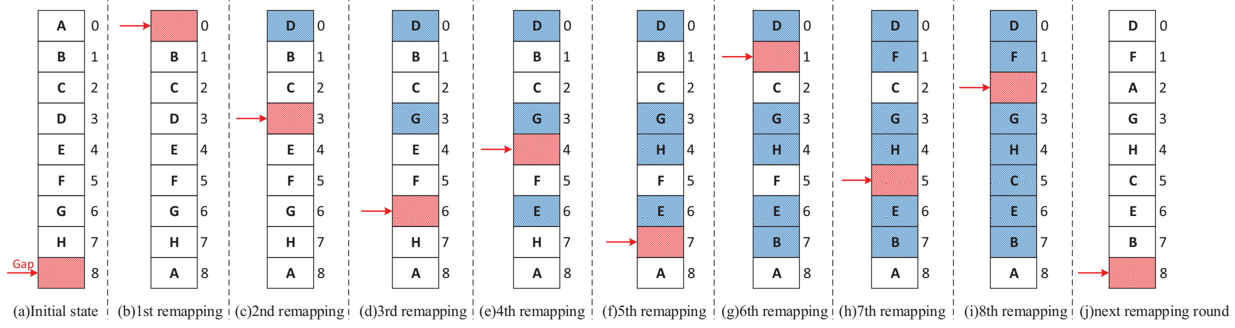

Fig. 8. An example of one complete Dynamic Feistel Network remapping round.

randomize the address space but more importantly provide security assurance.

To facilitate the remapping movement of DFN, an extra spare line, a register Gap and two register arrays K_c and K_p (the number of items is equal to the number of stages in DFN) are needed. The Gap register indicates the location of the 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 DFN by $ENC_{key}(\cdot)$ and $DEC_{key}(\cdot)$ respectively, where key can be K_p or K_c.

A complete remapping round of the proposed DFN on a memory system with 9 lines is shown in Fig. 8. Gap is highlighted with red color, while the blue color indicates that the corresponding memory lines have been remapped in the current round. At the beginning of a remapping round, the transformation of LA to IA is all given by $ENC_{K_p}(\cdot)$. A new secret key array K_c is then randomly generated. Line 8 is initially used as the extra spare line, which contains no data. The location of the spare line changes as the wear-leveling remapping goes on, which is always pointed by the Gap register.

First, the content of Line 0 is copied to Line 8, and Gap now points to Line 0. Then with every ψ writes, DFN will remap one line. Specifically, denoting LOC as the logical address which points to the Gap line under the current secret keys, i.e., $LOC = DEC_{Kc}(Gap)$, move the data of LOC (stored in Line $ENC_{K_p}(LOC)$) to the current Gap line and mark the previous line $ENC_{K_p}(LOC)$ as the new Gap line (by changing $GAP = ENC_{K_p}(LOC)$). Now we can see that the transformation of LOC is under the current secret keys, indicating that the remapping of LOC accomplishes. For example in Fig. 8(c), $LOC = D$ and $ENC_{Kc}(LOC) = 3$. Repeat the above remapping operation, except that after 7 movements, the resulting $ENC_{K_p}(LOC)$ is equal to 0. Noting that the old content of Line 0 has been moved to Line 8 in the first step, we need to move the content of Line 8 to Gap Line and let Gap point to Line 7. The above remapping movement is depicted in Fig. 9. It is not hard to see that, by repeating the above remapping, all the logical lines will be remapped to a new IA which is encrypted by the current secret keys. Therefore, at the end of the remapping round, the transformation of LA to IA will be given by $ENC_{K_c}(\cdot)$, instead of $ENC_{K_p}(\cdot)$. 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.

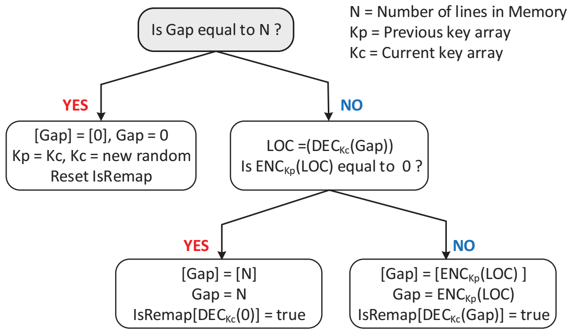

Fig. 9. Flowchart of DFN remapping movement.

The LA to be remapped is calculated in the running time. To facilitate address translation, each memory line needs an isRemap bit to mark if it has been remapped. Whenever 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. For each LA, if its corresponding isRemap bit is set (indicating that the LA has been remapped), use the current key array K_c to calculate its IA. If its corresponding isRemap bit is not set (indicating 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 Line 0 to the extra spare Line N at the beginning of every remapping round, if the resulting IA is 0 and the corresponding isRemap bit is not set, let IA be N if $Gap = 0$, otherwise be 0. The flowchart of calculating the mapping of LA to IA is given by Fig. 10.

Since the Round Function used in DFN is much more complex than the simple XOR, it takes much more writes to detect a single bit than that of Security Refresh. To say the least, suppose one bit of the secure key of DFN can be detected via N/R writes, as that of Security Refresh. To avoid the leakage of the secure key, the number of writes that required to detect the key should be larger than that of a remapping round. For a 1GB PCM bank with 256B 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 length of key array will make the detection fail, which is a 6-stage DFN in the 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 the security with the cost of

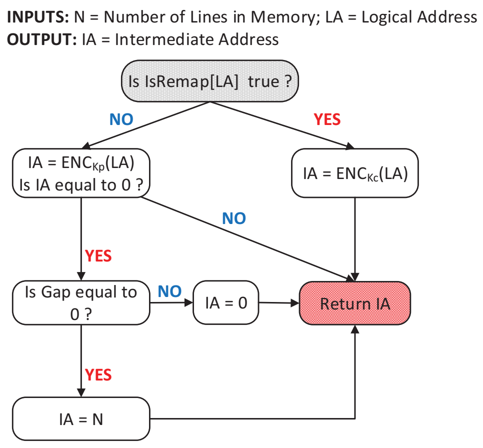

Fig. 10. Flowchart of DFN address translation.

addition key storage and operation circuit of each stage. Thus, we can make an efficient trade-off among security assurance and hardware / performance overhead by changing the number of stages of DFN. For a PCM configuration, we can estimate the security level beforehand and use a appropriate stages of DFN to prevent malicious attacks. How the number of stages affects the security will be discussed more in Section V.

V. Evaluation

In this section, we evaluate the efficiency of RTA against two state-of-the-art wear-leveling schemes (RBSG and two-level SR), as well as the robustness of Security RBSG against malicious attacks. We assume that: the total capacity is a 1GB memory bank; the PCM read time and write time is 125 ns and 1000 ns, respectively; the size of the block is the same as that of the last-level cache line (256B); the write endurance of storage cell is 10^8. Since the upper caches and buffers have been proved to be easily bypassed [7], the lifetime evaluations use a simple memory model which ignores caches and buffers.

A. *Efficiency of RTA against RBSG*

In this subsection, we evaluate the lifetime of PCM under RTA. For RBSG, to avoid RAA, the PCM bank should be divided into multiple regions and the number of lines in a region should be smaller than $Endurance/\psi$. Furthermore, to resist the BPA, there must be no more than $Endurance/(8 \times \psi)$ lines in a region. Thus, we vary the region number from 32 to 128 and the remapping interval from 16 to 100 (where 100 is recommended [5]). We make several observations from the evaluation as shown in the Fig. 11. First, the lifetime of RBSG under RTA decreases as the number of regions increases, since the less lines in a region, the less time it takes to detect the address sequence. Second, increasing remapping interval, which is an efficient way to prevent attacks such as RAA and BPA [15], makes PCM fail faster under RTA. Third, with the recommended configuration (32 regions and remapping interval as 100), RTA fails the PCM in 478 seconds, which is 27435X faster than RAA.

B. *Efficiency of RTA 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, inner-level remapping interval as 64, and outer-level remapping interval as 128. To measure the sensitivity of the

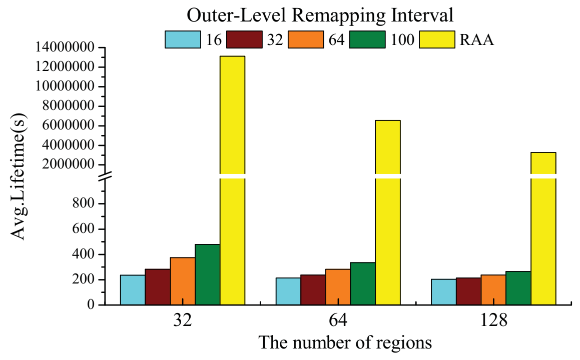

Fig. 11. The lifetime of RBSG under RTA and RAA.

PCM lifetime under RTA to the three parameters mentioned above, we vary the configurations as shown in Table I. As discussed in Section III, the key detection time varies from $(N/2) \times log_2(R)$ to $N \times log_2(R)$ with different outer-level wear-leveling keys. Therefore, for each configuration we run RTA five times, with each time randomly generating a key, and the average lifetime is used for evaluation.

TABLE I. Configuration of sub-region, 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

As shown in Fig. 12, we can draw the following observations. First, the lifetime of two-level SR under RTA decreases as the number of sub-regions increases, since there are less lines in the target sub-region to be attacked. Second, the lifetime decreases as outer-level remapping interval increases. As 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 wear out the target sub-region more times for each remapping round. Third, with the configuration suggested by SR, the lifetime of PCM is about 178.8 hours. Above all, the results are consistent with our previous theoretical analysis.

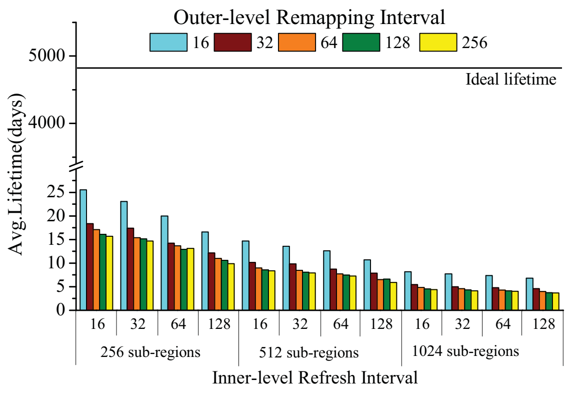

Fig. 12. Average lifetime of two-level SR under RTA.

To better evaluate the efficiency of RTA, RAA is also evaluated for comparison. For two-level SR, RAA has been proved to have the same effect with BPA [7]. As shown in Fig. 13, the lifetime of two-level SR under RAA is about 105 months, which is 322X longer than that under RTA.

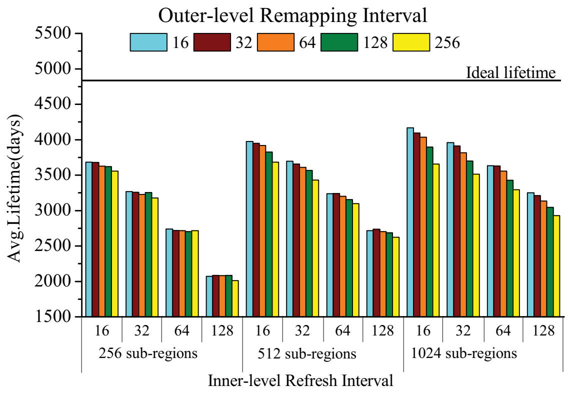

Fig. 13. Average lifetime of two-level SR under RAA.

C. Security Region Based Start-Gap

1) The Number of Stages: 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. As mentioned in Section IV, $K >= 6$ is capable to 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 SR and compare the lifetime of Security RBSG with that of two-level SR under RAA and BPA as well as the ideal lifetime. RBSG suggests that 3 stages is capable to randomize normal workload writes, but it can only achieve about 20% of the ideal lifetime under RAA. As shown in Fig. 14, to RAA, the lifetime of Security RBSG with 7 stages is 67.2% of the ideal lifetime and a little larger than the lifetime of two-level SR. To BPA, Security RBSG achieves 66.4% of the ideal lifetime. Note that BPA is insensitive to the number of stages since the attacked logical addresses are already chosen randomly. To ensure both security and lifetime with low overhead, we choose the number of stages as 7 in the following evaluations.

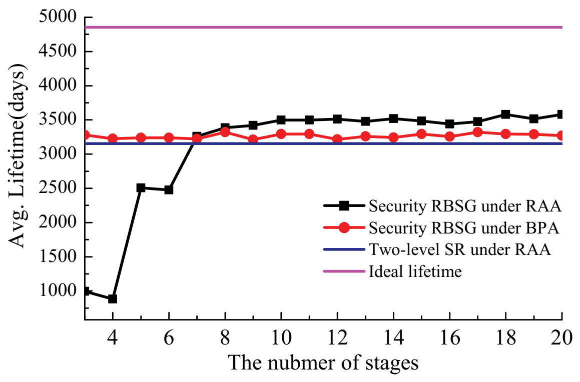

Fig. 14. Average lifetime of different DFN stages.

2) Wear-Leveling: We evaluate the lifetime of Security RBSG under RAA with the same configurations as described in Table I. From Fig. 15, we can see that the results shows similar characteristics as two-level SR. Increasing 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 outer-level remapping interval increases. This is because that Security RBSG adopts Start-Gap mapping for inner-level wear leveling, which makes writes from RAA distribute subsequently within a outer-level remapping round. Security RBSG could achieve comparable wear-leveling efficiency as two-level SR, in the same time providing higher security. With the recommended configuration, Security RBSB is able to endure more than 108 months.

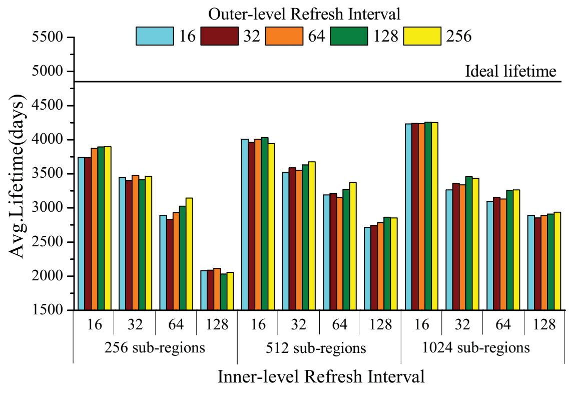

Fig. 15. Average lifetime of Security RBSG under RAA.

We also study how the writes generated by RAA are distributed across the memory space by Security RBSG. Fig. 16 shows the normalized accumulated number of writes for a given pinpointed physical address for 10^{10}, 10^{11}, 10^{12} and 10^{13} times with the recommended configuration. The y-axis of this chart plots the normalized accumulated number of writes across the memory addresses on the x-axis. Not surprisingly, as the number of writes increases, the writes are even better distributed. When the number of writes is increased to 10^{13}, the curve is approximate to linear, which means that the writes are distributed very evenly.

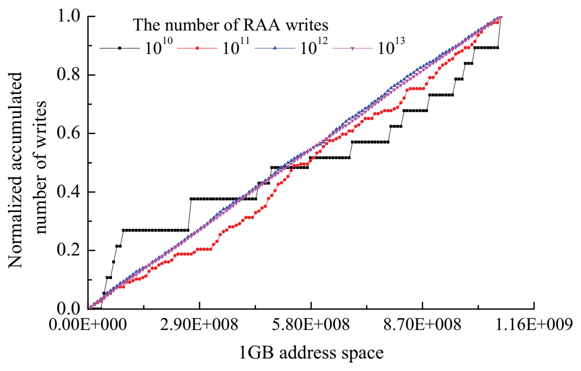

Fig. 16. Normalized accumulated number of write over the memory space.

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 B (i.e., $log_2(N)$) bit for Gap and $log_2(\psi)$ bit for write counter. For each stage, it requires B bit to store K_c and K_p. Assuming that there are S stages, totally $(S+1) \times B + log_2(\psi)$ bit register are needed. For the inner-level Start-Gap mapping, each sub-region requires $2 \times log_2(N/R)$ bits for $Start$ and Gap, and $log_2(\psi)$ bits for the write counter. Therefore, Security RBSG requires $(S+1) \times B + log_2(\psi) + R \times (2 \times log_2(N/R) + log2(\psi))$ bit register totally. For the recommended configuration, it costs about 2KB register for a 1GB bank. The outer-level and the inner-level wear leveling of each sub-region need an extra line, which is totally $(S+1) \times 256$ byte PCM. Each line needs a extra bit to indicate whether it has been remapped, which cost totally $log_2(N)$ bit (i.e., 0.5MB) SRAM.

We use the cubing function as the Round Function of Feistel Network. The cubing function can be considered as a multiplier after a squaring. Since squaring circuit requires approximately $0.5 \times B^2$ gates and multiplier circuit requires approximately B^2 gates [17], each cubing circuit requires about $(3/8) \times B^2$ gates. Therefore, Security RBSG with S stages requires $(3/8) \times S \times B^2$ gates.

4) Performance Impact: Finally, we evaluate the performance impact of the proposed Security RBSG using the Gem5 simulator [18]. In the experiment platform, the system consists of an 8-core processor (1 GHz) with a private 32KB L1 cache, a shared 256 KB L2 cache, and an 8 MB L3 DRAM cache. We use a queue with length of 32 and employ FR-FCFS scheduling scheme in the memory controller. Since each stage of the DFN costs 1 cycle and an access of SRAM takes 3 to 5 cycles, we assume that the address translation requires 10ns. We run 13 PARSEC benchmarks [19] and 27 SPEC CPU2006 benchmarks [20] and compare the IPC measure with the Baseline (without any wear-leveling schemes). The inner-level remapping interval of Security RBSG is 32/64/128, while the outer-level remapping interval is 128. For PARSEC benchmarks, the average IPC measure is decreased by 1.73%, 1.02%, and 0.68%, respectively. For SPEC CPU2006 benchmarks, the average performance degradation is all less than 0.5%. Some applications, such as *bzip2* and *gcc*, show no IPC degradation at all. This is because the memory accesses in these applications are relatively sparse and the remapping requests can be serviced during the idle periods. The results demonstrate that the performance impact of Security RBSG is arguably negligible.

VI. Conclusion

In this paper, we present RTA, a new type of timing attack that is based on the asymmetry in write time of PCM. RTA is shown to be more efficient in attacking the PCM with two state-of-the-art wear-leveling schemes (RBSG and SR). In order to resist RTA, we propose a novel wear-leveling scheme called Security RBSG, that employs a two-level strategy and dynamic Feistel Network to enhances the simple Start-Gap wear leveling with level-adjustable security assurance. The theoretical analysis and experimental evaluation demonstrate the feasibility of RTA in terms of failing a PCM and the robustness of Security RBSG in terms of resisting RAA, BPA, and especially RTA.

Acknowledgment

The work was partly supported by National High-tech R&D Program of China (863 Program) No. 2015AA015301; NSFC No. 61232004, 61502190, 61303046, 61173043, and 61402061; State Key Laboratory of Computer Architecture under Grant No. CARCH201505; Fundamental Research Funds for the Central Universities, HUST, under Grant No. 2014QNRC019.

References

[1] C. Lefurgy, K. Rajamani, F. Rawson, W. Felter, M. Kistler, and T. W. Keller, "Energy management for commercial servers," *Computer*, vol. 36, no. 12, pp. 39–48, 2003.

[2] S. Thoziyoor, J.-H. Ahn, M. Monchiero, J. Brockman, and N. Jouppi, "A comprehensive memory modeling tool and its application to the design and analysis of future memory hierarchies," in *Proc. ACM ISCA, 2008*.

[3] B. C. Lee, E. Ipek, and O. Mutlu, "Phase change memory architecture and the quest for scalability," *Communications of the ACM*, vol. 53, no. 7, pp. 99–106, 2010.

[4] R. F. Freitas and W. W. Wilcke, "Storage-class memory: The next storage system technology," *IBM Journal of Research and Development*, vol. 52, no. 4.5, pp. 439–447, 2008.

[5] M. K. Qureshi, J. Karidis, M. Franceschini, V. Srinivasan, L. Lastras, and B. Abali, "Enhancing lifetime and security of pcm-based main memory with start-gap wear leveling," in *Proc. ACM MICRO, 2009*.

[6] A. Seznec, "Towards phase change memory as a secure main memory," 2009.

[7] N. H. Seong, D. H. Woo, and H.-H. S. Lee, "Security refresh: prevent malicious wear-out and increase durability for phase-change memory with dynamically randomized address mapping," in *ACM SIGARCH computer architecture news*, vol. 38, no. 3. ACM, 2010, pp. 383–394.

[8] M. K. Qureshi, M. M. Franceschini, A. Jagmohan, and L. A. Lastras, "Preset: improving performance of phase change memories by exploiting asymmetry in write times," *ACM SIGARCH Computer Architecture News*, vol. 40, no. 3, pp. 380–391, 2012.

[9] A. J. Menezes, P. C. Van Oorschot, and S. A. Vanstone, *Handbook of applied cryptography*. CRC press, 1996.

[10] E. Gal and S. Toledo, "Algorithms and data structures for flash memories," *ACM Computing Surveys (CSUR)*, vol. 37, no. 2, pp. 138–163, 2005.

[11] P. Zhou, B. Zhao, J. Yang, and Y. Zhang, "A durable and energy efficient main memory using phase change memory technology," in *ACM SIGARCH Computer Architecture News*, vol. 37, no. 3. ACM, 2009, pp. 14–23.

[12] J. Dong, L. Zhang, Y. Han, Y. Wang, and X. Li, "Wear rate leveling: lifetime enhancement of pram with endurance variation," in *Proc. ACM DAC 2011*.

[13] J. Yun, S. Lee, and S. Yoo, "Bloom filter-based dynamic wear leveling for phase-change ram," in *Proc. IEEE DATE, 2012*.

[14] H. Yu and Y. Du, "Increasing endurance and security of phase-change memory with multi-way wear-leveling," *Computers, IEEE Transactions on*, vol. 63, no. 5, pp. 1157–1168, 2014.

[15] M. K. Qureshi, A. Seznec, L. Lastras, M. M. Franceschini *et al.*, "Practical and secure pcm systems by online detection of malicious write streams," in *Proc. IEEE HPCA, 2011*.

[16] J. Yue and Y. Zhu, "Accelerating write by exploiting pcm asymmetries," in *Proc. IEEE HPCA, 2013*.

[17] A. Liddicoat, M. J. Flynn *et al.*, "Parallel square and cube computations," in *Signals, Systems and Computers, 2000. Conference Record of the Thirty-Fourth Asilomar Conference on*, vol. 2. IEEE, 2000, pp. 1325–1329.

[18] N. Binkert, B. Beckmann, G. Black, S. K. Reinhardt, A. Saidi, A. Basu, J. Hestness, D. R. Hower, T. Krishna, S. Sardashti *et al.*, "The gem5 simulator," *ACM SIGARCH Computer Architecture News*, vol. 39, no. 2, pp. 1–7, 2011.

[19] C. Bienia, S. Kumar, J. P. Singh, and K. Li, "The parsec benchmark suite: Characterization and architectural implications," in *Proc. ACM PACT, 2008*.

[20] D. Gove, "Cpu2006 working set size," *ACM SIGARCH Computer Architecture News*, vol. 35, no. 1, pp. 90–96, 2007.