Protect Non-volatile Memory from Wear-out Attack based on Timing Difference of Row Buffer Hit/Miss

Haiyu Mao[†‡], Xian Zhang[§], Guangyu Sun[§] and Jiwu Shu[†‡*]
[†]Department of Computer Science and Technology, Tsinghua University, China
[‡]Tsinghua National Laboratory for Information Science and Technology, Beijing, China
[§]Center for Energy-efficient Computing and Applications, Peking University, China
[†‡]mhy15@mails.tsinghua.edu.cn, shujw@tsinghua.edu.cn, [§]{zhang.xian, gsun}@pku.edu.cn

***Abstract*—Non-volatile Memories (NVMs), such as PCM and ReRAM, have been widely proposed for future main memory design because of their low standby power, high storage density, fast access speed. However, these NVMs suffer from the write endurance problem. In order to prevent a malicious program from wearing out NVMs deliberately, researchers have proposed various wear-leveling methods, which remap logical addresses to physical addresses randomly and dynamically. However, we discover that side channel leakage based on NVM row buffer hit information can reveal details of address remappings. Consequently, it can be leveraged to side-step the wear-leveling. Our simulation shows that the proposed attack method in this paper can wear out a NVM within 137 seconds, even with the protection of state-of-the-art wear-leveling schemes. To counteract this attack, we further introduce an effective countermeasure named *Intra-Row Swap* (IRS) to hide the wear-leveling details. The basic idea is to enable an additional intra-row block swap when a new logical address is remapped to the memory row. Experiments demonstrate that IRS can secure NVMs with negligible timing/energy overhead, compared with previous works.**

I. Introduction

Non-volatile Memories (NVMs) have been considered as promising candidates for future main memories [1], [2], [3], [4]. NVMs are supposed to outperform traditional DRAM in standby power and storage density [5], [4]. In addition, NVMs demonstrate a comparable access speed to DRAM and are compatible to CMOS process technology [3]. However, these emerging NVMs all suffer from an endurance problem. Every NVM cell can sustain a very limited number of writes before the cell's worn-out [1], [2], [4]. Considering the nonuniform distribution of writes to NVMs, some cells may suffer concentrated writes and quickly break down [1], [6].

To mitigate the endurance problem, wear-leveling schemes are widely proposed [1], [6], [4]. In general, by dynamically remapping logical addresses (LA) to different physical addresses (PA) and swapping blocks, wear-leveling can uniformly allocate writes to different physical addresses to lengthen their lifetimes[6], [1]. Furthermore, inner states of wear-leveling should be kept confidential from the external due to security concerns. [2], [1], [6].

Unfortunately, a malicious program can still wear-out NVMs by exploring side channel leakage to reveal inner states of wear-leveling. Previous works have explored different sources of side channel leakage, such as bank conflict [2] and write asymmetry [7]. And corresponding countermeasures are proposed to mitigate the efficiency of attacks. For example, Seong *et. al.* have proposed a dynamic-remapping and bank-level wear-leveling scheme based on XOR operations [2]; Qureshi *et. al.* dynamically adjust the swap rate to mitigate malicious attack [8]; Huang *et. al.* obfuscate the remapping process using Feistel Network [7].

However, we observe that these existing approaches cannot protect NVMs from the attack based on side-channel leakages from memory row-buffer accesses. In this paper, we propose a novel wear-out attack based on the timing difference between row buffer hit and row buffer miss. Our attack is based on a simple fact: *Row buffer hit can reveal LAs mapped to the same physical row*. Thus, we can find out a group of LAs mapped to a certain physical row and keep detecting it. Then, we can figure out the new LA mapped to the PA written last, therefore crack the protection of wear-leveling.

To collapse the attack, *Intra-Row Swap* (IRS) is introduced. IRS can change the mappings of logical addresses to physical addresses in a row and hide actual physical addresses. Thus, attackers are obfuscated to determine the exact position of memory cells to keep attacking (writing). Our contributions are: 1. Introduce a novel wear-out attack based on the side channel leakage of NVM's row buffer hit/miss. 2. Evaluate the efficiency of our attack in various configurations. 3. Propose an efficient countermeasure named Intra-Row Swap and present comprehensive evaluation in performance/energy/lifetime overhead.

II. Preliminaries

This section introduces the background of life time problems of NVMs, heuristic wear-out attack proposed by previous works, and the row buffer structure in NVM main memories.

A. Wear-out Problem in NVMs

Due to the limitation of semiconductor fabrication and materials, NVMs all suffer from a wear-out problem. Every NVM cell can sustain a very limited number of writes [1], [4]. A general solution to wear-out problem is wear-leveling. In wear-leveling schemes, LAs are dynamically remapped to different PAs. Therefore, writes are dispatched to different cells, resulting in balanced wear-out intensity. During the dynamic remapping, a block swap occurs, which exchanges the bilateral locations of both blocks [1], [2], [7].

*Jiwu Shu is the corresponding author.

978-3-9815370-8-6/17/$31.00 ©2017 IEEE

One requirement for wear-leveling is that **the inner states of wear-leveling should remain unknown to the outside of the controller** [1], [7], such as which two blocks are being swapped. Unfortunately, malicious programs can still crack the protection of wear-leveling by mining the side channel leakage of memory accesses, which is described in the next subsection.

B. Wear-out Attack based on Side-channel Leakage

Side channel leakage may reveal details about LA-to-PA mappings in wear-leveling, which provides reference for wear-out attack. The most recent wear-out attack is Remapping Timing Attack proposed by Huang *et. al.* [7], which cracks the protection of security refresh wear-leveling [2] and start-gap wear-leveling [1]. To counteract Remapping Timing Attack, Security Region-based Start-gap (Security RBSG) is proposed. Block swap in Security RBSG is determined by Feistel Network. Every time, a LA is remapped to a random PA according to a periodically-changed mapping. Thus, it is difficult for attackers to infer the LA-to-PA mappings.

C. Row Buffer in NVMs

Row buffer is an essential component in DRAM or NVMs, which is integrated close to memory arrays to speed memory access [5]. Every time when a certain address is requested, memory controller first queries the row buffer for corresponding data. If data exist, row buffer will return the data immediately (i.e. buffer hit). Otherwise, data will be first loaded from the arrays to the row buffer along with other data in the same row, resulting in a long latency (i.e. buffer miss). In this paper, we adopt the structure and parameters of row buffer proposed in [5], and employ open-page strategy, following previous works[5], [9].

In the following section, we will propose heuristic wear-out attack based on the timing difference between row buffer hit and row buffer miss. Actually, different from Remapping Timing Attack, our attack flow is a general attack towards state-of-the-art wear-leveling schemes such as Start-gap, Security Refresh and Security RBSG. And our attack is efficient to all NVMs. Without loss of generosity, we focus our attack on PCM because endurance problem is the severest in PCM. For simplicity, we present our attack towards the security RBSG, which is the state-of-the-art secure wear-leveling scheme.

III. Wear-out Attack Based on Row Buffer Hit

In this section, we illustrate details of our wear-out attack. We first describe the attack model. Then, we present our attack flow. Last, we discuss some factors influencing our attack.

A. Attack Model

The attack model in this paper is similar to that in [2], [7]. We assume that a NVM main memory interacts with CPU, the attacker has the ability to compromise the OS, and the CPU cache can be turned off by the OS. In addition, states of memory bus can be captured by attackers. However, attackers cannot launch invasive attack [10] to physically probe the internal circuits of the NVM controllers. With these assumptions above, the goal of attackers is to wear-out one memory cell as soon as possible. To achieve this goal, the attacker send specific commands to NVMs and capture states of memory bus. By analyzing states, the attacker should infer LA-to-PA mappings, which indicate where to write. For the countermeasure against attacks, the goal is to lengthen the lifespan of every NVM cell as much as possible.

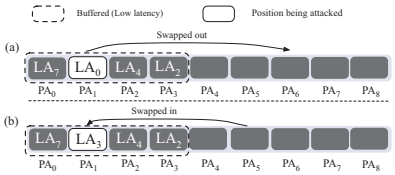

Fig. 1. Illustration of Row Buffer Hit Time Attack. (a) When LA_0 is swapped out, we will detect that it's no longer in this row. (b) When LA_3 is swapped in, we will detect that it's swapped with LA_0, then keep writing LA_3.

B. Row Buffer Hit Time Attack

Though prior wear-leveling schemes randomly remap LAs to PAs, attackers can easily detect **whether two LAs are mapped to the same physical row** by the timing difference between row buffer hits and row buffer misses. Then, by tracking the LAs mapped to this row, one can easily reveal which LA is newly mapped to this row and which LA is no longer in this row, because the LA-to-PA mappings are gradually changed [1], [2], [7] by swapping blocks. Thus, the attacker can focus on a certain LA to write. Once detecting that this LA is swapped out from the row, the attacker will focus on the LA' swapped in to write. This results in consistent writes to one PA and the block at PA will be worn out in the near future. We call this attack Row Buffer Hit Time Attack.

Figure 1 illustrates the attack flow. Symbols are listed in Table I. Now we present our attack flow, which aims to wear out the block at a certain physical address (it's PA_1 in Figure 1). There are four steps in Row Buffer Hit Time Attack:

- **Step 1:** Identify which LAs are in the same row of LA_0. To achieve this, read request sequences are sent to the NVMs: (LA_0, LA_1), (LA_0, LA_2),..., (LA_0, LA_{N-1}). If LA_0 and LA_k ($k \in \{1, 2, ..., N-1\}$) are in the same row, the read latency of LA_k should be equal to the latency of buffer hit. And if LA_k is not in the row containing LA_0, the read latency should be much longer, because of a buffer miss. After this step, we assume $\{LA_{i_0}(LA_0), LA_{i_1}, ..., LA_{i_{M-1}}\}$ are in the same row, which is called *Attacked Row Set (ARS)*. For example in Figure 1, the *ARS* is $\{LA_0, LA_2, LA_4, LA_7\}$.
- **Step 2:** Write consistently LA_0 until a block swap occurs. We focus on sending write requests about LA_0. After several writes, a block swap will be triggered for wear-leveling. Then, *ARS* is checked (with a similar method in Step 1). There are three cases:
  - Case-1: Swapping occurs outside *ARS*.
  - Case-2: LA_{i_k} ($k \in \{1, 2, ..., M-1\}$) is swapped out of *ARS*. Then, LA_{i_k} is updated as the incoming block address LA'_{i_k} in the *ARS*.

TABLE I
Symbols Used in This Work

Symbol	Description
N	Number of blocks in a NVM bank
M	Number of blocks in a NVM row buffer
LA_i	Logical address of memory blocks in a NVM
PA_i	Physical address of memory blocks in a NVM
LA_{i_k}	Logical address of memory blocks in the row of LA_0
LA'_{i_k}	Logical address swapped with LA_{i_k}
ARS	Logical address set $\{LA_0, LA_{i_1}, ..., LA_{i_{M-1}}\}$

- Case-3: LA_0 is swapped out of in *ARS*. Then, LA_0 is updated as the incoming block address LA'_0 in *ARS*. This case is shown in Figure 1, where LA'_0 is LA_3.

- **Step 3:** For Case-1 and Case-2 in Step 2, next time $LA' = LA_0$ will be written; For Case-3, next time $LA' = LA'_0$ (in the situation of Figure 1 is LA_3) will be written. This is because in Case-1 and Case-2, the mapping of LA_0 to PA_1 is not affected by the block swap, while in Case-3 LA_3 is remapped to PA_1.
- **Step 4:** Repeat Step-2 and Step-3 until PA_1 is worn-out.

In fact, there is an another case in Step 2: the block swap is in *ARS*. This case cannot be distinguished by the timing diagram of row buffer hit/miss. However, we discover that the possibility of this case is quite low in state-of-the-art wear-leveling schemesresults shown in Section V. Thus, this case is omitted in our attack flow.

IV. Intra-Row Swap

To counteract the Row Buffer Hit Time Attack, a straightforward method is to get rid of row buffer and eliminate the difference between buffer hits and buffer misses. However, this will introduce considerable overhead, which is demonstrated in the evaluation section.

As described in the last section, the block swaps in *ARS* cannot be detected by our attack, which disables tracing of logical address mapped to PA_1. Based on this observation, we propose Intra-Row Swap (IRS), an additional and compulsory swap after the block swap of original wear-leveling.

Figure 2 illustrates how IRS functions work after an original block swap. We suppose that the block at LA'_{i_k} (it's LA'_0 in Figure 2) is newly remapped into *ARS* in the original block swap. IRS is an additional swap that occurs right after the original swap. In IRS, the block at PA_i is swapped with block at PA_{M-1-i}. In other words, we swap two symmetric blocks in the row buffer.

To implement the IRS, Intra-Row Swap Vector (IRSV) is introduced. As shown in Figure 2, IRSV is a $\frac{M}{2}$-bit vector. The ith bit of it refers to that whether the ith block is swapped or not. After two blocks are swapped in IRS, corresponding bit reverses. For example, in Figure 2, the block at LA'_0 (mapped to PA_1) will swap with the block at LA_4 (mapped to PA_2), so the second bit of IRSV reverses. Note that, a block swap triggered by original wear-leveling does not affect the IRSV. To access one block corresponding to LA, original wear-leveling first transforms LA to "PA". Then, "PA" is transformed into the final actual PA according to the IRSV. Last, data at PA is returned. We can find that the hardware overhead of IRS is trivial, only $\frac{1}{4096}$ of NVM storage and negligible logic are required with configurations in Table II.

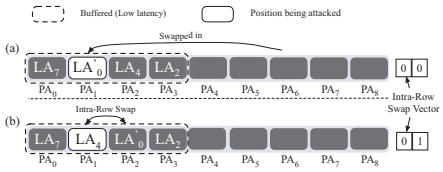

Fig. 2. Illustration of Intra-Row Swap

IRS can effectively obfuscate the attacker about new LA to write after last attacked LA is swapped. With the original wear-leveling, the LA-to-"PA" mapping is totally random. Thus, after IRS additionally swaps symmetric blocks in a row, the final LA to write remains random in the range of *ARS*. Thus, possible LAs mapping to a certain PA ramify after several block swaps, which severely mitigates the wear-out attack.

In fact, IRS does the wear-leveling based on both logical address and physical address. Though Zhou *et. al.* have proposed a wear-leveling method [3] based on physical address. But their method is unsafe when used alone and the time overhead is extremely high [2]. Though IRS introduces additional writes compared to a single wear-leveling scheme, combinational wear-leveling scheme still wear-out NVM uniformly. As described before, original wear-leveling ensures the randomness of LA-to-"PA" mappings. Thus, it is equivalent that IRS swaps two random LAs, which does not aggravate the non-uniformity of writes among the NVM blocks.

V. Evaluation

In this section, experimental setup, experiments on the lifetime of PCM under Row Buffer Hit Time Attack, experiments on the lifetime of PCM when Intra-Row Swap (IRS), and evaluations of the performance of IRS are presented.

A. Experiment Setup

A PCM main memory is employed for evaluation. Detailed parameters are listed in Table II. We first implement Security RBSG according to [7]. Then, we modify the wear-leveling flow to assist the combination of Security RBSG and Intra-Row Swapping. We evaluate performance with $gem5$ [11] connected to NVMain [12] and select 10 workloads from SPEC 2006 benchmark suite for simulation. We generate the latency and the power of IRS logic by synthesis tool [13].

We compare results of three memory systems with different designs, in respect of lifetime, performance and energy consumption. The baseline is a NVM main memory using Security RBSG only (labeled as *"Security RBSG"*), which is insecure under Row Buffer Hit Time attack. The second system is a straightforward countermeasure which directly gets rid of row buffer while Security RBSG is adopted (labeled as *"No Buffer"*). The third NVM main memory uses Security RBSG combined with IRS wear-leveling proposed in this work, which is labeled as *"IRS"*.

B. Attack evaluation

In Figure 3(a), we illustrate the average time to wear-out the PCM using Row Buffer Hit Time Attack. A two-level Security

TABLE II
Detailed Simulation Setup.

Processor Configuration
2GHz, Issue Width:8, Fetch Width:8, INT/FP FUs:8/8 LD/ST: 24/24, Branch penalty: 6 cycles ROB entries: 192, Fetch Q: 56, INT/FP registers: 128/128
Cache Configurations
DL1/IL1: 32/32KB, 2-way, 64B, R/W: 2/2-cycle, private L2: 1MB, 8-way, 64B, R/W: 10/10-cycle, share
Memory Configurations [7], [5]
8GB PCM, Security RBSG Wear leveling, 256 Byte per line 1GB per bank,1333MHz, read/write latency = 250/2000-cycle, Write Buffer: 16-entry, array read (write) = 2.47 (16.82) pJ/bit.
Row Buffer Configurations [5]
4KB row buffer, row buffer read/write = 0.93/1.02 pJ/bit FR-FCFS, Row hit/clean miss/dirty miss = 80/256/736 cycles. IRS control logic latency/IRSV access latency = 1/1-cycle

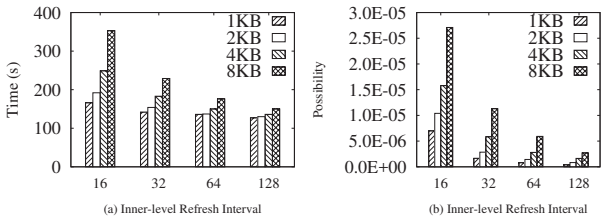

Fig. 3. (a)Attack Time (b)Intra-row Swap Possibility

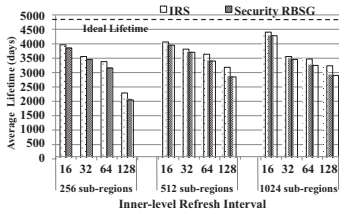

Fig. 4. Average Lifetime under Repeated Address Write

RBSG is adopted and the outer-level refresh interval is fixed at 128 [7]. The inner level refresh interval varies from 16 to 128. We can find that the lifetimes of PCMs are all below six minutes under attack. When the row buffer size increases or the inner-level refresh interval decreases, the Step 2 in attack flow costs more time, leading to an increase of attack time.

Moreover, the attack time is also influenced by the possibility that the attacked block swaps within the row. With higher possibility, more tries are required to successfully wear-out a line. With more blocks in the row buffer or higher frequency of block swaps, the possibility of swap between the attacked block and another block in the same row increases, which is shown in Figure 3(b). However, maximum of the possibility is 2.7×10^{-5}, which is negligible in practice.

We also evaluate the lifetime of different schemes in Figure 4. Due to the randomness of Security RBSG, we only investigate the lifetime of PCM under the Repeated Address Write, in which one logical address is consistently written but writes are actually distributed randomly over the memory bank [7]. "No Buffer" scheme does not change the LA-to-PA mappings. Thus, it has the same lifetime under attack and omitted in Figure 4. When IRS is combined with the Security RBSG, the lifetime increases because IRS introduces additional swaps to make writes more uniform [8]. To achieve the ten-year lifetime requirement [1], [2] with negligible overhead, we set the inner refresh level as 64 and the number of sub-regions as 512 for the rest of paper [7].

C. System-level evaluation

Figure 5 shows the total execution time normalized to that of Security RBSG. With respect to Security RBSG, an average of 2% timing overhead is introduced by IRS while 12% for "No Buffer" scheme. For $sjeng$, as much as 32% timing overhead is introduced in "No Buffer" scheme due to the decrease of row buffer hits. IRS introduces a very limited number of additional reads or writes becasue of the low block swap rate (one swap per sixty-four writes).

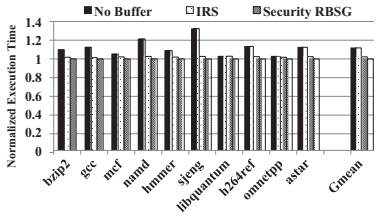

Fig. 5. Normalized execution time with different countermeasures

VI. Conclusion

In this paper, we observe that the side-channel leakage can introduce considerable vulnerability under heuristic wear-out attacks. We demonstrate this by introducing a novel wear-out attack towards NVMs, which explores the timing side channel leakage of row buffer hit. It can wear-out the PCM based NVM in 137 seconds even under the protection of state-of-the-art wear-leveling scheme. To overcome this problem, we propose so called Intra-Row Swap to hide the new logical address remapped to the physical address written last. Experiments demonstrate that Intra-Row Swap can ensure NVMs with enough lifetime under the aforementioned attack with negligible overhead in performance and energy consumption.

VII. Acknowledgments

This work is supported by National Natural Science Foundation of China (Grant No. 61232003, 61502266, 61572045), Beijing Municipal Science and Technology Commission of China (Grant No. D151100000815003), and China Postdoctoral Science Foundation (Grant No. 2016T90094, 2015M580098).

References

[1] M. K. Qureshi *et al.*, "Enhancing lifetime and security of pcm-based main memory with start-gap wear leveling," MICRO, pp. 14–23, ACM, 2009.
[2] N. H. Seong *et al.*, "Security refresh: prevent malicious wear-out and increase durability for phase-change memory with dynamically randomized address mapping," *ACM SIGARCH computer architecture news*, vol. 38, no. 3, pp. 383–394, 2010.
[3] P. Zhou *et al.*, "A durable and energy efficient main memory using phase change memory technology," ISCA, pp. 14–23, 2009.
[4] J. Wang *et al.*, "i2wap: Improving non-volatile cache lifetime by reducing inter- and intra-set write variations," HPCA, pp. 234–245, Feb 2013.
[5] H. Yoon *et al.*, "Row buffer locality aware caching policies for hybrid memories," ICCD, pp. 337–344, Sept 2012.
[6] J. Dong *et al.*, "Wear rate leveling: Lifetime enhancement of pram with endurance variation," DAC, (New York, NY, USA), pp. 972–977, ACM, 2011.
[7] F. Huang *et al.*, "Security rbsg: Protecting phase change memory with security-level adjustable dynamic mapping," IPDPS, pp. 1081–1090, May 2016.
[8] M. K. Qureshi *et al.*, "Practical and secure pcm systems by online detection of malicious write streams," HPCA, pp. 478–489, Feb 2011.
[9] A. Hassan *et al.*, "Software-managed energy-efficient hybrid dram/nvm main memory," Computing Frontiers, p. 23, ACM, 2015.
[10] O. Kömmerling *et al.*, "Design principles for tamper-resistant smartcard processors," WOST, pp. 2–2, USENIX Association, 1999.
[11] N. Binkert *et al.*, "The gem5 simulator," vol. 39, pp. 1–7, Aug. 2011.
[12] M. Poremba *et al.*, "Nvmain 2.0: A user-friendly memory simulator to model (non-)volatile memory systems," *IEEE Computer Architecture Letters*, vol. 14, pp. 140–143, July 2015.
[13] H. Bhatnagar, *Advanced ASIC Chip Synthesis: Using Synopsys® Design CompilerTM Physical CompilerTM and PrimeTime®*. Springer Science & Business Media, 2007.