Reliable Rowhammer Attack and Mitigation Based on Reverse Engineering Memory Address Mapping Algorithms

Saeyoung Oh and Jong Kim(✉)

Department of Computer Science and Engineering,
Pohang University of Science and Technology (POSTECH),
Pohang, Republic of Korea
{osy4997,jkim}@postech.ac.kr

Abstract. Rowhammer attacks intentionally induce bit flips to corrupt victim's data whose integrity must be guaranteed. To perform sophisticated rowhammer attacks, attackers need to repeatedly access the neighboring rows of target data. In DRAM, however, the physical addresses of neighboring rows are not always contiguous even if they are located before or after a target row. Hence, it is important to know the mapping algorithm which maps between physical addresses and physical row indexes not only for an attack but also for protection.

In this paper, we introduce a method to reverse engineer the exact mapping algorithm and demonstrate that the assumption in previous rowhammer work is faulty. In addition, we introduce a novel and efficient rowhammer method and improve existing mitigations that has a security hole caused by the faulty assumption. Finally, we evaluate the effectiveness of the proposed attack and show that the proposed mitigation almost perfectly defends against rowhammer attacks.

Keywords: Rowhammer bug · Reverse engineer · Memory address mapping

1 Introduction

In the last decades, DRAM has evolved dramatically. Researchers and manufacturers have devoted many efforts to increase the capacity of DRAM memory. As a consequence, the recent DRAM achieves high cell density, and thus, can hold large amounts of data. Despite the benefit, the development also brought side effects such as hardware faults which threaten data integrity.

A rowhammer attack is a representative attack abusing a disturbance error which is one of the high-density DRAM hardware faults. This attack corrupts target data by maximizing the effects of disturbance errors. In specific, the attacks repeatedly access the neighboring rows of the target row that contains the target data. Although rowhammer attacks exploit a hardware fault, the attacks are performed at the software level without direct access to the hardware.

One of the requirements for rowhammer attacks is that attackers need to access the neighboring rows of the target row in the same bank. However, the mapping information between the user-level locations (e.g., virtual addresses or physical addresses) and physical locations on DRAM (e.g., row and bank information) is not available publicly. Therefore, it is not straightforward to access the upper and lower neighboring rows in the same bank.

To find the mapping, previous work has reverse engineered the mapping algorithm between physical addresses and physical DRAM bank indexes [10,14]. Using the mapping algorithm for banks, it is possible to track down bank indexes with physical addresses. However, no method has been proposed to find the exact mapping algorithm for rows, although a part of the mapping algorithm is already revealed [4]. To defend against the rowhammer attacks, a mitigation also has to understand the mapping algorithm since it has to locate the exact neighboring rows. However, due to the difficulty of knowing row-mapping information, previous work has used a naïve assumption that DRAM rows are arranged identically to the sequence of physical addresses [1,3]. If the assumption is faulty, these mitigations will not work as the intended way.

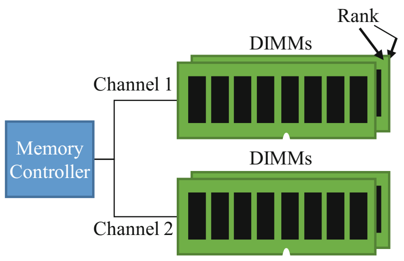
In this paper, we introduce a method to reverse engineer the mapping algorithm for rows. This method exploits the insight that rowhammer induces bit flips only on the neighboring rows. Based on this method, we demonstrate the invalidity of the commonly presumed assumption that physically contiguous rows have also contiguous physical addresses, and propose a novel and precise mapping algorithm-aware rowhammer attack. Also, we show that this invalid assumption causes a security hole in existing mitigation methods, and improve the existing mitigation by using our method to reverse engineer the mapping algorithm. Finally, we evaluate our attack and mitigation methods.

The contributions of our research are as follows:

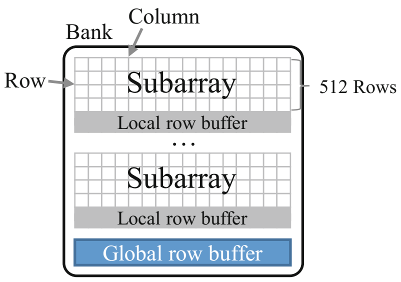
- We introduce a method to reverse engineer the DRAM row organization and discover the mapping algorithm for rows. To the best of our knowledge, no such method has been used by the previous work to reverse engineer the organization of DRAM modules.
- We propose a novel and precise rowhammer attack with exact mapping algorithm.
- We explain a security hole of existing mitigations and improve one of the mitigations using our exact mapping algorithm.

2 Background

In this section, we introduce DRAM organization and a rowhammer bug. In addition, we explain the prior work to reverse engineer the DRAM mapping algorithm.



(a) Memory controller, channels, DIMMs and ranks



(b) Subarrays within a bank

Fig. 1. DRAM organization.

2.1 DRAM Organization

DRAMs are hierarchically organized (Fig. 1a). Channels connect a memory controller to Dual Inline Memory Modules (DIMMs). Modern DIMMs can have at most eight ranks which denote sets of DRAM chips. Each rank has multiple banks and banks contain numerous cells that store the voltage representing logical data. All cells are connected horizontally and vertically through wordlines and bitlines. The horizontally and vertically connected cells are respectively called rows and columns (Fig. 1b).

The 2D array of cells is subdivided into several subarrays [7]. Each subarray is composed of 512 rows and has regularity, which indicates that the subarray internal organizations are same as each other. All the cells in one subarray are connected to one local row buffer, and all local row buffers are connected to one global row buffer (Fig. 1b). All row buffers store and amplify the voltage of DRAM data to a recognizable level.

The data on the DRAM cells are volatile because the voltage representing the data is leaked over time, leading to data loss. Therefore, the voltage is maintained by refreshing cells periodically before the voltage drops below the threshold (the indicator to determine whether the data is 0 or 1).

2.2 Rowhammer

Since DRAM chips have been compressed remarkably to increase the capacity within limited space, many hardware faults have emerged [2,5,6]. One of the hardware faults is a disturbance error which hazards data integrity due to interferences of adjacent cells.

Rowhammer is a method to intentionally induce the disturbance error. Rowhammer is performed by repeatedly and rapidly accessing a DRAM row (called an aggressor row). This process accelerates the effect of the disturbance error, that corrupts the neighboring rows (called target rows or victim rows) of the aggressor row.

Double-sided rowhammer, an effective method to induce bit flips, was introduced in [12]. Double-sided rowhammer repeatedly accesses both upper and lower neighboring rows of a target row, while the single-sided rowhammer repeatedly accesses only a single neighboring row to induce bit flips on the target row. To corrupt target data by using double-sided rowhammer (or single-sided rowhammer) exquisitely, it is necessary to know the user-level address (i.e., virtual address) of upper and lower neighboring rows in the same bank. Previous work [11,13] has introduced the methods to know the virtual addresses that correspond to the physical addresses of potential neighboring rows. However, since these methods are performed with the unproven assumption that contiguous rows have contiguous physical addresses, the potential neighboring rows may not be actual neighboring rows. Therefore, to precisely corrupt the target row, attackers need to know the exact physical addresses corresponding to the neighboring rows in the same bank.

2.3 Mapping Algorithm

To access data on DRAM module, the memory controller locates the data on DRAM by referring to mapping algorithm. The mapping algorithm determines the location of data on DRAM and is composed of several physical address bits. Since it must map the physical address to the hierarchies of DRAM, the algorithm consists of sub-mapping algorithms for channels, modules, ranks, banks, and rows.

The mapping algorithm for banks is reverse engineered in previous work [10,14]. By using a timing method which is related to row buffers, this work reveals that the mapping algorithm for banks is composed of XORed combinations of certain physical address bits. In contrast, the mapping algorithm for rows cannot be reverse engineered with the method which is used in the work. While it is possible to find which physical address bits compose the mapping algorithm for rows, it is impossible to identify how the composed bits of physical address determine the physical location of rows. This means that it is possible to distinguish whether the row of a certain physical address is different from the rowhammer target row in the same bank, but it is impossible to identify the exact upper and lower rows of the target row.

3 Reverse Engineering Mapping Algorithm

While the mapping algorithm for banks is reverse engineered, the mapping algorithm for rows cannot be reverse engineered using the timing method. Therefore, previous work [1,3,14] has naïve assumption that physical rows are arranged identically to physical addresses. According to the prior assumption, two rows are physically contiguous if the physical addresses of these rows are contiguous in the same bank.

Let r represent a row address composed of $\{b_0, b_1, \ldots, b_n\}$, b_i be the i^{th} row bit, where n is the number of bits on a logical row address. The row index function on prior assumption is:

$$LRow(r) = \sum_{i=0}^{n} 2^i b_i, \tag{1}$$

where r is a row address. These bits of a row address can be extracted from a physical address by using the previous methods [10,14]. We call this function (1) a logical row index function in this paper. Previous work has not demonstrated that the logical row index function is used in the real architecture.

The goal of this section is to find the real mapping algorithm. We can express the mapping algorithm as:

$$PRow(r) = \sum_{i=0}^{n} 2^i F_i(r) \tag{2}$$

We call this function (2) a physical row index function in this paper and the goal of this section is to find the $F_i(r)$ that is assumed to consist of physical address bits.

Before explaining our method in detail, we assume that 512 contiguous logical rows constitute one physical subarray and the mapping algorithm of one DRAM subarray also works for all DRAM subarrays. These assumptions seem reasonable due to the regularity of DRAM subarrays [5]. Therefore, we only focus on the lower 9 bits of a row address because each DRAM subarray consists of 512 ($=2^9$) rows.

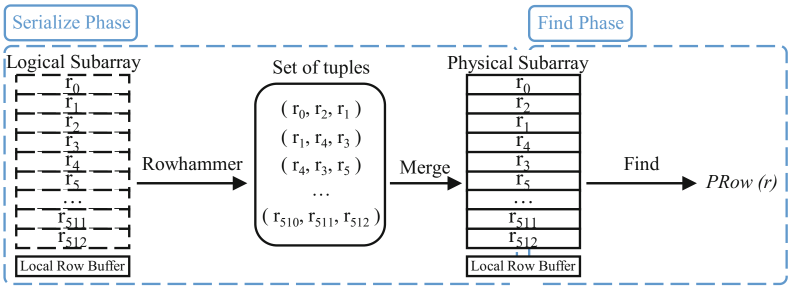


Fig. 2. Overall procedure of reverse engineering.

We reverse engineer the mapping algorithm in two steps. First, we arrange DRAM rows in sequential order by using rowhammer. We refer to this step as row serialization. Second, we manually find the mapping algorithm with the serialized rows, that is finding the function $F_i(r)$. The overall procedure of reverse engineering is shown in Fig. 2.

3.1 Row Serialization

In this subsection, we describe how to serialize rows of the single subarray in physical order to find the exact mapping algorithm. The key idea of row serialization is that *rowhammer induces bit flips only on the neighboring rows of an aggressor row.* We can identify the neighboring rows of aggressor rows by detecting the rows that have been corrupted by the rowhammer.

For convenience, we first define the following primitives:

- r_n is a row address which is composed of $\{b_0, b_1, b_2, ..., b_i\}$, where i is the number of bits on row address r_n.
- $sr_{n,m}$ is a row that is sandwiched between rows r_n and r_m.
- $Dhammer(r_n, r_m)$ returns the set of the rows that have been corrupted by double-sided rowhammer when r_n and r_m are aggressor rows.
- $Shammer(r)$ returns the set of the rows that have been corrupted by single-sided rowhammer when r is an aggressor row.

We perform double-sided rowhammer with all the combinations of two rows within a single DRAM subarray to find a row that is sandwiched between two rows. A single DRAM subarray consists of 512 rows, so the number of cases is at most $\binom{512}{2} = 130,816$. Thus, we argue that the time required to perform the rowhammer with two selected rows is feasible. We check which rows have been corrupted by double-sided rowhammer on the selected rows r_n and r_m. By this process, we can find the set of $Dhammer(r_n, r_m)$, which is related to $sr_{n,m}$. However, the set of $Dhammer(r_n, r_m)$ may contain rows that are adjacent to the other sides of the aggressor rows in addition to the sandwiched rows due to the effects of single-sided rowhammer. Therefore, we remove the effect of single-sided rowhammer by excluding the sets of $Shammer(r_n)$ and $Shammer(r_m)$ from $Dhammer(r_n, r_m)$ by performing additional single-sided rowhammer on r_m and r_n.

Ideally, $Dhammer(r_n, r_m) - Shammer(r_n) - Shammer(r_m)$ must have only one row $sr_{n,m}$ because only one sandwiched row can be between two aggressor rows. Therefore, except for two boundary rows (the 0th row and the 512th row) which are not affected by double-sided rowhammer, 510 tuples $(r_n, sr_{n,m}, r_m)$ can be obtained within a single DRAM subarray. This tuple indicates a sequence of physically contiguous rows. After obtaining the tuple, we can infer overall row arrangements due to the subarray regularity.

With the tuples, we can reconstruct a subarray by using the following rules:

- If two tuples, $(r_{n_1}, sr_{n_1,m_1}, r_{m_1})$ and $(r_{n_2}, sr_{n_2,m_2}, r_{m_2})$ exist such that $sr_{n_1,m_1} = r_{n_2}$ and $r_{m_1} = sr_{n_2,m_2}$, then the two tuples can be merged into $(r_{n_1}, r_{n_2}, r_{m_1}, r_{m_2})$.
- If two tuples, $(r_{n_3}, sr_{n_3,m_3}, r_{m_3})$ and $(r_{n_4}, sr_{n_4,m_4}, r_{m_4})$ exist such that $r_{m_3} = r_{n_4}$, then the two tuples can be merged into $(r_{n_3}, sr_{n_3,m_3}, r_{m_3}, sr_{n_4,m_4}, r_{m_4})$.

Using this method, we can ideally merge all of the tuples into one sequentially ordered tuple, that is, the physical sequence of 512 rows within a single DRAM subarray.

The success of row serialization depends on the number of corrupted rows after performing double-sided rowhammer. However, some rows might not be corrupted. To resolve this problem, we focus on the subarray regularity, which means that every subarray is organized identically. Due to this regularity, undiscovered tuples in one subarray can be obtained from another subarray.

3.2 Finding Mapping Algorithm

Next, we use the serialized rows to find the mapping algorithm of physical row indexes $PRow(r_n)$ (or $F_i(r)$). As we mentioned, the input of this function (2) is a row address and the output is a physical row index. We manually find the function $PRow(r)$ by exploiting the insight that the discovered tuple $(r_n, sr_{n,m}, r_m)$ implies $PRow(r_n) + 1 = PRow(sr_{n,m})$ and $PRow(r_m) - 1 = PRow(sr_{n,m})$, which means that the rows of the tuple are sequentially contiguous.

3.3 Row Address Mapping Schemes

We tested six DDR3 modules of three major DRAM manufacturers with this method to reverse engineer the mapping algorithm (Table 1). By using this method, we discovered two mapping algorithm schemes, which are address mirroring and row twisting. As a result, we demonstrate that the prior assumption is faulty.

Address Mirroring. The first scheme is observed only on one rank[1] of two-rank DRAM modules for all of the experimented manufacturers. The other rank is subject to either the logical row index function or row twisting. The physical row index function is (we mark the components which are different from the logical row index function in bold):

i	0	1	2	3	4	5	6	7	8
$F_i(r)$	b_0	b_1	b_2	$\mathbf{b_4}$	$\mathbf{b_3}$	$\mathbf{b_6}$	$\mathbf{b_5}$	$\mathbf{b_8}$	$\mathbf{b_7}$

JEDEC [4] documents this scheme, called address mirroring that is deployed to increase throughput by cross-wiring some wires on one rank. We can identify that our discovered physical row index is address mirroring because the index of address mirroring in the JEDEC standard is exactly identical with ours. The sameness implies that our method to reverse engineer is valid to find the exact physical address.

[1] In fact, it is impossible to distinguish the rank bits and bank bits from the existing reverse engineering method. However, since we can track the difference of mapping algorithm for each rank, we can infer which bit of the bank bits is a rank bit.

Row Twisting. The second scheme is observed only on DRAMs from A manufacturer, and we call the scheme row twisting. The physical row index function is (we mark the components which are different from the logical row index function in bold):

i	0	1	2	3	4	5	6	7	8
$F_i(r)$	b_0	$\mathbf{b_1 \oplus b_3}$	$\mathbf{b_2 \oplus b_3}$	b_3	b_4	b_5	b_6	b_7	b_8

According to our experimental result, the row twisting function is applied after address mirroring. For example, when a module is affected by both row twisting and address mirroring, the bit swap of b_3 and b_4 is first performed (due to address mirroring) and the swapped b_3 (originally b_4) is XORed with b_2 and b_1 later (due to row twisting). Therefore, we infer that row twisting is an internal mechanism of DRAM chip unlike address mirroring which is a mechanism at a circuit level. We also infer that row twisting is somehow related to twisted wordline schemes [8,9].

Table 1. Address mirroring and row twisting on DDR3 modules of three major manufacturers.

Tag	Manufacturer	Model	# of ranks	Mirroring	Twisting
A	SAMSUNG	M378B5273DH0-CK0	2	✓	✓
B		M378B5273DH0-CH9	2	✓	✓
C		M378B5173QH0-CK0	1		✓
D	HYNIX	HMT351U6CFR8C-PB	2	✓	
E		HMT351U6CFR8C-H9	2	✓	
F	MICRON	MT16JTF51264AZ-1G6M1*	2	✓	

*In this case, there are not enough bit flips to reverse engineer the mapping algorithm. However, since some bit flips are induced in rowhammer experiments with only address mirroring, we infer that this module is affected only by address mirroring.

4 Mapping Algorithm-Aware Rowhammer

If attackers know the exact mapping algorithm for rows, the attackers can perform more effective rowhammer attack by inducing more exploitable bit flips. In addition, attackers can exquisitely perform rowhammer attacks because the attackers can locate the exact upper/lower neighboring rows of the target row. We refer to this method as mapping algorithm-aware rowhammer.

We give an example of rows that are vulnerable to rowhammer attacks (Fig. 3). For simplicity, we assume that the DRAM module is affected by only address mirroring. When the faulty assumption is applied, the upper row is Row_b

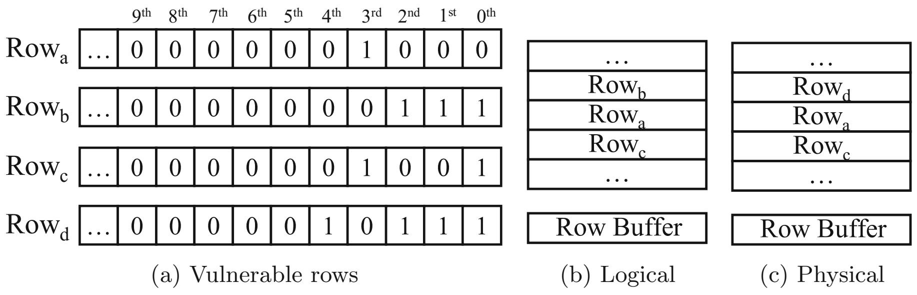

9th 8th 7th 6th 5th 4th 3rd 2nd 1st 0th
Rowa … 0 0 0 0 0 0 1 0 0 0
Rowb … 0 0 0 0 0 0 0 1 1 1
Rowc … 0 0 0 0 0 0 1 0 0 1
Rowd … 0 0 0 0 0 1 0 1 1 1
…
Rowb
Rowa
Rowc
…
Row Buffer
…
Rowd
Rowa
Rowc
…
Row Buffer
(a) Vulnerable rows
(b) Logical
(c) Physical

Fig. 3. Vulnerable rows on address mirroring. The upper row in logical index (b) and the upper row in physical index (c) is different about Row_a.

and the lower row is Row_c with respect to Row_a (Fig. 3b). However, the upper row of Row_a in address mirroring is actually Row_d, not Row_b (Fig. 3c). The logical row indexes of Row_a, Row_b, Row_c and Row_d are 8, 7, 9 and 23, but their physical row indexes are 16, 7, 17 and 15, respectively.

We evaluate the effectiveness of mapping algorithm-aware rowhammer. To evaluate the effectiveness, we measured the number of bit flips that were caused by double-sided rowhammer. We exploited three kinds of double-sided rowhammer:

(i) R1 is performed with the assumption that physically contiguous rows have contiguous physical addresses.
(ii) R2 is performed with the assumption that row indexes are affected only by address mirroring.
(iii) R3 is performed with the assumption that row indexes are affected by both address mirroring and row twisting.

We performed these three types of rowhammer on the memory space that is different from the subarray used in the experiment of Table 1. We tested on two modules: one was inferred to be affected only by address mirroring (module D); the other was inferred to be affected by both address mirroring and row twisting (module A) by our proposed method. We measured the number of bit flips on two processors, Sandy Bridge (i7-2600) and Haswell (i5-4460).

Although the prior assumption is faulty, R1 showed considerable bit flips (Fig. 4). The abnormal result is because some physical rows are sequential like logical rows, or single-sided rowhammer has shown an effect even if the attack uses double-sided rowhammer. However, module A shows the largest number of bit flips by R3 (25% more bit flips than R1 on average), and module D shows the largest number of bit flips by R2 (12% more bit flips than R1 on average). This experimental result shows that the proposed attack method, mapping algorithm-aware rowhammer, is more efficient than the conventional method regardless of processor types.

Also, this experimental result validates that our discovered mapping algorithm is correct.

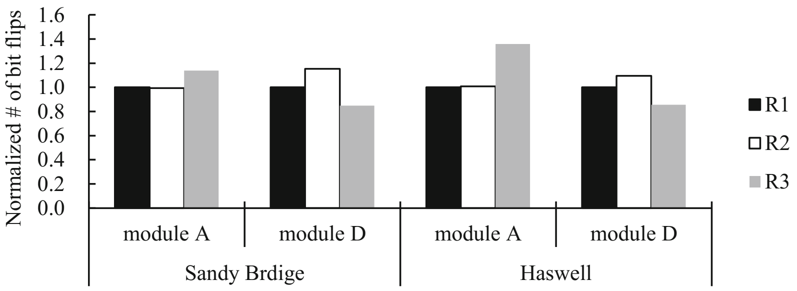


Fig. 4. Normalized number of bit flips. The number of bit flips for each case is normalized to R1.

5 Improved Mitigation

In this section, we explain existing mitigations and how the mitigations can be attacked by rowhammer attacks. Then, we improve Anvil, one of the existing mitigation methods, and evaluate its effectiveness as a proof-of-concept.

5.1 Existing Mitigations and a Security Hole

We study two representative mitigations, Anvil [1] and G-CATT [3], and explain how they are affected negatively by the faulty assumption using the aforementioned example of the vulnerable row (Fig. 3).

Anvil. Anvil defends rowhammer attacks by refreshing potential victim rows when the signs of rowhammer attacks are detected. The potential victim rows are the neighboring rows of the detected aggressor rows and thus Anvil refreshes the presumed neighboring rows.

However, according to the faulty assumption of Anvil, Anvil regards Row_b and Row_c as potential victim rows when Row_a is the aggressor row (Fig. 3b). Therefore, if attackers repeatedly access the aggressor row Row_a, bit flips may occur on Row_d, which was not refreshed by Anvil. As a result, Anvil can detect rowhammer attacks but cannot refresh all the victim rows.

G-CATT. G-CATT defends rowhammer attacks by putting a row between rows of different security domains (e.g., kernel and user) to physically separate the rows. Since rowhammer attacks only corrupt rows that are adjacent to aggressor rows, attackers cannot corrupt victim memory on G-CATT.

However, according to the faulty assumption of G-CATT, G-CATT expects that Row_a and Row_d are already separated (Fig. 3b). Therefore, if an attacker process is allocated in Row_d, Row_a is vulnerable to rowhammer attacks.

5.2 Method to Improve Existing Mitigations

To perfectly defend against rowhammer attacks, the existing mitigations must be improved with the proper physical row indexes. There are two methods to improve the existing mitigations with our discovered mapping algorithm.

The first method regards all possible neighboring rows of each and every case of the mapping algorithm schemes as the actual neighboring rows. This method takes additional runtime overhead to manage additional rows but our method to find the mapping algorithm is not required. The second method finds the mapping algorithm by using our proposed method before performing the existing mitigations. This method is only possible when the victim's DRAM modules are vulnerable enough to be able to be reverse engineered and initial time to find the mapping algorithm. However, this method does not need an additional runtime overhead compared to the first method.

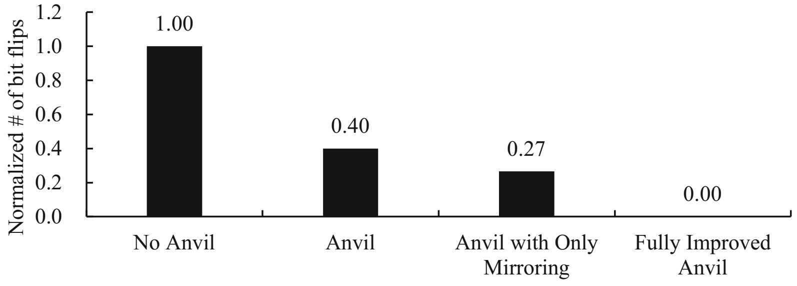


Fig. 5. Normalized number of bit flips with module A on Sandy Bridge.

5.3 Evaluation

We measured the bit flips by rowhammer on module A to evaluate the effectiveness of the improved mitigation that considers the exact mapping algorithm schemes. First, to show the security hole of existing mitigations, we performed single-sided rowhammer on Anvil with only the mapping algorithm for banks. Next, we improved Anvil into two cases: one is improved by considering only address mirroring, and the other is improved by considering both address mirroring and row twisting. We perform single-sided rowhammer again on those two improved Anvil methods (Fig. 5).

Anvil and Anvil with only mirroring might refresh actual victim rows because some refreshed rows are actual victim rows in spite of the faulty information about the neighboring rows. However, not all potential victim rows are actual victim rows, and thus Anvil and Anvil with only mirroring cannot refresh all of the actual victim rows. Fully improved Anvil, which is modified by using our mapping algorithm, shows no bit flips. This result shows that the fully improved mitigation can effectively defend against rowhammer attacks. In addition, we believe that G-CATT also properly prevents rowhammer attacks if the exact mapping algorithm is applied to G-CATT.

6 Conclusion

We introduced a method to reverse engineer the mapping algorithm for rows and revealed an exact mapping algorithm for rows. This method uses the feature that rowhammer induces bit flips only on the neighboring rows of aggressor rows. Using this method, we can infer the exact row arrangement. As a result, we demonstrate that previous work has faulty assumption that physically contiguous rows have also contiguous physical addresses.

Based on the exact physical row index, we can induce bit flips more efficiently than the conventional rowhammer method which does not consider the exact physical row index. Note that more bit flips make the rowhammer attack more successful because there are more candidates of exploitable bit flips. Also, we have shown that it is possible to make bit flips in the system with rowhammer mitigation methods if they were used with the faulty assumption. Finally, we have improved the existing mitigation and showed that the improved mitigation perfectly protects against rowhammer attacks.

Acknowledgement. This work was supported by the National Research Foundation of Korea (NRF) grant funded by the Korea government (MSIP) (No. 2017R1A2B4010914).

References

1. Aweke, Z.B., et al.: ANVIL: software-based protection against next-generation rowhammer attacks. ACM SIGPLAN Not. **51**(4), 743–755 (2016)
2. Baumann, R.: The impact of technology scaling on soft error rate performance and limits to the efficacy of error correction. In: International Electron Devices Meeting, IEDM 2002, pp. 329–332. IEEE (2002)
3. Brasser, F., Davi, L., Gens, D., Liebchen, C., Sadeghi, A.R.: Can't touch this: software-only mitigation against rowhammer attacks targeting kernel memory. In: Proceedings of the 26th USENIX Security Symposium (Security), Vancouver, BC, Canada (2017)
4. JEDEC: DDR3 SDRAM Unbuffered DIMM Design Specification, rev. 1.06 (2013)
5. Khan, S., Lee, D., Mutlu, O.: Parbor: an efficient system-level technique to detect data-dependent failures in dram. In: 2016 46th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN), pp. 239–250. IEEE (2016)
6. Kim, Y., et al.: Flipping bits in memory without accessing them: an experimental study of dram disturbance errors. In: 2014 ACM/IEEE 41st International Symposium on Computer Architecture (ISCA), pp. 361–372, June 2014
7. Kim, Y., Seshadri, V., Lee, D., Liu, J., Mutlu, O.: A case for exploiting subarray-level parallelism (SALP) in dram. ACM SIGARCH Comput. Arch. News **40**(3), 368–379 (2012)
8. Min, D.S., Langer, D.W.: Twisted line techniques for multi-gigabit dynamic random access memories, US Patent 6,034,879, 7 March 2000
9. Min, D.S., Seo, D.I., You, J., Cho, S., Chin, D., Park, Y.: Wordline coupling noise reduction techniques for scaled drams. In: 1990 Symposium on VLSI Circuits, Digest of Technical Papers, pp. 81–82. IEEE (1990)

10. Pessl, P., Gruss, D., Maurice, C., Schwarz, M., Mangard, S.: DRAMA: exploiting dram addressing for cross-CPU attacks. In: USENIX Security Symposium, pp. 565–581 (2016)
11. Razavi, K., Gras, B., Bosman, E., Preneel, B., Giuffrida, C., Bos, H.: Flip Feng Shui: hammering a needle in the software stack. In: USENIX Security Symposium, pp. 1–18 (2016)
12. Seaborn, M., Dullien, T.: Exploiting the DRAM rowhammer bug to gain kernel privileges (2015). https://googleprojectzero.blogspot.kr/2015/03/exploiting-dram-rowhammer-bug-to-gain.html
13. Van Der Veen, V., et al.: Drammer: deterministic rowhammer attacks on mobile platforms. In: Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security, pp. 1675–1689. ACM (2016)
14. Xiao, Y., Zhang, X., Zhang, Y., Teodorescu, R.: One bit flips, one cloud flops: cross-VM row hammer attacks and privilege escalation. In: USENIX Security Symposium, pp. 19–35 (2016)