WhistleBlower: A System-level Empirical Study on RowHammer

Wei He*, Zhi Zhang*, Yueqiang Cheng, Wenhao Wang✉,
Wei Song, Yansong Gao, Qifei Zhang, Kang Li, Dongxi Liu, Surya Nepal

Abstract—With frequent software-induced activations on DRAM rows, bit flips can occur on their physically adjacent rows (i.e., RowHammer). Existing studies leverage FPGA platforms to characterize RowHammer, which have identified key factors that contribute to RowHammer bit flips, e.g., data pattern. As the FPGA-based studies have removed the interference of the OS and the memory controller, their findings on the identified contributing factors do not always work as reported in a real-world computing system, resulting in negative effects on system-level RowHammer attacks and defenses.

In this paper, we carry out a system-level empirical study on factors from both the software side and the DRAM side that contribute to RowHammer. We conduct the study on 33 DRAM modules including both DDR4 and DDR3, with 292 DRAM chips from various vendors. Our experimental results from the software side show that some prior findings about existing factors are inconsistent with our observations, thus not applicable to a real-world system. Also, we contribute to identifying one new factor that effectively affects RowHammer bit flips. Our DRAM-side results identify three types of new contributing factors and indicate that DRAM modules are more vulnerable if they achieve better performance and lower power consumption. Particularly, Intel XMP, intended for improving DRAM performance, might be abused for RowHammer attacks.

Index Terms—RowHammer, DRAM, FPGA, Computing System.

1 Introduction

With the rapid progress of semiconductor technology, DRAM storage cells continue scaling down and distances between cells are getting smaller, resulting in electromagnetic coupling problems [1]. Among them, RowHammer has attracted the most attention from both academia and industry in recent years, as it poses a a serious challenge to system security. Specifically, RowHammer is a circuit-level interference phenomenon where repeatedly accessing DRAM rows (*aggressor rows*) can induce bit flips in data from nearby rows (*victim rows*) [2]. By exploiting RowHammer-induced bit flips, an unprivileged attacker can achieve privilege escalation [3], [4], [5], [6], [7], [8], [9], [10], sandbox escaping [6], [7], [11], [12], denial-of-service [13] and cryptographic key recovery [14], [15].

W. He and Z. Zhang are joint first authors.
Corresponding author: W. Wang (wangwenhao@iie.ac.cn).
W. He, W. Wang and W. Song are with SKLOIS, Institute of Information Engineering, CAS and School of Cyber Security, University of Chinese Academy of Sciences. E-mail: {hewei, wangwenhao, songwei}@iie.ac.cn.
Z. Zhang is with University of Western Australia. E-mail: zhi.zhang@uwa.edu.au. Most work was done when he was with Data61, CSIRO.
Y. Cheng is with NIO. E-mail: yueqiang.cheng@nio.io.
Y. Gao is with Data61, CSIRO, Australia and School of Computer Science and Engineering, NanJing University of Science and Technology, China. E-mail: yansong.gao@njust.edu.cn.
Q. Zhang is with School of Software, Zhejiang University. E-mail: cstzhangqf@zju.edu.cn.
K. Li is with Baidu. E-mail: kangli01@baidu.com.
D. Liu and S. Nepal are with Data61, CSIRO. E-mail: {dongxi.liu, surya.nepal}@data61.csiro.au.
This work was supported by the National Key R&D Program of China (Grant No. 2020YFB1805402), National Natural Science Foundation of China (No.62272452, No.62002167, No.62172406), and Natural Science Foundation of JiangSu (No.BK20200461).

To address the security challenge and mitigate the RowHammer attacks, researchers have spent great efforts characterizing RowHammer with the assistance of Field-Programmable Gate Array (FPGA) platforms [2], [16], [17], [18], [19], [20], [21], and their results have identified key factors that contribute to RowHammer bit flips both effectively and efficiently. Although FPGA platforms can serve as an alternative and controllable memory controller to obtain precise results, such platforms conceal the complexity of real-world computing systems, which are the primary targets of attackers. Thus, the identified RowHammer factors might not work as expected in a real-world scenario, as these factors were tested through the FPGA platform without the interference from the OS and memory controller.

For example, Kim et al. [2] leverage the FPGA platform to characterize RowHammer bit flips based on the DRAM cell type [22]. In particular, RowHammer causes a DRAM *true cell* flip from '1' to '0' and an *anti cell* flip from '0' to '1'. Following their work, CTA [23], as a system-level RowHammer defense, places all page tables onto high physical addresses of true cells and leverages the monotonic bit-flip direction of true cells to protect page tables from RowHammer attacks. However, in a real-world commodity system, the monotonic property does not hold, because the data scrambling feature deployed by the modern memory controller [24] enables a true cell to flip from either direction, breaking the security guarantee of CTA. Also, Cojocar et al. [16] utilize the FPGA and the UEFI firmware to record DDR commands and count DRAM activation rate. They show that using memory barriers (e.g., `mfence`) slows down the DRAM activation (`ACT`) rate and induces less bit flips. In contrast, we observe from a real-world system that using `mfence` for hammer can trigger much more bit flips.

Besides, hardware manufacturers frequently tune DRAM parameters to improve performance and reduce power consumption, which are likely to introduce more vulnerable DRAM modules. For example, eXtended Memory Profile (XMP) [25], proposed by Intel, optimizes DRAM process and parameters to overclock DRAM and improve DRAM performance. We observe that enabling XMP from hardware significantly increases DRAM susceptibility to RowHammer, indicating that an attacker might abuse XMP to mount a RowHammer attack. Unfortunately, many DRAM parameters have been ignored by prior works and a comprehensive evaluation of these parameters with regard to RowHammer is needed.

1.1 Our Work

To bridge the gap between FPGA-based findings and system-level RowHammer defenses and attacks, we perform an empirical study on factors contributing to RowHammer bit flips from a real-world system (i.e., a *system-level* study), using a popular Ubuntu OS and 33 different DRAM modules including both DDR3 and DDR4 (292 DRAM chips from 12 vendors). Specifically, our testbed consists of commercially available hardware, that is, Intel i3-10100 processor + MSI Z490 motherboard and Intel i7-4790 processor + ASUS Z97 motherboard. For the software of our testbed, we extend DRAMDig [28], a DRAM address mapping reverse engineering tool, to implement a RowHammer test. Our test supports 5 user-configurable parameters that have observable effects on bit flips, i.e., *hammer pattern*, *data pattern*, *hammer method*, *hammer count* and *multi-thread*. We first launch the RowHammer test with default DRAM parameters to perform memory templating and find a certain number of physical addresses vulnerable to bit flips. Based on the stored vulnerable addresses, we then investigate factors from both the software side and the DRAM side as follows.

Investigating factors from the software side. We attribute RowHammer factors that can be controlled by software into this category. They are known to have noticeable effects on bit-flip rates, and are thus crucial for RowHammer attacks and defenses. In our study, we first summarize prior findings about existing contributing factors. We then provide the RowHammer test with different parameter values to re-evaluate these findings in terms of their bit-flip rate on the vulnerable addresses. Table 1 shows a comparison of prior findings and our empirical observations. For the re-evaluation, we also leverage an FPGA board to reproduce some prior findings from prior FPGA-based studies, that is, we leverage SoftMC [29], an open-source FPGA-based platform to perform RowHammer tests on a Xilinx ML605 FPGA board. As a comparison, similar experiments are conducted in a Lenovo Thinkpad T420s laptop with Ubuntu Gnome environment installed. Both the FPGA board and the Thinkpad laptop use the same vulnerable single-rank Samsung DDR3 SODIMM (2 GiB with 8 DRAM chips). We summarize the experimental results as well as the new contributing factor identified in our study as follows.

• *Hammer pattern* (e.g., many-sided hammer [30]) specifies the number of rows being hammered. An effective hammer pattern results in frequent row activations and could bypass Target Row Refresh (TRR), a RowHammer defense implemented in present DDR4 modules [31]. We have examined 4 hammer patterns and observe that hammering fewer rows with the same access number for each row achieves better bit-flip effectiveness, being inconsistent with [17] which claims that if an n_1-side hammer ($n_1 > 2$) can successfully flip the bit, then n_2-side hammer ($n_2 > n_1$) is also successful to induce the bit flip if the same access number is applied. We note that we have successfully reproduced the claim from [17] using the FPGA board and the system-level observation using the Thinkpad laptop.

• *Data pattern* refers to data values stored in aggressor and victim rows (e.g., "RowStripe" [2]), which is critical in triggering bit flips. In [26], the "Killer" pattern is reported to be the most effective in triggering bit flips. In our study, we tested 10 data patterns including the "Killer" pattern. The results show that the "RowStripe" and "Checkered" patterns with their inverses are the most effective ones among the tested data patterns on the system-level. We also conduct the same experiments on both the laptop and the FPGA board. The results from the laptop are consistent with the system-level observation. While for the FPGA board, the results confirm prior works [2], [18] done at the FPGA level, that is, the 'RowStripe" pattern is the most effective one.

• *Hammer method* (e.g., `clflush` followed by a memory load and `mfence` [2]), bypasses CPU caches and enables DRAM memory accesses to a row being hammered. We have experimented with 12 hammer methods and showed that these methods have different RowHammer bit-flip effectiveness in given DRAM modules. We also observe that the order of bypassing caches and triggering memory accesses (known as the "gather" pattern and the "scatter" pattern) do not work the same as observed in [11]. The two patterns show distinct effectiveness in triggering bit flips against given DRAM modules. Besides, we note that a hammer method without a memory barrier (e.g., `mfence`) induces much fewer bit flips compared to that with a memory barrier on some given DRAM modules, which is inconsistent with the observation from [16] in the FPGA level, indicating that an improved DRAM activation rate made from the FPGA platform does not necessarily contribute to a more effective hammer method at the system level. As the number of row buffer conflicts is an strong indicator of hammering effectiveness, we leverage an Intel server (i.e., Intel Xeon E5-2660 v2), to collect the statistics of row buffer conflicts caused by different hammer methods with and without `mfence`. The results show that different hammer methods cause different row buffer conflict rates and thus cache-flush instructions with memory read are more effective than other hammer methods, while the `mfence`-based hammer methods do increase the row buffer conflict rate, resulting in an improved hammering effectiveness.

• *Minimal hammer count* is the least number of accesses required to each hammered row for inducing the first bit flip. We test the minimal hammer count (i.e., $\mathtt{HC}_{first}$ in [18]) from the system-level and find that the minimal hammer count of DDR4 modules (i.e., 10K) is same as previously reported [18] while the minimal hammer count of DDR3 modules is much higher than that from previous FPGA-level works [2], [18]. Our experiments on the laptop and

TABLE 1
A comparison of prior findings and our system-level empirical study.

Factor	Prior Finding	**Our Observation**
Hammer Pattern	In FPGA platforms, a vulnerable bit flippable with n_1-side hammer ($n_1 > 2$) is also flippable in n_2-sided ($n_2 > n_1$) hammer [17].	More hammered rows can prevent a vulnerable bit from flipping in our system-level study.
Data Pattern	The "Killer" data pattern is the most effective in triggering bit flips [26].	The "RowStripe" is more effective than the "Killer" data pattern.
Hammer Method	The non-temporal instructions can be used for RowHammer [12].	The non-temporal instructions are ineffective on certain DRAM modules.
	The "scatter" sequence cannot induce bit flips while the "gather" sequence can do so [11].	Both sequences can induce bit flips.
	The memory barriers decrease the `ACT` rates, resulting in a lower hammer effectiveness [16].	The memory barriers do have a role in effectively triggering bit flips from a real-world system.
Minimal Hammer Count	In FPGA platforms, the minimal hammer count can be as few as around 20K per row in DDR3 and 10K per row in DDR4 [18].	To induce bit flips in real-world systems, the minimal hammer count on DDR4 is the same as Kim et al. [18] reported, while much more in DDR3.
Multi-thread	Compared to single-thread, multi-thread for hammer is more effective in triggering bit flips on both DDR3 and DDR4 modules [8], [13], [26], [27].	The effectiveness of multi-thread-based hammer depends on the DRAM module type.
Bit-flip Direction	CTA [23] leverages *true cell* to enforce monotonic bit-flip direction on targeted DRAM regions without considering the data scrambling.	The data scrambling feature, employed by the memory controller, invalidates CTA as it allows a cell to flip from either '1' to '0' or '0' to '1'.

the FPGA board validate the DDR3-based finding, that is, the minimal hammer count on DDR3 in the system-level is much more that in the FPGA-level.

• *Multi-thread* is proposed to improve hammer effectiveness, as hammering multiple aggressor rows within a single thread is inefficient [8], [13], [26], [27]. Our system-level RowHammer test supports multi-thread hammer, based on the implementations of SGX-BOMB[1] and "rowhammer_armv8" [2]. The experiments show that multi-thread hammer is much less effective than single-thread hammer for DDR4 modules with TRR, and more effective for DRAM modules without TRR. The effectiveness difference is probably caused by the TRR's sampler, which might be ignored by prior works [8], [13], [26], [27].

• *Bit-flip Direction*. From our system-level RowHammer test, we observe that the bit-flip direction for a DRAM cell can be different if the system restarts, that is, a vulnerable DRAM cell can be flipped from either '0' to '1' or '1' to '0', which can be attributed to the data scrambling feature in modern commodity systems. The effect of data scrambling has been ignored by a recent RowHammer defense (i.e., CTA [23] in ASPLOS'19), which leverages different DRAM cell types to enforce monotonic bit-flip direction.

• *Running Environment*. We have identified a new contributing factor that significantly affects the number of bit flips that can be triggered. Particularly, we conduct our system-level RowHammer test on the Ubuntu Gnome environment and text-only terminal, respectively. The results show that the Ubuntu Gnome environment is much more effective in inducing bit flips (the number of bit flips can be two orders of magnitude more).

Identifying factors from the DRAM side. Motivated by XMP, we leverage the system-level RowHammer test with different DRAM parameters to explore the DRAM-side factors. Particularly, we examine the effectiveness of RowHammer with respect to major DRAM parameters, including the DRAM frequency, DRAM supply voltage, and DRAM timing parameters. The results in general imply that a DRAM module is more vulnerable to RowHammer if it is configured for better performance, similar to the aforementioned observation of XMP. We have identified three types of contributing factors as follows.

• *Frequency*. The frequency always facilitates RowHammer, as a higher DRAM clock rate improves memory-access throughput and triggers more bit flips.

• *Supply Voltage*. The higher supply voltage often suppresses RowHammer, as it may overcharge DRAM cells. Accordingly it is relatively more difficult for cells to leak enough charge, resulting in fewer bit flips.

• *Timing Parameters*. We select 17 timing parameters for configuration (16 for DDR4 and 10 for DDR3). From our experiments, RowHammer mainly correlates with 6 parameters, i.e., `tRCD`, `tRP`, `tRAS`, `tRFC`, `tREFI` and `tWR`. Interestingly, we observe that some of the parameters (e.g., `tRAS`) may affect the bit-flip effectiveness either positively or negatively, depending on the tested DRAM modules.

1.2 Contributions

In summary, this paper makes the following contributions.

• We conduct a comprehensive system-level empirical study of factors that contribute to Rowhammer bit flips. Our study examines factors from both the software side and the DRAM side based on extensive RowHammer tests.

• From the software side, we re-evaluate prior findings about 5 existing factors and a neglected feature, showing that most existing findings are inconsistent with our empirical observations and thus they are not widely applicable. Moreover, our study reveals a new contributing factor in affecting RowHammer bit-flip effectiveness.

• From the DRAM side, we quantify the impacts of major DRAM parameters on the effectiveness of Rowhammer, including DRAM frequency, DRAM supply voltage, and

1. https://github.com/sslab-gatech/sgx-bomb
2. https://github.com/VandySec/rowhammer_armv8

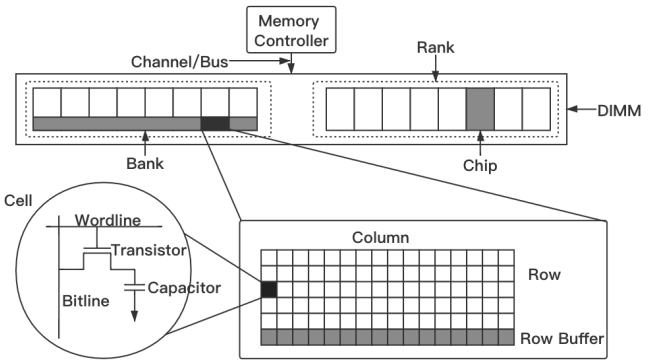


Fig. 1. The DRAM memory organization.

DRAM timing parameters. Our study identifies a potentially exploitable Intel feature and 3 types of contributing factors.

2 Background and Related Works

In this section, we first provide DRAM basics and then introduce RowHammer as well as related works. Please refer to the JEDEC standards [32], [33], [34] and two comprehensive surveys [1], [35] for more details about DRAM and RowHammer, respectively.

2.1 DRAM

DRAM Organization. Figure 1 presents an overview of a modern DRAM memory organization. Specifically, a memory controller (MC) transfers data and commands to and from DRAM modules through memory channels. A modern DRAM module known as the Dual In-line Memory Module (DIMM) is usually composed of one or two ranks. A rank consists of a set of DRAM chips that operate in lockstep to reply to commands from the MC. A DRAM chip has multiple cross-chip banks, and each bank has a sense amplifier and many subarrays. A sense amplifier, also called the *row buffer*, senses a row of data that has been recently accessed. A subarray is a two-dimensional array of DRAM cells, which is divided into rows and columns for its connected wordline and bitline. Each cell consists of an access transistor serving as a switch and a capacitor storing a single bit of either '1' or '0'. A cell has two types, i.e., *true cell* and *anti cell*. When the true cell's capacitor is charged (or discharged), it represents bit '1' (or bit '0'). The anti cell works in the opposite way.

DRAM Operations. A modern MC issues a set of DRAM commands to *read* (or *write*) data from (to) DRAM chips. First, an activate (`ACT`) command is sent to *open* a targeted row, whose data will then be copied into the row buffer. Second, a read/write (`RD/WR`) command is issued to select the desired cache lines from the row buffer for loading or storing data. Last, a precharge (`PRE`) command is used to *close* the row and clear the row buffer for subsequent access to another row.

As DRAM cells leak charge over time, a minimum time period that the cells maintain a correct bit is referred to as the *retention time*. The MC periodically issues a refresh (`REF`) command to the DRAM banks to ensure all cells are refreshed before the retention time expires. The standard refresh interval for a row is 64 ms [32], [34], within which at least 8192 `REF` commands need to be issued.

2.2 Related Works

RowHammer. Kim et al. [2] were the first to identify the existence of electromagnetic disturbance errors (the so-called RowHammer) in modern DIMMs. They observed that activating aggressor rows (i.e., hammering) frequently enough within the refresh interval can flip bits stored in adjacent victim rows. Even worse, recent DIMMs are more vulnerable to RowHammer than before, as DRAM manufacturers continue increasing DRAM storage density [18], [36].

There are several empirical studies on RowHammer [2], [16], [17], [18], [19], [20], [21], [26]. To characterize RowHammer and explore factors that contribute to RowHammer, some [2], [19], [20], [21] experiment with DDR3 modules while some others [16], [17], [18], [26] focus on both DDR3 and DDR4 modules.

All existing empirical studies above except [26] utilize FPGAs to characterize RowHammer. Particularly, Kim et al. [2] provide a relatively comprehensive RowHammer characterization and identify multiple factors triggering RowHammer such as access pattern, hammer count, data pattern, DRAM cell type, etc. Following their work, Park et al. [19], [20], [21] conduct experimental studies on minimal hammer count, data pattern, ambient temperature and `tRP` (i.e., a period for the DRAM `PRE` command). Kim et al. [18] examine a large amount of DDR3, DDR4 and lpDDR4 chips about minimal hammer count, data pattern and error spatial distribution, reporting that newer DRAM chips are more vulnerable to RowHammer. Jiang et al. [17] propose a mathematical model of capacitive-coupling in DRAM and analyze multiple factors in their proposed model contributing to RowHammer. Cojocar et al. [16] explore the DRAM internal address mapping and the hammer efficiency of different hammer methods using a DDR interposer and an FPGA on Intel server platforms booting into the UEFI mode. All these FPGA-based works have identified critical factors contributing to RowHammer. Besides the FPGA-based studies, Lanteigne [26] implements Memesis, a customized Linux kernel embedded enterprise memory test, to examine multi-threading hammer, regional RowHammer (i.e., 2 MB memory region as a Linux hugepage for hammering) and data pattern.

However, none of the above studies analyze RowHammer at a commodity-operating-system level, generating a non-neglectable gap between their findings and OS-level RowHammer attacks and defenses. To this end, multiple RowHammer attacks and defenses [11], [14], [23], [37], [38], [39], [40] spend efforts in analyzing and leveraging one or more RowHammer-relevant factors. For example, Radar [39] studies the impacts of different hammer methods on hammering efficiency while Smash [11] investigates the hammering effectiveness from different sequences of cache-flush instructions and memory accesses. RAMBleed [14] and Pinpoint [38] carefully craft data patterns to suppress unwanted RowHammer bit flips. ANVIL [37] reports that RowHammer still occurs in real-world systems even if the DRAM refresh interval is reduced by half. CTA [23] implements a system-level method to identify the DRAM cell type for a given DRAM row.

3 Evaluation Methodology

Our primary goal is to re-evaluate existing factors and explore new factors that contribute to RowHammer bit flips at the *system level*, i.e., on real-world computing systems. We consider both software factors (controlled by software) and DRAM factors (configured by the DRAM manufacturer or by the user through the BIOS). We conduct the experiment in the following three steps.

• *First*, to rule out the effect of robust DRAM cells, we need to collect enough vulnerable DRAM locations in advance. For this purpose, we conduct a RowHammer test with default parameters' values to find enough (i.e., more than 1000) vulnerable physical addresses per module where reproducible bit flips occur and collect these addresses offline.

• *Second*, we investigate the effect of each candidate factor from the software side on bit flips by varying the parameters' values for the RowHammer test with the default DRAM parameter configuration (refer to Table 7). We quantify the effectiveness of the candidate factor using the metric of *bit-flip rate*, i.e., the number of re-generated bit flips divided by the number of collected vulnerable bits.

• *Last*, we evaluate the effect of each candidate factor from the DRAM side on bit flips. We use the default system-level RowHammer test to further study the effects of candidate factors from the DRAM side.

In this section, we first discuss how to conduct the RowHammer test and describe our experimental setup. Deriving from our RowHammer test, we then elaborate our empirical study from both the software side (Sec. 4) and the DRAM side (Sec. 5) respectively.

System-level RowHammer Test. To trigger RowHammer bit flips and collect vulnerable physical addresses for a given DRAM module, we develop an effective Rowhammer test tool, which has 5 user-configurable parameters (i.e., *hammer pattern*, *data pattern*, *hammer method*, *hammer count* and *multi-thread*). The source code used for evaluation has been released at https://github.com/whistleblower2022/whistleblower_tool. We introduce how to implement the tool as follows.

Specifically, based on a distribution of rows being hammered (i.e., aggressor rows), we have multiple hammer patterns [5], [7], [30]. Among them, our extended RowHammer test selects double-sided hammer for DDR3 modules as it is the most efficient [7]. For DDR4 modules, we leverage TRRespass [30] to identify the best hammer pattern that produces most bit flips within a specified time frame, shown in Table 2. To implement the efficient hammer patterns, (partial) knowledge about the virtual-to-physical address mapping and physical-to-DRAM address mapping is required from the software perspective [3]. In our evaluation, we have access to the `/proc/pid/pagemap` interface for virtual-to-physical address mapping. We further reverse engineer the physical-to-DRAM address mapping to issue memory requests precisely by DRAMDig [28]. Then we can specify rows within the same bank for subsequent hammering under a given hammer pattern.

Previous works [2], [14], [18] have shown that data values stored in the aggressor and victim rows also have observable effects on bit flips, known as data pattern. There have been a number of proposed data patterns such as "Solid" and "RowStrip" [2], among which the difference in inducing bit flips can be in an order of magnitude. We incorporate 10 data patterns detailed in Sec. 4.2 into our test and select the "RowStripe" as the default[3].

After padding targeted rows with a distinct data pattern, we need an appropriate hammer method to enable direct memory accesses to every aggressor row. Existing hammer methods can be classified into three categories, cache eviction-based [4], [11], [37], [41], uncached memory-based [8], [42], [43], and explicit instructions-based [2], [5], [7], [15], [16], [30]. In our test, we implement 12 hammer methods and choose the `clflush+read`-based sequence with `mfence` that works for our Intel processors as the default hammer method.

Hammer count is the number of accesses to each hammered row in a finite loop. We choose 1000K based on previous works [7], [30], [41]. Each hardware thread of multiple threads can be used to hammers all aggressor rows [8], [13], or hammers some aggressor rows [27]. We use single-thread by default.

Experimental Setup. We use Ubuntu Gnome environment to run a system-level RowHammer test, which allocates 80% size of the total memory to find vulnerable physical addresses using default parameter values. To this end, we install Ubuntu systems on commodity platforms and focus on evaluating 31 DDR4 modules with 276 chips and 2 DDR3 modules with 16 chips, as DDR4 is the mainstream in the market and DDR3 is relatively outdated. Although 31 DDR4 modules have been tested, only 4 out of them have a statistical number of bit flips, which come from different vendors. Table 2 shows the experiment setup including the 6 vulnerable DRAM modules where our RowHammer test is conducted as well as three default hammer parameters.

Besides, we use an FPGA board to reproduce observations from prior FPGA-based studies to show the difference in hammering effectiveness from the FPGA level and the system level. Specifically, we leverage SoftMC [29], an open-source FPGA-based infrastructure to perform RowHammer tests on a Xilinx ML605 FPGA board. The FPGA board has a DDR3 SODIMM slot with a vulnerable single-rank Samsung DDR3 SODIMM (2 GiB with 8 DRAM chips) inserted, and a PCIe interface connecting itself to a Linux host machine. Please note that the DDR3 SODIMM has a part number of M471B5773DH0-CH9, which is different from the tested UDIMMs that are suitable only for a workstation as shown in Table 2. SoftMC is composed of 3 major parts, i.e., API, PCIe driver, and hardware and the general working flow among the three parts is as follows: the Linux host machine generates a set of SoftMC instructions by invoking the SoftMC API, which is sent by the SoftMC PCIe driver over a PCIe bus to the SoftMC hardware implemented on the FPGA board. After receiving the instructions, the SoftMC hardware can execute them. As a comparison, similar RowHammer tests are conducted in a Lenovo Thinkpad T420s laptop with Ubuntu Gnome environment installed and the same vulnerable DDR3 module used.

3. We pad memory with the "RowStripe" and its variant (i.e., its inverse) respectively in a RowHammer test and sum up their number of bit flips for the evaluation of "RowStripe".

TABLE 2
Experimental setup and three default hammer parameters.

Setting	DIMM	Vendor	Part Number	Size (GiB)	#Chips	#Banks	Hammer Pattern	Hammer Method	Data Pattern
MSI Z490 i3-10100	M0	Kingston	99P5701-005.A00G	8	16	32	3-sided	`clflush+ read` with `mfence`	"RowStripe" with its inverse
	M1	Apacer	D12.2324WC.001	8	8	16	2-sided		
	M2	Galaxy	—[1]	8	8	16	13-sided		
	M3	Samsung	M378A1G44AB0-CWE	8	4	8	18-sided		
ASUS Z97 i7-4790	M4	Hynix	HMT41GU6MFR8C-P8	8	16	16	2-sided		
	M5	G.Skill	F3-14900CL9-4GBSR	4	8	8	2-sided		

[1] The part number field of M2 is empty when read from both OS and BIOS.

TABLE 3
A comparison of Xilinx ML605 FPGA and Thinkpad T420s in bit-flip rates caused by different hammer patterns.

Bit-flip Rate	Hammer Pattern (n-sided)		
	3-sided	4-sided	5-sided
Xilinx ML605	88.82%	88.82%	89.28%
Thinkpad T420s	81.03%	29.58%	8.41%

4 Factors from the Software Side

From previous works, we have summarized 5 existing factors from the software perspective including the hammer pattern, data pattern, hammer method, hammer count and multi-thread. We also identify a new contributing factor, that is, *running environment*. In the following, we conduct quantitative experiments on these factors to re-evaluate previous findings and present new empirical observations at the system level. Particularly, we re-evaluate three factors, i.e., the hammer pattern, the data pattern and the minimal hammer count in both FPGA and OS contexts. We note that prior FPGA studies do not study other factors mentioned before as they are not supported by the FPGA platform. And we analyze the impact of data scrambling on RowHammer, which has been ignored by existing RowHammer characterization works.

4.1 Hammer Pattern

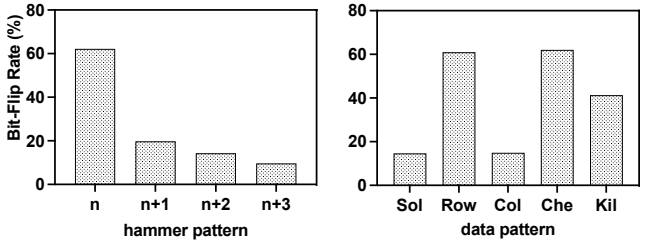


Fig. 2. Average bit-flip rate on tested modules when different hammer patterns (in the left plot) and different data patterns (in the right plot) are applied, respectively. Each bar in the right plot is a sum of bit-flip rates generated from a data pattern and its inverse. The "Sol", "Row", "Col", "Che", "Kil" are short for "Solid", "RowStripe", "ColStripe", "Checkered", "Killer".

The hammer pattern denotes the number of hammered rows and there are four uniform hammer patterns from prior works, i.e., single-sided hammer [2], [7], double-sided hammer [2], [7], one-location hammer [5] and many-sided hammer [30]. In this section, we evaluate the effect of the aggressor-row number on RowHammer.

Recently, Jiang et al. [17] report that if n_1-sided hammer ($n_1 > 2$) induces bit flips successfully, any n_2-sided hammer ($n_2 > n_1$) should also be successful when they apply the same hammer count to each hammered row. Specifically, they first find reproducible bit flips in certain DRAM cells by hammering and then apply less aggressor rows to these location. In our system-level experiment, we start with the default hammer pattern and increase the number of aggressor rows keeping hammer count the same. As the bit-flip rate tendency of different hammer pattern on each module is similar, we deliver the average bit-flip rate of all 6 modules in the left plot of Figure 2. The average bit-flip rate for each tested DIMMs decreases as the number of hammered rows increases. This is probably because that, when more rows are hammered, the time for hammering each row is reduced within the fixed DRAM refresh interval and victim rows are less likely to leak charge, thus generating less bit flips.

We reproduce the experiments above using the FPGA board and carry out the above system-level experiment on the Thinkpad laptop. To be specific, we randomly select more than 1000 bits that can be flipped using 5-sided RowHammer on the FPGA board. We then count the bit-flip rate of these flippable bits under 3/4/5-sided RowHammer. As we can see from Table 3, the bit-flip rate for each 3/4/5-sided RowHammer is similar to each other in the FPGA context, validating the previous observation in [17]. For the Thinkpad T420s, we collect vulnerable bits under 3-sided RowHammer and conduct the system-level experiments as above. The results on Thinkpad are also shown in Table 3. Clearly, increasing the number of hammered rows reduces hammer effectiveness towards the selected vulnerable bit at the system level .

Observation 1: *If row buffer is flushed and TRR is bypassed, more hammered rows trigger less bit flips at the system level.*

4.2 Data Pattern

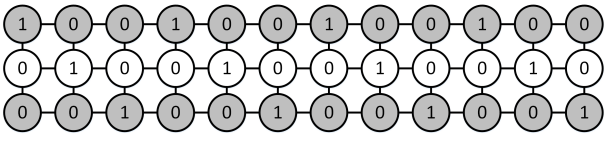


Fig. 3. "Killer" data pattern. The sandwiched victim row looks as '0x492492...' in hexadecimal.

Data pattern refers to the data values stored in aggressor rows and victim rows. Typically, there are four common data patterns [2]: "Solid" (all cells are padded with the same value '0' or '1'), "RowStripe" (rows padded with '0' are interleaved with rows padded with '1'), "ColStripe" (columns padded with '0' are interleaved with columns padded with '1') and "Checkered" (cells are padded with

TABLE 4
A comparison of Xilinx ML605 FPGA and Thinkpad T420s in bit-flip rates caused by different data patterns and their inverses.

Bit-flip Rate	Data Pattern				
	Sol	Row	Col	Che	Kil
Xilinx ML605	1.60%	94.41%	0.80%	44.46%	63.44%
Thinkpad T420s	3.10%	36.81%	2.71%	37.04%	26.49%

either '0' or '1' in a checkerboard pattern). Lanteigne [26] reports that the "Killer" data pattern (cells in a row are padded in a 3-bit cycle of either '010' or '101', making the row look as '0x492492...' or '0xb6db6d...' in hexadecimal, and the bit cycle shifted by one bit is used to pad nearby rows, shown in Figure 3) is the most effective in inducing bit flips among all data patterns on their tested modules. We re-evaluate these data patterns with their inverses on our testbed. Similar to the hammer pattern, we show the average bit-flip rate of all 6 modules in the right plot of Figure 2, manifesting that the "RowStripe" and the "Checkered" data patterns are the best while the "Killer" data pattern is not. Besides, as the bit-flip rate for a data pattern where cells of each row have the same values (e.g., "RowStripe") is much higher than that of a data pattern where cells of each column have the same values (e.g., "ColStripe"), aggressor cells and victim cells are more likely to reside in different rows rather than in different columns of the same row. Our results indicate the disturbance impact raised by different wordlines is larger than that from different bitlines, which is consistent with previous FGPA-based studies [2], [18].

We also use the Xilinx ML605 and Thinkpad to compare the hammering effectiveness from two levels and present results in Table 4. The results from the Thinkpad are consistent with previous system-level experiments shown in Figure 2, that is, the 'RowStripe" and "Checkered" perform the best at the system level. The results from the Xilinx ML605 are also consistent with prior works [2], [18] done at the FPGA level, that is, the 'RowStripe" is the most effective data pattern. But these are inconsistent with a previous observation done by [26] that reports that the "Killer" is the most effective one [4]. The different effectiveness for the same data pattern between the FPGA and OS contexts might be caused by the data-scrambling feature within the memory controller, as the "Killer" that is perceived by the OS might not be the "Killer" from the perspective of the FPGA.

Observation 2: *"Killer" is not as effective as previously reported [26] at the system level. "RowStripe" performs the best at the system level, consistent with prior works [2], [18] done at the FPGA level.*

```
mov (X), %rax      |   movnti %rax, (X)
clflush (X)        |   mov %rax, (X)
mov (Y), %rax      |   movnti %rax, (Y)
clflush (Y)        |   mov %rax, (Y)
mov (Z), %rax      |   movnti %rax, (Z)
clflush (Z)        |   mov %rax, (Z)
mfence             |   mfence
```

Listing 1. 3-sided hammer: clflush+r (left) and movnti+w (right)

4. We note that the observation is neither done from the FPGA level nor the real-world OS level. Instead, [26] uses Memesis, a Linux kernel embedded commercial memory test.

4.3 Hammer Method

Based on the default RowHammer test that implements `clflush`, we evaluate different instruction-based hammer methods, which are the most efficient for a local test in x86 architectures [39]. Specifically, we consider two types of instructions available on our Intel platforms: cache-flush instructions including `clflush` [7] and `clflushopt` [16] and non-temporal instructions [12] including `movnti` and `movntdq`. We provide our system-level RowHammer test with 12 different hammer methods and classify them into 3 categories depending on their memory-access type, that is, read (`r`), write (`w`), read-and-write (`rw`) (see two examples in Listing 1). Based on our experimental results, we have made 4 key observations as follows.

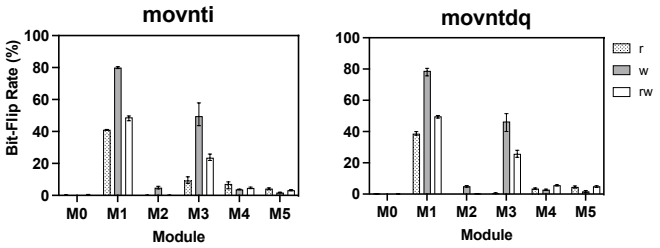


Fig. 4. Bit-flip rate on tested modules when different `non-temporal`-based hammer methods are applied.

First, Qiao et al. [12] observe that hammer methods with non-temporal instructions work as effectively as that with cache-flush instructions. However, a hammer method based on one of the observed instructions does not trigger bit flips in some tested DRAM modules. Particularly, we test `movnti+w` (identified by Qiao et al. [12]) against M0 and M1, showing that `movnti+w` is as effective as `clflush+r` in M1 but surprisingly induces no bit flips in M0.

Observation 3: *The effectiveness of non-temporal instruction based hammer methods is likely dependent on DRAM modules.*

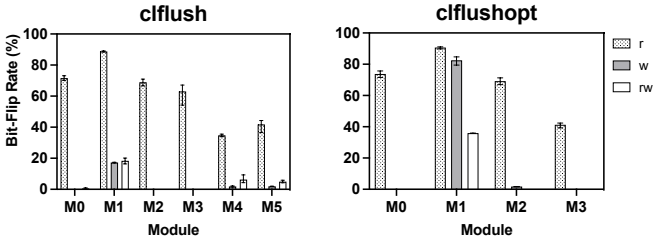


Fig. 5. Bit-flip rate on tested modules when different `flush`-based hammer methods are applied. The DDR3 modules of M4 and M5 are not illustrated in the right plot as the DDR3-based platform does not support `clflushopt` instruction.

Second, summarizing from all tested hammer methods manifested in Figure 4 and Figure 5, cache-flush instructions with memory read work better than that with memory write while memory write are better for non-temporal instructions. Considering that non-temporal instructions do not work consistently on each module, the cache-flush instructions are better choices.

Observation 4: *Cache-flush instructions with memory read are preferable to implement an effective instruction-based hammer method.*

```
mov (X), %rax      |   mov (X), %rax
mov (Y), %rax      |   clflush(X)
```

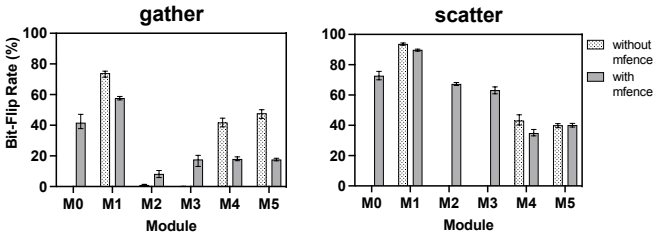


Fig. 6. Bit-flip rate on tested modules when different orders of hammer instruction sequence are applied.

```
...                    |   ...
clflush (X)            |   mov (Y), %rax
clflush (Y)            |   clflush(Y)
...                    |   ...
```

Listing 2. "gather" sequence (left) and "scatter" sequence (right).

Third, we re-evaluate a prior observation that the sequence of `clflush` and memory read significantly affects bit flip for many-sided hammer [11]. Particularly, Ridder et al. [11] find that the "gather" sequence can produce bit flips while the "scatter" sequence cannot. As shown in Listing 2, the "gather" sequence refers to a batch of memory requests followed by a batch of `clflush`. In the "scatter" sequence, `clflush` is interleaved with memory request. In our experiments, their finding does not apply to our tested DRAM modules. As shown in Figure 6 represented by the spotted column, both sequences cause bit flips in multiple modules. The "scatter" performs better on M1 and M4 while the "gather" is better on M5, and both hardly trigger bit flips on the other three DRAM modules (i.e., M0, M2, M3).

Observation 5: *The effectiveness of the order of `clflush` and memory access on RowHammer depends on DRAM modules.*

Last, memory-barrier instructions (e.g., `mfence`) make sure that data is flushed to memory before subsequent memory instruction is executed, as shown in Listing 1. Cojocar et al. [16] observe that a hammer method without the memory barrier presents a higher `ACT` rates (thus a higher hammer efficiency) than that with the memory barrier on Intel server processors booting into the UEFI mode, because the memory barrier introduces additional CPU cycles. Based on their observation, a hammer method without the memory barrier should generate more bit flips. However, as shown in Figure 6 where a spotted bar is for a hammer instruction sequence without `mfence` and the grey bar is for a sequence with `mfence`, hammer with `mfence` can trigger bit flips on every modules while hammer without `mfence` can only work on half of the test modules, which might be due to the CPU's optimization, that is, the CPU re-orders memory accesses for a given hammer instruction sequence without the memory barrier and serve the accesses from the cache, resulting in no bit flips in some modules.

Observation 6: *Although memory barriers decrease hammer efficiency [16], they can be counter-intuitively more effective in inducing bit flips in a DRAM module.*

Analyzing the hammering effectiveness in different combinations of instructions. To investigate the root cause behind the different hammering effectiveness, we observe that an Intel server can be of great help as it provides the statistics of row buffer conflicts. Specifically, an Intel CPU has a large part outside its actual cores, called "Uncore". The uncore part has LLCs, PCI-express, memory controller, etc, and provides a list of performance counter events to monitor its performance, among which an event called `PRE_COUNT.PAGE_MISS` can capture DRAM precharge events due to `page misses` [44]. A page miss is referred to as a "page/row buffer conflict" and occurs when a row buffer is open but has a wrong row in it. Derived from this event and other two events (i.e., `CAS_COUNT.RD` counts all DRAM read requests and `CAT_COUNT.WR` counts all DRAM write requests), another event called `PCT_REQUESTS_PAGE_MISS` reports the percentage of memory requests that result in row buffer conflicts, that is, `PRE_COUNT.PAGE_MISS / (CAS_COUNT.RD + CAS_COUNT.WR)`. As RowHammer requires accessing different rows frequently to trigger bit flips, it results in an abnormal number of row buffer conflicts. Clearly, the more row buffer conflicts within a given short period indicate more effective hammering.

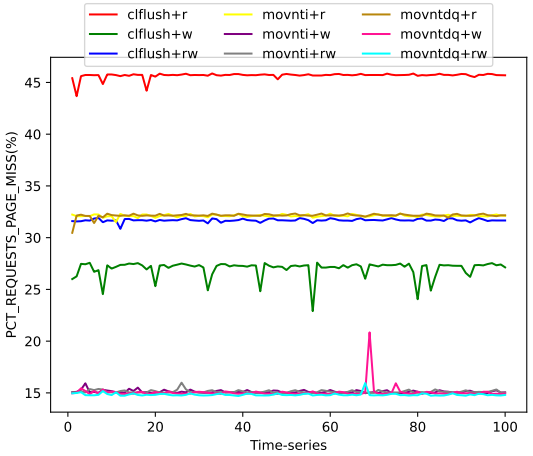


Fig. 7. A comparison of different hammer methods with regard to the percentage of row buffer conflicts they induced every 50 milliseconds in a given period of 5 seconds.

To this end, we leverage `PCT_REQUESTS_PAGE_MISS` to analyze the different combinations of instructions that present different hammering effectiveness. To be specific, we use an HP Z420 Workstation with Intel Xeon E5-2660 v2 and 64 GiB ECC-enabled Hynix DDR3 installed, and download an open-source tool[5] that is built on top of Linux `perf`. To evaluate the hammering effectiveness of each hammer method, we execute each to hammer a randomly selected pair of addresses in an infinite loop. The pair of addresses is from different rows within the same bank to trigger row buffer conflicts. In the meantime, this tool is launched for 5 seconds and reports `PCT_REQUESTS_PAGE_MISS` every 50 milliseconds, resulting in 100 values. Figure 7 shows `PCT_REQUESTS_PAGE_MISS` of each hammer instruction with a distinct memory type, followed by `mfence` by default (`clflushopt` is not supported in this microarchitecture). Clearly, `clflush+read` has the highest percentage of row buffer conflicts, indicating its highest hammering effectiveness.

5. https://github.com/andikleen/pmu-tools

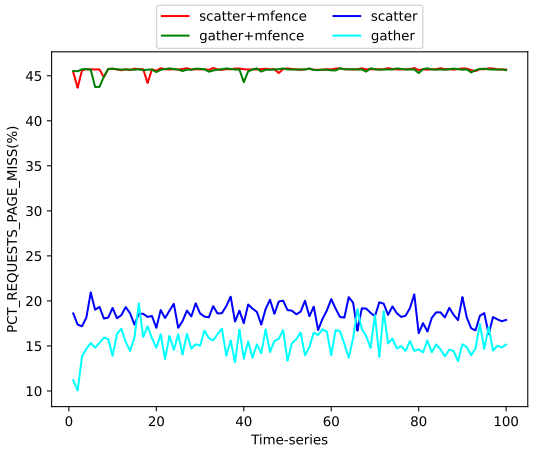


Fig. 8. A comparison of different instruction sequences with regard to the percentage of row buffer conflicts they induced every 50 milliseconds in a given period of 5 seconds.

TABLE 5
The minimal hammer count on tested modules.

Module	M0	M1	M2	M3	M4	M5
Minimal HC	110K	90K	30K	10K	260K	230K

We use the Intel server platform to analyze the impact of `mfence` as well. Figure 8 shows the impact of `mfence` in affecting the hammering effectiveness of `clflush+read` in both "scatter" and "gather" sequences (i.e., the order of `clflush` and memory access). Clearly, the hammering effectiveness of the hammer method with `mfence` is much better than that without `mfence` in either sequence.

4.4 Minimal Hammer Count

Kim et al. [18] report the minimal hammer count that can induce the first bit flip across different DRAM type-node configuration (i.e., 22.4K for DDR3, 10K for DDR4) using an FPGA platform with `REF` disabled. Considering the scheduling of memory requests and the translation from virtual address to physical address and to DRAM internal location, this minimal value may not be applicable to a real-world system. We examine the minimal hammer count at the system level. Specially, we utilize a prior TRR-fuzzing tool [30] to find the most efficient hammer pattern (i.e., the one that triggers most bit flips in a given time period) for TRR-protected DDR4 modules. However, after around 10 hours fuzzing (more than 1000 billion hammer times in total), most of our tested modules are robust enough that no bit flips occur. We thus regard these module's minimal hammer count as *infinite*. In our experiments, we find only one DDR4 module's minimal hammer count is 10K, consistent with the prior work [18], while the minimal values in other DDR4 modules are much higher than 10K, as shown in Table 5. For DDR3 modules at the system-level, the minimal hammer count is 230K, which is much more than prior reported number of 22.4K [18].

We also re-examine the minimal hammer count of the SODIMM on both the Xilinx ML605 and Thinkpad. Specifically, the respective minimal hammer count is 170 K at the system level and 90 K at the FPGA level, validating our observation that the minimal hammer count on DDR3 in the system-level is much more that in the FPGA-level.

4.5 Multi-thread

Previous works spawn multiple threads for hammer to improve RowHammer effectiveness [8], [13], [26], [27]. We divide these works into two categories, i.e., each thread hammers all aggressor rows [8], [13], and each thread hammers some of aggressor rows [27]. We re-examine these two categories on our test platforms. On DDR3 modules and certain DDR4 module TRR-unequipped (i.e., M1), all of them improve bit-flip rate as previously reported [8], [13], [26], [27]. While on DDR4 modules that support TRR, single-thread is much better as multi-thread hammer rarely induces bit flips. Take M0 as an example, the bit-flip rate of 2-thread hammer in the first category is 2.8% while it is more than 70% for the single-thread hammer. Also, all the tested multi-thread (i.e., 2, 3, 4) hammer of the second-category and more-thread (i.e., 3, 4) hammer of first-category do not flip any bit. On other TRR-employed DRAM modules (i.e., M2 and M3), they have similar bit-flip rates as M0. This is probably because leveraging multiple threads for many-sided hammer will asynchronously issue memory accesses and interfere with TRR's sampler [30], triggering additional `REF` commands issued to victim rows. A hammer thread should carefully synchronize with others to order the whole memory reads by the addresses in memory controller's read queue. We note that additional instructions required for the synchronization (e.g., lock and semaphore) can delay the memory reads, thus badly affecting the hammering efficiency. Thus, for TRR-protected DRAM modules, the single-sided hammer is probably better.

Observation 7: *On TRR-protected DDR4 modules, multi-thread hammer is not as effective as previously reported [8], [13], [26], [27] on DDR3 modules.*

4.6 Bit-flip Direction

The data scrambling feature, employed by the modern memory controller, applies pseudo-random patterns on the DDR data bus to minimize the impact of resonant frequency and cold-boot attacks [24]. Particularly, a DRAM cell's value, visible to the software, is the XORed output of the cell's logical value and a pseudo-random number generated by the data scrambling when the system boots up.

To verify whether the data scrambling is used in practice, we perform the following analysis of the RowHammer test results. From the collected vulnerable physical addresses, we select addresses that are monotonically flipped from '0' to '1' using the inverse "RowStripe" data pattern. These selected addresses are only mapped to DRAM anti-cells if the data scrambling is not in place. Then we restart the system and re-launch the test with both "RowStripe" and its inverse against the selected addresses. The results show that the bit-flip rate for the two data patterns is almost equal in all the tested DRAM modules and these addresses are flippable from either '0' to '1' or '1' to '0'. Due to data scrambling, both true-cell and anti-cell can be flipped in both directions at the system level.

With the above conclusion, we observe that the security guarantee of CTA [23] does not hold. Specifically, CTA (Cell-Type-Aware) [23] employs a two-step approach for protecting page tables from rowhammer attacks. In the first step, CTA puts all page tables into a dedicated region of the

physical memory. The physical addresses containing page-table pages are higher than that of user pages. In the second step, CTA ensures that these addresses are mapped to true cells which can be flipped monotonically from '1' to '0'. In the case of a bit flip in true-cells storing a page table entry (PTE), the new address pointed by the PTE will only be lower than the original address, thus the bit flip cannot change the PTE from pointing to a user page to pointing to a page-table page. However, with the data scrambling deployed, an attacker can bypass CTA by bit-flipping the PTE from '0' to '1' and gain unfettered access to page tables.

Observation 8: *Data scrambling enforced by the memory controller breaks the security guarantee of CTA [23].*

4.7 Running Environment

When the RowHammer test is running on a text-only terminal environment, the number of bit flips is surprisingly much lower than that with a Gnome Desktop environment (other parameters are the same). Take M2 as an example, the bit-flip rate produced on the text-only terminal environment is less than 1% while it is almost 70% on the Gnome Desktop environment. This unexpected case is reproducible on all tested DRAM modules, implying that the effect of running environment is independent on DRAM modules. Further experiment shows that when we implement RowHammer attack, if we run a helper thread to issue continuous memory accesses to an area larger than the Last-Level-Cache size (e.g., 6MB on the i3-10100), the bit-flip rate will surge. We will explore the root cause of RowHammer effectiveness in different running environments using some tools (e.g., HMTT [45]), as discussed in Sec. 6.

Observation 9: *The environment where a RowHammer test runs significantly affects bit flips: the Gnome environment is much more effective than the text-only terminal.*

5 Factors from the DRAM Side

With the extended RowHammer test as the basis, we identify DRAM parameters from DRAM-side that contribute to RowHammer. As the DRAM standards [32], [33], [34] specify numerous DRAM parameters, we focus on DRAM frequency, DRAM supply voltage and DRAM timing parameters (see Table 6) which are closely related to memory performance. We select the `clflush+r` with `mfence` as the default hammer method, "RowStripe" and its inverse as the default data pattern, 1000K as the default hammer count and apply the best hammer pattern to perform the RowHammer test using single thread.

TABLE 6
Major DRAM parameters we examined. ("✓" denotes that the parameter is configurable in BIOS.)

Parameters	Description	DDR4	DDR3
Frequency	DRAM data transfer rate.	✓	✓
Voltage	DRAM supply voltage.	✓	✓
`tCL`	CAS[1] read latency.	✓	✓
`tRCD`	`ACT` to internal read or write delay time.	✓	✓
`tRP`	`PRE` command period.	✓	✓
`tRAS`	`ACT` command to `PRE` command period.	✓	✓
`tRFC`	`REF` cycle time.	✓	✓
`tREFI`	`REF` interval time.	✓	✓
`tWR`	`WR` recovery time.	✓	
`tWTR_S`	Delay from start of internal write transaction to internal read command for different bank group.	✓	
`tWTR_L`	Delay from start of internal write transaction to internal read command for same bank group.	✓	
`tRRD`	`ACT` command to `ACT` command delay.		✓
`tRRD_S`	`ACT` command to `ACT` command delay to different bank group.	✓	
`tRRD_L`	`ACT` command to `ACT` command delay to same bank group.	✓	
`tRTP`	Internal `RD` command to `PRE` command delay.	✓	✓
`tFAW`	Four `ACT` window.	✓	✓
`tCWL`	CAS write latency.	✓	✓
`tCCD_S`	CAS_n to CAS_n delay for different bank group	✓	
`tCCD_L`	CAS_n to CAS_n delay for same bank group	✓	

[1] CAS stands for Column Address Strobe.

TABLE 7
Default DRAM parameters for each modules. Frequency is in the unit of Mega-transfer per second (`MT/s`). Voltage is in the unit of Volt (`V`). Timing parameter is in the unit of clock cycle.

Module	Freq.	Volt.	tRCD-tRP-tRAS-tRFC-tREFI-tWR
M0	2400	1.2	17 - 17 - 39 - 312 - 8316 - 18
M1	2666	1.2	19 - 19 - 43 - 467 - 10400 - 20
M2[1]	4000	1.35	20 - 20 - 40 - 700 - 15600 - 24
M3	3200	1.2	22 - 22 - 52 - 880 - 12480 - 24
M4	1600	1.5	11 - 11 - 28 - 208 - 6240 - [2]
M5[1]	1866	1.5	10 - 9 - 28 - 243 - 7283 - [2]

[1] The module is running with Intel XMP enabled.
[2] The `tWR` parameter cannot be configured from BIOS.

5.1 DRAM Frequency

DRAM modules have an internal clock for synchronization. Modern Double Data Rate (DDR) DRAM uses a single-edged clock to synchronize control and address transmissions, and a dual-edged clock for data transmissions. Thus, some data bits are transmitted on the data bus upon the rising edge of the clock and other bits are upon the falling edge, making the DDR DRAM channel data rate twice its bus clock rate. DRAM frequency denotes the DRAM channel data rate and its default value in each tested DRAM module is shown in Table 7. The unit of a timing parameter can be either nanosecond or clock cycle and the conversion between them is decided by the frequency as follows:

$$nanoseconds = 2 \times cycles/frequency \quad (1)$$

Considering that a higher DRAM frequency enables a faster access rate to a row and might induce more bit flips, we thus examine its effectiveness in triggering bit flips. We only decrease the frequency to quantify its effect in terms of bit-flip rate, as the system may not boot up when the frequency is set larger than the default value. As illustrated in the left plot of Figure 9, the bit-flip rate

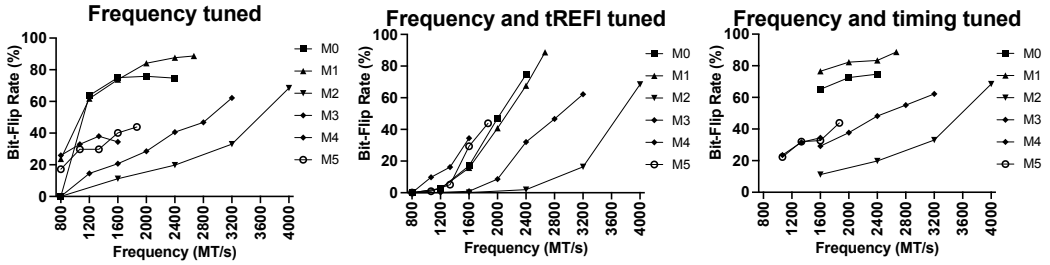


Fig. 9. Bit-flip rate on tested modules when frequency and different timing parameters are tuned. The right plot omits some low frequency cases because the BIOS cannot adjust timings to keep the same memory access latency.

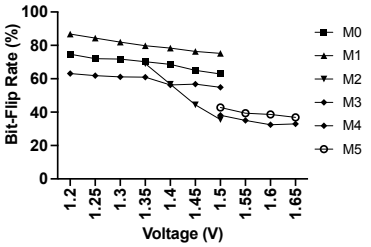


Fig. 10. Bit-flip rate on tested modules when DRAM supply voltage is tuned.

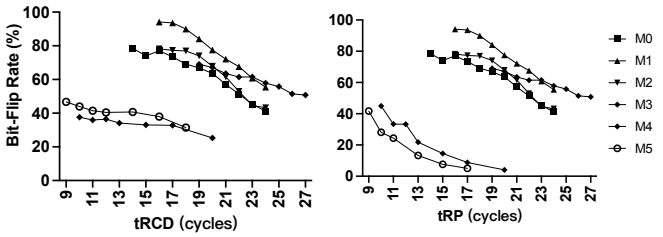


Fig. 11. Bit-flip Rate on tested modules when tRCD or tRP is tuned.

on M1-M5 monotonically decreases when the frequency is reduced with some exception of M0, M4, M5, where the bit-flip rate rises when the frequency drops in some cases. This is probably because that decreasing the frequency will increase the nanoseconds of timing parameters when their clock cycles remain unchanged, based on Equation 1. For these exceptions, the increased DRAM refresh interval has a greater impact on RowHammer than the decreased frequency. We then reduce frequency and the tREFI which determines DRAM refresh interval, finding that bit-flip rate reduces compared to the former as shown in the middle plot of Figure 9, proving this conjecture of refresh interval's disturbance. To evade latency's inference, we reduce DRAM frequency and all timing parameters to keep their nanoseconds unchanged as shown in the right plot of Figure 9. By doing so, the only effect on RowHammer is limited to the data transfer rate, and reducing it monotonically triggers fewer bit flips, because the hammer efficiency is decreased in the fixed refresh interval.

Observation 10: *Higher DRAM frequency triggers more bit flips.*

5.2 DRAM Supply Voltage

Voltage is supplied to the DRAM array and peripheral circuits through the power pins on a DRAM chip [46]. DDR4 is specified to operate at 1.2V [34] and DDR3 is at 1.5V [32], with a small deviation. Considering that RowHammer's electromagnetic coupling effect indirectly drains adjacent cells' charge, we investigate supply voltage's impact on RowHammer. Specifically, we provide the supply voltage from their default values to the maximum safe values, with a stride of 0.05V. For each supply voltage, we perform RowHammer test against each module. As shown in Figure 10, the bit-flip rate decreases when the supply charge increases. When victim cells are accessed, they may be overcharged by the row buffer. Thus, it is harder for them to be drained by electromagnetic coupling effect to lose enough charges and introduce bit flips.

Observation 11: *Higher DRAM supply voltage suppresses bit flips.*

5.3 DRAM Timing Parameters

Besides the frequency and supply voltage, there are numerous timing parameters defined in the DRAM standards [32], [33], [34]. In this section, we focus on 17 major timing parameters in total, which are briefly described in Table 6. We examine 16 timing parameters for DDR4 modules and 10 timing parameters for DDR3 modules, respectively. Specifically, we cannot configure tCCD_S from BIOS on DDR4 platform, otherwise, the system cannot boot up. For the remaining timing parameters, we change each one from a value below default to a rational maximum, most of which is the highest value can be configured. All their values are in the unit of clock cycles in BIOS. In the following, we summarize our empirical results about each timing parameter.

tRCD/tRP. tRCD and tRP are a pair that have the same value on DDR4 platforms, i.e., if either is modified from BIOS, the other will be changed automatically. On DDR3 platforms, they can be updated separately. Prior works [46], [47], [48], [49], [50] clearly demonstrate that reducing tRCD and tRP causes bit flips due to interrupted charge sharing, sense amplification and precharge processes before they are completed, thus benefiting RowHammer. Our experiments results support the demonstration at the system level, as illustrated in Figure 11.

Observation 12: *Lower tRCD/tRP contributes to RowHammer bit flips.*

tRAS. Similar to tRCD/tRP, reducing tRAS improves the RowHammer effectiveness on almost all the tested modules, probably due to the improved ACT rate or the reduced retention time caused by partially-refreshed or non-refreshed DRAM cells. An exception is M3 where reducing tRAS

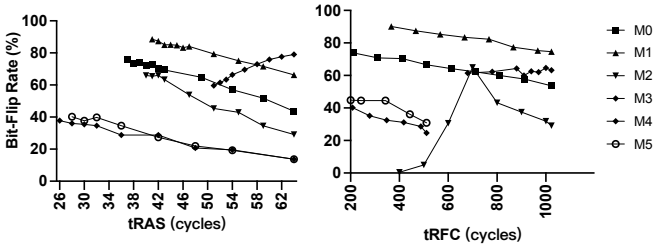


Fig. 12. Bit-flip Rate on tested modules when tRAS or tRFC is tuned.

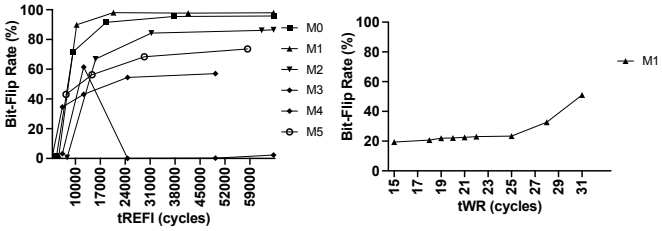


Fig. 13. Bit-flip Rate on tested modules when tREFI or tWR is tuned.

unexpectedly decreases the bit-flip rate that is shown in the left plot of Figure 12. This is probably because a lower tRAS cannot offer enough charge to M3, resulting in a weaker electromagnetic coupling effect than that of a higher tRAS.

Observation 13: *Lower tRAS contributes to bit flips.*

tRFC. tRFC decides the time period of refreshing a set of rows within a bank. Only after tRFC can the memory controller issue a valid command to DRAM[6], resulting in an interval of no memory access. Thus, increasing tRFC is expected to increase memory access latency and interfere with REF, further decrease hammer effectiveness as tRAS. The experimental results on M0 and M1 indeed validate our expectation, shown in the right plot of Figure 12. However, lower tRFC on M2 works the opposite, probably due to M2's different internal circuits and manufacturing process. The reason why tRFC on M3 has no effect is probably because that the value range of tRFC that M3 allows to change from the BIOS is too narrow to affect bit flips.

Observation 14: *Lower tRFC can contribute to bit flips.*

tREFI. tREFI decides the time interval of DRAM refresh. A higher tREFI prolongs the REF command interval and is supposed to induce a higher bit-flip rate. As shown in the left plot of Figure 13, almost all modules we tested work as expected. The only exception is M3 where default tREFI triggers the most bit flips compared to when it is scaled up to 2x, 4x or down to 0.5x, respectively, which is probably because that different TRR mechanisms are switched on upon different tREFI.

Observation 15: *Higher tREFI can contribute to bit flips.*

tWR. As shown in Table 6, tCWL, tWR, tWTR_S and tWTR_L are related to the DRAM WR command, we additionally replace the hammer method of test with clflush+w. However, our DDR3 motherboard cannot read tWR and the clflush+w hammer method only induces bit flips on M1 of all DDR4 modules. From our experiments, tCWL, tWTR_S and tWTR_L do not contribute to RowHammer statistically, while increasing tWR triggers more bit flips as shown in the right plot of Figure 13, different from the other timing parameter. Higher tWR time might overcharge and puncture the parasitic coupling capacitance between aggressor and victim rows, thus opening up the victim cell's access transistor and leaking the charge of the victim's capacitor.

Observation 16: *Higher tWR contributes to bit flips.*

Remaining Timing Parameters. By configuring each of the remaining parameters, we did not observe a clear difference in bit-flip rate.

Observation 17: *The timing parameters in Table 6, except tRCD, tRP, tRAS, tRFC, tREFI and tWR, contribute little to bit flips.*

5.4 Extreme Memory Profile on Rowhammer

Intel proposes XMP for system acceleration [25]. XMP has been widely supported by memory manufacturers, serving as an extension to standard JEDEC SPD specifications. XMP is intended to overclock DRAM and is accessible to users through profiles and predefined overclocking configurations that are known to be stable. Unlike JEDEC, XMP is designed for high performance and usually customized and tweaked to the physical characteristics of the chip.

We conduct our RowHammer test on M2 and M5 that support XMP. By default, M2's frequency is 2133MT/s and M5 is 1600MT/s. When XMP is enabled, M2's frequency is increased to 4000MT/s and M5's is increased to 1866MT/s with some parameters changed. Our results show that the bit-flip rate increases significantly from less than 10% with the default setting to more than 60% with XMP enabled on M2 and from around 10% to more than 40% on M5, implying that XMP is effective in inducing more bit flips.

Observation 18: *The XMP feature, intended for better system performance, might be abused for Rowhammer.*

6 Discussion

Our study has summarized multiple new observations on existing and new factors contributing to RowHammer bit flips. In this study we do not intend to explore the root causes for such effects in an exhaustive manner. For the following observations, we plan to find their root causes and shed more light on RowHammer characterization from the system level.

• In Sec. 4.7 we show that the running environment significantly affects the bit-flip effectiveness. To explain its root cause, a possible way is to collect all the memory traces coming from the Gnome environment and the text-only terminal respectively (e.g., using *HMTT* [45], a commercial hardware tool, to snoop on the memory bus), and perform a detailed analysis of the collected traces.

• Sec. 5 shows that some timing parameters (e.g., tRAS, tRFC and tREFI) do not work as expected on certain DRAM modules and we need more low-level experiments to explain the root causes behind these anomalies. A possible approach is to obtain and analyze the memory-access

6. In fact, only the DES (Device Deselected) command can be issued within tRFC [34].

information by capturing DRAM commands issued to targeted DRAM banks.

7 Conclusion

Previous studies have identified several factors that contribute to RowHammer bit flips such as data pattern and hammer method. As these works mainly relied on FPGA-based test platforms to characterize RowHammer, their findings on the identified factors may not work in a real-world computing system where the OS and the memory controller inevitably interfere. In this paper, we presented a system-level empirical study on the key factors that affect the RowHammer effectiveness. Our study reported some new observations from both the software and DRAM side, which we believe can benefit future RowHammer research.

References

[1] O. Mutlu and J. S. Kim, "Rowhammer: A retrospective," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 39, no. 8, pp. 1555–1571, 2019.

[2] Y. Kim, R. Daly, J. Kim, C. Fallin, J. H. Lee, D. Lee, C. Wilkerson, K. Lai, and O. Mutlu, "Flipping bits in memory without accessing them: an experimental study of DRAM disturbance errors," in *International Symposium on Computer Architecture*, 2014, p. 361–372.

[3] Y. Cheng, Z. Zhang, S. Nepal, and Z. Wang, "CATTmew: Defeating software-only physical kernel isolation," *IEEE Transactions on Dependable and Secure Computing*, 2019.

[4] P. Frigo, C. Giuffrida, H. Bos, and K. Razavi, "Grand pwning unit: accelerating microarchitectural attacks with the GPU," in *IEEE Symposium on Security and Privacy*, 2018.

[5] D. Gruss, M. Lipp, M. Schwarz, D. Genkin, J. Juffinger, S. O'Connell, W. Schoechl, and Y. Yarom, "Another flip in the wall of rowhammer defenses," in *IEEE Symposium on Security and Privacy*, 2018, pp. 245–261.

[6] D. Gruss, C. Maurice, and S. Mangard, "Rowhammer.js: A remote software-induced fault attack in JavaScript," in *Detection of Intrusions and Malware, and Vulnerability Assessment*, 2016, pp. 300–321.

[7] M. Seaborn and T. Dullien, "Exploiting the DRAM rowhammer bug to gain kernel privileges," in *Black Hat'15*, 2015.

[8] A. Tatar, R. K. Konoth, E. Athanasopoulos, C. Giuffrida, H. Bos, and K. Razavi, "Throwhammer: Rowhammer attacks over the network and defenses," in *USENIX Annual Technical Conference*, 2018.

[9] Z. Zhang, Y. Cheng, D. Liu, S. Nepal, Z. Wang, and Y. Yarom, "Pthammer: Cross-user-kernel-boundary rowhammer through implicit accesses," in *International Symposium on Microarchitecture*, 2020.

[10] Z. Zhang, W. He, Y. Cheng, W. Wang, Y. Gao, D. Liu, K. Li, S. Nepal, A. Fu, and Y. Zou, "Implicit hammer: Cross-privilege-boundary rowhammer through implicit accesses," *IEEE Transactions on Dependable and Secure Computing*, 2022.

[11] F. de Ridder, P. Frigo, E. Vannacci, H. Bos, C. Giuffrida, and K. Razavi, "{SMASH}: Synchronized many-sided rowhammer attacks from javascript," in *USENIX Security Symposium*, 2021.

[12] R. Qiao and M. Seaborn, "A new approach for rowhammer attacks," in *Hardware Oriented Security and Trust*, 2016, pp. 161–166.

[13] Y. Jang, J. Lee, S. Lee, and T. Kim, "Sgx-bomb: Locking down the processor via rowhammer attack," in *Proceedings of the 2nd Workshop on System Software for Trusted Execution*, 2017, pp. 1–6.

[14] A. Kwong, D. Genkin, D. Gruss, and Y. Yarom, "RAMBleed: Reading bits in memory without accessing them," in *IEEE Symposium on Security and Privacy*, 2020.

[15] K. Razavi, B. Gras, E. Bosman, B. Preneel, C. Giuffrida, and H. Bos, "Flip Feng Shui: Hammering a needle in the software stack," in *USENIX Security Symposium*, 2016, pp. 1–18.

[16] L. Cojocar, J. Kim, M. Patel, L. Tsai, S. Saroiu, A. Wolman, and O. Mutlu, "Are we susceptible to rowhammer? an end-to-end methodology for cloud providers," in *IEEE Symposium on Security and Privacy*, May 2020.

[17] Y. Jiang, H. Zhu, D. Sullivan, X. Guo, X. Zhang, and Y. Jin, "Quantifying rowhammer vulnerability for dram security," in *Design Automation Conference*, 2021.

[18] J. S. Kim, M. Patel, A. G. Yaglikci, H. Hassan, R. Azizi, L. Orosa, and O. Mutlu, "Revisiting rowhammer: An experimental analysis of modern dram devices and mitigation techniques," in *International Symposium on Computer Architecture*, 2020.

[19] K. Park, S. Baeg, S. Wen, and R. Wong, "Active-precharge hammering on a row induced failure in ddr3 sdrams under 3× nm technology," in *2014 IEEE International Integrated Reliability Workshop Final Report (IIRW)*. IEEE, 2014, pp. 82–85.

[20] K. Park, C. Lim, D. Yun, and S. Baeg, "Experiments and root cause analysis for active-precharge hammering fault in ddr3 sdram under 3× nm technology," *Microelectronics reliability*, vol. 57, pp. 39–46, 2016.

[21] K. Park, D. Yun, and S. Baeg, "Statistical distributions of row-hammering induced failures in ddr3 components," *Microelectronics Reliability*, vol. 67, pp. 143–149, 2016.

[22] J. Liu, B. Jaiyen, R. Veras, and O. Mutlu, "Raidr: Retention-aware intelligent dram refresh," in *International Symposium on Computer Architecture*, 2012, pp. 1–12.

[23] X.-C. Wu, T. Sherwood, F. T. Chong, and Y. Li, "Protecting page tables from rowhammer attacks using monotonic pointers in DRAM true-cells," in *Architectural Support for Programming Languages and Operating Systems*, 2019, pp. 645–657.

[24] J. A. Halderman, S. D. Schoen, N. Heninger, W. Clarkson, W. Paul, J. A. Calandrino, A. J. Feldman, J. Appelbaum, and E. W. Felten, "Lest we remember: cold-boot attacks on encryption keys," *Communications of the ACM*, vol. 52, no. 5, pp. 91–98, 2009.

[25] Intel Corporation, "Intel® extreme memory profile (intel® xmp) and overclock ram," https://www.intel.com/content/www/us/en/gaming/extreme-memory-profile-xmp.html.

[26] M. Lanteigne, "How rowhammer could be used to exploit weaknesses in computer hardware," *SEMICON China*, 2016.

[27] Z. Zhang, Z. Zhan, D. Balasubramanian, X. Koutsoukos, and G. Karsai, "Triggering rowhammer hardware faults on arm: A revisit," in *Proceedings of the 2018 Workshop on Attacks and Solutions in Hardware Security*, 2018, pp. 24–33.

[28] M. Wang, Z. Zhang, Y. Cheng, and S. Nepal, "Dramdig: A knowledge-assisted tool to uncover dram address mapping," in *Design Automation Conference*, 2020.

[29] H. Hassan, N. Vijaykumar, S. Khan, S. Ghose, K. Chang, G. Pekhimenko, D. Lee, O. Ergin, and O. Mutlu, "Softmc: A flexible and practical open-source infrastructure for enabling experimental dram studies," in *High Performance Computer Architecture*, 2017, pp. 241–252.

[30] P. Frigo, E. Vannacci, H. Hassan, V. van der Veen, O. Mutlu, C. Giuffrida, H. Bos, and K. Razavi, "TRRespass: Exploiting the many sides of target row refresh," in *IEEE Symposium on Security and Privacy*, 2020.

[31] Micron, Inc., "DDR4 SDRAM Datasheet," https://www.micron.com/products/dram/ddr4-sdram/, 2015.

[32] JEDEC Solid State Technology Association., "Ddr3 sdram standard," https://www.jedec.org/standards-documents/docs/jesd-79-3d, 2012.

[33] ——, "Low power double data rate 4 (LPDDR4)," https://www.jedec.org/standards-documents/docs/jesd209-4b, 2015.

[34] ——, "Ddr4 sdram standard," https://www.jedec.org/standards-documents/docs/jesd79-4a, 2017.

[35] Z. Zhang, J. Qi, Y. Cheng, S. Jiang, Y. Lin, Y. Gao, S. Nepal, Y. Zou, J. Zhang, and Y. Xiang, "A retrospective and futurespective of rowhammer attacks and defenses on dram," *arXiv preprint arXiv:2201.02986v2*, 2022.

[36] P. Jattke, V. van der Veen, P. Frigo, S. Gunter, and K. Razavi, "Blacksmith: Scalable rowhammering in the frequency domain," in *2022 IEEE Symposium on Security and Privacy (SP)*, vol. 1, 2022.

[37] Z. B. Aweke, S. F. Yitbarek, R. Qiao, R. Das, M. Hicks, Y. Oren, and T. Austin, "ANVIL: Software-based protection against next-generation rowhammer attacks," in *Architectural Support for Programming Languages and Operating Systems*, 2016, pp. 743–755.

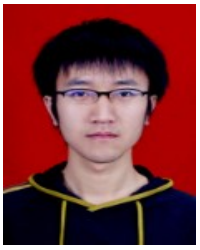
[38] S. Ji, Y. Ko, S. Oh, and J. Kim, "Pinpoint rowhammer: Suppressing unwanted bit flips on rowhammer attacks," in *Asia Conference on Computer and Communications Security*, 2019, pp. 549–560.

[39] Z. Zhang, Z. Zhan, D. Balasubramanian, B. Li, P. Volgyesi, and X. Koutsoukos, "Leveraging EM side-channel information to detect rowhammer attacks," in *IEEE Symposium on Security and Privacy*, 2020.

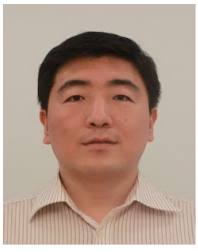
[40] Z. Zhang, Y. Cheng, M. Wang, W. He, W. Wang, N. Surya, Y. Gao, K. Li, Z. Wang, and C. Wu, "Softtrr: Protect page tables against

rowhammer attacks using software-only target row refresh," in *USENIX Annual Technical Conference*, 2022.
[41] D. Gruss, C. Maurice, and S. Mangard, "Program for testing for the DRAM rowhammer problem using eviction," https://github.com/IAIK/rowhammerjs, May 2017.
[42] M. Lipp, M. T. Aga, M. Schwarz, D. Gruss, C. Maurice, L. Raab, and L. Lamster, "Nethammer: Inducing rowhammer faults through network requests," *arXiv preprint arXiv:1805.04956*, 2018.
[43] V. van der Veen, Y. Fratantonio, M. Lindorfer, D. Gruss, C. Maurice, G. Vigna, H. Bos, K. Razavi, and C. Giuffrida, "Drammer: Deterministic rowhammer attacks on mobile platforms," in *ACM SIGSAC Conference on Computer and Communications Security*, 2016, pp. 1675–1689.
[44] Intel, Inc., "Intel xeon processor e5-2600 product family uncore performance monitoring guide," 2012.
[45] C. A. o. S. Institute of Computer Technology, "Hmtt: Hybrid memory trace toolkit," http://asg.ict.ac.cn/hmtt.
[46] K. K. Chang, A. G. Yağlıkçı, S. Ghose, A. Agrawal, N. Chatterjee, A. Kashyap, D. Lee, M. O'Connor, H. Hassan, and O. Mutlu, "Understanding reduced-voltage operation in modern dram devices: Experimental characterization, analysis, and mechanisms," *Proceedings of the ACM on Measurement and Analysis of Computing Systems*, vol. 1, no. 1, pp. 1–42, 2017.
[47] K. K. Chang, A. Kashyap, H. Hassan, S. Ghose, K. Hsieh, D. Lee, T. Li, G. Pekhimenko, S. Khan, and O. Mutlu, "Understanding latency variation in modern dram chips: Experimental characterization, analysis, and optimization," in *Proceedings of the 2016 ACM SIGMETRICS International Conference on Measurement and Modeling of Computer Science*, 2016, pp. 323–336.
[48] J. Kim, M. Patel, H. Hassan, and O. Mutlu, "Solar-dram: Reducing dram access latency by exploiting the variation in local bitlines," in *2018 IEEE 36th International Conference on Computer Design (ICCD)*. IEEE, 2018, pp. 282–291.
[49] D. Lee, S. Khan, L. Subramanian, S. Ghose, R. Ausavarungnirun, G. Pekhimenko, V. Seshadri, and O. Mutlu, "Design-induced latency variation in modern dram chips: Characterization, analysis, and latency reduction mechanisms," *Proceedings of the ACM on Measurement and Analysis of Computing Systems*, vol. 1, no. 1, pp. 1–36, 2017.
[50] D. Lee, Y. Kim, G. Pekhimenko, S. Khan, V. Seshadri, K. Chang, and O. Mutlu, "Adaptive-latency dram: Optimizing dram timing for the common-case," in *2015 IEEE 21st International Symposium on High Performance Computer Architecture (HPCA)*. IEEE, 2015, pp. 489–501.

Wei He received the B.S. degree from Tongji University, and now is a Ph.D. candidate at the University of Chinese Academy of Sciences and SKLOIS, Institute of Information Engineering, CAS. His research interests include system security.



Zhi Zhang is a lecturer in the University of Western Australia. Prior to that, he was a Research Scientist at CSIRO's Data61, Australia. He received his Ph.D. in Computer Science from the University of New South Wales. His research interests are in the areas of system security, rowhammer and adversarial artificial intelligence.



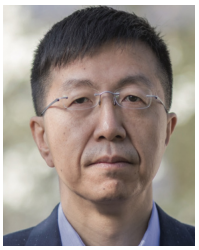
Yueqiang Cheng is head of security research at NIO. He earned his Ph.D. degree in School of Information Systems from Singapore Management University under the guidance of Professor Robert H. Deng and Associate Professor Xuhua Ding. His research interests are system security, trustworthy computing, software-only root of trust and software security.

Wenhao Wang received the B.S. degree from Ocean University of China in 2009, and the Ph.D. degree from University of Chinese Academy of Sciences in 2015. He is an Associate Professor in Institute of Information Engineering, Chinese Academy of Sciences. His research interests include system security, trusted execution environment and cryptography.

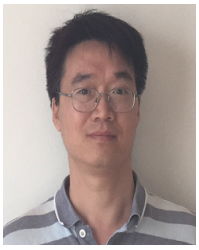
Wei Song received the Ph.D. degree in computer science from the University of Manchester, Manchester, U.K., in 2011. He is currently an Associate Professor with the Institute of Information Engineering, CAS. His current research focuses on the security enhancement of computer architectures, such as the defenses for cache side channel and control-flow hijacking attacks.

Yansong Gao received his M.Sc degree from University of Electronic Science and Technology of China in 2013 and Ph.D degree from the School of Electrical and Electronic Engineering in the University of Adelaide, Australia, in 2017. His current research interests are AI security and privacy, hardware security and system security.

Qifei Zhang serves as deputy director of IoT and Intelligent Computing center in School of Software, Zhejiang University. He is a member of Association Committee on IoT and Embedded System of Zhejiang Province, China and a member of Expert Committee on intelligent manufacturing of Zhejiang Province, China. He received his Ph.D. in Computer Science from Zhejiang University. His main research interests include System Security, IoT Application.



Kang Li is the director of Baidu X-Lab. He received his B.S degree from Tsinghua University, Master degree from Yale Law School, and Ph.D degree from Oregon Graduate Institute at OHSU. His research interests are system security and privacy.



Dongxi Liu is a Principal Research Scientist in CSIRO. His research interests include applied cryptography, light weight encryption, and system security. His recent work aims to design public key encryption based on checkable hardness facts and design new proof-of-work blockchain protocol for crowdmining.

Surya Nepal is a Senior Principal Research Scientist at CSIRO Data61 and leads the distributed system security research group. His main research focus has been in the area of distributed systems, with a specific focus on security, privacy and trust. He has more than 200 peer-reviewed publications to his credit. He currently serves as an associate editor in IEEE Transactions on Service Computing and IEEE Transactions on Dependable and Secure Computing.