Understanding the Security Benefits and Overheads of Emerging Industry Solutions to DRAM Read Disturbance

Oğuzhan Canpolat[§†] A. Giray Yağlıkçı[§] Geraldo F. Oliveira[§] Ataberk Olgun[§]
Oğuz Ergin[†] Onur Mutlu[§]
[§]*ETH Zürich* [†]*TOBB University of Economics and Technology*

We present the first rigorous security, performance, energy, and cost analyses of the state-of-the-art on-DRAM-die read disturbance mitigation method, widely known as Per Row Activation Counting (PRAC), with respect to its description in the updated (as of April 2024) JEDEC DDR5 specification. Unlike prior state-of-the-art that advises the memory controller to periodically issue a DRAM command called refresh management (RFM), which provides the DRAM chip with time to perform its countermeasures, PRAC introduces a new back-off signal. PRAC's back-off signal propagates from the DRAM chip to the memory controller and forces the memory controller to 1) stop serving requests and 2) issue RFM commands. As a result, RFM commands are issued only when needed as opposed to periodically, reducing the performance overhead of RFM. We analyze PRAC in four steps. First, we define a security-oriented adversarial access pattern that represents the worst-case for the security of PRAC. Second, we investigate PRAC's different configurations and their security implications. Our security analyses show that PRAC can be configured for secure operation as long as no bitflip occurs before accessing a memory location 20 times. Third, we evaluate the performance impact of PRAC and compare it against prior works using an open-source cycle-level simulator, Ramulator 2.0. Our performance analysis shows that while PRAC incurs less than 13% performance overhead on benign applications for today's DRAM chips, its performance overheads can reach up to 94% (85% on average across 60 workloads) for future DRAM chips that are more vulnerable to read disturbance bitflips. Fourth, we define an availability-oriented adversarial access pattern that exacerbates PRAC's performance overhead to perform a memory performance attack, demonstrating that such an adversarial pattern can hog up to 94% of DRAM throughput and degrade system throughput by up to 95% (87% on average). We discuss PRAC's implications on future systems and foreshadow future research directions. To aid future research, we open-source our implementations and scripts at `https://github.com/CMU-SAFARI/ramulator2`.

1. Introduction

To ensure system robustness (including reliability, security, and safety), it is critical to maintain memory isolation: accessing a memory address should *not* cause unintended side-effects on data stored on other addresses [1]. Unfortunately, with aggressive technology scaling, DRAM [2], the prevalent main memory technology, suffers from increased *read disturbance*: accessing (reading) a row of DRAM cells (i.e., a DRAM row) degrades the data integrity of other physically close but *unaccessed* DRAM rows. RowHammer [1] is a prime example of DRAM read disturbance, where a DRAM row (i.e., victim row) can experience bitflips when at least one nearby DRAM row (i.e., aggressor row) is repeatedly activated (i.e., hammered) [1, 3–69] more times than a threshold, called the *minimum hammer count to induce the first bitflip* (N_{RH}). *RowPress* [70] is another prime example of DRAM read disturbance that amplifies the effect of RowHammer and consequently reduces N_{RH}.

A simple way of mitigating DRAM read disturbance is to preventively refresh potential victim rows before bitflips occur. Doing so comes at the cost of potential performance degradation [1, 36, 42, 71–91]. To provide DRAM chips with the necessary flexibility to perform preventive refreshes in a timely manner, recent DRAM standards (e.g., DDR5 [92, 93]) introduce 1) a command called *refresh management (RFM)* [92] and 2) a mechanism called *Per Row Activation Counting (PRAC)* [93]. RFM is a DRAM command that provides the DRAM chip with a time window (e.g., 195 ns [93]) to perform preventive refreshes. Specifications before 2024 (e.g., DDR5 [92]) advise the memory controller to issue RFM when the number of row activations in a bank or a logical memory region exceeds a threshold (e.g., 32 [93]). A recent update as of April 2024 of the JEDEC DDR5 specification [93, 94] introduces a new on-DRAM-die read disturbance mitigation mechanism called PRAC. PRAC has two key features. First, PRAC maintains an activation counter per DRAM row [1] to accurately identify when a preventive refresh is needed. PRAC increments a DRAM row's activation counter while the row is being closed, which increases the latency of closing a row, i.e., the precharge latency (t_{RP}) and row cycle time (t_{RC}) timing parameters. Second, PRAC proposes a new *back-off* signal to convey the need for preventive refreshes from the DRAM chip to the memory controller, similar to what prior works propose [76, 85, 87, 95, 96]. The DRAM chip asserts this back-off signal when a DRAM row's activation count reaches a critical value. Within a predefined time window (e.g., 180 ns [93]) after receiving the back-off signal, the memory controller has to issue an RFM command so that the DRAM chip can perform the necessary preventive refresh operations. PRAC aims to 1) avoid read disturbance bitflips by performing necessary preventive refreshes in a timely manner and 2) minimize unnecessary preventive refreshes by accurately tracking each row's activation count. Unfortunately, *no* prior work rigorously investigates the impact of PRAC on security, performance, energy, and cost for modern and future systems.

This paper performs the first rigorous analysis of PRAC in four steps. First, we define a security-oriented adversarial access pattern that achieves the highest possible activation count in systems protected by PRAC. Second, we conduct a security analysis by evaluating the highest possible activation count that a DRAM row can reach under different configurations of PRAC. Our analysis shows that PRAC can be configured for secure operation against an N_{RH} value of 20 or higher. Third, we evaluate the impact of PRAC on performance and energy using Ramulator 2.0 [97, 98], an open-source cycle-level simulator extended with DRAMPower [99]. Our results across 60 different four-core multiprogrammed benign workload mixes show that PRAC incurs an average (maximum) 9.9% (13.1%) system performance and 18.5% (22.7%) DRAM energy overheads for modern DRAM chips with any of the N_{RH} values of 10K [61], 4.8K [61], and 1K [70]. These overheads are similar across the specified N_{RH} values because they are mainly a result of the increased critical DRAM access latencies (i.e., t_{RP} and t_{RC}). PRAC's average (maximum) overheads reach 10.1% (13.3%), 11.8% (15.7%), and 84.7% (94.0%) for performance and 18.8% (22.9%), 20.8% (25.7%), and 13x (18x) for DRAM energy on future DRAM chips with N_{RH} values of 128, 64, and 20, respectively. We attribute these large overheads to many preventive refresh operations being performed (even under benign workloads) as N_{RH} values decrease. We compare PRAC to three state-of-the-art mitigation mechanisms: 1) Graphene [75], 2) Hydra [86], and 3) PARA [1]. Our results across 60 different four-core multiprogrammed benign workloads show that PRAC performs 1) better than PARA at N_{RH} values lower than 1K and 2) comparably to Graphene and Hydra at N_{RH} values lower than 256, as it performs preventive refreshes less aggressively, i.e., when a row activation counter gets close to N_{RH}. Fourth, we define an availability-oriented adversarial access pattern that exacerbates the performance overhead of PRAC to perform a memory performance attack and show that this adversarial access pattern 1) hogs up to 94% of DRAM throughput and 2) reduces system performance by up to 94.5% (86.8% on average) across 60 workloads.

We make the following contributions:

- We present the first security analysis of PRAC and provide robust PRAC configurations against its worst-case access pattern.
- We rigorously evaluate the performance, energy, and cost implications of PRAC's different configurations for modern and future DRAM chips. Our results show that PRAC incurs non-negligible overheads, even for DRAM chips with N_{RH} values higher than 1K, because it increases critical DRAM timing parameters.
- We compare PRAC to three state-of-the-art mitigation mechanisms for modern and future DRAM chips. Our results show that PRAC 1) underperforms against two of the three mitigation mechanisms for modern DRAM chips with relatively high (i.e., $\geq$1K) N_{RH} values and 2) performs comparably to all three mitigation mechanisms for future DRAM chips with lower N_{RH} values, because it performs preventive refreshes in a timely manner.
- We mathematically and empirically show that an attacker can exploit PRAC's preventive refreshes to mount memory performance attacks [100–106] and hog a large fraction of DRAM throughput, which in turn, significantly degrades system performance.
- To aid future research in a transparent manner, we open-source our implementations and scripts at `https://github.com/CMU-SAFARI/ramulator2`.

2. Background & Motivation

Organization. A memory channel connects the processor to a set of DRAM chips, called *DRAM rank*. Each chip has multiple DRAM banks, in which DRAM cells are organized as a two-dimensional array of rows and columns.

Operation. The memory controller serves memory access requests with four main DRAM commands [92, 107–115]. First, the memory controller issues an *ACT* command alongside the bank address and row address corresponding to the memory request's address, which opens (activates) one DRAM row in a DRAM bank. Second, the memory controller can read/write data from/to an activated row using *RD*/*WR* commands. Third, to access another row in an already activated DRAM bank, the memory controller must issue a *PRE* command to close the opened row and prepare the bank for a new activation. The memory controller obeys many timing constraints to guarantee correct operation [93, 113–115]. Three constraints on minimum delay between commands are 1) charge restoration latency (t_{RAS}), from an *ACT* command to the next *PRE* command; 2) t_{RP}, from a *PRE* command to the next *ACT* command; and 3) t_{RC}, between two *ACT* commands as the sum of t_{RAS} and t_{RP}.

Refresh. To maintain data integrity, a DRAM cell is periodically refreshed [116–118] with a time interval called the *refresh window (t_{REFW})*, which is typically 64 ms (e.g., [111, 112, 119]) or 32 ms (e.g., [92, 107, 108]). The memory controller periodically issues a refresh (REF) command with a time interval called the *refresh interval (t_{REFI})*, typically 7.8 µs (e.g., [111, 112, 119]) or 3.9 µs (e.g., [92, 107, 108]). When a rank- or bank-level refresh [118] is issued, the DRAM chip internally refreshes several DRAM rows, during which the whole rank or bank is busy. This operation's latency is called the *refresh latency (t_{RFC})*.

Read Disturbance. Read disturbance is the phenomenon that reading data from a memory device causes electrical disturbance on another piece of data that is *not* accessed but physically located nearby the accessed data. Two prime examples of read disturbance in modern DRAM chips are RowHammer [1] and RowPress [70], where repeatedly activating (i.e., hammering) or keeping active (i.e., pressing) a DRAM row induces bitflips in physically nearby DRAM rows. In RowHammer and RowPress terminology, a row that is hammered or pressed is called the *aggressor* row, and the row that experiences bitflips the *victim* row. For read disturbance bitflips to occur, 1) an aggressor row needs to be activated more than a certain threshold value, defined as N_{RH} [1] and/or 2) the time that an aggressor row stays active, i.e., aggressor row's on-time (t_{AggOn}) [70] needs to be large-enough [70]. One way to avoid read disturbance is to identify potential aggressor rows and preventively refresh their potential victim rows [1,36,42,71–91].

PRAC. Various prior works discuss the use of per-row activation counters to detect how many times each row in DRAM is activated within a refresh interval [1, 77, 85, 95, 120]. A recent update (as of April 2024) of the JEDEC DDR5 specification [93,94] introduces a similar on-DRAM-die read disturbance mitigation mechanism called PRAC (explained in §3), which aims to ensure robust operation at low overhead by preventively refreshing victim rows when necessary. Although PRAC is a promising DRAM specification advancement, *no* prior work rigorously analyzes PRAC's impact on security, performance, energy, and cost for modern and future systems.

3. A Brief Summary of RFM and PRAC

This section briefly explains the RFM command, PRAC mechanism, and assumptions we use for our evaluations.

RFM Command. RFM is a DRAM command that provides the DRAM chip with a time window (e.g., 195 ns [93]) so that the DRAM chip preventively refreshes potential victim rows. The DRAM chip is responsible for identifying and preventively refreshing potential victim rows, and the memory controller is responsible for issuing RFM commands.

PRAC Overview. PRAC implements an activation counter for each DRAM row, and thus accurately measures the activation counts of *all* rows. When a row's activation count reaches a threshold, the DRAM chip asserts a back-off signal which forces the memory controller to issue an RFM command. The DRAM chip preventively refreshes potential victim rows upon receiving an RFM command.

Assumptions about the PRAC Mechanism. We make two assumptions: 1) PRAC always refreshes potential victims of the row with the maximum activation count during each RFM command (even if the maximum activation count is *not* close to N_{RH})[1] and 2) physically-adjacent DRAM rows can experience bitflips when a DRAM row is activated more than a threshold value, denoted as N_{RH}.

PRAC's Operation and Parameters. PRAC increments the activation count of a DRAM row while the row is being closed (i.e., during precharge), which increases t_{RP} and t_{RC} [93]).[2] The DRAM chip asserts the back-off signal when a row's activation count reaches a fraction of N_{RH}, denoted as the back-off threshold (N_{BO}), where the fraction can be configured to either 70%, 80%, 90%, or 100% [93]. The memory controller receives the back-off signal between the time after a command that closes rows (e.g., precharge or refresh) is issued and a small latency after the same command's completion (e.g., ≈5 ns [93]). The memory controller and the DRAM chip go through three phases when the back-off signal is asserted. First, during *the window of normal traffic* ($t_{ABO_{ACT}}$) [93], the memory controller has a limited time window (e.g., 180 ns [93]) to serve requests after receiving the back-off signal. A DRAM row can receive up to $t_{ABO_{ACT}}/t_{RC}$ activations in this window. Second, during the *recovery period* [93], the memory controller issues a number of RFM commands, which we denote as N_{Ref} (e.g., 1, 2 or 4 [93]). An RFM command can further increment the activation count of a row before its potential victims are refreshed. Third, during the *delay period* or the delay until a new back-off can be initiated ($t_{BackOffDelay}$) [93], the DRAM chip *cannot* reassert the back-off signal until it receives a number of activate (ACT) commands, which we denote as N_{Delay} (e.g., 1, 2 or 4 [93]).[3] Considering these three phases, §5 calculates the highest achievable activation count to any DRAM row in a PRAC-protected system.

RFM and PRAC Implementations. We analyze four different RFM and PRAC implementations: 1) *Periodic RFM (PRFM)*, where the memory controller issues an RFM command *periodically* when the total number of activations to a bank reaches a predefined threshold value called *bank activation threshold to issue an RFM command* (RFM_{th}) with *no* back-off signal from the DRAM chip, as described in early DDR5 standards [92]; 2) *PRAC-N*, where the memory controller issues N back-to-back RFM commands *only* after receiving a back-off signal from the DRAM chip, as described in the latest JEDEC DDR5 standard [93,94]; 3) *PRAC+PRFM*, where the memory controller issues an RFM command *i*) when the total number of activations to a bank reaches RFM_{th} or *ii*) it receives a back-off signal from the DRAM chip. PRAC-N implementations are *not* secure at N_{RH} values lower than 20. Therefore, combining PRAC and PRFM enables security at lower N_{RH} values at the cost of potentially refreshing the victims of aggressor rows whose activation counts are *not* close to N_{RH}; and 4) *PRAC-Optimistic*, which is the same as the default PRAC configuration advised by JEDEC [93] (i.e., PRAC-4) *without* any change in timing parameters (including t_{RP} and t_{RC} [93]).

4. Adversarial Access Pattern: The Wave Attack

Threat Model. To account for the worst case, we assume that the attacker 1) knows the physical layout of DRAM rows (as in [121]), 2) accurately detects when a row is internally refreshed (preventively or periodically as in U-TRR [42]), and 3) precisely times *all* DRAM commands except REF and RFM commands (as in [42,121]).

Overview. The adversarial access pattern aims to achieve the highest number activation count for a given row in a PRAC-protected DRAM chip by overwhelming PRAC using a number of decoy rows, similar to the *wave attack* [85,87]. In this access pattern, the attacker hammers a number of rows in a balanced way, such that PRAC can perform preventive refreshes *only* for a small subset of the hammered rows when an RFM is issued.

[1] The specification does *not* enforce refreshing the victims of the aggressor with the maximum activation count [93].

[2] When PRAC is enabled, activation counters are incremented with internal reads and writes before a row is closed. The counter update causes a delay between the time of receiving a precharge command and *actually* precharging the row. Because of this delay, 1) t_{RP} increases by 21 ns (+140%) and 2) t_{RAS}, t_{RTP}, and t_{WR} reduces by 16 ns (-50%), 2.5 ns (-33%), and 20 ns (-66%) [93]. Combined effect of these timing parameters result in a t_{RC} increase of 5 ns (+10%) (for DDR5-3200AN speed bin [93]).

[3] Current DDR5 specification [93] notes that N_{Ref} and N_{Delay} always have the same value. To comprehensively assess PRAC's security guarantees, we use different values for the two parameters *only* in our security analysis of PRAC (§5).

When an aggressor row's victims are refreshed, the attacker excludes the aggressor row in the next round of activations. By doing so, this adversarial access pattern achieves the highest possible activation count for the row whose victims are preventively refreshed latest.

5. Configuration of PRAC and Security Analysis

This section investigates different RFM and PRAC configurations and their impact on security under the wave attack.

Notation. We denote the set of rows that the wave attack hammers in round i as R_i and the number of rows in R_i as $|R_i|$.

Key Parameters. We assume a *blast radius* of 2 [45], a t_{RC} of 52 ns [93], and a refresh management latency (t_{RFM}) of 350 ns [93], which allows an RFM command to refresh four victim rows of one aggressor row.

PRFM. In round 1, the wave attack hammers each row in R_1 once, causing the memory controller to issue $\lfloor(|R_1|/RFM_{th})\rfloor$ RFM commands, each refreshing the four victims of one aggressor row. In round 2, the wave attack continues hammering the non-refreshed rows R_2, where $|R_2| = |R_1| - \lfloor(|R_1|/RFM_{th})\rfloor$. By repeating this calculation i times, Equation 1 evaluates the number of rows with victims that are not refreshed at an arbitrary round i ($|R_i|$).

$$|R_i| = |R_1| - \left\lfloor \frac{\sum_{k=1}^{i-1} |R_k|}{RFM_{th}} \right\rfloor \qquad (1)$$

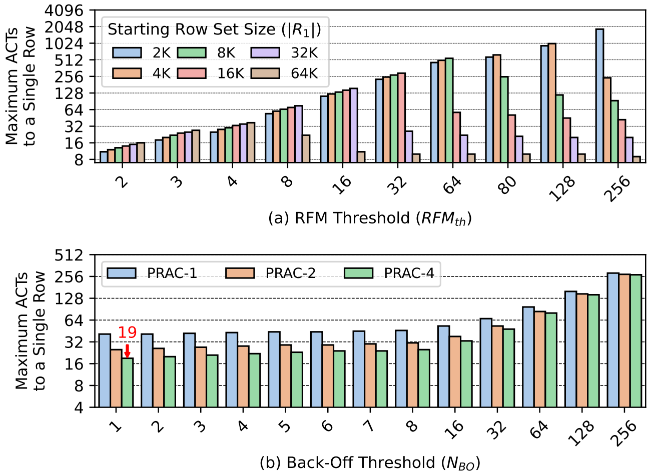
To cause bitflips, the wave attack must make sure that 1) at least one aggressor row's victims are *not* refreshed by an RFM command at round N_{RH}, i.e., $|R_{N_{RH}}| > 0$, and 2) the time taken by the attacker's row activations and RFM preventive refreshes do *not* exceed t_{REFW}, i.e., aggressor's victims are *not* periodically refreshed before being activated N_{RH} times. We rigorously sweep the wave attack's configuration parameters and identify the maximum hammer count of an aggressor row before its victims are refreshed.

PRAC-N. We adapt our PRFM wave attack analysis to PRAC-N by leveraging two key insights: First, PRAC-N mechanism will *not* preventively refresh any row until a row's activation count reaches N_{BO}. We prepare rows in R_1 such that each row is already hammered N_{BO}-1 times. Doing so, the number of rounds necessary to induce a bitflip is reduced by N_{BO}-1. Second, at least one row's activation counter remains above N_{BO} across *all* rounds after initialization until the end of the wave attack. This causes PRAC-N to assert the back-off signal as frequently as possible, i.e., with a time period containing a recovery period ($N_{Ref} \times t_{RFM}$), a delay period ($t_{BackOffDelay}$), and a window of normal traffic ($t_{ABO_{ACT}}$). Leveraging these insights, we update Equation 1 to derive Equation 2.

$$|R_i| = |R_1| - N_{Ref} \times \left\lfloor \frac{\sum_{k=1}^{i-1} |R_k|}{N_{Delay} + (t_{ABO_{ACT}}/t_{RC})} \right\rfloor \qquad (2)$$

For a PRAC-N system to be secure, an attacker should *not* be able to obtain $|R_{N_{RH}-N_{BO}}| > 0$ within t_{REFW} for any R_1. We analyze the maximum hammer count of an aggressor row before its victims are refreshed in a PRAC-N-protected system for a wide set of N_{BO} and $|R_1|$ configurations.

Configuration Sweep. Fig. 1 shows the maximum activation count an aggressor row can reach before its victims are refreshed (y-axis) for PRFM and PRAC-N in Figs. 1a and 1b, respectively. Fig. 1a shows the bank activation threshold to issue an RFM command (RFM_{th}) on the x-axis and starting row set size ($|R1|$) color-coded. Fig. 1b shows the back-off threshold (N_{BO}) on the x-axis and N_{Ref} color-coded.



(a) RFM Threshold (RFM_{th})

(b) Back-Off Threshold (N_{BO})

Figure 1: Maximum activations to a row allowed by (a) PRFM and (b) PRAC-N

From Fig. 1a, we observe that to prevent bitflips for very low N_{RH} values (e.g., 32 on the y axis), RFM_{th} should be configured to very low values (e.g., <4), as only such RFM_{th} values results in activation counts less than N_{RH} for all $|R_1|$ values. From Fig. 1b, we observe that PRAC-N provides security at N_{RH} values as low as 20 (because a row can receive 19 activations as annotated) when configured to 1) trigger a back-off as frequently as possible ($N_{BO} = 1$) and 2) issue four RFMs in the recovery period (i.e., PRAC-4). For the remainder of our study, we assume we can accurately determine N_{RH} and configure PRFM and PRAC protected systems to avoid all bitflips using these secure thresholds (which is a difficult problem in itself, given that determining N_{RH} for every row is not easy, as shown by multiple works [1, 61, 62, 70, 122–124]).

6. Experimental Evaluation

We evaluate PRAC's overheads on system performance, DRAM energy consumption, and DRAM chip area for existing and future DRAM chips, by sweeping N_{RH} from 1K down to 20. We compare PRAC's overheads against three read disturbance mitigation mechanisms: 1) Graphene [75], the state-of-the-art mechanism that maintains row activation counters completely within the processor chip; 2) PARA [1], the state-of-the-art mechanism that does *not* maintain any counters; and 3) Hydra [86], the state-of-the-art mechanism that maintains counters in the DRAM chip and caches them in the processor chip. To evaluate performance and DRAM energy consumption, we conduct cycle-level simulations using Ramulator 2.0 [97, 98], integrated with DRAMPower [99]. We extend Ramulator 2.0 [97, 98] with the implementations of PRAC, RFM, and the back-off signal (as specified in the latest JEDEC DDR5 DRAM specification as of April 2024 [93]). We evaluate system performance using the weighted speedup metric [125, 126].

Table 1 shows our system configuration. We assume a realistic quad-core system, connected to a dual-rank memory with eight bank groups, each containing four banks (64 banks in total). The memory controller employs the FR-FCFS memory scheduler [127, 128] with a Cap on Column-Over-Row Reordering (FR-FCFS+Cap) of four [102]. We extend the memory controller to delay the requests that *cannot* be served within $t_{ABO_{ACT}}$.

Table 1: Simulated System Configuration

Processor	4.2 GHz, 4-core, 4-wide issue, 128-entry instr. window
Last-Level Cache	64-byte cache line, 8-way set-associative, 8 MB
Memory Controller	64-entry read/write request queues; Scheduling policy: FR-FCFS+Cap of 4 [102]; Address mapping: MOP [129]
Main Memory	DDR5 DRAM [98], 1 channel, 2 ranks, 8 bank groups, 4 banks/bank group, 64K rows/bank

Workloads. We evaluate applications from five benchmark suites: SPEC CPU2006 [130], SPEC CPU2017 [131], TPC [132], MediaBench [133], and YCSB [134]. We group all applications into three memory-intensity groups based on their row buffer misses per kilo instructions (RBMPKIs), similar to prior works [135, 136]. These groups are High (H), Medium (M), and Low (L) for the lowest MPKI values of 10, 2, and 0, respectively. Then, we create 60 workload mixes with 10 of each HHHH, MMMM, LLLL, HHMM, MMLL, and LLHH combination types. We simulate each workload mix until all cores execute 100M instructions or 5 billion cycles.

6.1. Performance Evaluation

Fig. 2 presents the performance overheads of the evaluated read disturbance mitigation mechanisms as N_{RH} decreases. Axes respectively show the N_{RH} values (x axis) and system performance (y axis) in terms of weighted speedup [125, 126] normalized to a baseline with *no* read disturbance mitigation (higher y value is better). Different bars identify different read disturbance mitigation mechanisms and red edge color indicates read disturbance vulnerable configurations.

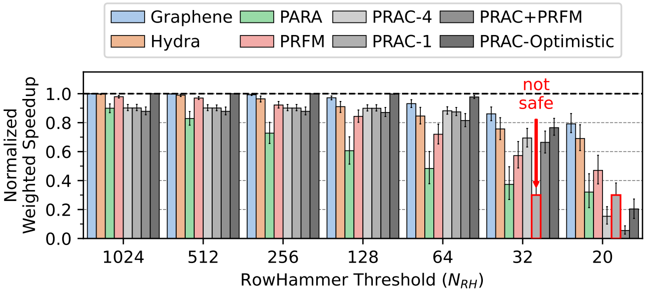


Figure 2: Performance impact of evaluated read disturbance mitigation mechanisms on 60 benign four-core workloads

We make nine observations from Fig. 2. First, as N_{RH} decreases, performance overheads of all studied mitigation mechanisms increases, as expected, due to the more frequent mitigating actions (i.e., preventive refreshes) performed.

Effect of Increased Timing Parameters. Second, at an N_{RH} of 1K, PRAC-4 performs similarly to PARA and is outperformed by Graphene, Hydra, and PRFM averaged across all workloads. We attribute PRAC's non-negligible overhead at relatively high N_{RH} values (i.e., 9.9% average and 13.1% maximum) to PRAC's increased DRAM timing parameters [93] (see §3) as PRAC-Optimistic (i.e., PRAC-4 *without* increased DRAM timing parameters) leads to an average (maximum) system performance overhead of only 0.002% (0.01%) across the same workloads at the same 1K threshold. Third, between N_{RH} values of 1K and 64, PRAC-Optimistic outperforms all evaluated mitigation mechanisms, demonstrating that PRAC without increased timing parameters has good potential.

PRAC Scales Well Until $N_{RH} = 64$. Fourth, when N_{RH} decreases from 1K to 64, PRAC-4's average (maximum) system performance overhead across all workloads increases from 9.9% (13.1%) to 11.8% (15.7%). In contrast, Graphene and Hydra's average (maximum) system performance overheads across all workloads increase from 0.03% (0.1%) and 0.2% (1.1%) to 6.9% (13.0%) and 15.4% (25.9%), respectively. Fifth, between N_{RH} values of 128 and 64, PRAC-4 outperforms Hydra and performs similarly to Graphene. We attribute the relative improvement against Graphene and Hydra as N_{RH} decreases to PRAC's more accurate tracking of aggressor row activations and the resulting less aggressive preventive refreshes performed.

PRAC Overheads Shoot Up at $N_{RH} \leq 64$. Sixth, between N_{RH} values of 64 to 20, PRAC-4 and PRAC-Optimistic's average (maximum) system performance overheads across all workloads significantly increase from 11.8% (15.7%) and 2.3% (4.7%) to 84.7% (94.0%) and 79.6% (90.8%), respectively. We attribute the significant increase in system performance overhead to PRAC performing more frequent preventive refreshes. For example, with PRAC-4 at N_{RH} values of 64 and 20, the four-core benign workload of *523.xalancbmk*, *435.gromacs*, *459.GemsFDTD*, and *434.zeusmp* trigger 11.9 and 129.0 recovery periods per million cycles, resulting in 11.2% and 87.7% system performance overhead, respectively. Seventh, at an N_{RH} of 20, Graphene and Hydra outperform *all* evaluated PRAC and PRFM implementations.

PRFM Performs Poorly. Eighth, when N_{RH} decreases from 1K to 20, the average (maximum) system performance overhead of PRFM increases from 2.1% (4.0%) to 53.0% (68.7%). We attribute this significant overhead increase to PRFM's configuration against the wave attack drastically increasing the frequency of preventive refreshes as N_{RH} decreases, similar to PRAC for N_{RH} between 64 and 20. Ninth, pairing PRAC-4 with PRFM increases PRAC's system performance overhead by an average (maximum) of 24.4% (66.1%) across all N_{RH} values. This is because PRAC's secure configurations (§5) already preventively refresh all rows before they reach a critical level. Therefore, pairing PRAC's secure configurations with PRFM causes performance degradation due to unnecessary preventive refreshes.

We conclude that 1) PRAC's increased DRAM timing parameters incur significant overheads even under infrequent preventive refreshes for modern DRAM chips (i.e., $N_{RH} = 1$K), 2) PRAC shows similar performance to Graphene and outperforms Hydra as N_{RH} values decrease (until 32, where PRAC starts performing significantly worse), 3) PRFM does *not* scale well with decreasing N_{RH} values and incurs significant system performance loss, and 4) pairing PRAC with PRFM provides no system performance advantage.

6.2. Energy Evaluation

Fig. 3 presents the energy consumption of the evaluated read disturbance mitigation mechanisms (y-axis) as N_{RH} decreases (x-axis). Energy consumption is normalized to a baseline with *no* read disturbance mitigation.

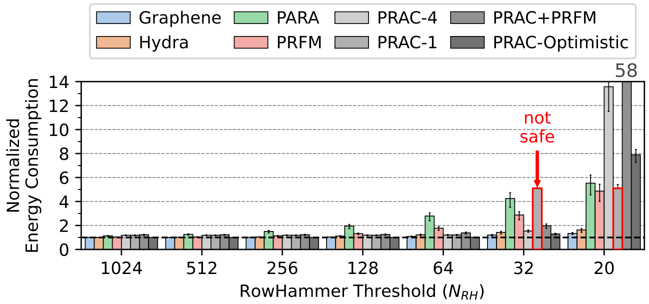


Figure 3: Energy impact of evaluated read disturbance mitigation mechanisms on 60 benign four-core workloads

We make four observations from Fig. 3. First, as N_{RH} decreases, the DRAM energy overhead of all studied mitigation mechanisms increases. Second, when N_{RH} decreases from 1K to 20, PRFM's average DRAM energy overhead increases from 3.2% to 4x. Third, when N_{RH} decreases from 1K to 20, PRAC-4's average DRAM energy overhead significantly increases from 18.5% to 13x. In contrast, PRAC-Optimistic's (i.e., PRAC *without* increased DRAM timing parameters) average DRAM energy overhead increases from 0.001% to 7x. Therefore, a significant portion of PRAC's DRAM energy is likely due to the increased timing parameters. We attribute the high increase in DRAM energy overheads of PRFM and PRAC as N_{RH} decreases to 1) their conservative preventive refresh thresholds against the wave attack and 2) benign applications triggering many preventive refreshes. Fourth, as N_{RH} decreases from 1K to 20, average DRAM energy overheads of Graphene and Hydra increase from 0.01% and 0.3% to 33.2% and 62.7%, respectively.

We conclude that 1) PRAC and PRFM already incur relatively high DRAM energy overheads for modern DRAM chips (i.e., $N_{RH} = 1K$), 2) energy overheads of all evaluated mitigations mechanisms significantly increase for future DRAM chips that are more vulnerable to read disturbance, and 3) Graphene and Hydra outperform all PRAC and PRFM implementations at all evaluated N_{RH} values across all workloads (except PRAC-Optimistic at N_{RH} values higher than 32).

6.3. Storage Evaluation

Fig. 4 presents the storage requirements of the evaluated read disturbance mitigation mechanisms as N_{RH} decreases. Axes respectively show the N_{RH} values (x axis) and storage (y axis) in mebibytes (MiB).

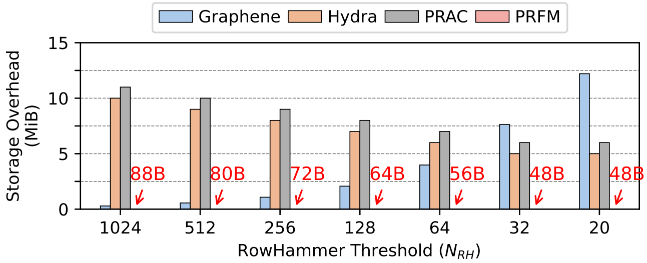


Figure 4: Storage used by evaluated read disturbance mitigation mechanisms as a function of RowHammer threshold

From Fig. 4, we make four observations. First, as N_{RH} decreases from 1K to 20, Graphene's storage overhead in CPU increases significantly (by 50.3x) due to the need to track many more rows. Second, as N_{RH} decreases from 1K to 20, PRAC and Hydra's storage overheads in DRAM reduce by 45.5% and 50.0%, respectively. We note that while Hydra's cache structure in CPU does *not* change, the overall cache size reduces with N_{RH} (by 43.9% from 1K to 20) as smaller cache entries are sufficient to track activations. Fourth, PRFM incurs the least storage overhead in the CPU among the evaluated mitigation techniques as it only requires only one counter per bank.

We conclude that PRAC, PRFM, and Hydra incur low storage overheads and scale well with decreasing N_{RH} values as they either *i*) keep counters in DRAM where a large amount of storage is available at high density or *ii*) require only a small set of counters.

7. Performance Degradation Attack

An attacker can take advantage of PRAC to mount memory performance (or denial of memory service) attacks [137] by triggering many preventive actions (e.g., back-off signals and RFMs). This section presents 1) the theoretical maximum downtime of a PRAC-protected system and 2) simulation results.

Theoretical Analysis. We calculate the maximum possible fraction of time that preventive actions take in a PRAC-protected system. First, triggering a back-off signal takes $N_{BO} \times t_{RC}$, which causes N_{Ref} RFM commands, blocking the bank for a time window of $N_{Ref} \times t_{RFM}$. Therefore, an attacker can block a DRAM bank for $(N_{Ref} \times t_{RFM})/(N_{Ref} \times t_{RFM} + N_{BO} \times t_{RC})$ of time. We configure N_{BO}, N_{Ref}, t_{RFM}, and t_{RC} as 1, 4, 350 ns, and 52 ns against an N_{RH} of 20, based on the DDR5-3200AN DRAM timing constraints specified in the JEDEC standard [93]. We observe that an attacker can *theoretically* consume 94% of DRAM throughput by triggering back-offs.

Simulation. To understand the system performance degradation an attacker could cause by hogging the available DRAM throughput with preventive refreshes, we simulate 60 four-core workload mixes of varying memory intensities where one core maliciously hammers 8 rows in each of 4 banks.[4]

Our results for N_{RH} values of 128, 64, 32, and 20 show that PRAC reduces system performance (based on the weighted

[4] We experimentally found these values to yield the highest performance overhead for PRAC in our system configuration. We open-source our attacker trace generator with the rest of our implementation to aid reproducibility [98].

speedup metric [125, 126]) on average (maximum) by 18.4% (29.0%), 22.9% (34.0%), 49.2% (75.0%), and 86.8% (94.6%) with a maximum slowdown [43, 103] on a single application of 64.5%, 66.8%, 75.0%, and 97.7%, respectively. These results indicate that memory performance attacks can exploit PRAC and future research should tackle PRAC's performance overheads to avoid such denial of service attacks.

8. Summary and Future Research Directions

We show that PRAC ensures secure operation even for very low N_{RH} values (e.g., as low as 20, see §5). However, PRAC still incurs high performance and energy overheads especially at low N_{RH} values (e.g., ≤ 32, see §6.1), which can be maliciously exacerbated to mount memory performance attacks (§7). Therefore, reducing PRAC's performance and energy overheads and avoiding its denial of service vulnerability are still important research problems.

We identify at least four directions to explore. A first direction is to reduce the t_{RP} and t_{RC} timing constraints that increase when PRAC is enabled. These increased DRAM timing parameters incur non-negligible system performance overheads even at high N_{RH} values. This reduction can be done by 1) leveraging large safety margins associated with timing parameters (as shown in [63, 114, 138–152]) or 2) modifying the DRAM circuitry to separate the counters from data arrays to parallelize row activation counter accesses [95]. A second direction is to overlap the latencies of preventive refreshes and other memory operations. A workload triggers more preventive actions as N_{RH} decreases, as even a benign application starts activating DRAM rows too many (i.e., closer to or more than N_{RH}) times. Overlapping the latencies of preventive refreshes is possible by 1) leveraging subarray-level parallelism [113, 118, 147] or 2) eliminating the blocking nature of preventive refreshes [96, 153]. A third direction is to leverage the significant variation in read disturbance vulnerability across DRAM rows to avoid overprotecting the vast majority of the rows [62, 154]. This direction requires profiling a given chip with fast, accurate, and comprehensive (and likely online [117, 155–157]) profiling methodologies, which addresses several aspects, including RowHammer's complex interaction with temperature [46, 62] and new read disturbance phenomena like RowPress [70]. A fourth direction is to defend against malicious attackers that exploit preventive refreshes. Attackers can trigger increasing amounts of preventive refreshes as N_{RH} decreases, allowing a new attack vector to conduct memory performance attacks [137]. Preventing these performance attacks may be possible by accurately detecting and throttling workloads that trigger many preventive refreshes [121, 158].

9. Related Work

This is the first work that rigorously analyses the security and performance of PRAC, a key feature introduced in the latest JEDEC DDR5 DRAM specification [93]. §6.1 qualitatively and quantitatively compares PRAC to several prominent RowHammer mitigation mechanisms [1, 75, 86]. There are various other mitigation mechanisms that can be implemented in the memory controller [1, 22, 29, 36, 42, 57, 69, 71, 74–84, 86–88, 90, 91, 96, 120–122, 147, 159–195] or in the DRAM chip [72, 73, 85, 87, 89, 92, 95, 187, 196–201]. We leave a rigorous comparison of PRAC to this broader set of RowHammer mitigation techniques to future work.

10. Conclusion

We presented the first rigorous security, performance, energy, and cost analyses of Per Row Activation Counting (PRAC), the state-of-the-art RowHammer mitigation technique that is recently adopted by industry in the DDR5 standard [93]. We show that PRAC 1) has non-negligible overheads due to increased DRAM timing parameters for today's DRAM chips, 2) incurs significant system performance and DRAM energy overheads by triggering increasingly more back-off requests for future DRAM chips with higher read disturbance vulnerabilities, 3) can be used as a memory performance attack vector to consume a significant portion of the available DRAM throughput and thus degrade overall system performance, and 4) provides secure operation for N_{RH} values as low as 20.

We conclude that more research is needed to improve PRAC by *i*) reducing the high system performance and DRAM energy overheads due to increased DRAM timing parameters, *ii*) solving the exacerbated performance impact as N_{RH} decreases, and *iii*) stopping its preventive refreshes from being exploited by memory performance attacks.

Acknowledgments

We thank the anonymous reviewers of DRAMSec 2024 for the positive feedback. We thank the SAFARI Research Group members for valuable feedback and the stimulating scientific and intellectual environment they provide. We acknowledge the generous gift funding provided by our industrial partners (especially Google, Huawei, Intel, Microsoft, VMware), which has been instrumental in enabling the research we have been conducting on read disturbance in DRAM since 2011 [202]. This work was also in part supported by the Google Security and Privacy Research Award and the Microsoft Swiss Joint Research Center.

References

[1] 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 *ISCA*, 2014.

[2] R. H. Dennard, "Field-Effect Transistor Memory," U.S. Patent 3,387,286, 1968.

[3] A. P. Fournaris, L. Pocero Fraile, and O. Koufopavlou, "Exploiting Hardware Vulnerabilities to Attack Embedded System Devices: A Survey of Potent Microarchitectural Attacks," *Electronics*, 2017.

[4] D. Poddebniak, J. Somorovsky, S. Schinzel, M. Lochter, and P. Rösler, "Attacking Deterministic Signature Schemes using Fault Attacks," in *EuroS&P*, 2018.

[5] A. Tatar, R. K. Konoth, E. Athanasopoulos, C. Giuffrida, H. Bos, and K. Razavi, "Throwhammer: Rowhammer Attacks Over the Network and Defenses," in *USENIX ATC*, 2018.

[6] S. Carre, M. Desjardins, A. Facon, and S. Guilley, "OpenSSL Bellcore's Protection Helps Fault Attack," in *DSD*, 2018.

[7] A. Barenghi, L. Breveglieri, N. Izzo, and G. Pelosi, "Software-Only Reverse Engineering of Physical DRAM Mappings for Rowhammer Attacks," in *IVSW*, 2018.

[8] Z. Zhang, Z. Zhan, D. Balasubramanian, X. Koutsoukos, and G. Karsai, "Triggering Rowhammer Hardware Faults on ARM: A Revisit," in *ASHES*, 2018.

[9] S. Bhattacharya and D. Mukhopadhyay, "Advanced Fault Attacks in Software: Exploiting the Rowhammer Bug," *Fault Tolerant Architectures for Cryptography and Hardware Security*, 2018.

[10] M. Seaborn and T. Dullien, "Exploiting the DRAM Rowhammer Bug to Gain Kernel Privileges," http://googleprojectzero.blogspot.com.tr/2015/03/exploiting-dram-rowhammer-bug-to-gain.html, 2015.

[11] SAFARI Research Group, "RowHammer — GitHub Repository," https://github.com/CMU-SAFARI/rowhammer.

[12] M. Seaborn and T. Dullien, "Exploiting the DRAM Rowhammer Bug to Gain Kernel Privileges," *Black Hat*, 2015.

[13] 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 *CCS*, 2016.

[14] D. Gruss, C. Maurice, and S. Mangard, "Rowhammer.js: A Remote Software-Induced Fault Attack in Javascript," arXiv:1507.06955 [cs.CR], 2016.

[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*, 2016.

[16] P. Pessl, D. Gruss, C. Maurice, M. Schwarz, and S. Mangard, "DRAMA: Exploiting DRAM Addressing for Cross-CPU Attacks," in *USENIX Security*, 2016.

[17] Y. Xiao, X. Zhang, Y. Zhang, and R. Teodorescu, "One Bit Flips, One Cloud Flops: Cross-VM Row Hammer Attacks and Privilege Escalation," in *USENIX Security*, 2016.

[18] E. Bosman, K. Razavi, H. Bos, and C. Giuffrida, "Dedup Est Machina: Memory Deduplication as An Advanced Exploitation Vector," in *S&P*, 2016.

[19] S. Bhattacharya and D. Mukhopadhyay, "Curious Case of Rowhammer: Flipping Secret Exponent Bits Using Timing Analysis," in *CHES*, 2016.

[20] W. Burleson, O. Mutlu, and M. Tiwari, "Invited: Who is the Major Threat to Tomorrow's Security? You, the Hardware Designer," in *DAC*, 2016.

[21] R. Qiao and M. Seaborn, "A New Approach for RowHammer Attacks," in *HOST*, 2016.

[22] F. Brasser, L. Davi, D. Gens, C. Liebchen, and A.-R. Sadeghi, "Can't Touch This: Software-Only Mitigation Against Rowhammer Attacks Targeting Kernel Memory," in *USENIX Security*, 2017.

[23] Y. Jang, J. Lee, S. Lee, and T. Kim, "SGX-Bomb: Locking Down the Processor via Rowhammer Attack," in *SOSP*, 2017.

[24] M. T. Aga, Z. B. Aweke, and T. Austin, "When Good Protections Go Bad: Exploiting Anti-DoS Measures to Accelerate Rowhammer Attacks," in *HOST*, 2017.

[25] O. Mutlu, "The RowHammer Problem and Other Issues We May Face as Memory Becomes Denser," in *DATE*, 2017.

[26] A. Tatar, C. Giuffrida, H. Bos, and K. Razavi, "Defeating Software Mitigations Against Rowhammer: A Surgical Precision Hammer," in *RAID*, 2018.

[27] 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 *S&P*, 2018.

[28] M. Lipp, M. T. Aga, M. Schwarz, D. Gruss, C. Maurice, L. Raab, and L. Lamster, "Nethammer: Inducing Rowhammer Faults Through Network Requests," arXiv:1805.04956 [cs.CR], 2018.

[29] V. van der Veen, M. Lindorfer, Y. Fratantonio, H. P. Pillai, G. Vigna, C. Kruegel, H. Bos, and K. Razavi, "GuardION: Practical Mitigation of DMA-Based Rowhammer Attacks on ARM," in *DIMVA*, 2018.

[30] P. Frigo, C. Giuffrida, H. Bos, and K. Razavi, "Grand Pwning Unit: Accelerating Microarchitectural Attacks with the GPU," in *S&P*, 2018.

[31] L. Cojocar, K. Razavi, C. Giuffrida, and H. Bos, "Exploiting Correcting Codes: On the Effectiveness of ECC Memory Against Rowhammer Attacks," in *S&P*, 2019.

[32] S. Ji, Y. Ko, S. Oh, and J. Kim, "Pinpoint Rowhammer: Suppressing Unwanted Bit Flips on Rowhammer Attacks," in *ASIACCS*, 2019.

[33] O. Mutlu and J. S. Kim, "RowHammer: A Retrospective," *TCAD*, 2019.

[34] S. Hong, P. Frigo, Y. Kaya, C. Giuffrida, and T. Dumitraş, "Terminal Brain Damage: Exposing the Graceless Degradation in Deep Neural Networks Under Hardware Fault Attacks," in *USENIX Security*, 2019.

[35] A. Kwong, D. Genkin, D. Gruss, and Y. Yarom, "RAMBleed: Reading Bits in Memory Without Accessing Them," in *S&P*, 2020.

[36] 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 *S&P*, 2020.

[37] 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 *S&P*, 2020.

[38] Z. Weissman, T. Tiemann, D. Moghimi, E. Custodio, T. Eisenbarth, and B. Sunar, "JackHammer: Efficient Rowhammer on Heterogeneous FPGA–CPU Platforms," arXiv:1912.11523 [cs.CR], 2020.

[39] Z. Zhang, Y. Cheng, D. Liu, S. Nepal, Z. Wang, and Y. Yarom, "PTHammer: Cross-User-Kernel-Boundary Rowhammer Through Implicit Accesses," in *MICRO*, 2020.

[40] F. Yao, A. S. Rakin, and D. Fan, "Deephammer: Depleting the Intelligence of Deep Neural Networks Through Targeted Chain of Bit Flips," in *USENIX Security*, 2020.

[41] 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*, 2021.

[42] H. Hassan, Y. C. Tugrul, J. S. Kim, V. van der Veen, K. Razavi, and O. Mutlu, "Uncovering In-DRAM RowHammer Protection Mechanisms: A New Methodology, Custom RowHammer Patterns, and Implications," in *MICRO*, 2021.

[43] P. Jattke, V. van der Veen, P. Frigo, S. Gunter, and K. Razavi, "Blacksmith: Scalable Rowhammering in the Frequency Domain," in *SP*, 2022.

[44] M. C. Tol, S. Islam, B. Sunar, and Z. Zhang, "Toward Realistic Backdoor Injection Attacks on DNNs using RowHammer," arXiv:2110.07683v2 [cs.LG], 2022.

[45] A. Kogler, J. Juffinger, S. Qazi, Y. Kim, M. Lipp, N. Boichat, E. Shiu, M. Nissler, and D. Gruss, "Half-Double: Hammering From the Next Row Over," in *USENIX Security*, 2022.

[46] L. Orosa, U. Rührmair, A. G. Yaglikci, H. Luo, A. Olgun, P. Jattke, M. Patel, J. Kim, K. Razavi, and O. Mutlu, "SpyHammer: Using RowHammer to Remotely Spy on Temperature," IEEE Access 2024, 2022.

[47] 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," *TDSC*, 2022.

[48] L. Liu, Y. Guo, Y. Cheng, Y. Zhang, and J. Yang, "Generating Robust DNN with Resistance to Bit-Flip based Adversarial Weight Attack," *TC*, 2022.

[49] Y. Cohen, K. S. Tharayil, A. Haenel, D. Genkin, A. D. Keromytis, Y. Oren, and Y. Yarom, "HammerScope: Observing DRAM Power Consumption Using Rowhammer," in *CCS*, 2022.

[50] M. Zheng, Q. Lou, and L. Jiang, "TrojViT: Trojan Insertion in Vision Transformers," arXiv:2208.13049 [cs.LG], 2022.

[51] M. Fahr Jr, H. Kippen, A. Kwong, T. Dang, J. Lichtinger, D. Dachman-Soled, D. Genkin, A. Nelson, R. Perlner, A. Yerukhimovich *et al.*, "When Frodo Flips: End-to-End Key Recovery on FrodoKEM via Rowhammer," *CCS*, 2022.

[52] Y. Tobah, A. Kwong, I. Kang, D. Genkin, and K. G. Shin, "SpecHammer: Combining Spectre and Rowhammer for New Speculative Attacks," in *SP*, 2022.

[53] A. S. Rakin, M. H. I. Chowdhuryy, F. Yao, and D. Fan, "DeepSteal: Advanced Model Extractions Leveraging Efficient Weight Stealing in Memories," in *SP*, 2022.

[54] K. Park, D. Yun, and S. Baeg, "Statistical Distributions of Row-Hammering Induced Failures in DDR3 Components," *Microelectronics Reliability*, 2016.

[55] K. Park, C. Lim, D. Yun, and S. Baeg, "Experiments and Root Cause Analysis for Active-Precharge Hammering Fault in DDR3 SDRAM under 3xnm Technology," *Microelectronics Reliability*, 2016.

[56] C. Lim, K. Park, and S. Baeg, "Active Precharge Hammering to Monitor Displacement Damage Using High-Energy Protons in 3x-nm SDRAM," *TNS*, 2017.

[57] S.-W. Ryu, K. Min, J. Shin, H. Kwon, D. Nam, T. Oh, T.-S. Jang, M. Yoo, Y. Kim, and S. Hong, "Overcoming the Reliability Limitation in the Ultimately Scaled DRAM using Silicon Migration Technique by Hydrogen Annealing," in *IEDM*, 2017.

[58] D. Yun, M. Park, C. Lim, and S. Baeg, "Study of TID Effects on One Row Hammering using Gamma in DDR4 SDRAMs," in *IRPS*, 2018.

[59] T. Yang and X.-W. Lin, "Trap-Assisted DRAM Row Hammer Effect," *EDL*, 2019.

[60] A. J. Walker, S. Lee, and D. Beery, "On DRAM RowHammer and the Physics on Insecurity," *IEEE TED*, 2021.

[61] J. S. Kim, M. Patel, A. G. Yağlıkçı, H. Hassan, R. Azizi, L. Orosa, and O. Mutlu, "Revisiting RowHammer: An Experimental Analysis of Modern Devices and Mitigation Techniques," in *ISCA*, 2020.

[62] L. Orosa, A. G. Yağlıkçı, H. Luo, A. Olgun, J. Park, H. Hassan, M. Patel, J. S. Kim, and O. Mutlu, "A Deeper Look into RowHammer's Sensitivities: Experimental Analysis of Real DRAM Chips and Implications on Future Attacks and Defenses," in *MICRO*, 2021.

[63] A. G. Yağlıkçı, H. Luo, G. F. De Oliviera, A. Olgun, M. Patel, J. Park, H. Hassan, J. S. Kim, L. Orosa, and O. Mutlu, "Understanding RowHammer Under Reduced Wordline Voltage: An Experimental Study Using Real DRAM Devices," in *DSN*, 2022.

[64] M. N. I. Khan and S. Ghosh, "Analysis of Row Hammer Attack on STTRAM," in *ICCD*, 2018.

[65] S. Agarwal, H. Dixit, D. Datta, M. Tran, D. Houssameddine, D. Shum, and F. Benistant, "Rowhammer for Spin Torque based Memory: Problem or Not?" in *INTERMAG*, 2018.

[66] H. Li, H.-Y. Chen, Z. Chen, B. Chen, R. Liu, G. Qiu, P. Huang, F. Zhang, Z. Jiang, B. Gao, L. Liu, X. Liu, S. Yu, H.-S. P. Wong, and J. Kang, "Write Disturb Analyses on Half-Selected Cells of Cross-Point RRAM Arrays," in *IRPS*, 2014.

[67] K. Ni, X. Li, J. A. Smith, M. Jerry, and S. Datta, "Write Disturb in Ferroelectric FETs and Its Implication for 1T-FeFET AND Memory Arrays," *IEEE EDL*, 2018.

[68] P. R. Genssler, V. M. van Santen, J. Henkel, and H. Amrouch, "On the Reliability of FeFET On-Chip Memory," *TC*, 2022.

[69] O. Mutlu, A. Olgun, and A. G. Yaglikci, "Fundamentally Understanding and Solving RowHammer," in *ASP-DAC*, 2023.

[70] H. Luo, A. Olgun, A. G. Yağlıkcı, Y. C. Tuğrul, S. Rhyner, M. B. Cavlak, J. Lindegger, M. Sadrosadati, and O. Mutlu, "RowPress: Amplifying Read Disturbance in Modern DRAM Chips," in *ISCA*, 2023.

[71] E. Lee, I. Kang, S. Lee, G. Edward Suh, and J. Ho Ahn, "TWiCe: Preventing Row-Hammering by Exploiting Time Window Counters," in *ISCA*, 2019.

[72] S. M. Seyedzadeh, A. K. Jones, and R. Melhem, "Counter-Based Tree Structure for Row Hammering Mitigation in DRAM," *CAL*, 2017.

[73] S. M. Seyedzadeh, A. K. Jones, and R. Melhem, "Mitigating Wordline Crosstalk Using Adaptive Trees of Counters," in *ISCA*, 2018.

[74] I. Kang, E. Lee, and J. H. Ahn, "CAT-TWO: Counter-Based Adaptive Tree, Time Window Optimized for DRAM Row-Hammer Prevention," *IEEE Access*, 2020.

[75] Y. Park, W. Kwon, E. Lee, T. J. Ham, J. H. Ahn, and J. W. Lee, "Graphene: Strong yet Lightweight Row Hammer Protection," in *MICRO*, 2020.

[76] M. J. Kim, J. Park, Y. Park, W. Doh, N. Kim, T. J. Ham, J. W. Lee, and J. H. Ahn, "Mithril: Cooperative Row Hammer Protection on Commodity DRAM Leveraging Managed Refresh," in *HPCA*, 2022.

[77] D.-H. Kim, P. J. Nair, and M. K. Qureshi, "Architectural Support for Mitigating Row Hammering in DRAM Memories," *CAL*, 2014.

[78] K. Bains, J. Halbert, C. Mozak, T. Schoenborn, and Z. Greenfield, "Row Hammer Refresh Command," U.S. Patent 9,117,544, 2015.

[79] K. S. Bains and J. B. Halbert, "Distributed Row Hammer Tracking," U.S. Patent

Patent 9,299,400, 2016.

[80] K. S. Bains and J. B. Halbert, "Row Hammer Monitoring Based on Stored Row Hammer Threshold Value," U.S. Patent 9,384,821, 2016.

[81] 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 *ASPLOS*, 2016.

[82] Apple Inc., "About the Security Content of Mac EFI Security Update 2015-001," https://support.apple.com/en-us/HT204934, June 2015.

[83] M. Son, H. Park, J. Ahn, and S. Yoo, "Making DRAM Stronger Against Row Hammering," in *DAC*, 2017.

[84] J. M. You and J.-S. Yang, "MRLoc: Mitigating Row-Hammering Based on Memory Locality," in *DAC*, 2019.

[85] A. G. Yağlıkçı, J. S. Kim, F. Devaux, and O. Mutlu, "Security Analysis of the Silver Bullet Technique for RowHammer Prevention," arXiv:2106.07084 [cs.CR], 2021.

[86] M. Qureshi, A. Rohan, G. Saileshwar, and P. J. Nair, "Hydra: Enabling Low-Overhead Mitigation of Row-Hammer at Ultra-Low Thresholds via Hybrid Tracking," in *ISCA*, 2022.

[87] F. Devaux and R. Ayrignac, "Method and Circuit for Protecting a DRAM Memory Device from the Row Hammer Effect," U.S. Patent 10,885,966, 2021.

[88] G.-H. Lee, S. Na, I. Byun, D. Min, and J. Kim, "CryoGuard: A Near Refresh-Free Robust DRAM Design for Cryogenic Computing," in *ISCA*, 2021.

[89] M. Marazzi, P. Jattke, F. Solt, and K. Razavi, "ProTRR: Principled yet Optimal In-DRAM Target Row Refresh," in *S&P*, 2022.

[90] Z. Zhang, Y. Cheng, M. Wang, W. He, W. Wang, S. Nepal, Y. Gao, K. Li, Z. Wang, and C. Wu, "SoftTRR: Protect Page Tables against Rowhammer Attacks using Software-Only Target Row Refresh," in *USENIX ATC*, 2022.

[91] B. K. Joardar, T. K. Bletsch, and K. Chakrabarty, "Learning to Mitigate RowHammer Attacks," in *DATE*, 2022.

[92] JEDEC, *JESD79-5: DDR5 SDRAM Standard*, 2020.

[93] JEDEC, *JESD79-5c: DDR5 SDRAM Standard*, 2024.

[94] S. Saroiu, "DDR5 Spec Update Has All It Needs to End Rowhammer: Will It?" https://stefan.t8k2.com/rh/PRAC/index.html.

[95] T. Bennett, S. Saroiu, A. Wolman, and L. Cojocar, "Panopticon: A Complete In-DRAM Rowhammer Mitigation," in *Workshop on DRAM Security (DRAMSec)*, 2021.

[96] H. Hassan, A. Olgun, A. G. Yaglikci, H. Luo, and O. Mutlu, "A Case for Self-Managing DRAM Chips: Improving Performance, Efficiency, Reliability, and Security via Autonomous in-DRAM Maintenance Operations," arXiv:2207.13358 [cs.AR], 2022.

[97] H. Luo, Y. C. Tuğrul, F. N. Bostancı, A. Olgun, A. G. Yağlıkçı, , and O. Mutlu, "Ramulator 2.0: A Modern, Modular, and Extensible DRAM Simulator," arXiv:2308.11030 [cs.AR], 2023.

[98] SAFARI Research Group, "Ramulator V2.0," https://github.com/CMU-SAFARI/ramulator2.

[99] K. Chandrasekar, B. Akesson, and K. Goossens, "Improved Power Modeling of DDR SDRAMs," in *DSD*, 2011.

[100] T. Moscibroda and O. Mutlu, "Memory Performance Attacks: Denial of Memory Service in Multi-Core Systems," in *USENIX Security*, 2007.

[101] O. Mutlu and T. Moscibroda, "Parallelism-Aware Batch Scheduling: Enhancing Both Performance and Fairness of Shared DRAM Systems," in *ISCA*, 2008.

[102] O. Mutlu and T. Moscibroda, "Stall-Time Fair Memory Access Scheduling for Chip Multiprocessors," in *MICRO*, 2007.

[103] Y. Kim, M. Papamichael, O. Mutlu, and M. Harchol-Balter, "Thread Cluster Memory Scheduling: Exploiting Differences in Memory Access Behavior," in *MICRO*, 2010.

[104] Y. Kim, D. Han, O. Mutlu, and M. Harchol-Balter, "ATLAS: A Scalable and High-Performance Scheduling Algorithm for Multiple Memory Controllers," in *HPCA*, 2010.

[105] L. Subramanian, D. Lee, V. Seshadri, H. Rastogi, and O. Mutlu, "BLISS: Balancing Performance, Fairness and Complexity in Memory Access Scheduling," *TPDS*, 2016.

[106] L. Subramanian, D. Lee, V. Seshadri, H. Rastogi, and O. Mutlu, "The Blacklisting Memory Scheduler: Achieving High Performance and Fairness at Low Cost," in *ICCD*, 2014.

[107] JEDEC, *JESD209-5A: LPDDR5 SDRAM Standard*, 2020.

[108] JEDEC, *JESD209-4B: Low Power Double Data Rate 4 (LPDDR4) Standard*, 2017.

[109] JEDEC, *JESD235C: High Bandwidth Memory (HBM) DRAM*, 2020.

[110] JEDEC, *JESD79F: Double Data Rate (DDR) SDRAM Standard*, 2008.

[111] JEDEC, *JESD79-4C: DDR4 SDRAM Standard*, 2020.

[112] JEDEC, *JESD79-3: DDR3 SDRAM Standard*, 2012.

[113] Y. Kim, V. Seshadri, D. Lee, J. Liu, O. Mutlu, Y. Kim, V. Seshadri, D. Lee, J. Liu, and O. Mutlu, "A Case for Exploiting Subarray-Level Parallelism (SALP) in DRAM," in *ISCA*, 2012.

[114] 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 *HPCA*, 2015.

[115] D. Lee, Y. Kim, V. Seshadri, J. Liu, L. Subramanian, and O. Mutlu, "Tiered-Latency DRAM: A Low Latency and Low Cost DRAM Architecture," in *HPCA*, 2013.

[116] J. Liu, B. Jaiyen, R. Veras, and O. Mutlu, "RAIDR: Retention-Aware Intelligent DRAM Refresh," in *ISCA*, 2012.

[117] J. Liu, B. Jaiyen, Y. Kim, C. Wilkerson, O. Mutlu, J. Liu, B. Jaiyen, Y. Kim, C. Wilkerson, and O. Mutlu, "An Experimental Study of Data Retention Behavior in Modern DRAM Devices," in *ISCA*, 2013.

[118] K. K. Chang, D. Lee, Z. Chishti, A. R. Alameldeen, C. Wilkerson, Y. Kim, and O. Mutlu, "Improving DRAM Performance by Parallelizing Refreshes with Accesses," in *HPCA*, 2014.

[119] Micron Inc., "SDRAM, 4Gb: x4, x8, x16 DDR4 SDRAM Features," 2014.

[120] W. Kim, C. Jung, S. Yoo, D. Hong, J. Hwang, J. Yoon, O. Jung, J. Choi, S. Hyun, M. Kang *et al.*, "A 1.1 V 16Gb DDR5 DRAM with Probabilistic-Aggressor Tracking, Refresh-Management Functionality, Per-Row Hammer Tracking, a Multi-Step Precharge, and Core-Bias Modulation for Security and Reliability Enhancement," in *ISSCC*, 2023.

[121] A. G. Yağlıkçı, M. Patel, J. S. Kim, R. Azizibarzoki, A. Olgun, L. Orosa, H. Hassan, J. Park, K. Kanellopoullos, T. Shahroodi, S. Ghose, and O. Mutlu, "BlockHammer: Preventing RowHammer at Low Cost by Blacklisting Rapidly-Accessed DRAM Rows," in *HPCA*, 2021.

[122] S. Saroiu and A. Wolman, "How to Configure Row-Sampling-Based Rowhammer Defenses," *DRAMSec*, 2022.

[123] A. Olgun, M. Osseiran, A. G. Yaglikci, Y. C. Tugrul, H. Luo, S. Rhyner, B. Salami, J. G. Luna, and O. Mutlu, "Understanding Read Disturbance in High Bandwidth Memory: An Experimental Analysis of Real HBM2 DRAM Chips," arXiv:2310.14665 [cs.AR], 2023.

[124] R. Zhou, J. Liu, S. Ahmed, N. Kochar, A. S. Rakin, and S. Angizi, "Threshold Breaker: Can Counter-Based RowHammer Prevention Mechanisms Truly Safeguard DRAM?" *arXiv preprint arXiv:2311.16460*, 2023.

[125] S. Eyerman and L. Eeckhout, "System-Level Performance Metrics for Multiprogram Workloads," *IEEE Micro*, 2008.

[126] A. Snavely and D. M. Tullsen, "Symbiotic Job Scheduling for A Simultaneous Multithreaded Processor," in *ASPLOS*, 2000.

[127] S. Rixner, W. J. Dally, U. J. Kapasi, P. Mattson, and J. D. Owens, "Memory Access Scheduling," in *ISCA*, 2000.

[128] W. K. Zuravleff and T. Robinson, "Controller for a Synchronous DRAM That Maximizes Throughput by Allowing Memory Requests and Commands to Be Issued Out of Order," 1997, U.S. Patent 5,630,096.

[129] D. Kaseridis, J. Stuecheli, and L. K. John, "Minimalist Open-Page: A DRAM Page-Mode Scheduling Policy for the Many-Core Era," in *MICRO*, 2011.

[130] Standard Performance Evaluation Corp., "SPEC CPU 2006," http://www.spec.org/cpu2006/, 2006.

[131] Standard Performance Evaluation Corp., "SPEC CPU 2017," http://www.spec.org/cpu2017, 2017.

[132] Transaction Processing Performance Council, "TPC Benchmarks," http://tpc.org/.

[133] J. E. Fritts, F. W. Steiling, J. A. Tucek, and W. Wolf, "MediaBench II Video: Expediting the Next Generation of Video Systems Research," *MICPRO*, 2009.

[134] B. Cooper, A. Silberstein, E. Tam, R. Ramakrishnan, and R. Sears, "Benchmarking Cloud Serving Systems with YCSB," in *SoCC*, 2010.

[135] A. Olgun, Y. C. Tugrul, N. Bostanci, I. E. Yuksel, H. Luo, S. Rhyner, A. G. Yaglikci, G. F. Oliveira, and O. Mutlu, "ABACuS: All-Bank Activation Counters for Scalable and Low Overhead RowHammer Mitigation," *USENIX Security*, 2024.

[136] F. N. Bostanci, I. E. Yüksel, A. Olgun, K. Kanellopoulos, Y. C. Tuğrul, A. G. Yağliçi, M. Sadrosadati, and O. Mutlu, "CoMeT: Count-Min-Sketch-Based Row Tracking to Mitigate RowHammer at Low Cost," in *HPCA*, 2024.

[137] T. Moscibroda and O. Mutlu, "Memory Performance Attacks: Denial of Memory Service in Multi-Core Systems," in *USENIX Security*, 2007.

[138] K. Chandrasekar, S. Goossens, C. Weis, M. Koedam, B. Akesson, N. Wehn, and K. Goossens, "Exploiting Expendable Process-Margins in DRAMs for Run-Time Performance Optimization," in *DATE*, 2014.

[139] 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 *SIGMETRICS*, 2016.

[140] 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," in *SIGMETRICS*, 2017.

[141] K. K. Chang, "Understanding and Improving the Latency of DRAM-Based Memory Systems," Ph.D. dissertation, Carnegie Mellon University, 2017.

[142] J. S. Kim, M. Patel, H. Hassan, and O. Mutlu, "Solar-DRAM: Reducing DRAM Access Latency by Exploiting the Variation in Local Bitlines," in *ICCD*, 2018.

[143] Y. Wang, A. Tavakkol, L. Orosa, S. Ghose, N. M. Ghiasi, M. Patel, J. S. Kim, H. Hassan, M. Sadrosadati, and O. Mutlu, "Reducing DRAM Latency via Charge-Level-Aware Look-Ahead Partial Restoration," in *MICRO*, 2018.

[144] 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," in *SIGMETRICS*, 2017.

[145] A. Olgun, M. Patel, A. G. Yağlıkçı, H. Luo, J. S. Kim, N. Bostancı, N. Vijaykumar, O. Ergin, and O. Mutlu, "QUAC-TRNG: High-Throughput True Random Number Generation Using Quadruple Row Activation in Commodity DRAM Chips," in *ISCA*, 2021.

[146] J. S. Kim, M. Patel, H. Hassan, L. Orosa, and O. Mutlu, "D-RaNGe: Using Commodity DRAM Devices to Generate True Random Numbers with Low Latency and High Throughput," in *HPCA*, 2019.

[147] A. G. Yağlıkçi, A. Olgun, M. Patel, H. Luo, H. Hassan, L. Orosa, O. Ergin, and O. Mutlu, "HiRA: Hidden Row Activation for Reducing Refresh Latency of Off-the-Shelf DRAM Chips," in *MICRO*, 2022.

[148] J. S. Kim, M. Patel, H. Hassan, and O. Mutlu, "The DRAM Latency PUF: Quickly Evaluating Physical Unclonable Functions by Exploiting the Latency–Reliability Tradeoff in Modern Commodity DRAM Devices," in *HPCA*, 2018.

[149] F. Gao, G. Tziantzioulis, and D. Wentzlaff, "ComputeDRAM: In-Memory Compute Using Off-the-Shelf DRAMs," in *MICRO*, 2019.

[150] F. Gao, G. Tziantzioulis, and D. Wentzlaff, "FracDRAM: Fractional Values in Off-the-Shelf DRAM," in *MICRO*, 2022.

[151] L. Orosa, Y. Wang, M. Sadrosadati, J. S. Kim, M. Patel, I. Puddu, H. Luo, K. Razavi, J. Gómez-Luna, H. Hassan, N. Mansouri-Ghiasi, S. Ghose, and O. Mutlu, "CODIC: A Low-Cost Substrate for Enabling Custom In-DRAM Functionalities and Optimizations," in *ISCA*, 2021.

[152] B. M. S. Bahar Talukder, B. Ray, D. Forte, and M. T. Rahman, "PreLatPUF: Exploiting DRAM Latency Variations for Generating Robust Device Signatures," *IEEE Access*, 2019.

[153] K. Nguyen, K. Lyu, X. Meng, V. Sridharan, and X. Jian, "Nonblocking Memory Refresh," in *ISCA*, 2018.

[154] A. G. Yağlıkcı, Y. C. Tuğrul, G. F. De Oliviera, I. E. Yüksel, A. Olgun, H. Luo, and O. Mutlu, "Spatial Variation-Aware Read Disturbance Defenses: Experimental Analysis of Real DRAM Chips and Implications on Future Solutions," in *HPCA*, 2024.

[155] M. Patel, J. S. Kim, and O. Mutlu, "The Reach Profiler (REAPER): Enabling the Mitigation of DRAM Retention Failures via Profiling at Aggressive Conditions," in *ISCA*, 2017.

[156] M. Qureshi, D.-H. Kim, S. Khan, P. Nair, and O. Mutlu, "AVATAR: A Variable-Retention-Time (VRT) Aware Refresh for DRAM Systems," in *DSN*, 2015.

[157] S. Khan, D. Lee, Y. Kim, A. R. Alameldeen, C. Wilkerson, and O. Mutlu, "The Efficacy of Error Mitigation Techniques for DRAM Retention Failures: A Comparative Experimental Study," in *SIGMETRICS*, 2014.

[158] O. Canpolat, A. G. Yağlıkçı, A. Olgun, İ. E. Yüksel, Y. C. Tuğrul, K. Kanellopoulos, O. Ergin, and O. Mutlu, "Leveraging Adversarial Detection to Enable Scalable and Low Overhead RowHammer Mitigations," arXiv:2404.13477 [cs.CR], 2024.

[159] Hewlett-Packard Enterprise, "HP Moonshot Component Pack Version 2015.05.0," 2015.

[160] Lenovo Group Ltd., "Row Hammer Privilege Escalation," 2015. [Online]. Available: https://support.lenovo.com/us/en/product_security/row_hammer

[161] Z. Greenfield and T. Levy, "Throttling Support for Row-Hammer Counters," U.S. Patent 9,251,885, 2016.

[162] B. Aichinger, "DDR Memory Errors Caused by Row Hammer," in *HPEC*, 2015.

[163] K. Bains *et al.*, "Row Hammer Refresh Command," U.S. Patent 14/068,677, 2014.

[164] G. Irazoqui, T. Eisenbarth, and B. Sunar, "MASCAT: Stopping Microarchitectural Attacks Before Execution," *IACR Cryptology*, 2016.

[165] C.-M. Yang, C.-K. Wei, H.-P. Chen, J.-S. Luo, Y. J. Chang, T.-C. Wu, and C.-S. Lai, "Scanning Spreading Resistance Microscopy for Doping Profile in Saddle-Fin Devices," *TNANO*, 2017.

[166] G. Saileshwar, B. Wang, M. Qureshi, and P. J. Nair, "Randomized Row-Swap: Mitigating Row Hammer by Breaking Spatial Correlation Between Aggressor and Victim Rows," in *ASPLOS*, 2022.

[167] R. K. Konoth, M. Oliverio, A. Tatar, D. Andriesse, H. Bos, C. Giuffrida, and K. Razavi, "ZebRAM: Comprehensive and Compatible Software Protection Against Rowhammer Attacks," in *OSDI*, 2018.

[168] S. Vig, S. Bhattacharya, D. Mukhopadhyay, and S.-K. Lam, "Rapid Detection of Rowhammer Attacks Using Dynamic Skewed Hash Tree," in *HASP*, 2018.

[169] S. Gautam, S. Manhas, A. Kumar, M. Pakala, and E. Yieh, "Row Hammering Mitigation Using Metal Nanowire in Saddle Fin DRAM," *IEEE TED*, 2019.

[170] J. Juffinger, L. Lamster, A. Kogler, M. Eichlseder, M. Lipp, and D. Gruss, "CSI: Rowhammer-Cryptographic Security and Integrity against Rowhammer," in *SP*, 2023.

[171] A. Saxena, G. Saileshwar, P. J. Nair, and M. Qureshi, "AQUA: Scalable Rowhammer Mitigation by Quarantining Aggressor Rows at Runtime," in *MICRO*, 2022.

[172] S. Enomoto, H. Kuzuno, and H. Yamada, "Efficient Protection Mechanism for CPU Cache Flush Instruction Based Attacks," *IEICE Transactions on Information and Systems*, 2022.

[173] E. Manzhosov, A. Hastings, M. Pancholi, R. Piersma, M. T. I. Ziad, and S. Sethumadhavan, "Revisiting Residue Codes for Modern Memories," in *MICRO*, 2022.

[174] S. M. Ajorpaz, D. Moghimi, J. N. Collins, G. Pokam, N. Abu-Ghazaleh, and D. Tullsen, "EVAX: Towards a Practical, Pro-active & Adaptive Architecture for High Performance & Security," in *MICRO*, 2022.

[175] B. K. Joardar, T. K. Bletsch, and K. Chakrabarty, "Machine Learning-Based Rowhammer Mitigation," *TCAD*, 2022.

[176] Z. Zhang, Z. Zhan, D. Balasubramanian, B. Li, P. Volgyesi, and X. Koutsoukos, "Leveraging EM Side-Channel Information to Detect Rowhammer Attacks," in *SP*, 2020.

[177] K. Loughlin, S. Saroiu, A. Wolman, and B. Kasikci, "Stop! Hammer Time: Rethinking Our Approach to Rowhammer Mitigations," in *HotOS*, 2021.

[178] J. Han, J. Kim, D. Beery, K. D. Bozdag, P. Cuevas, A. Levi, I. Tain, K. Tran, A. J. Walker, S. V. Palayam *et al.*, "Surround Gate Transistor With Epitaxially Grown Si Pillar and Simulation Study on Soft Error and Rowhammer Tolerance for DRAM," *TED*, 2021.

[179] A. Fakhrzadehgan, Y. N. Patt, P. J. Nair, and M. K. Qureshi, "SafeGuard: Reducing the Security Risk from Row-Hammer via Low-Cost Integrity Protection," in *HPCA*, 2022.

[180] S. Saroiu, A. Wolman, and L. Cojocar, "The Price of Secrecy: How Hiding Internal DRAM Topologies Hurts Rowhammer Defenses," in *IRPS*, 2022.

[181] K. Loughlin, S. Saroiu, A. Wolman, Y. A. Manerkar, and B. Kasikci, "MOESI-Prime: Preventing Coherence-Induced Hammering in Commodity Workloads," in *ISCA*, 2022.

[182] R. Zhou, S. Tabrizchi, A. Roohi, and S. Angizi, "LT-PIM: An LUT-Based Processing-in-DRAM Architecture with RowHammer Self-Tracking," *IEEE CAL*, 2022.

[183] A. Di Dio, K. Koning, H. Bos, and C. Giuffrida, "Copy-on-Flip: Hardening ECC Memory Against Rowhammer Attacks," in *NDSS*, 2023.

[184] S. Sharma, D. Sanyal, A. Mukhopadhyay, and R. H. Shaik, "A Review on Study of Defects of DRAM-RowHammer and Its Mitigation," *Journal For Basic Sciences*, 2022.

[185] J. Woo, G. Saileshwar, and P. J. Nair, "Scalable and Secure Row-Swap: Efficient and Safe Row Hammer Mitigation in Memory Systems," in *HPCA*, 2023.

[186] J. H. Park, S. Y. Kim, D. Y. Kim, G. Kim, J. W. Park, S. Yoo, Y.-W. Lee, and M. J. Lee, "Row Hammer Reduction Using a Buried Insulator in a Buried Channel Array Transistor," *IEEE TED*, 2022.

[187] M. Wi, J. Park, S. Ko, M. J. Kim, N. S. Kim, E. Lee, and J. H. Ahn, "SHADOW: Preventing Row Hammer in DRAM with Intra-Subarray Row Shuffling," in *HPCA*, 2023.

[188] C. Gude Ramarao, K. T. Kumar, G. Ujjinappa, and B. V. D. Naidu, "Defending SoCs with FPGAs from Rowhammer Attacks," *Material Science*, 2023.

[189] K. Guha and A. Chakrabarti, "Criticality Based Reliability from Rowhammer Attacks in Multi-User-Multi-FPGA Platform," in *VLSID*, 2022.

[190] L. France, F. Bruguier, M. Mushtaq, D. Novo, and P. Benoit, "Modeling Rowhammer in the gem5 Simulator," in *CHES*, 2022.

[191] L. France, F. Bruguier, D. Novo, M. Mushtaq, and P. Benoit, "Reducing the Silicon Area Overhead of Counter-Based Rowhammer Mitigations," in *18th CryptArchi Workshop*, 2022.

[192] K. Arıkan, A. Palumbo, L. Cassano, P. Reviriego, S. Pontarelli, G. Bianchi, O. Ergin, and M. Ottavi, "Processor Security: Detecting Microarchitectural Attacks via Count-Min Sketches," *VLSI*, 2022.

[193] C. Tomita, M. Takita, K. Fukushima, Y. Nakano, Y. Shiraishi, and M. Morii, "Extracting the Secrets of OpenSSL with RAMBleed," *Sensors*, 2022.

[194] A. Saxena, G. Saileshwar, J. Juffinger, A. Kogler, D. Gruss, and M. Qureshi, "PT-Guard: Integrity-Protected Page Tables to Defend Against Breakthrough Rowhammer Attacks," in *DSN*, 2023.

[195] R. Zhou, S. Ahmed, A. S. Rakin, and S. Angizi, "DNN-Defender: An In-DRAM Deep Neural Network Defense Mechanism for Adversarial Weight Attack," arXiv:2305.08034 [cs.CR], 2023.

[196] JEDEC, *JESD79-4C: DDR4 SDRAM Standard*, 2020.

[197] H. Gomez, A. Amaya, and E. Roa, "DRAM Row-Hammer Attack Reduction Using Dummy Cells," in *NORCAS*, 2016.

[198] C. Yang, C. K. Wei, Y. J. Chang, T. C. Wu, H. P. Chen, and C. S. Lai, "Suppression of RowHammer Effect by Doping Profile Modification in Saddle-Fin Array Devices for Sub-30-nm DRAM Technology," *TDMR*, 2016.

[199] H. Hassan, M. Patel, J. S. Kim, A. G. Yağlıkçı, N. Vijaykumar, N. Mansouri Ghiasi, S. Ghose, and O. Mutlu, "CROW: A Low-Cost Substrate for Improving DRAM Performance, Energy Efficiency, and Reliability," in *ISCA*, 2019.

[200] S. Hong, D. Kim, J. Lee, R. Oh, C. Yoo, S. Hwang, and J. Lee, "DSAC: Low-Cost Rowhammer Mitigation Using In-DRAM Stochastic and Approximate Counting Algorithm," arXiv:2302.03591 [cs.CR], 2023.

[201] M. Marazzi, F. Solt, P. Jattke, K. Takashi, and K. Razavi, "REGA: Scalable Rowhammer Mitigation with Refresh-Generating Activations," in *SP*, 2023.

[202] O. Mutlu, "Retrospective: Flipping bits in memory without accessing them: An experimental study of dram disturbance errors," *Retrospective Issue for ISCA-50*, 2023.