2024 57th IEEE/ACM International Symposium on Microarchitecture (MICRO)

MINT: Securely Mitigating Rowhammer with a Minimalist In-DRAM Tracker

Moinuddin Qureshi
Georgia Tech
moin@gatech.edu

Salman Qazi
Google
sqazi@google.com

Aamer Jaleel
NVIDIA
ajaleel@nvidia.com

***Abstract*—This paper investigates secure low-cost in-DRAM trackers for mitigating Rowhammer (RH). In-DRAM solutions have the potential to solve the RH problem within the DRAM chip without relying on other parts of the system. However, in-DRAM mitigation suffers from two key challenges: First, the mitigations are synchronized with refresh, which means that we cannot mitigate at arbitrary times. Second, the SRAM area available for aggressor tracking is limited to only a few bytes. Existing low-cost in-DRAM trackers (such as TRR) have been broken by well-crafted access patterns, whereas, secure counter-based schemes require impractical overheads of hundreds or thousands of entries per bank. The goal of our paper is to develop an ultra-low-cost secure in-DRAM tracker.**

Our solution is based on a simple observation: If only one row can be mitigated at refresh, we should ideally need to track only one row. We propose a *Minimalist In-DRAM Tracker (MINT)*, which provides secure mitigation with just a single entry. Unlike prior trackers that decide the row to be mitigated based on the past behavior (select based on activation counts) or solely based on the current activation (select with some probability), MINT decides which row in the future will get mitigated. At each refresh, MINT probabilistically decides which activation in the upcoming interval will be selected for mitigation at the next refresh. MINT provides guaranteed protection against classic single and double-sided attacks. We also derive the minimum RH threshold (TRH*) tolerated by MINT across all patterns. MINT has a TRH* of 1482, which can be lowered to 356 with RFM. The TRH* of MINT is lower than a prior counter-based design with 677 entries per bank, and is within 2x of the TRH* of an idealized design that stores one-counter-per-row. We also analyze the impact of refresh postponement on the TRH* of low-cost in-DRAM trackers, and propose an efficient solution to make such trackers compatible with refresh postponement.

I. Introduction

Rowhammer (RH) is a data-disturbance error that occurs when rapid activations of a DRAM row causes bit-flips in neighboring rows [23]. Rowhammer is a serious security threat, as it gives an attacker a powerful tool to flip bits in protected data structures, such as page tables, which can result in privilege escalation [3], [5], [8]–[10], [41], [44] and breach of confidentiality [25]. The RH problem has been difficult to solve because the *Rowhammer Threshold (TRH)*, which is the number of activations required to induce a bit-flip, has continued to reduce with successive generations of DRAM, reducing from 140K [23] to about 4.8K [19] in the last decade. Thus, Rowhammer solutions must scale to low TRH.

Typical hardware-based mitigation for RH relies on a *tracking* mechanism to identify the aggressor rows (i.e., the rows that get activated repeatedly) and issue a refresh to neighboring victim rows [11]. Hardware-based RH mitigation can be deployed at the Memory-Controller (MC) or within the DRAM chip (in-DRAM). The in-DRAM approach has the advantage that it can solve the RH problem transparently within the DRAM chip without relying on changes to other parts of the system. The in-DRAM approach also has the advantage that memory vendors can tune their mitigation solution to target the TRH observed in their chips. This work focuses on low-cost in-DRAM RH mitigation.

In-DRAM RH mitigation suffers from two key constraints. **First**, the RH mitigation is performed transparently within the refresh operation (REF). For example, for DDR5, the REF operation occurs every 3.9 microseconds, and the DRAM chip can steal some of the time reserved for REF for performing RH mitigation (refresh the victim rows of a selected aggressor row). It is not possible for DRAM chips to sometimes take longer to do REF if there are more aggressor rows, as this would violate the deterministic timing guarantees of DDR5. This restriction means that it is not enough to track aggressor rows, but we must *spread* the mitigation of aggressor rows over as many REF periods, as we cannot support *bursty* mitigation. **Second**, the SRAM budget available for tracking aggressor rows is quite small (often a few bytes per bank) and this budget is insufficient for tracking all the aggressor rows. For example, DDR4 devices contain *Targeted Row Refresh (TRR)* tracker containing 1-30 entries [11], however, such trackers have been defeated with attack patterns such as TRRespass [8] and Blacksmith [14]. Thus, systems continue to be prone to RH attacks even in the presence of such deployed mitigations.

Designing secure low-cost in-DRAM trackers has proven to be a significant challenge. The recent solutions from industry have also not proven to be secure. For example, the DSAC [12] tracker from Samsung is vulnerable to Blacksmith patterns (when designed for a TRH of 2K, DSAC results in 9K unmitigated activations on an aggressor row with Blacksmith). Similarly, the PAT [22] tracker from Hynix claims 30% lower failure rate than TRR (however, as TRR can be broken within a few minutes, PAT can also be broken within a few minutes).

Prior studies, such as ProTRR [29] and Mithril [20], bound the minimum number of tracking entries required to deterministically and securely mitigate a given TRH. These studies show that an *optimal* number of entries in the tracker would be several thousand per bank for current TRH (e.g. 1400 entries for TRH of 2K). This SRAM overhead is prohibitively large for practical adoption in DRAM chips. Any tracker with number of entries below the optimal will always suffer from a non-zero probability of failure [29].

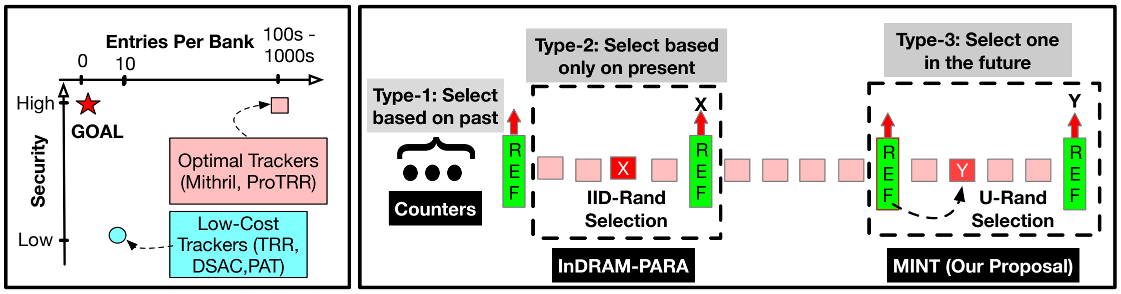
Entries Per Bank
0
10
100s - 1000s
High
Low
Security
GOAL
Optimal Trackers (Mithril, ProTRR)
Low-Cost Trackers (TRR, DSAC,PAT)
Type-1: Select based on past
Counters
Type-2: Select based only on present
REF
X
IID-Rand Selection
InDRAM-PARA
Type-3: Select one in the future
Y
U-Rand Selection
MINT (Our Proposal)

Fig. 1. (a) Our goal is to develop secure low-cost trackers. (b) In-DRAM RH mitigation is performed at REF and the tracker decides which row to mitigate. Trackers can be categorized into three types (1) past-centric, such as counter-based tracking (2) present-centric, such as selecting the currently activated row with some probability (3) future-centric, our design, which decides at each REF which row will be picked for mitigation in the upcoming interval.

Figure 1 (a) provides an overview of the landscape of in-DRAM trackers. Current in-DRAM trackers are either insecure or incur a prohibitively high-cost. The goal of our paper is to develop a low-cost secure in-DRAM tracker. Our solution is based on a simple insight: if in-DRAM mitigation is limited to mitigating at-most one aggressor row per REF, ideally we should need a tracker with only a single-entry (that identifies the aggressor row to be mitigated). We develop a classification of in-DRAM trackers that helps in guiding our solution. Figure 1 (b) shows an overview of in-DRAM rowhammer mitigation. A REF occurs at each tREFI interval and there are activations between REF. At each REF, one aggressor row is selected for mitigation. Depending on how this selection is made, we can classify the in-DRAM trackers into three types. First, **past-centric**, which makes the selection decision based on past behavior. For example, counter-based schemes (ProTRR and Mithril) select a row with the maximum counter value. Such trackers need a large amount of storage. Second, **present-centric**, which makes the *selection* decision probabilistically based only on the currently activated row. For example, a PARA [23] like in-DRAM scheme would select each activated row with an *Independent and Identically Distributed (IID)* probability of p. If selected, the row address is stored in a single-entry tracker for getting mitigated at the next REF. The problem with such an *InDRAM-PARA* design is that a selected row can be *over-written* before reaching REF by another selected row and thus miss the chance of mitigation. Furthermore, even if the same row gets activated throughout the tREFI interval, there is still a non-negligible likelihood that the row will still not be selected for mitigation. Ideally, we want a single-entry tracker that avoids both problems of InDRAM-PARA (no over-writing a selected entry and guaranteed selection of one row in tREFI). The *Minimum TRH (TRH*)* tolerated by the InDRAM-PARA is 7.6K.

We propose a *Minimalist In-DRAM Tracker (MINT)* that provides secure RH mitigation with a single-entry. MINT offers a new category, the third-type, **future-centric**. Let there be a maximum of M activations in tREFI. At each REF, MINT decides which of the M activations in the *upcoming* interval should be selected for mitigation at the *next* REF. MINT performs this selection using a *Uniform Random (U-RAND)* choice of all 1 to M positions. This position is stored in a *Selected Activation Number (SAN)*. Each activation in tREFI is given a sequence number, and when this number reaches SAN, the row is designated for mitigation at next REF.

By design, MINT avoids overwriting the selection. Secondly, MINT guarantees selection of one row (from M activations), thus it provides guaranteed protection against classic single-sided and double-sided attacks, if such attacks are done continuously. We analyze the worst-case pattern for MINT for determining the TRH*. We also analyze *Transitive Attacks* [24] that leverage victim refresh to cause RH failures in a distant row. Our analysis shows that MINT has a TRH* of 2800 (1400 for double-sided, TRH-D*). The TRH* of MINT is similar to Mithril [20] with 677 entries per bank.

DDR5 allows the postponement of up-to four REF operations. Delayed refreshes are especially problematic for low-cost trackers as they track limited entries, which may be dislodged (without mitigation). We analyze the impact of delayed refreshes on low-cost trackers and propose a generalized solution, the *Delayed Mitigation Queue (DMQ)*, which allows low-cost trackers to operate with REF postponement. With DMQ, the TRH-D* of MINT is 1482.

The threshold tolerated by MINT (with support for refresh postponement) is lower than a Mithril tracker with 677-entries per bank, and is within **2x** of an idealized tracker that has one counter per row. Thus, our proposal bounds the gap between the lowest-cost tracker (single-entry) and an idealized tracker (per-row counters) to a narrow range (2x).

Overall, this paper makes the following contributions:

1) We propose *Minimalist In-DRAM Tracker (MINT)* that provides secure RH mitigation with a single entry.
2) We show that MINT has a TRH* of 2800 (TRH-D* 1400), similar to a 677-entry tracker.
3) To the best of our knowledge, this is the first paper to study the impact of refresh postponement on low-cost trackers. We propose *Delayed Mitigation Queue (DMQ)* to enable low-cost tracking with refresh postponement. MINT+DMQ has a TRH-D* of 1482.
4) We combine MINT with the RFM feature of DDR5 to to obtain a TRH-D* as low as 356.

The storage overhead of MINT is four bytes (per bank) and the performance and power overheads are negligibly small.

II. Background and Motivation

A. DRAM Architecture and Parameters.

To access data from DRAM, the memory controller must first issue an activation (ACT) for the DRAM row. To ensure data retention, the memory controller sends a refresh command every tREFI that refreshes a subset of rows. Table I shows the DDR5 parameters, derived from DDR5 datasheet (DDR5-5200B bin with 32Gb chips). The two critical parameters for our study are: (1) the maximum number of ACT (MaxACT) possible within tREFI is 73 and (2) we assume that the device performs one Rowhammer mitigation at each refresh event.

TABLE I
DRAM Parameters (from DDR5 datasheet)

Parameter	Explanation	Value
tREFW	Refresh Window	32 ms
tREFI	Time interval between REF Commands	3900 ns
tRFC	Execution Time for REF Command	410 ns
tRC	Time between successive ACTs to a bank	48 ns
MaxACT	M = (tREFI - tRFC) / tRC	73

B. Threat Model

Our threat model assumes that an attacker can issue memory requests for arbitrary addresses. We assume that the attacker knows the defense algorithm but does not have physical access to the system (e.g., the outcome of random-number generator). Our defense aims to prevent Rowhammer against all access patterns, including Blacksmith [14], and Half-Double [24]. To keep our analysis simple, we do not consider the Row-Press [28] attack. Our recent work [38] shows that Row-Press can be easily mitigated by converting the row-open time into an equivalent number of activations (Appendix C shows how our solution can be modified to tolerate Row-Press).

C. DRAM Rowhammer Attacks

Rowhammer [23] occurs when a row (aggressor) is activated frequently, causing bit-flips in nearby rows (victim). The minimum number of activations to an aggressor row to cause a bit-flip in a victim row is called the *Rowhammer Threshold (TRH)*. TRH can be reported for a single-sided pattern *(TRH-S)* or a double-sided pattern *(TRH-D)*. As shown in Table II, TRH has dropped significantly, from 139K (TRH-S) in 2014 [23] to 4.8K (TRH-D) in 2020 [19].

TABLE II
Rowhammer Threshold Over Time

DRAM Generation	TRH-S (Single-Sided)	TRH-D (Double-Sided)
DDR3-old	139K [23]	–
DDR3-new	–	22.4K [19]
DDR4	–	10K [19] - 17.5K [19]
LPDDR4	–	4.8K [19] - 9K [24]

Rowhammer is a serious security threat, as an attacker can use it to flip bits in the page table to perform privilege escalation [8], [9], [41], [50] or break confidentiality [25].

Solutions for mitigating Rowhammer typically rely on a mechanism to identify the aggressor rows and then perform a mitigation by refreshing the victim rows. The identification of aggressor rows can be done either at the Memory Controller (MC) or within the DRAM chip (in-DRAM).

D. Memory-Controller Based Mitigation

Memory-Controller (MC) based solutions identify aggressor rows either using counters [26] [33] [34] or probabilistically [18] [23]. These solutions suffer from three major shortcomings. First, DRAM chips internally use proprietary mapping, so this solution must rely on the *Directed RFM (DRFM)* command to perform mitigation. DRFM incurs a latency of tRFC (410ns), resulting in significant slowdown, and there is a rate limit of one DRFM per two tREFI, placing a high limit on the TRH that can be tolerated. Second, the solution must be conservatively designed for the lowest TRH, across vendors and over the years of the system lifetime. Third, the cost and complexity of tracking can deter some processor vendors from adoption, leading to fragmented protection.

E. In-DRAM Mitigation

The advantage of in-DRAM mitigation is that it can solve Rowhammer within the DRAM chips, without relying on other parts of the system. Furthermore, DRAM manufacturers can tune their solution to the TRH of their chips.

In-DRAM mitigation typically performs the mitigation transparently during the time provisioned for the refresh operations (commodity DRAM is a deterministic device, so it is not possible to do mitigative activations while servicing the normal demand accesses). A recent study [29] observes that DDR5 chips support mitigating one aggressor row at each tREFI, or one per two tREFI. In our paper, we assume a default rate of mitigating one aggressor row per tREFI.

F. Low-Cost In-DRAM Trackers: Not Secure

For guaranteed protection, the in-DRAM tracker must be able to identify *all* aggressor rows and mitigate them before they receive TRH activations. Unfortunately, the SRAM budget available for tracking within the DRAM chip is limited to only a few bytes. Therefore, practical in-DRAM trackers (such as **TRR** from DDR4, **DSAC** [12] from Samsung, and **PAT** [22] from SK Hynix) are limited to only tracking a few entries (1-30), and can be broken within a few minutes using patterns that target a large number of aggressor rows [8] or use decoy rows [14]. Thus, the system remains vulnerable to RH attacks, even in the presence of such low-cost trackers.

G. Optimal In-DRAM Trackers: Not Practical

The minimum number of entries needed for an in-DRAM tracker to deterministically and securely tolerate a threshold of TRH is determined by the rate of mitigation (e.g. one per tREFI). If several rows get identified as aggressor rows at a similar time, then in-DRAM solution would need to spread their mitigation over several tREFI intervals, leading to more activations on the unmitigated aggressor rows. Two concurrent works [20] [29], establish the bounds on TRH tolerated by in-DRAM mitigation arising from such a restriction. They also bound the optimal number of entries needed to tolerate a given TRH. We call the designs that have the minimum number of tracking entries to tolerate a given threshold as *optimal* trackers. Examples of optimal trackers include:

Mithril [20]: Mithril uses a *Counter-based Summary* algorithm to track the activation counts of heavily activated rows. At mitigation, the row with the highest counter value is mitigated and the counter value is reduced by the min count.

ProTRR [29]: ProTRR performs *victim tracking* using a *Misra-Gries* tracker to identify the top victim rows. At mitigation, the victim row(s) with the highest counter value get refreshed and removed from the tracker.

Given the TRH and the rate of mitigation, we can determine the number of entries in the optimal trackers. For example, for a mitigation rate of 1 per tREFI, for a TRH-D of 1K, Mithril would require approximately 1400 entries. Note that each bank requires an independent tracker, so the total number of tracker entries to protect the entire DRAM rank (32 banks) would be approximately 45K. Unfortunately, the limited SRAM budget within the DRAM renders such trackers impractical.

H. Per-Row Counter-Table (PRCT)

The TRH tolerated by a in-DRAM mitigation depends on both, the number of entries in the tracker and the rate of mitigation (e.g. one per tREFI). To understand the limit of in-DRAM mitigation, under a given rate-of-mitigation, we also study an idealized design, *Per-Row Counter-Table (PRCT)*, which stores one counter per row in an SRAM table. Given the large overheads, such a design is not practical, however, it can still help us understand the TRH* gap between a practical design and an idealized design. The TRH* tolerated by PRCT is purely determined by the rate of mitigation. For example, at 1 mitigation per tREFI, PRCT can let two aggressor rows reach 623 activations each (victim subjected to 1226 activations). Thus, the TRH* of PRCT is 1226 (623 double-sided).

I. Understanding the Impact of Refresh Postponement

DDR5 allows the postponement of up-to four REF operations. Delayed refreshes can cause a row selected for mitigation to be further subjected to an additional 292 (73x4) activations. For counter-based mitigations [20], [22], [29] refresh postponement increases the tolerated TRH by 292. For example, the TRH* tolerated by PRCT with refresh postponement becomes 1518 (double-sided 759). Delayed refreshes are especially challenging for low-cost trackers with only a few entries, as those entries may get dislodged before getting mitigated during the period of refresh postponement. For example, a PARFM tracker [20] that tolerates a threshold of a few thousand (without refresh postponement) can be made to deterministically cause **487K (!)** activations on an attack row without any mitigation under refresh postponement.

J. Goal of our Paper

The goal of our paper is to develop an ultra low-cost and secure in-DRAM tracker with a tolerable threshold that is close to the idealized design (PRCT). Furthermore, we want our proposed design to be fully compatible with refresh postponement, as handling such a feature is a non-negotiable requirement for practical adoption. We first discuss the pitfalls of simply extending PARA to an in-DRAM setting.

III. Fundamental Pitfalls of In-DRAM-PARA

PARA [23] is a memory controller scheme that mitigates each activated row with probability *p*. In this section, we show that applying PARA to in-DRAM setting (InDRAM-PARA) suffers from two fundamental shortcomings: (a) non-uniform mitigation probability, depending on where the row activation lies in the tREFI interval (b) frequent non-selection of any activated row in the tREFI interval even if all the activation slots are used. These shortcomings cause the threshold tolerated by InDRAM-PARA to be **2.7x higher** than an idealized policy that mitigates all activations with equal probability.

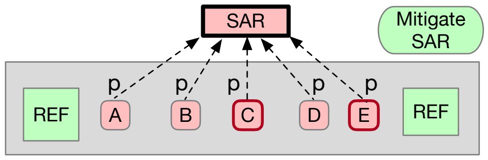

Fig. 2. Design of InDRAM-PARA. Each activation is sampled with a probability p and stored in SAR. At REF the row in SAR (if valid) is mitigated.

A. InDRAM-PARA: Design and Analysis

Figure 2 shows the overview of InDRAM-PARA. Each activation is sampled with *Sampling Probability (p)*. If sampled, the row-address is stored in *SAR (Sampled Address Register)*. At REF, if SAR is valid, the row is mitigated. For a row to get mitigated it must be both *sampled* and it must *survive* until REF in SAR. For example, if Row-C is sampled, then later sampling of Row-E evicts Row-C in SAR. For our design, all activations are sampled with a uniform probability (p=1/73), so the mitigation probability ($P_{mitigation}$) is proportional to the survival probability ($P_{survive}$), as shown in Equation 1.

$$P_{mitigation} = p \cdot P_{survive} \tag{1}$$

Model for Survival Probability: Let there be M activations between two refreshes (tREFI). The window starts with an empty SAR. Let Row-A be accessed at the Kth activation and get sampled into SAR. SAR will retain this entry if there is no other insertion in the remaining (M-K) activations. If p (we use $p = 1/73$) is the sample probability, then the *Survival Probability* (S_K) for position K is given by Equation 2.

$$S_K = (1 - p)^{(M-K)} \tag{2}$$

Fig 3 shows the survival-probability (S_K) as the position (K) is varied from 1 (earliest in tREFI) to 73 (last in tREFI).

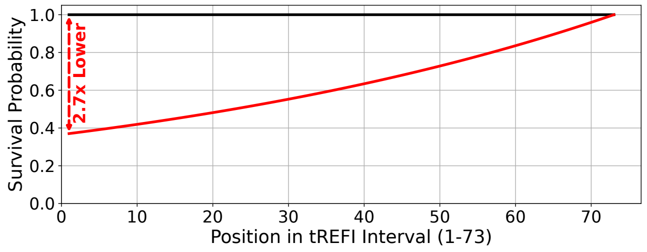

Fig. 3. Highly Non-Uniform Survival Probability for InDRAM-PARA

The first position has the lowest survival-probability (0.37) whereas the last position has the highest survival-probability (1). Thus, with this design, the most vulnerable position has **2.7x lower mitigation probability** compared to an idealized scheme that mitigates all positions with probability p.

B. InDRAM-PARA: No-Overwrite Version

It is intuitive to think that the non-uniform mitigation of InDRAM-PARA can easily be addressed by simply avoiding the overwrite of SAR, if SAR had a valid entry. Figure 4 shows an overview of such an InDRAM-PARA (No-Overwrite) design. While such a design guarantees 100% survival probability, it suffers from another equally critical problem of non-uniform sampling. If a row gets sampled (e.g. Row-C), then the probability of sampling for all later rows (e.g. Row-D and Row-E) becomes zero. As survival probability is 1, the mitigation probability of a activation is equal to the sampling probability of that position within the tREFI interval.

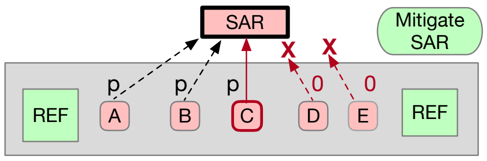

Fig. 4. Design of InDRAM-PARA that avoids overwriting SAR. While this design has 100% survival probability, it has non-uniform sampling probability.

Model for Non-Uniform Sampling Probability: Let M activations between two refreshes (tREFI). The window starts with an empty SAR. Let p be the designated sampling probability at the start of the tREFI interval. Then the first row will be selected with probability p. For all subsequent rows, the sampling probability is either p or 0, depending whether something was selected before. The sampling probability at position K (P_K) is given by Equation 3.

$$P_K = p \cdot (1-p)^K \tag{3}$$

Fig 3 shows the sampling probability (P_K) as the position (K) is varied from 1 (earliest in tREFI) to 73 (last in tREFI), normalized to the first position in the window (which equals p=1/73). We note that the sampling probability is highly non-uniform, reducing to about 0.37x for the last position in the window (so absolute sampling probability has reduced from 1/73 to 1/73 * 0.37 = 1/200). Thus, even with this design, the most vulnerable position has **2.7x lower mitigation probability** compared to an idealized scheme that performs mitigation of all positions with probability p.

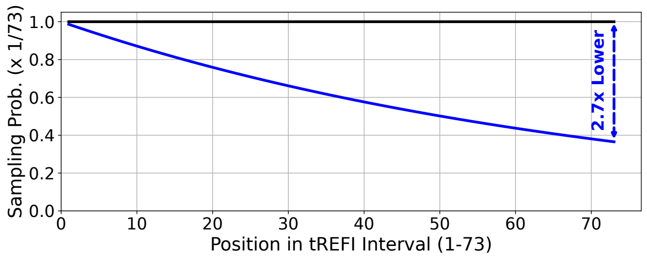

Fig. 5. Sampling Probability for InDRAM-PARA (No-Overwrite)

C. Impact of Non-Uniform Mitigation on Security

Figure 6 shows the mitigation probability of InDRAM-PARA and InDRAM-PARA(No-Overwrite) normalized to an ideal policy that mitigates each position with probability p=1/73, as the position in the tREFI window is varied. Both versions of InDRAM-PARA Designs have non-uniform mitigation, just that the most vulnerable position is different for them (either first or last). For both designs, the most vulnerable position has 2.7x lower mitigation rate than the ideal policy.

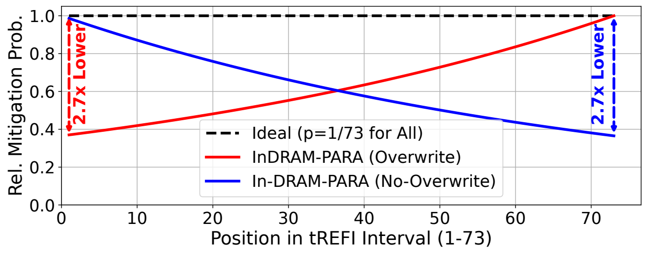

Fig. 6. Mitigation Probability of InDRAM-PARA, InDRAM-PARA (No-Overwrite) normalized to ideal policy with uniform mitigation (p=1/73).

Impact: Given any In-DRAM scheme, the attacker will focus on the most-vulnerable position, so the overall security of the design is determined by the most-vulnerable position. We note that exploiting the non-uniform mitigation probability of In-DRAM Trackers is a well-known technique in Rowhammer attacks. For example, both SMASH [6] and BlackSmith [14] use *Refresh Interval Synchronization* to converge on the most vulnerable position within the tREFI window. Thus, non-uniform mitigation has significant security implications.

Given 2.7x reduced mitigation probability, the threshold tolerated by InDRAM-PARA is about 2.7x higher than an ideal policy that mitigates all positions uniformly. For example, the TRH* of Ideal is 2.8K and InDRAM-PARA is 7.6K.

D. The Problem of Non-Selection with InDRAM-PARA

Another shortcoming of InDRAM-PARA is that even if all the activation slots of a tREFI window are used, it can still have a significant probability that nothing will get selected, thus missing out on the possibility of performing mitigation at REF. Equation 4 shows the probability that nothing will be selected in a tREFI window if M activations occur.

$$P_{NoSelect} = (1-p)^M = (1-1/73)^{73} = 0.37 \tag{4}$$

In our case, p=1/73, and M can be up-to 73. Thus, even if all activations slots are used, InDRAM-PARA will **skip mitigation 37% of the time**. The non-selection of InDRAM-PARA allows stressful attack patterns, such as the classic Single-Side and Double-Sided Rowhammer attacks that continuously activate the same one or two rows over the entire tREFI.

Key Takeaways: InDRAM-PARA suffers from non-uniform mitigation over the tREFI interval, and such non-uniformity has a significant impact on security and threshold. It also suffers from non-selection. An ideal In-DRAM solution must provide uniform mitigation probability for all activations, and avoid non-selection. We propose such an ideal solution.

IV. Methodology For Analyzing Security

We consider an event of one or more bitflips from Rowhammer as a failure. Thus, if any row receives TRH activations, without an intervening mitigation, we declare it as a failure.

A. *Model for Failure Probability in tREFW Window*

We divide the time into windows of tREFW, as all rows get refreshed every tRFW. We want to determine the probability of failure at the kth activation, given the row is mitigated with probability p at each activation. To the best of our knowledge, the most complete analytical model for estimating the probability of failure is by Sariou and Wolman [37] (it corrects an off-by-one error of Mithril [20] and also incorporates auto-refreshes). The probability of failure (P_k) at any given activation (k) for a TRH of T is given by Equations 1-3.

$$P_k = 0 \qquad \text{if } k < T \qquad (5)$$

$$P_k = (1-p)^T \qquad \text{if } k = T \qquad (6)$$

$$P_k = p \cdot (1-p)^T \cdot (1 - P_{\text{k-T-1}}) + P_{\text{k-1}} \quad \text{if } k > T \qquad (7)$$

Equation 1 and Equation 2 are trivial: (1) No failure with less than T activations and (2) at K=T activations, failure happens if all activations escape selection. For K>T activations, the recurrence is based on a powerful insight. For the row to fail exactly at the Kth activation, following must be true: (1) The position K-T must have received a mitigation (hence the term p) (2) No mitigation since (hence the term $(1-p)^T$) (3) Position K-T-1 must not already start with failure (hence the term (1-$P_{\text{k-T-1}}$)). Finally, as failures are cumulative, the term $P_{\text{k-1}}$. Figure 7 shows an overview of this model for T=4.

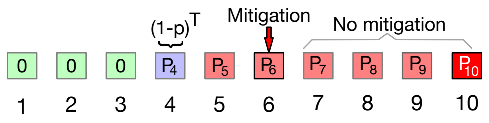

Fig. 7. Sariou-Wolman model for getting P_k for T=4. For failure at position-10, position-6 must be failure-free, must get a mitigation, and none after.

B. *Computing the Mean Time-to-Failure (MTTF)*

For a given number of activations to an attack row within tREFW, we use Equations 5-7 to estimate the failure probability (P_{REFW}) per tREFW. Based on Sariou-Wolman model [37], we account for auto-refresh by reducing P_{REFW} by a factor of (1-N/8192), where N denotes the length of the successful sequence in terms of tREFI. Equation 8 shows the *Mean Time-to-Failure (MTTF)* for a bank. The MTTF for a system with B banks would be approximately B times lower.

$$\text{Mean-Time-to-Failure (Bank)} = \frac{1}{P_{REFW}} \cdot tREFW \qquad (8)$$

C. *Minimum-Tolerated TRH: Key Figure-of-Merit*

We use a default *Target-MTTF* (per-bank) of 10,000 years (this is similar to the per-bank failure rate from naturally occurring errors [4], sensitivity in Section VIII-B). We define the *Minimum Tolerated TRH* (**TRH***) as the *lowest TRH* for which the design can meet the Target-MTTF. We denote **TRH-D*** as the per-row TRH* for a double-sided pattern.

V. Minimalist In-DRAM Tracker (MINT)

Secure trackers require significant storage, whereas existing low-cost trackers have been rendered insecure. To better understand this dichotomy, and develop a low-cost and secure tracker, we classify tracker design-space into three types, depending on how they select the row to be mitigated at a given REF. First, *past-centric*, which considers past behavior, for example, selecting the row with the highest counter value. To do a reliable selection, significant amount of past behavior is needed. Second, *present-centric*, which makes selection decision purely based on the currently activated row (say select the given row with a given probability and store it in a single-entry tracker). Unfortunately, such *InDRAM-PARA*, suffers from the problems of non-uniform mitigation probability over tREFI and non-selection (even if the row is continuously activated in the tREFI interval). Therefore, it has a high TRH*. We propose a *Minimalist In-DRAM Tracker (MINT)*, which is *future-centric* and provides a secure mitigation with just a single-entry. We first provide an overview and design of MINT, then derive the security for worst-case pattern, then discuss the impact of *Transitive Attacks* and *Spatial Attacks*.

A. *Overview of MINT: A Single-Entry Tracker*

Figure 8 shows an overview of MINT. Lets say each tREFI window can have up-to M activations. At each REF, MINT decides which of the *future* M activations must be selected for mitigation at the next REF. It uses a *Uniform Random (URAND)* selection for all possible M positions. Each activation within tREFI is given a sequence number and when it reaches the selected number, the given row (e.g. Row-C), is selected for getting mitigated at the next REF. The process repeats at each subsequent REF with a new URAND selection.

Note that the selection of MINT is done without knowing which address will appear at the selected activation number. By design, MINT does not suffer from overwrite problem of InDRAM-PARA (as no more than one row can be selected for mitigation). Furthermore, if a given address occurs M times in the window, then it is guaranteed to be selected by MINT.

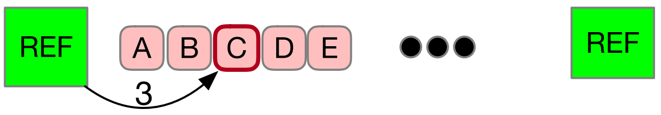

Fig. 8. Overview of MINT. At each REF, MINT decides (a priori) which activation number will get selected for mitigation at next REF.

Note that, if an access pattern has fewer than M activations within tREFI, then MINT can sometimes not select any row for mitigation at the next REF. However, this does not impact the security of MINT as the slot not used for activation can be treated, for the purpose of analyzing security, as an activation to a decoy (benign) row. For the attacker to have the highest chance of success, an attack pattern must use all the activation slots for causing *damage*. Therefore, for our security analysis, it is safe to assume that all M activations are used in an attack.

B. Design and Operation of MINT

Figure 9 shows the design and operation of MINT. MINT consists of three registers: (1) *Selected Activation Number (SAN)*, which stores the activation number that will be selected in the upcoming interval (2) *Current Activation Number (CAN)*, which provides a sequence number to each activation in the tREFI window, and (3) *Selected Address Register (SAR)*, which stores the address of the row to be mitigated at the next REF. SAR contains a valid bit to indicate if the SAR is filled.

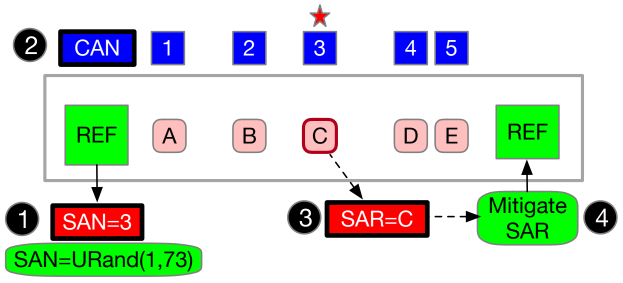

Fig. 9. Design and Operation of MINT. At each REF, SAN is set with URAND. During tREFI, CAN tracks sequence number for ACT. If CAN=SAN, the row address is stored in SAR. At REF, SAR is mitigated.

The maximum number of activations within the tREFI window is 73 (so, M=73). ❶ At each REF, the SAN is initialized based on a URAND selection of all 1 to 73 slots. The CAN is reset to 0. The SAR is set to invalid. ❷ During the tREFI, the CAN provides a sequence number to each activation. For example, the first activation has CAN=1, second has CAN=2, and so on. ❸ When CAN is equal to SAN, it indicates that the given address is selected for mitigation, which means the row address is copied to SAR and SAR is set to be valid. ❹ At the next REF, if the SAR is valid, the row-address stored in SAR gets mitigated. For mitigation, we assume that the number of victim rows equivalent to the *Blast Radius* is refreshed on either side of the given aggressor row. The process repeats at each subsequent REF. MINT is a *single-entry tracker*, as only the *Selected Address Register (SAR)* holds the address of the row to be mitigated.

C. Impact of Classic Attacks: The Need for New Patterns

By design, MINT is robust against classic single-sided and double-side patterns that operate continuously during tREFI.

Single-Sided Attack: If a pattern repeatedly activates a given row, say Row-A (we use closed-page policy), during the entire tREFI window, then MINT is guaranteed to select this row for mitigation. Thus, MINT would limit such a classic single-sided attack to at-most M activations on the attacked row.

Double-Sided Attack: If a pattern repeatedly activates a pair of rows in alternating fashion and the pair shares a victim, say Row-A and Row-C with shared victim Row-B, then MINT is guaranteed to select one of the two rows (Row-A or Row-C), which will cause a refresh of victim (Row-B). Thus, MINT would limit the effect on victim to at-most M activations.

As classic attack patterns are not useful for analyzing the security of MINT, we investigate the worst-case pattern for MINT and use it to determine the TRH*.

D. Estimating the TRH* of MINT

Our analysis is based on three observations for MINT: (1) The selection decisions are localized to within the tREFI, so what happens in the previous or next tREFI does not impact the selection during current tREFI. (2) The probability of a row getting selected does not depend on the position in the tREFI interval, as all positions are equally likely to get selected, therefore reordering the address pattern within tREFI does not impact security (3) If a given row is activated n times within the same tREFI, it is n times more likely to get selected for mitigation (in the limit, if n=73, the row has guaranteed selection). So, successful attacks must avoid having a large number of activations to the same row within the single tREFI. These properties can help us identify the worst-case pattern.

Pattern-1: Single-Row, Single-Copy: This pattern focuses the attack on a single row (Row-A). During each tREFI, it performs only a single activation on Row-A and the remaining 72 activation slots remain unused. The pattern repeats 8192 times. During each tREFI, each activation of Row-A will get selected with probability p=1/73. We use the Sariou-Wolman model to determine the probability of failure (P_{REFW}) on the 8192nd activation. We use the P_{REFW} to determine TRH*. For this pattern, the **TRH* is 2461**.

Pattern-2: Multi-Row, Single-Copy: For faster attacks, the pattern must try to use all of the 73 slots in the tREFI interval. A simple way to achieve this is to perform a single activation to k attack rows within the single tREFI. If $P_{REFW}(1)$ is the failure probability for pattern-1, then with k lines, the failure probability increases by k times. So, $P_{REFW}(k) = k \cdot P_{REFW}(1)$. We use $P_{REFW}(k)$ to determine the TRH* for each k. Figure 10 shows the TRH* as k is varied from 1 to 73. We also evaluate a *multi-TREFI* attack containing more than 73 rows. The TRH* increases with k, peaks at k=73, and reduces thereafter for multi-TREFI pattern, as activations per row reduces. For pattern, with k=73, the **TRH* is 2763**.

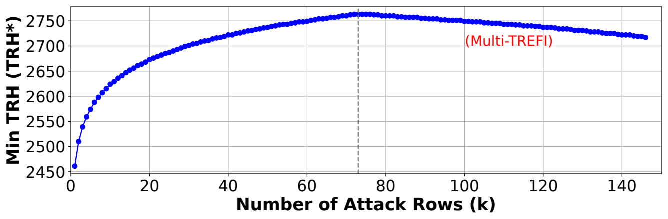

Fig. 10. TRH* for pattern-2 as the number of attack lines (k) is varied.

Pattern-3: Multi-Row, Multi-Copy: An attacker could spend the 73 activations on attacking k rows but activate each row c times. This attack is less stealthy as each row has c times higher chance of getting selected in each tREFI. Figure 11 shows the TRH* of MINT as the number of copies (c) per row is varied from 1 to 73 (the number of rows is adjusted to fit in one tREFI). With few copies (1-3), the TRH* of pattern-3 remains similar (within 0.5%) to pattern-2, however, it drops significantly for 4+ copies. Thus, having a large number of copies within tREFI is not an effective attack.

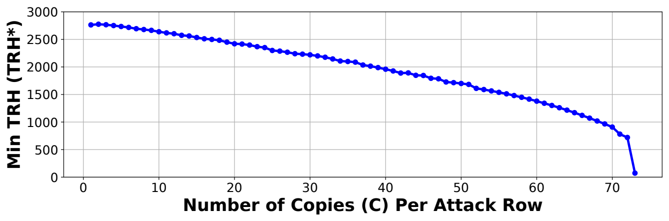

Fig. 11. TRH* for pattern-3 as the number of times a given attack row is activated within the pattern (copies) gets varied from 1 to 73.

Key Takeaway: To develop the most effective *direct* attack for MINT, the attacker is forced to have only 1 activation to any attack row within tREFI to maximize stealth. Under this constraint, we can use our analysis with pattern-2 (with 73 attack rows) to determine that MINT has **TRH* of 2763**.

E. Impact of Transitive Attacks

Thus far, we have only been focused on *direct* attacks, which aim to directly cause failures in the victim rows of a given row. *Transitive Attacks* [43], such as Half-Double [24], offer another indirect way to cause failure. Figure 12 (a) shows an example of such an attack, where Row-C receives continuous activations (using a single-sided attack). MINT will mitigate the neighbors of Row-C at each REF, causing 8192 victim refreshes on Row-B and Row-D. The activations from these mitigative refreshes are silent (not observable by MINT) and can cause failures in Row-A and Row-E. Thus, such an attack increases TRH* of MINT to 8192. Note that refreshing two rows on either side of an aggressor does not mitigate transitive attacks, as the third row now experiences failures.

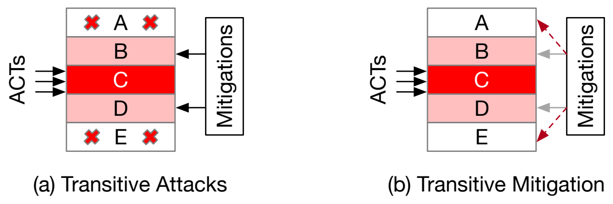

Fig. 12. (a) Transitive Attacks (B) Extending MINT with Transitive Mitigation, which performs mitigative refreshes on the victim of victim-rows.

We extend MINT to be secure against transitive attacks by using *transitive mitigation*. A transitive mitigation refreshes the victim of victim-rows. For example, a regular mitigation for Row-C would refresh Row-B and Row-D, however, a transitive mitigation would refresh Row-A and Row-E. Ideally, on each activation, if we do normal mitigations with probability p, we should do transitive mitigations with probability p^2.

Instead of choosing from 73 slots, we modify MINT to select from 74 slots (the extra slot indicating transitive mitigation for the recently mitigated row). As there are 74 slots, now URAND must be modified to select (0-73) positions, where 0 indicates transitive mitigation for the current address in SAR (if SAN is 0, SAR is preserved at REF, and indicates transitive mitigation). We note that the transitive mitigation can be applied recursively [13] (distance is increased if SAN=0 comes consecutively). As the probability of selection is reduced to 1/74, per pattern-2, the **TRH* of MINT is 2800.**

F. Impact of Spatial-Correlation Attacks

Our analysis thus far has assumed that the rows in a multi-row attack (pattern-2) are not spatially correlated. However, an attacker could try to sandwich a victim-row, between two attack rows to increase the hammers suffered by the victim row. This type of spatial correlation attack is present in the double-sided pattern, as shown in Figure 13(a), where victim Row-C is between two aggressor rows, Row-B and Row-D.

Such a pattern is challenging for prior counter-based trackers, that determine the selection decision based on counter-values. If both aggressor rows perform T activations before either one gets selected for a mitigation, then the victim-row is subjected to 2T activations. Thus, the effective threshold tolerated by counter-based schemes get doubled due to such a pattern. MINT is immune against such spatial patterns.

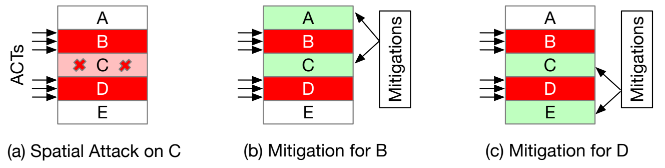

Fig. 13. Spatial correlation attacks (a) Double-sided attack on Row-C (b) Mitigation for B refreshes A and C (c) Mitigation for D refreshes C and E. Thus, C has as many chances of mitigations as summed over B and D.

MINT performs probabilistic selection. If Row-B is selected, then Row-A and Row-C get refreshed, as shown in Figure 13(b). If Row-D is selected, then Row-C and Row-E get refreshed, as shown in Figure 13(c). Thus, Row-C has as many chances for refresh as offered by *both* Row-B and Row-D. For example, if Row-B and Row-D each had 2800 activations (TRH*), then Row-C gets 5600 chances of refresh.

The TRH* of MINT indicates how many chances of mitigation can be escaped successfully. Thus, the total number of activations over Row-B and Row-D cannot exceed 2800, so, on average, each of the two rows can have up-to 1400 activations. This analysis also helps us bound the *Minimum Tolerated TRH (TRH-D*)* for a double-sided pattern. Note that, device characterization studies typically report TRH in terms of per-row activations for a double-sided pattern, so they report TRH-D*. The **TRH-D* of MINT is 1400**.

G. Comparison with Prior Trackers

We compare MINT with four designs, two counter-based (PRCT and Mithril [20]) and three probabilistic (PARFM [20], PrIDE [13], and InDRAM-PARA). InDRAM-PARA is our adaptation of PARA [23] to the in-DRAM setting, using a single-entry tracker, to highlight the differences between MINT and an alternative probabilistic selection. We compare these designs in terms of TRH-D*, the number of tracking entries, and the impact of Transitive Attacks.

PRCT: This is an idealized past-centric scheme that maintains a counter for each row. As all activations (including from victim refresh) increment the counter, PRCT is immune to transitive attacks. To determine TRH-D* of PRCT, we use the *Feinting Attack* [29], and start by spreading the available activations across 8192 aggressor rows. At each REF, the row with the highest counter value gets mitigated and removed from the list of aggressor rows. In the second-to-last round (8191), all the activations within tREFI are focused on the last two remaining aggressor rows. To determine, TRH-D*, we assume that the victim is placed in between the two aggressor rows. The TRH-D* of PRCT is 623.

Mithril: This counter-based past-centric design tracks only a subset of rows, and uses a proactive mitigation (i.e. it selects the row with the highest counter value for mitigation at each REF). As activations from mitigative refreshes increment the counter, this design is immune to transitive attacks. We use the Theorem-1 provided in the Mithril paper [20] to derive the number of counters required for a particular TRH-D*. Mithril requires at-least 677 entries to get TRH-D* of 1400.

PARFM: This is a past-centric probabilistic design that buffers all the activations during the tREFI window, and on reaching REF, it randomly selects one of the buffered entries to get mitigated, and invalidates all the buffered entries to free up space for the next tREFI. As there could be up-to 73 activations within the tREFI interval, this design requires an overhead of 73 entries per bank. As only the demand activations are used for selection, this design is vulnerable to transitive attacks. The TRH-D* of PARFM is 4096.

InDRAM-PARA: This is a current-centric probabilistic design. On each activation, a row is selected with probability *p* (p=1/73 for our study) and stored in a single-entry tracker. This row is mitigated only if it can *survive* until REF. We derive the TRH-D* of the InDRAM-PARA to be 3732. As the TRH* of this design is relatively high, direct attacks allow more unmitigated activations on a given row than transitive attacks, therefore, this design is immune to transitive attacks.

PrIDE [13]: This is a recently proposed probabilistic tracker. On each activation, PrIDE selects a row with probability *p* (e.g. p=1/73) and stores it in a four-entry FIFO buffer. At REF, the oldest entry in the FIFO is mitigated. PrIDE improves the *survival* probability but suffers from *Tardiness*. We derive the TRH-D* of the PrIDE to be 1750.

Table III compares the five trackers with MINT in terms of type, TRH-D*, entries, and vulnerability to Transitive Attacks. MINT has a TRH* similar to a 677-entry Mithril tracker, and has 2.25x the TRH* of the idealized PRCT design. This bound with PRCT becomes within 2x under refresh postponement, which we discuss next.

TABLE III
COMPARISON OF IN-DRAM TRACKERS

Design	Type (Centric)	TRH-D* (Threshold)	Entries (Per-Bank)	Transitive Attacks
PRCT	Past	623	128K	Immune
Mithril	Past	1400	677	Immune
PARFM	Past	4096	73	Vulnerable
InDRAM-PARA	Current	3732	1	Immune
PrIDE	Current	1750	4	Immune
MINT	Future	1400	1	Immune

VI. HANDLING REFRESH POSTPONEMENT

Thus far, we assumed that a refresh is performed at every tREFI. However, in reality, DDR5 specifications allow the postponement of up-to 4 refresh operations. A maximum of 5 refreshes can be batched and performed together, as shown in Figure 14. Refresh postponement increases the number of activations between refresh from 73 to 365.

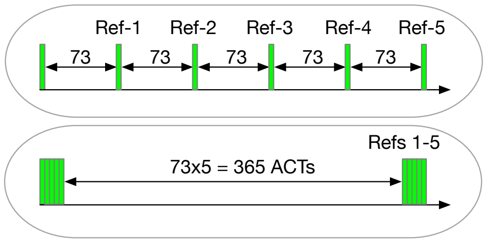

Fig. 14. Refresh postponement in DDR5 (top) Timely refresh (bottom) Batches of 5 refresh allowing up-to 365 ACTs between refresh.

A. Impact of Refresh Postponement on Optimal Trackers

Refresh postponement can allow the attacker to cause more activations on the aggressor row (selected for mitigation) during the period of refresh postponement. The impact of refresh postponement is easier to understand for counter-based trackers [29], as the threshold gets revised by an amount equal to the additional ACTs due to refresh postponement (so 73x4 = 292 in our case, split equally on either side of a double-sided attack). Thus, the TRH-D* of PRCT increases from 623 to 769. Similarly, the number of entries required by Mithril to achieve TRH-D* of 1400 increases from 677 to 827.

B. Impact of Refresh Postponement on Low-Cost Trackers

To the best of our knowledge, no prior work has studied the impact of refresh postponement on low-cost trackers. Refresh postponement is especially problematic for low-cost trackers as they track only a few entries, and are tailored for a given rate of mitigation (e.g. one per tREFI).

If the maximum number of activations within tREFI is M (73 in our case), then refresh postponement can make all activations past M *invisible* to the tracking mechanism. This is the case for both MINT and PARFM, as they are designed for only M activations within tREFI and a refresh thereafter. With refresh postponement, the attacker can perform activations on decoy rows during the first M activations, and then deterministically perform 4M activations on the attack row (without receiving any mitigations). Thus, refresh postponement can allow the attacker to deterministically perform **478K** (!) activations on a row every 32ms using MINT and PARFM.

For the InDRAM-PARA, the extra activations during the period of refresh postponement can cause the tracked entry to get dislodged easily. If the attacker places the attack row in the first position, then the probability of survival after another 364 activations is 0.66%. Refresh postponement increases the TRH-D* of the InDRAM-PARA from 3.7K to **21.3 K.**

Thus, refresh postponement demolishes existing low-cost trackers. We propose a generalized design that makes low-cost trackers compatible with refresh postponement.

C. Practical Solution: Delayed Mitigation Queue (DMQ)

As refresh postponement is in DDR5 standards, all trackers must support refresh postponement. To achieve this, we propose a general solution, *Delayed-Mitigation Queue (DMQ)*. Figure 15 shows the overview of our design. We consider a generic low-cost tracker (e.g. MINT, PARFM, InDRAM-PARA etc.). DMQ requires the tracker to count the number of activations *(NumACTs)* since the last refresh (MINT already does this with CAN, but other trackers may need an additional register). If NumACTs exceeds the maximum number of activations in tREFI (e.g. 73), it is reset to 1, and the tracker performs a *pseudo-mitigation*. During pseudo-mitigation, the tracker provides the address of the selected aggressor row (e.g. stored in SAR for MINT and InDRAM-PARA), which is inserted into a FIFO buffer called the *DMQ*. The DMQ has four entries (as up-to four refreshes can be postponed). On REF, if the DMQ contains at-least one valid entry, the oldest entry from the DMQ is mitigated. Else, the tracker selects and mitigates normally, similar to no refresh postponement.

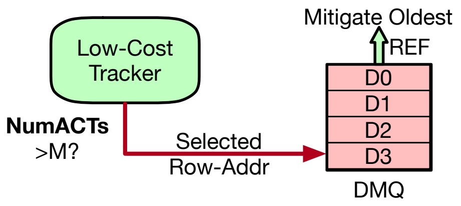

Fig. 15. Design of Delayed-Mitigation Queue (DMQ). DMQ makes low-cost in-DRAM trackers compliant with refresh postponement.

D. Bounding the Impact of DMQ on Tolerable TRH

As DMQ is a FIFO, the maximum number of activations for which the row selected by the tracker will get delayed in receiving a mitigation while waiting in the DMQ is 292 (73x4). At the worst-case, the pattern may be accessing the same row continuously, in which case the row may receive up-to 292 activations while in the DMQ, so the TRH* of the tracker would increase by 292, or equivalently, the TRH-D* would increase by 146. Thus, DMQ makes the impact of refresh postponement on low-cost trackers, similar to counter-based trackers. Table IV shows the impact of refresh postponement (with and without DMQ). As MINT forces a pattern that has a single activation of a row within tREFI, delaying the mitigation by four tREFI causes only four more activations, so the TRH-D* of MINT increases by 4 to 1404 (it can be increased to 1482* using an adaptive attack, shown in Appendix B). **The TRH-D* of MINT is 1482, thus (outperforming 677-entry Mithril and 1.9x of PRCT)**.

TABLE IV
IMPACT OF REFRESH POSTPONEMENT AND DMQ ON TRACKERS

Design	Entries (Bank)	TRH-D* (NoPostpone)	TRH-D* (No DMQ)	TRH-D* (with DMQ)
PRCT	128K	623	769	769
Mithril	677	1400	1546	1546
PARFM	73	4096	478K	4242
InDRAM-PARA	1	3732	21.3K	3650
PrIDE	4	1750	6500	1900
MINT	1	1400	478K	1404/1482*

VII. Scaling to Lower Thresholds with RFM

The TRH* of MINT can be reduced by increasing the mitigation-rate. DDR5 specifications include *Refresh Management (RFM)* that enables the memory controller to provide more time to the DRAM chip to perform mitigation. RFM enables the memory controller to issue additional mitigations (RFM commands) to the DRAM when the activations per bank crosses a threshold. The memory controller maintains a *Rolling Accumulation of ACTs (RAA)* counter [20]) per bank. When any RAA counter reaches a given threshold (*RAAMMT*), the memory controller issues an RFM and reduces the RAA counters by a given amount (*RAAIMT*). Furthermore, RAA counter is reduced by a given amount (e.g. RAAIMT) on REF.

We co-design MINT with RFM to scale to lower thresholds. We evaluate two variations: with a RAAIMT of 32 (MINT+RFM32) and 16 (MINT+RFM16). As the number of activations within the mitigation period is now 32 or 16, we modify MINT to select URAND(0,32) or URAND(0,16).

Table V shows the TRH-D* for MINT and MINT+RFM. We evaluate various mitigation rates, including 0.5x (one mitigation every two tREFI), 1x (one mitigation every tREFI), RFM32 ($\sim$2x mitigation rate), and RFM16 ($\sim$4x mitigation rate). We use RAAMMT of 5 times RAAIMT, thus, our solution must allow RFM postponing by up-to 4x. We implement MINT+RFM with DMQ and report the threshold under an adaptive attack. MINT+RFM16 has a **TRH-D* of 356**.

TABLE V
THE TRH-D* OF MINT AND MINT+RFM (INCLUDES DMQ)

Scheme	Relative Mitigation Rate	TRH-D*
MINT	0.5x (one per two tREFI)	2.70K
MINT	1x (one per tREFI)	1.48K
MINT+RFM32	2x (approx two per tREFI)	689
MINT+RFM16	4x (approx four per tREFI)	**356**

VIII. Results

A. Impact on Performance

We model MINT and MINT+RFM in a detailed memory system simulator (memsim [2]). Table VI shows our configuration. We model a 8-core out-of-order CPU with DDR5, using Micron Datasheet [31]. We evaluate our design with 22 SPEC2017 [42], 6 GAP [35] and 4 stream workloads. All workloads are run in 8-core rate mode. We use a representative slice of 100 million instructions. Per DDR5 specifications, we assume tDRFM$_{sb}$ is equal to tRFC (240ns) while tRFM$_{sb}$ is equal to half the time of tRFC (205ns). We assume that both RFM and DRFM stall the same bank in all bankgroups (so, for each RFM/DRFM, 8 banks get stalled).

TABLE VI
BASELINE SYSTEM CONFIGURATION

Out-of-Order Cores	8 core, 4GHz, 4-wide, 256-ROB
Last Level Cache (Shared)	8MB, 16-Way, 64B lines
Memory specs	32 GB, DDR5
t_{RCD}-t_{CL}-t_{RP}-t_{RC}	12-12-32-48 ns
Banks x Ranks x Sub-Channels	32$\times$1$\times$2
Rows	128K rows, 4KB row buffer

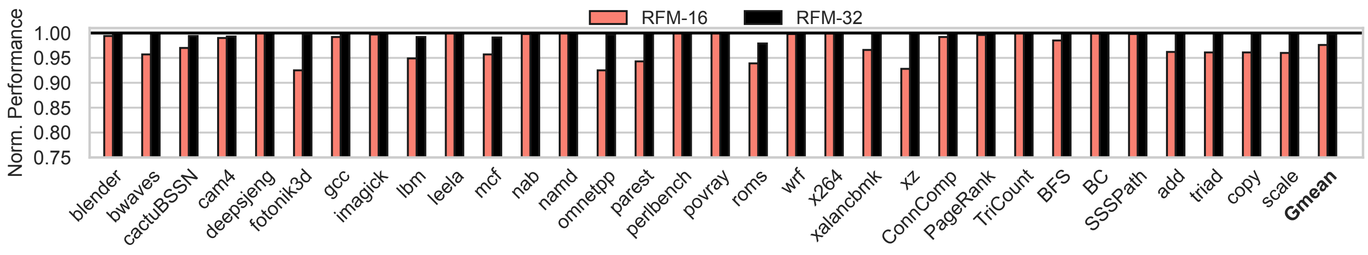

Fig. 16. Normalized performance. MINT incurs zero slowdown. The bar labeled Gmean represents the Geometric mean over all the 32 workloads. MINT + RFM32 and MINT + RFM16 incur a 0.2% and 2. 5% slowdown, respectively.

Figure 16 shows the relative performance of MINT and MINT+RFM, normalized to the DDR5 baseline. MINT incurs zero slowdown, as the mitigations required for MINT are performed within the tRFC period as mentioned in DDR5 specifications. MINT+RFM32 incurs negligible slowdown and MINT+RFM16 incurs an average slowdown of 2.5%. Thus, MINT and MINT+RFM provide a scalable and low overhead mitigation for Rowhammer, even at low thresholds.

B. Impact of Target Time-to-Fail on Threshold

As MINT is a probabilistic design; we deem it to be secure if the time-to-fail is greater than the *Target Time-to-Fail (Target-TTF)*. We used a default Target-TTF of 10,000 years per bank as it is similar to per-bank failure-rate similar from naturally occurring errors [4]. Table VII shows TRH-D* of MINT for varying Target-TTF (per-bank) and MTTF for our system (64 banks, but only 22 can be used concurrently due to tFAW). MINT provides several decades/centuries of protection even under continuous attacks on all available banks.

TABLE VII
THE TRH-D* OF MINT FOR VARIOUS TARGET-TTF

Target-TTF (Bank)	MTTF (System)	TRH-D* MINT	TRH-D* (+RFM32)	TRH-D* (+RFM16)
1K years	45 years	1.40K	651	336
10K years	**450 years**	**1.48K**	**689**	**356**
100K years	4.5K years	1.57K	726	375
1Million years	45K years	1.64K	763	395

C. Storage Overheads

MINT requires a CAN (7-bits), SAN (7-bits) and SAR (18 bits), for a total of 32 bits (4 bytes). The DMQ requires 4 entries (19 bits, including one for transitive-mitigation) for a total of 9.5 bytes. Thus, MINT+DMQ requires less than 15 bytes per bank. It also requires an in-DRAM pseudo-random number generator, similar to DSAC [12] and PAT [22].

D. Energy Overheads

The energy overheads of MINT can be attributed to three sources (1) the random number generator (RNG) that is consulted at each tREFI to derive the SAN, (2) the additional structure of DMQ, and (3) the extra activations to perform mitigative refreshes. Our energy overheads include the energy incurred in all three sources.

MINT uses a 7-bit TRNG [17], [49], which consumes 90 micro-watts of static power and 200 micro-watts of dynamic power. The total power of the TRNG (290 micro-watts) is three orders of magnitude lower than the DRAM power.

We use CACTI-6.5 to estimate the DMQ power. DMQ consumes static power of 48 micro-watts and dynamic power of 38 micro-watts. The total power of DMQ (86 microwatts) is three orders of magnitude lower than the DRAM power.

Table VIII shows the relative memory energy consumption (including DMQ and RNG) of MINT and MINT+RFM normalized to the baseline. To determine DRAM energy, we use the Micron power calculator [32]. We split the DRAM energy into three parts: ACT+RD/WR, Background, and Refresh. The mitigations due to MINT increase the refresh energy. MINT increases the DRAM energy by 0.8%, and when combined with RFM it increases the average energy by 2% or 4.4%.

TABLE VIII
MEMORY ENERGY OVERHEADS OF MINT AND MINT+RFM

Config	ACT+RD/WR	Background	Refresh	Total
Base (No Mitig)	66.3%	22.3%	11.3%	100%
MINT	66.3%	22.3%	12.1%	100.8%
MINT+RFM32	66.3%	22.4%	13.3%	102.0%
MINT+RFM16	66.3%	22.9%	15.2%	104.4%

E. Comparison with Memory-Controller-Based PARA

We compare MINT with PARA implemented on the MC-side (MC-PARA). Fig 17 shows the average slowdown of MC-PARA and MINT tuned for similar TRH*. MINT can do mitigations transparently (within REF) and incur RFM overheads only when ACT count exceeds RFMTH. MC-PARA relies on DRFM, which means all mitigations block the bank from service. On average, MC-PARA incurs a slowdown of 1.8%-8%, whereas MINT incurs a slowdown of 0%-2.5%.

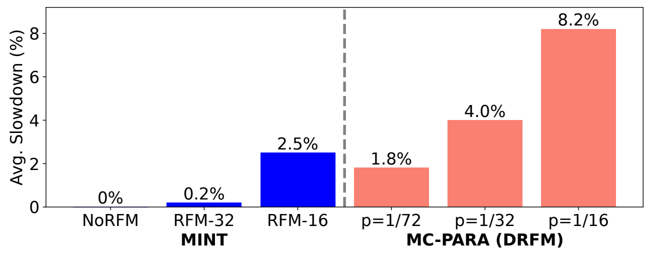

Fig. 17. Performance of MINT and MC-PARA (DRFM). DRFM-based implementation incurs higher performance overheads.

IX. Related Work

Per-Row-Activation-Counting (PRAC): Concurrent to our submission, JEDEC announced [1] an update to DDR5 specifications with *PRAC* [15]. PRAC extends the DRAM array to have a counter and uses that counter to track activations. The timing of memory operations (e.g. tRC) is changed to perform a read-modify-write of the counter for each activation. While PRAC is a principled defense for Rowhammer, we note that PRAC is an optional feature and DRAM manufacturers are not required to support PRAC. A recent Hynix [22] design showed that the area overhead of supporting per-row counters is approximately 9%. Furthermore, PRAC specifications change the tRC timings from the current 46ns-48ns to 52ns (almost 10% higher). DRAM industry is extremely cost-sensitive. If there is a secure low-cost method to mitigate Rowhammer, then the memory companies can avoid the significant area, power, and timing overheads of PRAC, and choose the low-cost alternative. MINT offers such an alternative. Our analysis with feinting-style attacks shows that the threshold of MINT is within 2x of Per-Row-Counter-Table (PRCT).

Other One-Counter-Per-Row Designs: CRA [18] and Hydra [34] keep the counters in DRAM and use caches or filters to reduce the DRAM lookups for counters. However, they can incur large slowdowns for the worst-case patterns.

Efficient-Counters: Several proposals reduce the SRAM overhead of aggressor-row tracking. Table IX compares the per-bank SRAM overheads of Graphene [33]. MINT has significantly lower SRAM overheads, especially at low TRH.

TABLE IX
Per-Bank SRAM Overhead of Trackers (per-rank will be 32x)

Name	Device TRH-D=3K	Device TRH-D=300
Graphene	56.5 KB	565 KB
MINT+DMQ	**15 bytes**	**15 bytes**

Secure Low-Cost Tracker: Our recent work proposes *PrIDE* [13], a secure low-cost in-DRAM tracker. The InDRAM-PARA design we discuss in this paper is equivalent to single-entry PrIDE. PrIDE uses a 4-entry FIFO to reduce the *loss probability* (from 63% to 10%) but suffers from *Tardiness*. In the PrIDE terminology, MINT has zero loss-probability and zero Tardiness (pattern-2). The TRH-D* of PrIDE is 1750 (25% higher than MINT). Refresh postponement causes the TRH-D* of PrIDE to increase to 6.5K. PrIDE with DMQ has a TRH-D* of 1900 (28% higher than MINT+DMQ).

Mitigating-Actions: For MINT, we use victim refresh for mitigation. RRS [36], AQUA [40], SRS [46], SHADOW [45] perform mitigation with row migration, whereas, Blockhammer [47] uses rate limits. REGA [30] changes the DRAM circuitry to provide mitigating refresh on each demand activation. HiRA [48] changes the interface to allow multiple activations per bank. MINT avoids changes to DRAM array and interface.

ECC-Codes: SafeGuard [7], CSI-RH [16], PT-Guard [39], and Cube [21] use ECC codes to tolerate Rowhammer failures, however, uncorrectable failures can still cause data loss.

X. Conclusion

Current in-DRAM trackers for Rowhammer mitigation are either insecure or require prohibitively-high cost. This paper develops *Minimalist In-DRAM Tracker (MINT)* to provide secure Rowhammer mitigation with a single entry. The key insight that enables MINT is that instead of selecting the aggressor row based on the past or the current, MINT selects one in the future (a random row in the upcoming interval). We also study the compatibility of low-cost trackers with refresh postponement, and propose *Delayed Mitigation Queue (DMQ)* as a generalized solution. We show that MINT can securely protect devices with a double-sided TRH of 1482 and as low as 356 when combined with RFM. The storage, performance, and energy overheads of MINT are negligible.

Acknowledgements

We thank the reviewers of MICRO-2024 for their feedback. This research was supported in part by a gift from Intel. Thanks to Hritvik Taneja for DRAMSim data (Appendix-D).

Appendix A: Impact of MaxACT

One of the key parameters that determines the efficacy of in-DRAM trackers is the mitigation rate. Our default mitigation rate is 1 aggressor row per tREFI. Given our default timing parameters, we can have a maximum activations (MaxACT) of 73 per tREFI. In this section, we vary MaxACT.

JEDEC specifies a range of memory timings and memory companies decide which specifications to support. For example, for DDR5, JEDEC specifies 11 data transfer rates (DDR5-3200 to DDR5-7200, once every 400), and for each rate, they specify four *speed-bins* (A, AN, B, BN). Across all these 44 specifications, the minimum tRC is 46ns and maximum tRC is 49.5ns (52ns for revised specs with PRAC). For DDR5, tREFW=32ms, and tRFC=350ns or 410ns. Thus, MaxACT can range from 67.2 to 77.2 for the entire DDR5 specs.

Figure 18 shows the TRH-D* supported by MINT and InDRAM-PARA as the MaxACT is varied from 65 to 80. The viable range of MaxACT for the DDR5 specifications is highlighted in green. The relative difference between MINT and InDRAM-PARA remains at 2.7x throughout the DDR5 range (and even outside). Thus, the advantage MINT is not limited to a specific choice of MaxACT.

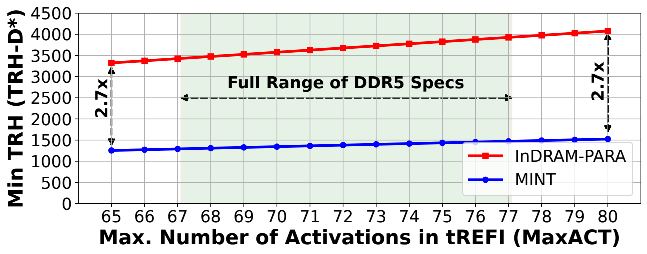

Fig. 18. Impact of varying the maximum number of activations (MaxACT) within tREFI for InDRAM-PARA and MINT. MINT continues to have 2.7x lower TRH-D* than InDRAM-PARA across the entire range of MaxACT.

Appendix B: Adaptive Attacks on MINT+DMQ

The Pattern: The best attack for MINT is to activate a given aggressor row within tREFI only once (to evade selection). The best attack for DMQ is to activate the same row repeatedly, as it allows more activations on the selected row while waiting in the DMQ for mitigation. Therefore, using a single pattern is not ideal for the MINT+DMQ. We develop an Adaptive Attack (ADA) on DMQ that changes the pattern from what is optimal for MINT (pattern-2) to what is optimal for DMQ (repeated activations) at a predefined *morphing-point (MP)*. Figure 19 shows an overview of ADA.

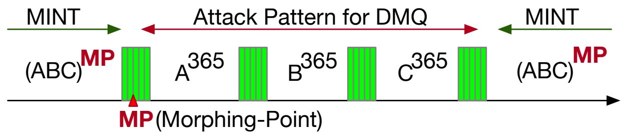

Fig. 19. ADA morphs the pattern from MINT to DMQ at morphing-point. We show 3 rows for simplicity, MINT pattern has 73 lines.

As refresh postponement can allow 365 activations on a given row, this switching can allow ADA to cause 365 activations on an aggressor row (without any intervening mitigation), however, after ADA, this row is guaranteed to get mitigated. Thus, if the row had A activations at the morphing-point, then ADA can cause the row to have $A + 365$ activations.

The Model for Threshold: For ADA to be effective it must find a row with at-least (TRH* - 365) activations, otherwise the row is guaranteed to get mitigated before reaching TRH*. Similarly, if the row already has TRH*, it fails without requiring ADA. As the attacker does not know the activation-counts of a given row within the given tREFW window, the key decision for ADA is to set the morphing-point (in terms of tREFI), with the useful range from (TRH* - 365) to (8192 - 365). To compute the probability of finding a given row with a given activation count (A) at a given tREFI interval, we use a Markov-Chain, as shown in Figure 20. For pattern-2, the row can have an activation-count from 1 to 8192 (states above TRH* indicate failure with MINT). We determine the probability of finding a row with A activations and then increase it by 365 due to ADA. We use these probabilities to compute the P_{REFW}, MTTF, and TRH*.

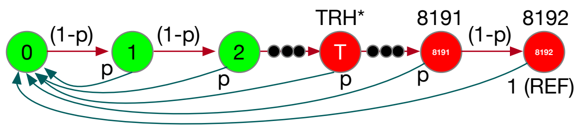

Fig. 20. Markov-Chain for activation count of a row. Each state denotes the activation counts since the last mitigation or refresh. The parameter p denotes the probability of mitigation. Red/Green denotes states with/without an error.

The Impact: Figure 21 shows the TRH* of ADA as the morphing-point is varied from 500 to 8000. We evaluate both single-sided and double-sided versions. For single-sided version, ADA starts to become effective only after MP of 2400 (before this the TRH* remains 2763, same as without ADA), and provides the highest TRH* of 2899 at MP between 2533-3730, beyond this the TRH* drops to 2847. The reason for this behavior is a smaller MP allows for the attack to be repeated multiple times within the same tREFW. For the double-sided pattern, the attack starts to become effective after MP of 1200, and provides the highest TRH-D* of 1282 between MP of 1299 and 1456, after which TRH-D* reduces to 1474. The earlier success point for double-sided attack is because TRH-D* (without ADA) is much lower than TRH* (without ADA). Overall, this analysis shows that under adaptive attacks, **TRH-D* of MINT+DMQ is 1482**.

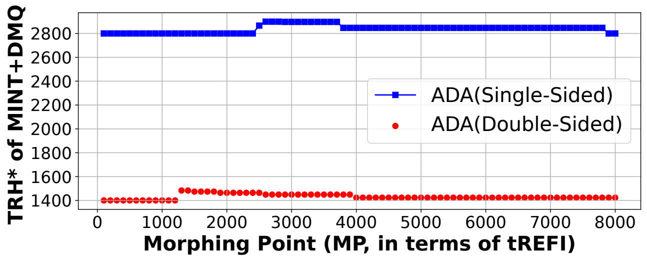

Fig. 21. Threshold of MINT+DMQ with ADA, as the Morphing-Point (MP) is varied. ADA has TRH* of 2899 and TRH-D* of 1482. The double-sided attack has shorter period for DMQ attack and an earlier MP.

Appendix C: Tolerating Row-Press with MINT

Row-Press [28] is a new vulnerability that arises when a row is kept open for a long time. The charge in the nearby rows continues to slowly leak through the bit lines. Each round of Row-Press incurs one activation and keeps the row open for a period of tON (tON can be between tRAS and 5*tREFI). Due to the extra charge leaked during tON, Row-Press can cause bit flips in much fewer activations than TRH.

Our concurrent work, ImPress [38], enables in-DRAM trackers to mitigate Row-Press without affecting the tolerated TRH. The key idea is to convert the row open time into an equivalent number of activations (EACT) for Rowhammer mitigation. Thus, rows that are kept open for a longer time have higher EACT and therefore a higher rate of mitigation. EACT is given by Equations 9.

$$EACT = (tON + tPRE)/tRC \tag{9}$$

ImPress requires a timer to track tON. The division (with tRC) is implemented with a shift operation. EACT can have up to 7-bits of fractional part. To tolerate Row-Press with MINT, we must change the 7-bit CAN register to a fixed-point register (7+7=14 bits). For each activation, CAN is incremented by an amount equal to EACT. When the value of CAN crosses SAN, the row causing the activation is stored in SAR. MINT combined with ImPress can tolerate Row-Press without affecting the TRH*. With ImPress, the total storage overhead increases from 15 to 17 bytes per bank.

Thus, MINT when combined with ImPress offers a practical solution to tolerate both Rowhammer and Row-Press with a minimalist hardware solutions that incurs negligible storage overhead and negligible performance overhead.

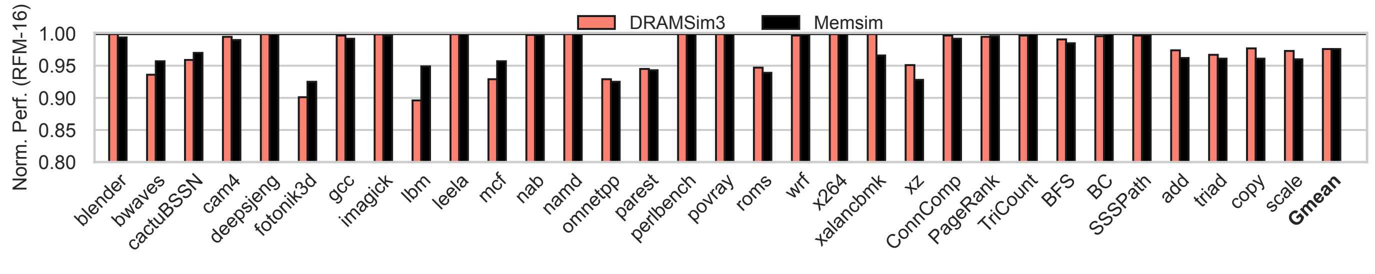

Fig. 22. Performance Impact of MINT+RFM16 using two simulators (a) DRAMSim3 (b) Memsim. The bar labeled Gmean represents the Geometric mean over all the 32 workloads. Both setups show an average slowdown of 2.4%.

Appendix D: Validation with DRAMSim3

We use our memory system simulator *memsim* [2] for performance evaluations. Memsim models the memory system in detail (e.g. banks, channels, queues, refresh, mapping, and scheduling) with key timing parameters. It is designed for speed and ease of use. We validate memsim with a cycle-accurate memory system simulator, DRAMSim3 [27]. Figure 22 shows the slowdown of MINT+RFM16 with memsim and DRAMSim3. We observe a good correlation between the two simulators (the average difference is 0.9%). Both memsim and DRAMSim3 show an average slowdown of 2.4%.

Appendix-E: Artifact

A. Abstract

This artifact presents the code for MINT, our Rowhammer mitigation. We provide the C++ code for security analysis, determining the minimum Rowhammer threshold (TRH*) protected by MINT and other related schemes.

We provide scripts and code for security analysis and recreate the key results – Figure 3, Figure 5, Figure 6, Figure 10 Figure 11, Figure 18, and Figure 21. We also show key results for prior schemes shown in Table III. We also provide (optional) artifacts to generate performance results as shown in Figure 16. The artifact is available at: https://doi.org/10.5281/zenodo.13363054. The latest version of the Memsim simulator is available at https://github.com/mqureshi4/memsim

B. Artifact check-list (meta-information)

- **Algorithm**: MINT mitigation.
- **Compilation**: Tested with g++ (Apple clang-1500.3.9.4)
- **Run-time environment**: Tested on Mac OS Sonoma 14.5.
- **Hardware**: The artifact can be run on a laptop (we used the Macbook Air M2) or a Linux machine.
- **Execution**: Analytical models and Performance Models.
- **Metrics**: TRH* (single-sided and double-sided)
- **Output**: Recreating security results: Figure 3, Figure 5, Figure 6, Figure 10 Figure 11, Figure 18, and Figure 21.
- **Experiments**: Instructions to run the analysis and plot graphs are available in the README file.
- **How much disk space required (approximately)?**: 1 GB.
- **How much time is needed to prepare workflow (approximately)?**: 5 minutes
- **How much time is needed to complete experiments (approximately)?**: 5 minutes (security) + 2 hours (performance)
- **Publicly available?**: Yes.
- **Workflow framework used?**: Analytical Models.
- **Archived (provide DOI)?**: Yes, the DOI for the artifact is https://doi.org/10.5281/zenodo.13363054.

C. Description

1) How to access: The code and instructions are at: https://zenodo.org/records/13731504.

2) Hardware dependencies: Security evaluations can be run on most generic Linux machines or a laptop.

3) Software dependencies: g++ is used for compilation and Python3 with matplotlib and numpy to plot graphs.

D. Experiment workflow

The README provides detailed instructions to reproduce the results of the paper:

- Run the ./gengraphs.sh script to perform the security analysis and display the corresponding figures.

E. Evaluation and expected results

The artifact provides the scripts to run both the security evaluations and display the graphs. The relevant commands are provided in the artifact README. The expected results from this artifact are to recreate the key security results – Figure 3, Figure 5, Figure 6, Figure 10 Figure 11, Figure 18, and Figure 21. We also show key results for the previous schemes shown in Table III.

F. Experiment customization

Scripts are self-contained and do not need customization.

G. Optional: Performance Evaluation

MINT itself does not incur any performance overheads, as the mitigation happens under REF. MINT can optionally be combined with RFM to scale to a lower threshold. However, the performance overhead of RFM is agnostic to the underlying implementation of the in-DRAM tracker and is purely dependent on activation counts and RFM thresholds.

We also provide optional artifacts to reproduce the results for Fig. 16. To run these experiments, go to `memsim/SCRIPTS` and type `./runall.sh`. These experiments may take 2-3 hours (on an 8-core laptop). Once the experiments are completed, the script should automatically generate Figure 16. For any glitches, see the error messages.

References

[1] "JEDEC Updates JESD79-5C DDR5 SDRAM Standard: Elevating Performance and Security for Next-Gen Technologies," https://www.jedec.org/news/pressreleases/jedec-updates-jesd79-5c-ddr5-sdram-standard-elevating-performance-and-security.

[2] "Memsim": Simulating Memory Systems with Ease and Speed. https://github.com/mqureshi4/memsim.

[3] 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," *ACM SIGPLAN Notices*, vol. 51, no. 4, pp. 743–755, 2016.

[4] M. V. Beigi, Y. Cao, S. Gurumurthi, C. Recchia, A. Walton, and V. Sridharan, "A systematic study of ddr4 dram faults in the field," in *2023 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*, 2023.

[5] L. Cojocar, K. Razavi, C. Giuffrida, and H. Bos, "Exploiting correcting codes: On the effectiveness of ecc memory against rowhammer attacks," in *2019 IEEE Symposium on Security and Privacy (SP)*. IEEE, 2019, pp. 55–71.

[6] 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 21*, 2021.

[7] 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 *2022 IEEE International Symposium on High Performance Computer Architecture (HPCA)*. IEEE, 2022.

[8] P. Frigo, E. Vannacc, 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 *2020 IEEE Symposium on Security and Privacy (SP)*. IEEE, 2020, pp. 747–762.

[9] 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 *2018 IEEE Symposium on Security and Privacy (SP)*. IEEE, 2018, pp. 245–261.

[10] 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*, J. Caballero, U. Zurutuza, and R. J. Rodríguez, Eds. Cham: Springer International Publishing, 2016, pp. 300–321.

[11] 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-54: 54th Annual IEEE/ACM International Symposium on Microarchitecture*, 2021, pp. 1198–1213.

[12] 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 preprint arXiv:2302.03591*, 2023.

[13] A. Jaleel, G. Saileshwar, S. Keckler, and M. Qureshi, "PrIDE: Achieving Secure Rowhammer Mitigation with Low-Cost In-DRAM Trackers," in *Annual International Symposium on Computer Architecture*, 2024.

[14] P. Jattke, V. van der Veen, P. Frigo, S. Gunter, and K. Razavi, "BLACKSMITH: Rowhammering in the Frequency Domain," in *43rd IEEE Symposium on Security and Privacy'22 (Oakland)*, 2022, https://comsec.ethz.ch/wp-content/files/blacksmith_sp22.pdf.

[15] JEDEC, "JESD79-5C: DDR5 SDRAM Specifications," 2024.

[16] J. Juffinger, L. Lamster, A. Kogler, M. Eichlseder, M. Lipp, and D. Gruss, "Csi: Rowhammer-cryptographic security and integrity against rowhammer," in *2023 IEEE Symposium on Security and Privacy (SP)*. IEEE Computer Society, 2022, pp. 236–252.

[17] O. Katz, D. A. Ramon, and I. A. Wagner, "A robust random number generator based on a differential current-mode chaos," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 16, no. 12, pp. 1677–1686, 2008.

[18] D.-H. Kim, P. J. Nair, and M. K. Qureshi, "Architectural support for mitigating row hammering in dram memories," *IEEE CAL*, vol. 14, no. 1, pp. 9–12, 2014.

[19] J. S. Kim, M. Patel, A. G. Yağlıkçı, H. Hassan, R. Azizi, L. Orosa, and O. Mutlu, "Revisiting rowhammer: An experimental analysis of modern dram devices and mitigation techniques," in *ISCA*. IEEE, 2020, pp. 638–651.

[20] 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 *2022 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*. IEEE, 2022, pp. 1156–1169.

[21] M. J. Kim, M. Wi, J. Park, S. Ko, J. Choi, H. Nam, N. S. Kim, J. H. Ahn, and E. Lee, "How to kill the second bird with one ecc: The pursuit of row hammer resilient dram," in *56th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, 2023.

[22] W. Kim, C. Jung, S. Yoo, D. Hong, J. Hwang, J. Yoon, O. Jung, J. Choi, S. Hyun, M. Kang, S. Lee, D. Kim, S. Ku, D. Choi, N. Joo, S. Yoon, J. Noh, B. Go, C. Kim, S. Hwang, M. Hwang, S.-M. Yi, H. Kim, S. Heo, Y. Jang, K. Jang, S. Chu, Y. Oh, K. Kim, J. Kim, S. Kim, J. Hwang, S. Park, J. Lee, I. Jeong, J. Cho, and J. Kim, "A 1.1v 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 *2023 IEEE International Solid- State Circuits Conference (ISSCC)*, 2023, pp. 1–3.

[23] 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," *ISCA*, 2014.

[24] 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 Symposium*, 2022.

[25] A. Kwong, D. Genkin, D. Gruss, and Y. Yarom, "Rambleed: Reading bits in memory without accessing them," in *2020 IEEE Symposium on Security and Privacy (SP)*. IEEE, 2020, pp. 695–711.

[26] E. Lee, I. Kang, S. Lee, G. E. Suh, and J. H. Ahn, "TWiCe: preventing row-hammering by exploiting time window counters," in *ISCA*, 2019.

[27] S. Li, Z. Yang, D. Reddy, A. Srivastava, and B. L. Jacob, "DRAMsim3: A Cycle-Accurate, Thermal-Capable DRAM Simulator," *IEEE Comput. Archit. Lett.*, vol. 19, no. 2, pp. 110–113, 2020.

[28] H. Luo, A. Olgun, A. G. Yağlıkçı, Y. C. Tuğrul, S. Rhyner, M. B. Cavlak, J. Lindegger, M. Sadrosadati, and O. Mutlu, "RowPress: amplifying read disturbance in modern dram chips," in *Proceedings of the 50th Annual International Symposium on Computer Architecture*, ser. ISCA '23. New York, NY, USA: Association for Computing Machinery, 2023. [Online]. Available: https://doi.org/10.1145/3579371.3589063

[29] M. Marazzi, P. Jattke, F. Solt, and K. Razavi, "Protrr: Principled yet optimal in-dram target row refresh," in *IEEE Symposium on Security and Privacy (SP)*. IEEE, 2022, pp. 735–753.

[30] M. Marazzi, F. Solt, P. Jattke, K. Takashi, and K. Razavi, "REGA: Scalable Rowhammer Mitigation with Refresh-Generating Activations," in *IEEE Symposium on Security and Privacy (SP)*. IEEE, 2023.

[31] "DDR5 SDRAM Datasheet: Directed Refresh Management (DRFM), Page-290," Micron Technology Inc., 2022. [Online]. Available: https://www.micron.com/-/media/client/global/documents/products/data-sheet/dram/ddr5/ddr5_sdram_core.pdf

[32] Micron Technology Inc., "System Power Calculators," https://www.micron.com/support/tools-and-utilities/power-calc.

[33] 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*. IEEE, 2020, pp. 1–13.

[34] 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 *Proceedings of the 49th Annual International Symposium on Computer Architecture*, 2022, pp. 699–710.

[35] K. A. S. Beamer and D. Patterson, "The gap benchmark suite," in *arXiv preprint arXiv:1508.03619*, 2015.

[36] 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, pp. 1056–1069.

[37] S. Saroiu and A. Wolman, "How to configure row-sampling-based rowhammer defenses," in *Workshop on DRAM Security (DRAMSec)*, 2022.

[38] A. Saxena, A. Jaleel, and M. Qureshi, "Impress: Securing dram against data-disturbance errors via implicit row-press mitigation," in *2024 57th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, 2024.

[39] 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 *IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*, 2023.

[40] A. Saxena, G. Saileshwar, P. J. Nair, and M. Qureshi, “Aqua: Scalable rowhammer mitigation by quarantining aggressor rows at runtime,” in *MICRO*. IEEE, 2022, pp. 108–123.

[41] M. Seaborn and T. Dullien, “Exploiting the DRAM rowhammer bug to gain kernel privileges,” *Black Hat*, vol. 15, p. 71, 2015.

[42] “SPEC CPU2017 Benchmark Suite,” Standard Performance Evaluation Corporation. [Online]. Available: http://www.spec.org/cpu2017/

[43] L. C. Stefan Saroiu, Alec Wolman, “The price of secrecy: How hiding internal dram topologies hurts rowhammer defenses,” in *Proceedings of International Reliability Physics Symposium (IRPS)*, 2022.

[44] 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*, 2016.

[45] 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)*. IEEE, 2023, pp. 333–346.

[46] J. Woo, G. Saileshwar, and P. J. Nair, “Scalable and secure row-swap: Efficient and safe row hammer mitigation in memory systems,” in *HPCA*. IEEE, 2023.

[47] A. G. Yağlikçi, M. Patel, J. S. Kim, R. Azizi, A. Olgun, L. Orosa, H. Hassan, J. Park, K. Kanellopoulos, T. Shahroodi *et al.*, “Blockhammer: Preventing rowhammer at low cost by blacklisting rapidly-accessed dram rows,” in *HPCA*. IEEE, 2021, pp. 345–358.

[48] A. G. Yağlikçi, 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.

[49] F. Yu, L. Li, Q. Tang, S. Cai, Y. Song, and Q. Xu, “A survey on true random number generators based on chaos,” *Discrete Dynamics in Nature and Society*, vol. 2019, pp. 1–10, 2019.

[50] Z. Zhang, Y. Cheng, D. Liu, S. Nepal, Z. Wang, and Y. Yarom, “Pthammer: Cross-user-kernel-boundary rowhammer through implicit accesses,” in *MICRO*. IEEE, 2020, pp. 28–41.