PrIDE: Achieving Secure Rowhammer Mitigation with Low-Cost In-DRAM Trackers

Aamer Jaleel
NVIDIA
ajaleel@nvidia.com

Gururaj Saileshwar*
University of Toronto
gururaj@cs.toronto.edu

Stephen W. Keckler
NVIDIA
skeckler@nvidia.com

Moinuddin Qureshi
Georgia Tech
moin@gatech.edu

***Abstract*—Rowhammer-induced bit-flips are a threat to DRAM security. To mitigate Rowhammer, DDR4 devices employ TRR, an in-DRAM tracker, to identify aggressor rows. In-DRAM trackers tend to be severely resource-constrained (1-30 entries), which means they cannot reliably track all the aggressor rows and are bound to fail for some access patterns. Unfortunately, for existing in-DRAM trackers, it is difficult to a priori determine how often they will fail when subjected to the worst-case pattern. Unsurprisingly, all the current low-cost in-DRAM trackers have been broken with specific access patterns within a few minutes. While provably secure alternatives for in-DRAM tracking exist, they require thousands of tracking entries, making them unappealing for commercial adoption. The goal of our paper is to develop a low-cost in-DRAM tracker that is secure (guarantees a time-to-failure in the range of years) against all access patterns.**

We contend that the root cause of the vulnerability of current low-cost in-DRAM trackers stems from the use of activation-counters to direct policy decisions (e.g. which rows to insert, which to evict, and which to mitigate). Therefore, an attacker can perform frequent accesses to dummy rows to evade the mitigation of an aggressor row. The key insight of our paper is that to ensure security, the policy decisions of an in-DRAM tracker must not depend on the access pattern. To that end, we propose a secure and low-cost in-DRAM tracker called *PrIDE*, which consists of a FIFO buffer with probabilistic insertion. As the policy decisions of PrIDE do not depend on the access pattern, we develop a framework to calculate the time-to-failure. Our analysis with DDR5 shows that PrIDE (with 4 entries, 10-byte storage) can tolerate Rowhammer thresholds of 1.9K while guaranteeing per-bank time-to-failure of more than 10,000 years for all access patterns. We also co-design PrIDE with RFM to tolerate thresholds as low as 400 with only 1.6% slowdown. To the best of our knowledge, PrIDE is the first low-cost in-DRAM tracker to achieve provably secure Rowhammer mitigation.

I. Introduction

The increased inter-cell interference with DRAM scaling causes Rowhammer [20], whereby rapid activations of a DRAM row causes bit-flips in neighboring rows. Rowhammer is a major security threat that can result in system hijacking [1], [4], [6]–[8], [36], [41]. The number of activations required to induce bit-flips, called the *Rowhammer threshold (TRH)*, has dropped from 140K [20] to 4.8K [16] over the last decade. Therefore, we need effective mitigations.

To mitigate Rowhammer, DRAM manufacturers have designed DDR4 devices with an *in-DRAM* mitigation called *Targeted Row Refresh (TRR)*. TRR, and many subsequent solutions, rely on a *tracking* mechanism to identify the aggressor rows and then issue a *mitigative action* by refreshing the neighboring victim rows [9]. The tracker typically consists of a group of counters within each DRAM bank that counts row activations and performs mitigation transparently within the time period reserved for regular refresh operations.

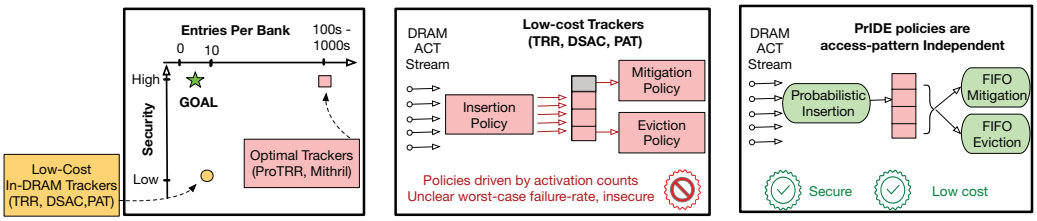
The optimal number of tracking entries required to ensure that all aggressor rows get mitigated is determined by both the maximum number of activations possible within the refresh interval and TRH. For example, for DDR5, we can perform 650K activations within the refresh interval of 32ms, so we can have up-to 200 aggressor rows that can be activated 3.25K times (per bank). Optimal in-DRAM trackers, such as ProTRR [27] and Mithril [17], have enough entries to track all the aggressor rows. Any tracker with less than the optimal number of entries will incur a non-zero failure rate [27].

Unfortunately, given the limited storage capabilities within the logic space of a DRAM module, in-DRAM trackers are typically limited to only a few entries. For example, TRR implementations use 1-30 counters per bank [9], [12]; the recently proposed in-DRAM tracker from Samsung, *DSAC* [10], uses 20 entries per bank, and the recently proposed in-DRAM tracker form SK Hynix, *PAT* [19], uses 8 tracking entries per bank. For these trackers, it is hard to a priori determine how often they will fail when subjected to worst-case patterns.

All TRR designs of DDR4 can be broken within a few minutes with TRRespass [6] and Blacksmith [12] patterns. DSAC is vulnerable to Blacksmith (Section VII-F). While the design details of PAT are not publicly available, the paper [19] mentions a 30% lower failure rate than *conventional designs*; however, if the conventional designs fail within a few minutes, PAT will still fail within a few minutes. Thus, all known low-cost in-DRAM trackers have been broken by some *well-crafted* attack patterns. Looking at how easily in-DRAM trackers get broken, it may seem futile to develop new low-cost in-DRAM trackers, as they would also inevitably be broken by some *well-crafted* patterns. In fact, the inability to securely mitigate Rowhammer with low-cost in-DRAM trackers has caused the DRAM industry to consider radical changes [3], [19], including per-row counters, which requires significant area overheads (9% [19]) and write operation after read, which incurs significant performance overheads. The goal of this paper is to develop a low-cost in-DRAM tracker that is secure, in that it guarantees time-to-fail in years for all access patterns. Figure 1 (a) shows an overview of current in-DRAM trackers.

*Gururaj performed part of this work while he was affiliated with NVIDIA.

979-8-3503-2658-1/24/$31.00 ©2024 IEEE
DOI 10.1109/ISCA59077.2024.00087



(a) In-DRAM trackers incur high storage or are non-secure

(b) Hard to ensure low worst-case failure rate across all patterns (not secure)

(c) PrIDE guarantees negligibly low failure rates across all patterns

Fig. 1. (a) Our goal is to develop secure low-cost trackers. (b) Typical low-cost in-DRAM trackers use counter values to determine policy decisions, which makes it hard to bound the worst-case failure rate due to dependence on frequencies in the access pattern (c) We propose PrIDE (comprising probabilistic insertion and FIFO queue), which makes policy decisions independent of access patterns, making it possible to establish the worst-case failure rate.

To analyze the failure rates of trackers, we first develop an abstraction for in-DRAM trackers to reason about them analytically. We can model an in-DRAM tracker as a multi-entry structure managed by three policies. First, an *insertion policy*, that determines if an accessed row enters the tracker. Second, an *eviction policy*, that determines which entry is evicted to store the incoming entry. Third, a *mitigation policy*, that determines which entry gets selected for mitigation. With three policies, there are three sources of failure. An aggressor row could encounter a *Tracker Insertion Failure (TIF)*, wherein the row is never inserted in the tracker within TRH activations. Similarly, an aggressor row could encounter *Tracker Retention Failure (TRF)* where the inserted entry gets evicted before mitigation. Finally, *Tardiness*, which indicates the extra activations a row can receive between insertion and mitigation, allowing the tracked row to cross TRH activations while still in the tracker. If we could estimate TIF, TRF, and Tardiness, we can determine the failure rate of the tracker.

Existing resource-constrained in-DRAM trackers (TRR, DSAC, PAT) try to emulate optimal trackers, in that they maintain activation counters in the tracking entries, and use these counter values to drive at-least one of the three policy decisions. Unfortunately, this also means that the corresponding failure metrics become dependent on the relative activation-counts of different rows within the access pattern (Figure 1(b)). This access-pattern dependent nature of failures means it is hard to a priori determine the worst-case pattern and how often the tracker will fail for such a pattern. This has allowed the attackers to craft specific access-patterns and break the tracker.

Insight: Rather than trying to emulate optimal trackers, low-cost in-DRAM trackers must be designed with policies that are independent of the access-patterns, and with the goal of guaranteeing negligibly low failure-rate across all patterns. Otherwise, these trackers will be broken by specific patterns.

To this end, this paper proposes *PrIDE (Probabilistic In-DRAM Tracker)*, a counterless in-DRAM tracker with a low bounded failure-rate across *any* pattern. PrIDE is carefully designed such that all three policy decisions are not influenced by the access pattern, which makes it possible to estimate the worst-case time-to-failure across all access patterns. Figure 1(c) shows an overview of PrIDE. PrIDE consists of an N-entry FIFO buffer, where the insertion policy is based on probabilistic sampling of the request stream. Sampling not only reduces the number of insertions in the buffer but also makes the insertion decision independent of the access pattern. The eviction policy is FIFO, which means an entry can get evicted (without mitigation) after N insertions into the FIFO buffer, and is independent of the addresses accessed. Finally, the mitigation policy is also FIFO, which means the maximum Tardiness is bounded by the size of the FIFO buffer and the rate of mitigation (e.g. one mitigation per refresh). We develop analytical models to estimate TIF and TRF, and a simple linear equation to bound Tardiness. Our analysis with DDR5 shows that PrIDE with a 4-entry tracker can handle TRH of 1.92K per row for a double-sided attack with a guaranteed per-bank time-to-fail of 10,000+ years across all patterns.

While PrIDE is an in-DRAM mitigation, we also compare PrIDE with prior probabilistic mitigation PARA [20], at the Memory Controller (MC). DDR5 supports MC-side mitigation with Directed Refresh Management (DRFM) [29], however, DRFM can be performed at most once per two refresh periods [29]. For a double-sided attack, PARA-DRFM can handle a threshold of 8.5K, compared to 1.92K with PrIDE.

The TRH tolerated by PrIDE depends on the mitigation rate (we assume 1 mitigation per tREFI). DDR5 introduces the RFM command, which enables the MC to allot time to the DRAM chips to perform additional mitigations. We co-design PrIDE with RFM to handle a threshold of 400 per row in double-sided attack, while incurring only 1.6% slowdown.

Overall, this paper makes the following contributions:

1) We contend that low-cost trackers must be designed with the goal of bounded failure rate across all patterns.
2) We observe that the root cause of vulnerability of low-cost in-DRAM trackers is their counter-driven policy decisions, which makes it hard to bound the failure rates.
3) We propose *PrIDE*, a simple probabilistic 4-entry tracker that guarantees low failure rates across all patterns. PrIDE tolerates a Rowhammer threshold of 1.9K.
4) We co-design PrIDE with RFM to tolerate even lower thresholds (400) with a slowdown of only 1.6%.

To our knowledge, PrIDE is the first design to achieve provably secure Rowhammer mitigation with low-cost in-DRAM trackers (requiring just 10-bytes SRAM per bank).

II. Background and Motivation

A. Threat Model

Our threat model assumes an attacker with native execution capability, that can issue memory requests for arbitrary addresses. We assume the attacker knows the defense algorithm, but does not have physical access to the systems (e.g., attacker cannot access the seed used for random-number generator). Our defense aims to prevent Rowhammer against all possible access patterns (including but not limited to TRRespass [6], Blacksmith [12], and Half-Double [21]). The recent RowPress [26] attack is out-of-scope, since its effects are orthogonal to Rowhammer and can be mitigated with a paging policy that limits the time a row is kept open.

B. DRAM Architecture and Parameters.

To access data from DRAM, the memory controller (MC) must first issue an activation (ACT) for the DRAM row. To ensure data retention of DRAM, the MC sends a refresh command every tREFI that refreshes a subset of rows. Table I shows the DRAM parameters. The two critical parameters for our study are: (1) the maximum number of activations within a tREFI interval (set at 79) and (2) the device performs one Rowhammer mitigation at each refresh event.

TABLE I
DRAM Parameters

Parameter	Explanation	Value
tREFW	Refresh Period	32 ms
tREFI	Time between successive REF Commands	3900 ns
tRFC	Execution Time for REF Command	350 ns
tRC	Time between successive ACTs to a bank	45 ns
ACTs-per-tREFI	(tREFI - tRFC) / tRC	79

C. DRAM Rowhammer Attacks

Rowhammer [20] occurs when a row (aggressor) is accessed 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 either 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 [20] to 4.8K (TRH-D) in 2020 [16].

TABLE II
Rowhammer Threshold Over Time

DRAM Generation	TRH-S (Single-Sided)	TRH-D (Double-Sided)
DDR3-old	139K [20]	–
DDR3-new	–	22.4K [16]
DDR4	–	10K [16] - 17.5K [16]
LPDDR4	–	4.8K [16] - 9K [21]

Rowhammer is not only a reliability issue [24] but also a serious security threat as an attacker can use it to cause privilege escalation [6], [7], [36], [47] or break confidentiality [22].

Solutions for mitigating Rowhammer typically rely on tracking the aggressor rows and providing a mitigating action (e.g. refresh the victim rows). The focus of our paper is tracking. Rowhammer mitigations are typically implemented either at the Memory Controller (MC) or within the DRAM chip.

D. Memory-Controller Side Mitigation

Typical Memory-Contoller (MC) based solutions identify aggressor rows either using counters [23] [30] [31] or probabilistically [15] [20]. Once the aggressor is identified, the MC sends mitigating refreshes to the rows adjacent to the aggressor rows. The key assumption in this class of solutions is that the MC is able to identify the neighbors of a given aggressor row. Unfortunately, DRAM chips internally use proprietary mappings, which makes it hard to identify the row adjacency information, making MC-based solutions harder to adopt.

More importantly, the mitigative refreshes within the DRAM chip are not visible to the MC, so the MC cannot accurately count activations, and such solutions become vulnerable to *Transitive Attacks* [40] such as Half-Double [21]. These attacks use undetected mitigative activations to silently cause failures to the neighbor of refreshed rows (we provide an overview in Figure 10). Finally, counter-based MC solutions typically require significant storage overhead (10s to 100s of KB), which means not all processor vendors are likely to adopt these solutions, leading to a fragmented landscape, where only the systems designed by specific vendors will have protection.

To enable MC-based solutions, the emerging DDR5 standard supports the Directed-RFM (DRFM) command, which allows the MC to send the address of the aggressor rows to the DRAM chip and request a mitigation. Unfortunately, DDR5 limits the rate of DRFM (once every two tREFI), which means MC-based solutions are limited to low rates of mitigation.

E. In-DRAM Mitigation

The advantage of in-DRAM mitigation is the potential to solve the DRAM Rowhammer problem within the DRAM chips, thus avoiding a fragmented landscape. Furthermore, DRAM manufacturers would have more information about their chips (e.g. TRH), so they can optimize their solutions.

In-DRAM mitigation typically relies on an in-DRAM tracker to identify the aggressor rows, and perform the mitigative refreshes transparently during the time window of refresh operations (DRAM is a deterministic device, so doing mitigative activations while servicing the normal demand accesses is not practical for commodity DRAM). For DDR5, the chips are required to support either one mitigation every tREFI or one mitigation every two tREFI [27]. In our paper, we assume a default rate of one mitigation per tREFI.

For guaranteed protection, the in-DRAM tracker must be able to identify *all* aggressor rows and mitigate them before they receive TRH activations. The optimal number of entries in the tracker is dictated by both the number of activations within tREFW (650K for DDR5) and the Rowhammer threshold. For example, we could have up-to 200 aggressor rows (per bank) that can each be activated 3.25K times, and the tracker must be able to identify all of the aggressor rows (in practice, a tracker does not know a priori, which rows will become aggressors, so it will need to track additional rows). We call the trackers that can reliably identify all aggressor rows, as *optimal* in-DRAM trackers. Examples of optimal in-DRAM trackers include ProTRR [27] and Mithril [17].

F. Low-Cost In-DRAM Mitigation

For current Rowhammer threshold, the optimal trackers require several hundred entries, which increases to thousands of entries as the TRH reduces below 1K. Each bank requires an independent tracker, so the total number of tracker entries to protect the entire DRAM rank (32 banks) would be in the range of 10s-100s of thousands of entries. Unfortunately, the storage capabilities within the logic space of a DRAM module are severely limited, so practical in-DRAM trackers typically have only a few entries (few tens) per bank. It can be proven that any tracker with less than the optimal number of entries will always incur a non-zero failure rate for some access patterns [27]. In fact, all low-cost trackers have been broken by well-crafted access patterns (within a few minutes). Examples of low-cost trackers include:

TRR (DDR4): DDR4 modules support in-DRAM mitigation called *Targeted Row Refresh (TRR)* [6] using a tracker of up to 30 entries [9], [12]. The exact implementation details are not publicly known and often vary from vendor to vendor. But TRR has been broken using *TRRespass* [6], which hammers a large number of rows exceeding the tracker capacity, causing tracked aggressors to be evicted, and the *Blacksmith* [12] attacks which exploit the deterministic insertion policy of TRR and hammer at specific instances to evade insertion.

DSAC (Samsung) [10]: A recent proposal from Samsung uses an in-DRAM tracker with 20 entries. It tries to approximate the optimal trackers with modified insertion (an accessed row replaces the min-counter entry with a probability inversely proportional to min-counter+1). DSAC can be easily broken with TRRespass and Blacksmith patterns (not evaluated in the original paper), which cause more than 9K activations to an aggressor row, even when the tracker is designed for a TRH of 500 (we provide the comparative results in Section VII-F).

PAT (SK Hynix) [19]: This recent proposal from SK Hynix uses an in-DRAM tracker with 8 entries (7 entries have a counter). The exact details of PAT are not publicly available. The paper reports that PAT has 30% lower failure rate than conventional designs. However, given that conventional low-cost trackers can typically be broken within a few minutes, it would mean that PAT can also be broken within a few minutes.

Given that the above designs were broken easily, it may seem futile to develop new low-cost in-DRAM trackers, as they can also be inevitably broken by some *well-crafted* access pattern. However, we contend that the root-case of these trackers getting broken is because they are usually designed to mimic optimal trackers with much fewer counters, and without a clear understanding of how frequently the tracker will fail when subjected to the tracker-specific worst-case access pattern. Ideally, for a low-cost tracker, we seek a design that can bound the worst-case failure rate across *any* access pattern. To that end, we develop a framework to analyze the failure modes of low-cost in-DRAM trackers and use that to design a *secure* low-cost tracker that guarantees a negligible failure rate (one failure per 10,000 years) across all patterns.

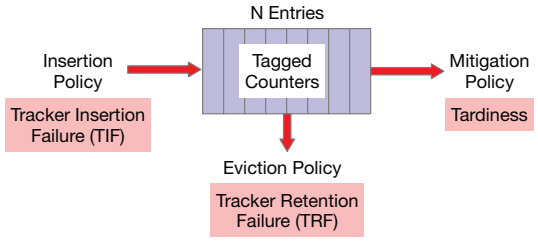


Fig. 2. In-DRAM tracker has N entries, modulated by three policies: insertion, eviction, and mitigation. Three policies cause three types of failures.

G. How and Why Low-Cost In-DRAM Trackers Fail?

Figure 2 shows an overview of a canonical low-cost tracker. It consists of an N-entry structure (each entry contains row address and some metadata such as counters). The structure is managed by three policies which determine the three different types of failure that can occur in the tracker. First, an *insertion policy* that determines if an aggressor row enters the tracker before reaching TRH; if not we would deem that event as a *Tracker Insertion Failure (TIF)*. Second, an *eviction policy* that determines which tracker entry is evicted to store the incoming entry. The episode of losing a tracking entry before mitigation is called as the *Tracker Retention Failure (TRF)*. Third, a *mitigation policy* that determines which entry gets selected for mitigation at the time of a given refresh operation. This policy determines *Tardiness*, which indicates the extra activations a row can receive between insertion and mitigation. Tardiness can cause a row to reach TRH while still in the tracker waiting for mitigation. If we could determine the TIF, TRF, and Tardiness across all patterns, then we could bound the worst-case failure-rate of the tracker across all patterns.

Unfortunately, it is difficult to understand and bound the failure rates of existing low-cost trackers. These trackers are typically designed to emulate optimal trackers – for example, optimal trackers have counters and make counter-dependent policy decisions (when to insert, what to evict, what to mitigate). Similarly, the low-cost trackers also use counter values to drive the policy decisions (e.g. DSAC uses min-counter value to drive insertion, min-counter entry for eviction, and max-counter entry for mitigation). An attacker can influence the policies by manipulating the access counts of dummy rows. With counter-based decisions, the corresponding failure metrics (TIF, TRF, and Tardiness) become dependent on the relative access frequencies of different rows. This access-pattern dependent nature of failure-rate is what makes current low-cost in-DRAM trackers insecure.

H. Goal of our Paper

Our key insight is that low-cost trackers can be made secure with policy decisions that are access-pattern independent (not influenced by which address is accessed), thus making it possible to determine the time-to-fail in the worst-case. The goal of this paper is to develop a low-cost secure in-DRAM tracker that guarantees a negligibly small failure rate (time-to-fail in the ranges of years) across all access patterns.

III. Methodology For Analyzing Failures

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 would declare it as a failure. We are interested in estimating the *Time-to-Fail* considering an attack with the worst-case access pattern.

A. Attack Round and Round Failure Probability

An *Attack Round* is a sequence of (up to TRH) accesses to a row (other rows may be accessed in between, see Figure 3). An attack round for row-A always begins with a mitigation to row-A and ends with one of two possibilities: (1) a failure (no mitigation), or (2) a mitigation, which instantly ends the round. Our definition of Attack Round can support arbitrary patterns (see Appendix-B). Let *Round Failure Probability* (P_{RF}) be the probability that the attack round results in a failure.

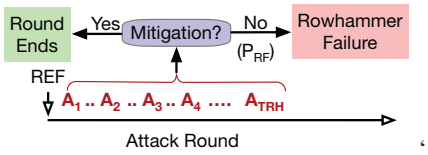


Fig. 3. An Attack Round consisting of TRH activations on row A. The round ends either with mitigation or with Rowhammer failure (probability P_{RF}).

B. Computing the Time-to-Fail for Bank and System

As each mitigation starts a new attack round, it is beneficial for the attacker to attempt as many attack rounds as the number of mitigations in a given time. With 1 mitigation per tREFI, the expected time-to-fail for a bank is given by Equation 1. The time-to-fail for a system with B banks is B times lower.

$$\text{Time-to-Fail (Bank)} = \frac{1}{P_{RF}} \cdot tREFI \qquad (1)$$

C. Defining Probabilistic Security with Target Time-to-Failure

We deem our probabilistic design to be secure *only* if the time-to-fail exceeds a defined *Target Time-to-Fail (Target-TTF)*. We use Target-TTF of 10,000 years per bank (P_{RF} is 1.24×10^{-17}), which ensures that our bank-FIT rate is similar to the bank-FIT rate from naturally occurring errors [2]. Section VII-B analyzes Target-TTF sensitivity.

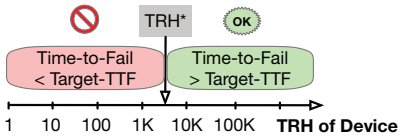


Fig. 4. TRH* is the minimum TRH that meets the Target Time-to-Fail. We use TRH* as the key metric to compare designs (lower TRH* is better).

D. Critical Rowhammer Threshold: Key Figure-of-Merit

The time-to-fail depends on the TRH. We define the *Critical Rowhammer Threshold (***TRH***)* as the *lowest TRH* for which the design meets the TargetTTF, as shown in Figure 4. We use TRH* as the key figure-of-merit to compare designs (lower is better). We derive TRH* for a single-sided pattern and show in Section VI that TRH* is 2x lower for double-side pattern.

IV. PrIDE: Probabilistic In-DRAM Tracker

This paper proposes *PrIDE (Probabilistic In-DRAM Tracker)*, a low-cost secure tracker. PrIDE is specifically designed such that we can determine the worst-case time-to-fail across all patterns. We first provide an overview of PrIDE, then describe how to analyze the impact of the three failure metrics (tracker insertion failure, tracker retention failure, and tardiness), and finally, discuss modifications to ensure that the tracker is tolerant to transitive attacks. We also compare PrIDE with other probabilistic policies.

A. Overview of PrIDE

Figure 5 shows an overview of PrIDE. PrIDE consists of an N-entry FIFO buffer regulated by a probabilistic insertion policy. The eviction policy and the mitigation policy are both FIFO (selecting the oldest entry). The FIFO is implemented as a circular buffer: a pointer *(PTR)* points to the oldest entry in the buffer and a register *(Occ)* keeps track of the number of valid entries in the buffer. Each entry of the buffer contains the row address. Note that the tracking entries do not contain any counters for counting activations.

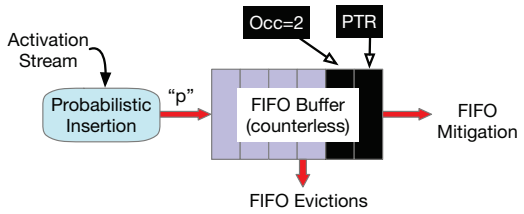


Fig. 5. Overview of PrIDE. It contains an N-entry FIFO buffer managed by probabilistic insertion policy. The eviction and mitigation policy are independent of access patterns to make it possible to bound the failure rate.

On a DRAM activation, the insertion policy of PrIDE samples it for insertion with an *insertion probability (p)*, using a pseudo-random-number generator (PRNG). Thus each activation enters the buffer with probability 'p'. On each insertion, the incoming address is inserted at a distance of *Occ* away from the *PTR*, and *Occ* is incremented. If the buffer is full, the oldest entry (pointed by *PTR*) is evicted, *Occ* is decremented and *PTR* is incremented, making room for the incoming entry. Once every tREFI, a mitigation is performed if *Occ* is non-zero: the oldest resident entry pointed by *PTR* is mitigated, the entry invalidated, *PTR* is incremented, and *Occ* is decremented. We assume that mitigation is done using a victim refresh of all neighboring rows in its blast radius.

The policies of PrIDE are designed such that their decisions are access-pattern independent (i.e. the decisions are not influenced by which address is accessed). For example, regardless of the access pattern, each activation has the same probability 'p' of being inserted into the tracker. Once an address enters the FIFO buffer, it can get evicted (assuming no mitigation) after N insertions into the buffer, regardless of which addresses get accessed. Similarly, if an address enters the buffer when the occupancy was *K*, it will be mitigated after *K* mitigations, (assuming no evictions), and is not influenced by addresses accessed. We analyze the failure modes of PrIDE next.

B. Analyzing the Impact of Insertion Policy

Our insertion policy must ensure that an attacker cannot manipulate the access pattern to reduce the likelihood of an aggressor row entering the buffer.

Key Requirements: For security, the probability of insertion *must* not be influenced by the state of the buffer. In particular:

R1. The insertion decision must be agnostic to the existence of invalid entries in the buffer (so, even if there is any invalid entry, we must still insert only with probability 'p').

R2. The insertion decision must be agnostic to the existence of a matching entry in the buffer (so, in such a case, we still insert a duplicate entry in the buffer with probability 'p').

Violating these two seemingly innocuous requirements can result in a significant increase in the overall failure rate of PrIDE. For example, always inserting if there is any invalid entry increases the average occupancy and chances of the tracker being thrashed which significantly increases tracker retention failures (TRF). Similarly, not installing a matching entry leads to higher TRF, as the existing entry may get evicted.

Estimating TIF: As the attack round can reach up to TRH activations, the probability of tracker insertion failure (TIF) for an attack round is given by Equation 2:

$$TIF = (1-p)^{TRH} \tag{2}$$

We need to carefully set the value of p. If p is too low then the given aggressor row may not get selected before reaching TRH, thus increasing insertion failures (TIF). If p is too high, it increases the retention failures (TRF). If the rate of mitigation is once per W activations, then, selecting $p = 1/W$ ensures that the insertion rate matches with the mitigation rate. As our DRAM configuration can perform up to 79 activations within tREFI, for the default mitigation rate of 1 per tREFI, we select a default insertion probability of $p = 1/79$.

Estimating TRH due to TIF: If the failures due to TRF and Tardiness were zero (i.e. we could instantly mitigate the selected row), the overall failures would solely be determined by TIF. Thus, round failure probability (P_{RF}) would equal TIF, and we can use P_{RF} to compute TRH^*_{TIF}, the TRH due to TIF which can be obtained using Equation 1 and Equation 2.

$$\text{MTTF} = \frac{\text{tREFI}}{(1-p)^{TRH}} \implies TRH = ln(^{\text{tREFI}}/_{\text{MTTF}})/ln(1-p) \tag{3}$$

In our case, MTTF=10,000 years, and tREFI is 3.9 microseconds, therefore, Equation 3 simplifies to :

$$TRH^*_{TIF} = \frac{-38.93}{ln(1-p)} \tag{4}$$

As we use $p = 1/79$, the TRH tolerated by such an idealized design, limited only by insertion failures, is given by $\boldsymbol{TRH^*_{TIF}}$ **= 3.06K**. In reality, PrIDE will have a higher TRH* due to failures from TRF and Tardiness.

C. Analyzing the Impact of Eviction Policy

When a row is inserted in the N-entry FIFO buffer with an occupancy of K, it will receive a mitigation at (K+1)th mitigation event. However, in the interim, the entry can get evicted without any mitigation, if there are N insertions before the given entry receives a mitigation. The failure due to such evictions is captured with *Tracker Retention Failure (TRF).*

The Lossy-Buffer Model: We can view the storage structure of PrIDE as a lossy buffer, where an inserted entry has a *Loss Probability* (L) of getting lost (evicted without mitigation) from the buffer. Figure 6 provides an overview of the tracker with such a lossy buffer model.

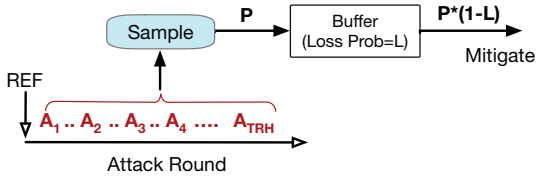


Fig. 6. Modeling the Insertion and Eviction Failures in PrIDE.

Any given reference has a probability p of getting sampled and inserted into the buffer. Assuming the buffer provides an *independent-and-identically distributed (IID)* loss-probability of L for each insertion, then the probability that the insertion survives in the buffer would be (1-L). Thus, the probability that a given address gets inserted and mitigated becomes $\hat{p} = p \cdot (1-L)$. Therefore, the failure rate of this system with a lossy buffer would be similar to a system that has only TIF failures, but where the insertion probability is revised from p to $\hat{p}$. We can use this in Equation 3 to determine the TRH* of a system that has only TIF and TRF failures, which we will denote as $TRH^*_{(TIF+TRF)}$.

$$TRH^*_{(TIF+TRF)} = ln(^{\text{tREFI}}/_{\text{MTTF}})/ln(1-p\cdot(1-L)) \tag{5}$$

In our case, MTTF=10,000 years, and tREFI is 3.9 microseconds, therefore, Equation 5 simplifies to:

$$TRH^*_{(TIF+TRF)} = \frac{-38.93}{ln(1-p\cdot(1-L))} \tag{6}$$

This analysis relies on a critical assumption that the loss probability (L) is IID for all insertions. While this assumption is generally not true (e.g. addresses accessed earlier in the tREFI window are more likely to get lost), we can still perform the analysis, if we could determine and use the worst-case loss probability as our estimate of L for all insertions. By design, using the worst-case L for all references will necessarily give a pessimistic TRH* (the actual value of TRH* will be lower than our estimates), however, we use this assumption as it greatly simplifies the analysis and provides access-pattern-independence for our estimates. We discuss how to determine the worst-case estimates for L next.

Estimating Worst-Case Loss Probability: As insertions are independent events, the loss probability is not dependent on which addresses are accessed before or after the insertion of a given entry. Furthermore, as we insert a duplicate entry even if the given address is already present in the buffer, the loss probability is not dependent on which addresses are present in the buffer at the time of insertion. Thus, the loss probability has *address independence*. However, the loss probability can still be influenced by the *position* of the attacked address within the tREFI window (we call this *positional dependence*). To achieve positional *independence*, we estimate the worst-case loss probability across all positions both analytically and empirically (using Monte-Carlo)[1]. Both approaches provide similar results. Below we describe the analysis first for a single-entry buffer, and then for a multi-entry buffer.

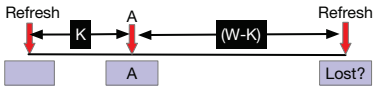


Fig. 7. Timeline for evictions from a single-entry tracker

Loss Probability for Single-Entry Tracker: Let there be W activations in a window between two refreshes (tREFI), as shown in Figure 7. The window starts with an empty buffer (mitigation at refresh). Lets assume Row-A is inserted into the buffer at the Kth activation. The buffer will lose this entry if there is any other insertion in the remaining (W-K) activations. If p (1/W) is the insertion probability, then the loss probability (L_K) for an insertion at Kth position is given by Equation 7.

$$L_K = 1 - (1 - \frac{1}{W})^{(W-K)} \tag{7}$$

Figure 8 shows the loss probability (L_K) as the position (K) of the attacked line is varied from 1 (earliest in the tREFI window) to 79 (last position in the tREFI window). The first position has the highest loss-probability (0.63) whereas the last position has zero loss-probability (nothing else left in the window to dislodge the entry from the tracker).

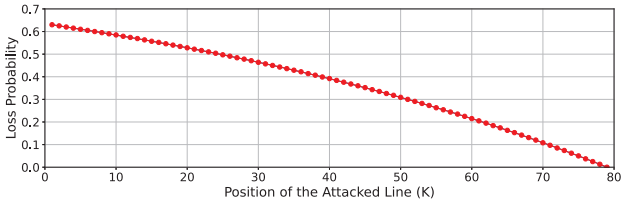


Fig. 8. The loss probability of a single-entry tracker when attacking various positions in tREFI window. Loss probability peaks at K=1, is zero at K=79.

[1]For Monte-Carlo simulations, we run 100 million tREFI periods, each with W (79) activations and corresponding insertion probability (p=1/W). For each of the W positions in the tREFI window, we keep track of the number of insertions and the number of times the entry got evicted without mitigation to compute the per-position loss probability. As placing the aggressor row at the first activation within the tREFI interval provides the highest loss probability (more time to get lost), so we report the loss probability for this worst position.

To determine an upper-bound on the loss-probability of our tracker, we make the following two pessimistic assumptions:

1) *Pessimistic self-eviction*: Even if the tracker entry is evicted by the insertion of the same address, we still count this eviction as a *loss* (even though in practice the tracker still gets another copy of the lost address).
2) *Pessimistic position for all*: We assume that every insertion in the tracker will suffer the loss-probability of the worst-case position (even though in practice some entries will incur much lower loss probability).

Therefore, we can guarantee that the loss probability of a single entry tracker can *never* be more than 0.63 (loss probability of the first position). By design, there can be no pattern that will have a higher loss probability than our estimates (see Appendix-C for validation with attack patterns). We use our estimate of L=0.63 in Equation 6 to obtain the $TRH^*_{(TIF+TRF)}$ of the single-entry tracker as 8.3K.

Loss Probability for Multi-Entry Tracker: For a multi-entry tracker, the loss probability is also influenced by the occupancy of the tracker at the beginning of a tREFI (e.g. inserting into an empty buffer has a lower loss probability compared to inserting into a full buffer). An N-entry tracker can start a tREFI window in N possible states (0 to N-1 entries occupied, as at least one entry is always invalid due to mitigation at refresh).

We split the loss probability analysis into two parts: first, we use a model similar to the single-entry tracker to obtain the worst-case loss probability (L_x) when the buffer begins the tREFI window with x entries occupied (x is varied from 0 to N-1). Second, we use a Markov-Chain model to determine the probabilities (P_x) that the buffer starts a tREFI window with x entries valid (at each tREFI the occupancy decreases by 1 if occupancy was greater than zero, and each of the W activations increases the occupancy by probability p, limited by the buffer size). The effective loss probability for each x is $P_x \cdot L_x$. The overall loss probability (L) is simply the summation of the effective loss probability over all the N cases (x varied from 0 to N-1), *i.e.*, $L = \sum_{x=0}^{N-1} P_x \cdot L_x$ (See Appendix-A)

Results for Loss Probability and TRH: Table III shows our worst-case loss-probability (L), estimated using our analytical model, as the buffer size is varied. We use Equation 6 to compute $TRH^*_{(TIF+TRF)}$.

For a 16-entry buffer, PrIDE gets $\boldsymbol{TRH^*_{(TIF+TRF)}}$ **= 3.15K**, if the failures were limited to only insertion and eviction. However, in reality, there are failures also due to Tardiness. We analyze this in the next section.

TABLE III
THE LOSS PROBABILITY AND $TRH^*_{(TIF+TRF)}$

Buffer Size	Loss Prob (L)	$TRH^*_{(TIF+TRF)}$
1	0.630	8.29K
2	0.305	4.40K
4	0.119	3.47K
8	0.060	3.25K
16	0.030	3.15K

D. Bounding Maximum Tardiness

When a row is inserted in the buffer, it can continue to receive more activations between insertion and mitigation, a metric we call *Tardiness*. If there are W activations within tREFI, and the row is inserted at the end of an N-entry buffer in the worst-case, with one mitigation per tREFI, the maximum activations between insertion and mitigation would be $N \cdot W$. For an open-row policy, there must be an intervening access to another row to cause multiple activations to the same target row. Thus, the total number of activations for a given aggressor row can be at-most half the number of activations in the access pattern (i.e., $W/2$). However, we assume a *closed-page policy* (which represents the best-case for an attacker), therefore, the TRH* tolerated by PrIDE when Tardiness is included in Equation 6, is shown in Equation 8.

$$TRH^* = \frac{-38.93}{ln(1 - p \cdot (1 - L))} + N \cdot W \tag{8}$$

Figure 9 shows the TRH* (with and without Tardiness) as the buffer size is varied. The TRH* reduces until buffer-size of 5 (due to reduced loss probability) and then starts increasing (due to the increase in Tardiness). The TRH* at buffer-size of 4 is 3.79K, 5 is 3.78K, and 16 is 4.42K. Thus, *larger buffers are not always beneficial*. Our default configuration uses PrIDE with 4 entries, and provides the **TRH* of 3.79K**.

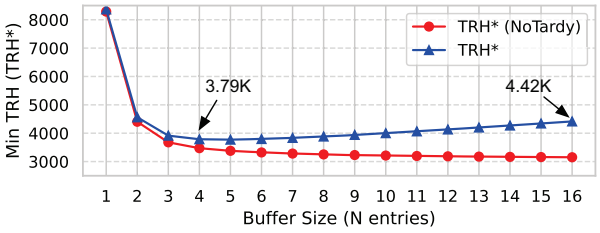


Fig. 9. TRH* and TRH* (ignoring Tardiness) as the buffer-size is varied. The lowest TRH* for PrIDE occurs around a buffer size of 4 (and not at 16).

E. Tolerating Transitive Attacks

The security of PrIDE stems from the fact that all activations are equally likely to get mitigated, given that they will be selected by the insertion policy with probability 'p'. However, the scheme described thus far is vulnerable to *Transitive Attacks* [21], [40], where the attacker uses the mitigating activations themselves to cause an attack, as shown in Figure 10.

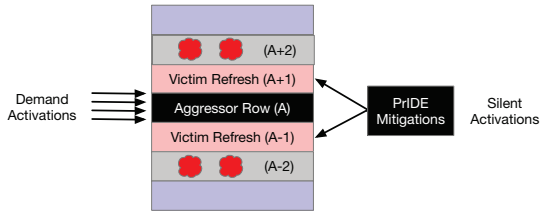


Fig. 10. Transitive Attacks [40] exploit silent activations due to mitigations to cause failure in neighbors of the rows undergoing frequent victim refresh.

For example, for DDR5, we can perform approximately 650K activations within the refresh period. If we perform an ABABAB pattern on the bank, then we can access the two rows up to 325K times. PrIDE (with p=1/79) will mitigate both rows on average 4.1K times. However, these mitigations can *silently* (without getting sampled) incur 4.1K activations, leading to failures at rows that are at a distance of 2 from the aggressor. Thus, TRH* will increase by an amount equal to the silent activations. Note that simply refreshing two rows on either side of an aggressor does not mitigate transitive attacks, as now the third row on either side will experience failures.

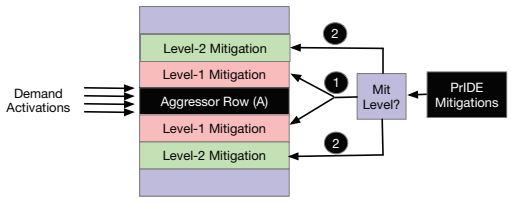


Fig. 11. Modifying PrIDE to Defend Against Transitive Attacks.

We extend PrIDE to be secure against transitive attacks by probabilistically sampling the mitigative actions and progressively changing the distance of mitigation. Figure 11 shows our implementation. We define a multi-level mitigation[2] as follows: an entry inserted due to a demand activation receives a mitigation of level 1, i.e., only refreshing immediate neighbors (say, A+1 and A-1). A mitigation level of 2 means neighbors at distance 2 are refreshed for a given row A (say A+2 and A-2). In general, a mitigation level m refreshes m-th neighbors of a given row A (A+m and A-m). We extend each tracker entry to contain a 3-bit mitigation level. Whenever, a mitigation with level m is performed, the row address is reinserted into the buffer with probability p (same insertion rate as regular activations), with the mitigation-level incremented $(m + 1)$.

This design provides a principled mechanism to provide mitigations for the mitigating activations (and instead of inserting two entries, one on each side, we are simply inserting one entry, as the aggressor row A does not need to be refreshed). As the number of items that can enter the buffer within the mitigation window are not W, but W+1, the insertion probability is reduced to p=1/(W+1). PrIDE (with 4-entries) with transitive attack protection provides a **TRH* of 3.83K**.

F. Tying it All Together: Low-Cost and Secure PrIDE

Our default design of PrIDE uses a 4-entry buffer with p=1/80. Each entry contains a row-address and a mitigation-level. On each demand activation, the row is inserted with probability 'p'. If the buffer is full, oldest entry is evicted. At tREFI, we mitigate the oldest entry using the given mitigation level, and remove it from the buffer (reinsert it with probability 'p' with an incremented mitigation level). This design can tolerate a Rowhammer Threshold of 3.83K, while guaranteeing a time-to-fail of 10,000+ years (per bank). Thus, PrIDE enables low-cost and secure in-DRAM mitigation.

[2]We describe a *Blast Radius (R) of 1 for simplicity. This design works for larger R also. For example, level-1 mitigation will refresh the adjacent +/- R rows, level-2 mitigation will refresh +/- R rows beyond that, and so on.*

G. Comparison with PARA

PARA [20] is a memory controller (MC) based defense, which on a demand activation, issues a refresh to the adjacent rows with a probability 'p'. However, DRAM chips may use proprietary mappings, so the MC may not know the addresses of adjacent rows to a given aggressor row. To support such MC-based mitigation, DDR5 introduces the DRFM command, whereby the MC can ask the DRAM chip to issue mitigation for a given aggressor row address. However, DRFM rate is limited (1 per two tREFI in DDR5) [29]. Furthermore, as the mitigative activations performed within the DRAM chip are not visible to the MC, PARA is vulnerable to transitive attacks.

We modify PARA for DRFM, called PARA-DRFM, such that PARA selects a row with probability p (we use p=1/160) if the DRFM limit (1 per two tREFI) is not reached. We also evaluate an enhanced scheme PARA-DRFM+ where the DRFM rate is 1 per tREFI (p=1/80). For the purpose of our analysis we assume that DRFM internally provides protection against transitive attacks [29]. Table IV shows the effective threshold (TRH*) for PARA-DRFM and PARA-DRFM+ and compares it with PrIDE. PARA-DRFM has a TRH* of 17K and PARA-DRFM+ has a TRH* of 8.4K, whereas, PrIDE provides a much lower TRH* of about 3.8K.

TABLE IV
THE TRH* OF PARA AND PRIDE

Scheme	Type of Defense	TRH*
PARA-DRFM	MC	17K
PARA-DRFM+	MC	8.4K
PrIDE	In-DRAM	**3.8K**

V. SCALING PRIDE TO LOWER THRESHOLDS WITH RFM

To reduce the TRH tolerated by PrIDE we need to increase the mitigation rate. DDR5 specifications include *Refresh Management (RFM)* that enables the memory controller to provide more time to the DRAM chip to perform mitigations. We briefly describe RFM and then propose PrIDE+RFM.

A. Refresh Management (RFM)

RFM enables the memory controller to issue additional mitigations (RFM commands) to the DRAM when the activations per bank crosses a threshold. However, the actual row to be mitigated still depends on the in-DRAM tracker. The memory controller maintains a *Rolling Accumulation of ACTs (RAA)* counter [17]) per bank. When any RAA counter crosses a threshold, RFM_{TH}, the memory controller resets the RAA counter and issues an RFM to the corresponding DRAM bank.

B. PrIDE+RFM: Co-designing PrIDE with RFM

We co-design PrIDE with RFM to scale to lower thresholds. Figure 12 describes our proposed combination, *PrIDE+RFM*. We evaluate two variations, each with a threshold of 16 or 40 activations (PrIDE+RFM16 and PrIDE+RFM40, respectively). We revise the insertion probability 'p' of PrIDE+RFM40 to 1/41 and PrIDE+RFM16 to 1/17. We note that the FIFO buffer of PrIDE remains unmodified.

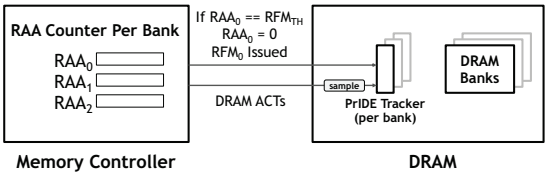


Fig. 12. An Overview of PrIDE+RFM

Table V shows the TRH* for PrIDE and PrIDE+RFM. We evaluate various mitigation rates, including 0.5x (one mitigation every 2 tREFI), 1x (one mitigation every tREFI), RFM40 (2x mitigation rate), and RFM16 (approximately 5x mitigation rate). PrIDE+RFM16 provides a **TRH* of 823** .

TABLE V
THE TRH* OF PRIDE AND PRIDE+RFM

Scheme	Relative Mitigation Rate	TRH*
PrIDE	0.5x (one per two tREFI)	7.52K
PrIDE	1x (one per tREFI)	3.83K
PrIDE+RFM40	2x (approx two per tREFI)	1.98K
PrIDE+RFM16	5x (approx five per tREFI)	**823**

C. Comparison with PARFM

We also evaluate a design that combines PARA with RFM, called PARFM [17]. PARFM buffers all the row addresses since the last mitigation, and at mitigation (at tRFC or RFM), PARFM randomly selects one of the buffered entries for mitigation, after which it invalidates all the entries (to free up the buffer for the requests in the next epoch). PARFM needs a large buffer (166 entries for DDR4 and 79 entries for DDR5) compared with only 4 entries for PrIDE+RFM. Furthermore, the PARFM design is vulnerable to transitive attacks, which *worsens* with lower RFM thresholds (more silent activations). PARFM has a TRH* of 7.1K, compared to 3.83K for PrIDE.

VI. THE IMPACT OF VICTIM-SHARING ATTACKS

The security of PrIDE stems from the guarantee that the likelihood that a row will encounter TRH* activations without getting mitigated is negligibly small (once per 10,000 years). An attacker could also try to launch an attack pattern that shares a victim row, (*e.g.*, a *double-sided* attack) to try to cause more hammers on the shared victim row before a mitigation. Figure 13 shows the victim-sharing attacks for a system designed for a Rowhammer Threshold of T. For Blast-Radius (BR) of 1, the victim-row C is shared between aggressor rows B and D. For BR=2, the victim-row C is shared between four aggressor rows A, B, D, and E.

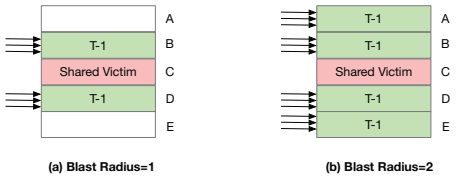


Fig. 13. Victim sharing attacks for Blast Radius of 1 and 2.

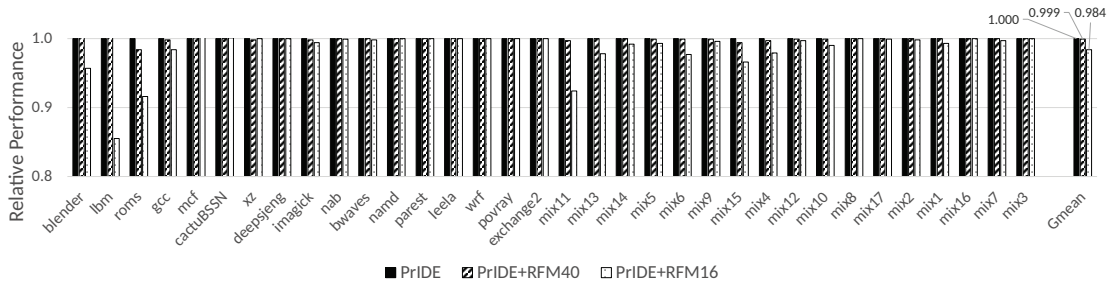


Fig. 14. Normalized performance. PrIDE incurs zero slowdown. PrIDE+RFM40 and PrIDE+RFM16 incur 0.1% and 1.6% slowdown, respectively.

If we had a counter-based mitigation [17], [23], [27], [30], [31], [37] that issues a mitigation when counter reaches T, such victim sharing attacks can subject the victim row to 2 * (T - 1) or up-to 4 * (T - 1) hammers without triggering a refresh of the victim, thereby reducing the effective threshold handled by the counter-based mitigation schemes. Thus, while victim-sharing attacks are effective with counter-based mitigations, we demonstrate that such attacks are ineffective against PrIDE.

We observe that PrIDE is probabilistic and causes a refresh of the shared victim-row if any of the aggressors within the BR gets selected for mitigation (as mitigations refresh BR rows on either side of the aggressor row). For example, for BR=1, an activation of Row-B or Row-D can trigger a refresh of the victim Row-C. Therefore, the total number of chances the victim-row has of getting refreshed is the sum of the activation counts across all the aggressor rows within the BR.

We can view TRH* of PrIDE as the consecutive number of chances of mitigation that remain unmitigated. Thus, with PrIDE, for BR=1 attacks, the average number of activations on the two aggressor rows can be at most half of TRH* while escaping mitigation (for BR=2 attack, the four aggressor rows can have an average activation counts of only a quarter of TRH* while escaping mitigation). Thus, PrIDE is robust against victim-sharing attacks as mitigation is proportional to the total number of hammers suffered by the victim.

We can use this analysis to also determine the Rowhammer Threshold (TRH-D*) tolerated by PrIDE for the double-sided rowhammer pattern (pattern is identical to BR=1 victim-sharing attack). In the double-sided pattern, each aggressor row is accessed TRH-D times, thus the victim row has 2*TRH-D chances of getting mitigated. For example, if TRH-D was 3.8K, PrIDE would get 7.6K chances to refresh the victim row. Thus, the TRH* of PrIDE can be halved for a double-sided pattern, while still retaining the same target failure rate. Table VI shows the single-sided (TRH-S*) and double-side (TRH-D*) thresholds tolerated by PrIDE and PrIDE+RFM. PrIDE can tolerate (the typically reported TRH) **TRH-D* of 1.9K**, and PrIDE+RFM16 can tolerate **TRH-D* of 412**.

TABLE VI
THE TRH-S* AND TRH-D* OF PARA-DRFM, PRIDE AND PRIDE+RFM

Scheme	TRH-S* (Single-Sided)	TRH-D* (Double-Sided)
PARA-DRFM	17K	8.5K
PrIDE	3.83K	1.92K
PrIDE+RFM40	1.98K	992
PrIDE+RFM16	823	**412**

VII. RESULTS

A. Impact on Performance

We model PrIDE and PrIDE+RFM using the Gem5 [25] simulator. Table VII shows our configuration. We evaluate our design with 17 SPEC2017 [39] *rate* workloads and 17 mixed workloads. We skip 25 billion instructions and simulate 250 million instructions. For RFM, we assume that the bank is unavailable for 180ns (enough to refresh 2 rows on each side).

TABLE VII
BASELINE SYSTEM CONFIGURATION

Out-of-Order Cores	4 core, 3GHz, 8-wide fetch, 192 entry ROB
Last Level Cache (Shared)	4MB, 16-Way, 64B lines
Memory specs	32 GB, DDR5
t_{RCD}-t_{CL}-t_{RP}-t_{RC}	14.2-14.2-14.2-45 ns
Banks x Ranks x Channels	32×1×1
Rows	128K rows, 8KB row buffer

Figure 14 shows the relative performance of PrIDE and PrIDE+RFM, normalized to the DDR5 baseline. PrIDE incurs zero slowdown, as the mitigations required for PrIDE are performed within the tRFC period specified by DDR5 specifications. PrIDE+RFM40 incurs negligible slowdown (0.1%) and PrIDE+RFM16 incurs an average slowdown of 1.6%. Thus, PrIDE and PrIDE+RFM provide a scalable and low overhead mitigation for Rowhammer, even at low thresholds.

B. Impact of Target Time-to-Fail (TTF) on Threshold

As PrIDE provides probabilistic security, we deem the design 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. Table VIII shows TRH* of PrIDE for varying Target-TTF (per-bank) and for our system (64 banks but only 22 can be used concurrently due to tFAW constraints). We note that increasing the default Target-TTF reduces the required round-failure probability, which increases the TRH*. Hence, we observe that TRH* increases with Target-TTF.

TABLE VIII
THE TRH-S* AND TRH-D* OF PRIDE FOR VARIOUS TARGET-TTF

Target-TTF (Bank)	MTTF (System)	TRH-S*	TRH-D*
100 years	4.5 years	3.42K	1.71K
1K years	45 years	3.63K	1.81K
10K years	**454 years**	**3.83K**	**1.92K**
100K years	4.5K years	4.04K	2.02K
1 Million years	45K years	4.25K	2.12K

C. Impact of Device Threshold on Time-to-Fail

Thus far, we have computed the TRH* for PrIDE and PrIDE+RFM assuming that the system will be made of modules with TRH greater than TRH*. We now analyze the effectiveness of our solutions as the device TRH is varied. Given that modern devices are characterized with TRH-D and the most recent (publicly available) characterization study observed at TRH-D of 4.8K, we are interested in the failure of the system when devices with varying TRH-D up to the current TRH-D are used in the system. Without loss of generality, for our analysis, we assume our system with 64 banks (only 22 can be used concurrently due to tFAW), and compute the expected time to failure for the server, assuming that all banks are concurrently and continuously attacked for the entire duration.

TABLE IX
AVERAGE TIME TO SYSTEM FAILURE FOR PRIDE AND PRIDE+RFM ON DEVICES WITH GIVEN TRH (MLN DENOTES MILLION)

Device TRH-D	PrIDE	PrIDE+RFM40	PrIDE+RFM16
4800 (now)	> 1 Mln years	> 1 Mln years	> 1 Mln years
2000	2936 years	> 1 Mln years	> 1 Mln years
1800	**36 years**	> 1 Mln years	> 1 Mln years
1600	153 days	> 1 Mln years	> 1 Mln years
1400	2 days	> 1 Mln years	> 1 Mln years
1200	32 mins	> 1 Mln years	> 1 Mln years
1000	23 sec	**674 years**	> 1 Mln years
800	< 1 sec	42 days	> 1 Mln years
600	< 1 sec	10 min	> 1 Mln years
400	< 1 sec	< 1 sec	**140 years**
200	< 1 sec	< 1 sec	3 sec

Table IX shows the expected time-to-fail (TTF) for a system with PrIDE, PrIDE+RFM40, and PrIDE+RFM16. For current thresholds (TRH-D of 4800) all three designs provide a time-to-fail exceeding 1 million years. For a TRH-D of 2000, standalone PrIDE can provide a time-to-fail of 2.9K years, thus the co-design with RFM may not be required. However, at lower thresholds, standalone PrIDE can cause frequent failures (almost once every 20 secs with TRH-D of 1000). For such devices, we suggest PrIDE+RFM. With PrIDE+RFM40, with devices of TRH-D of up to 1000, we get a time-to-fail exceeding 670 years. With PrIDE+RFM16, we get a time-to-fail exceeding 100 years for devices up to TRH-D of 400.

D. Storage Overheads

PrIDE requires a 4-entry buffer where each entry (20-bits) contains a row-id (17-bits) and a mitigation-level (3-bits). Thus, PrIDE requires a storage overhead of 10 bytes per bank. Additionally, it requires a pseudo random number generator, similar to DSAC [10] and PAT [19]. For RFM, the memory controller needs one counter per bank (1 byte) for the RAA.

E. Energy Overheads

PrIDE incurs energy overheads due to mitigative refreshes to victim rows and the RNG being consulted on each activation. Table X shows the average energy overheads for PrIDE and PrIDE+RFM using a 7-bit TRNG [14], [46] which has an area of 0.00025mm^2 and consumes 0.08mW leakage power per bank and 24.9pJ per activation in 10 nm technology.

PrIDE incurs 0.6% extra energy over a baseline with no energy overheads for mitigation, and PrIDE+RFM incurs 0.8% – 2.4% extra energy. PrIDE increases ACT energy by 5%–23% due to extra mitigative activations and RNG access; but as activation energy is only 13% of the total energy, its impact on overall energy is small. PrIDE increases non-ACT energy by 0.2%-1% due to RNG's leakage power (2.56mW for 32 banks), and a minor increase in execution time with RFM.

TABLE X
DRAM ENERGY OVERHEADS OF PRIDE AND PRIDE+RFM

Config	ACT Energy	Non-ACT Energy	Tot. Energy
Base (No Mitig)	1x (13% overall)	1x (87% overall)	1x
PrIDE	1.054x	1.002x	1.006x
PrIDE+RFM40	1.086x	1.002x	1.008x
PrIDE+RFM16	1.226x	1.010x	1.024x

F. Empirically Comparing Security Using Attack Patterns

Using a methodology similar to an earlier version of this work [11], we test PrIDE with 500 randomly generated uniform and non-uniform attack patterns based on TRRespass [6] and Blacksmith [12], and a Half-Double Pattern [21]. For each pattern, we run 100 trials, each with a different seed. Across all patterns and seeds, we measure *Maximum Disturbance*, the maximum activations to any row before a mitigation.

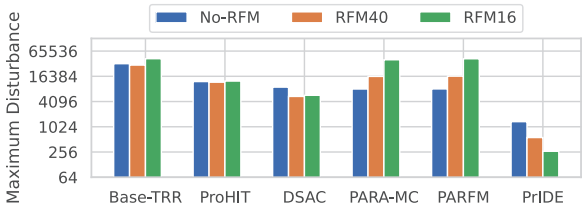


Fig. 15. Maximum Disturbance across 500 patterns for PrIDE and other probabilistic mitigations (repeated across 100 seeds).

Figure 15 shows the maximum-disturbance of different trackers: PRoHIT [38], DSAC [10], an idealized PARA [20] (PARA-MC), and PARFM [17]. For PrIDE, the maximum-disturbance is 1.3K (NoRFM), 566 (at RFM40) and 266 (at RFM16). Thus, the maximum disturbance is below our computed TRH*. For other schemes, the maximum disturbance is above 8000 (without RFM, with RFM PrIDE is even better).

VIII. OTHER RELATED WORK

PROTEAS [11]: An earlier version of this work proposed PROTEAS, a 16-entry in-DRAM tracker with probabilistic insertion-policy, random eviction-policy, and a counter-based mitigation-policy. As counter-based mitigation-policy suffers from access-pattern dependence, it becomes hard to determine the TRH* of PROTEAS. These challenges resulted in the key insight that for bounding TRH*, all three policies *must* have access pattern independence. While PrIDE uses FIFO for both eviction-policy and mitigation-policy, we also analyzed a design with Random eviction-policy and Random mitigation-policy. While this design also has access pattern independence, Random eviction-policy has higher loss-probability than FIFO, and Random mitigation-policy has unbounded Tardiness.

One-Counter-Per-Row: To mitigate Rowhammer, Panopticon [3] and PRHT [19] store per-row counters within the DRAM array. These designs change the DRAM array, require a write after each read, and incur significant area overheads (9% for PRHT). Furthermore, these designs can *still* fail (due to Tardiness) when several lines reach TRH activations within the same tREFI. For example, PRHT reports a relative failure rate of 10%. CRA [15] and Hydra [31] keep the counters in DRAM and use caches or filters to reduce the lookups, however, these schemes incur high worst-case slowdowns.

Efficient-Counters: Several proposals reduce the SRAM overhead of aggressor-row tracking. Table XI compares the per-bank SRAM overheads of PrIDE with CAT [37], TWiCE [23], and Graphene [30] (variant used in both Mithril [17] and ProTRR [27]) for devices with TRH-D of 4000 (current) and 400 (future). PrIDE has significantly lower SRAM overheads.

TABLE XI
PER-BANK SRAM OVERHEAD OF TRACKERS (PER-RANK WILL BE 32X)

Name	Device TRH-D=4K	Device TRH-D=400
Graphene	42.5 KB	425 KB
TWiCe	300KB	3 MB
CAT	196 KB	1.96 MB
PrIDE	**10 bytes**	**10 bytes**

Mitigating-Actions: For PrIDE, we use victim refresh for mitigating. RRS [32], AQUA [35], SRS [43] perform mitigation with row migration, whereas, Blockhammer [44] uses rate limits. These schemes are implemented at the memory controller and are not compatible with PrIDE. However, the in-DRAM row migration of SHADOW [42] is compatible with PrIDE. REGA [28] changes the DRAM array to provide mitigating refresh on each demand activation. HiRA [45] changes the interface to allow multiple activations to the bank. Our solution avoids changes to DRAM array and interface.

ECC-Codes: SafeGuard [5], CSI-RH [13], and PT-Guard [34] use codes to detect Rowhammer failures; however, uncorrectable failures can still cause data loss. Rowhammer failures may also be reduced by re-organizing the data layout [18].

IX. CONCLUSION

The inability to securely mitigate Rowhammer with low-cost in-DRAM trackers is causing the DRAM industry to consider radical changes (such as including per-row counters and performing a write on each read). In this paper, we provide the key insight that low-cost in-DRAM trackers can be made secure by designing them with the goal of guaranteeing a negligibly-low failure rate across all access patterns. Our proposal, PrIDE, enables this with probabilistic insertion and a FIFO buffer. PrIDE securely tolerates Rowhammer thresholds of 1.9K while requiring just 10 bytes of SRAM per bank. When co-designed with RFM, PrIDE tolerates even lower thresholds (400) at negligible slowdown (1.6%).

ACKNOWLEDGEMENTS

We thank Salman Qazi, Kuljit Bains, and the reviewers of ISCA for their feedback. This work was funded, in part, by an NSF grant 2333049 and an NSERC grant RGPIN-2023-04796.

APPENDIX A: ANALYZING LOSS PROBABILITY

We describe an analytical model to estimate the loss probability of a multi-entry tracker. While we use the model for estimating the loss probability for all sizes (1-16 entries), we describe the details only for a two-entry tracker, and then provide the general process for obtaining the loss probability of an N-entry tracker. The loss probability obtained with our analytical model matches those obtained using Monte-Carlo.

For a two-entry tracker, there are two starting points: S_0, both entries are invalid, and S_1, one entry is valid.

For S_0, let's say A appears in the first position (best case for attacker) and gets inserted. A will get mitigated unless 2+ more lines get inserted (loss prob for S_0 = P2+ = 26%).

For S_1, when A is inserted, both entries are occupied. If 2+ more lines get inserted, A will be lost (loss = P2+ = 26%). If only 1 more got inserted, A becomes the oldest and mitigated (no loss). If no more entry got inserted, the other entry will get mitigated at the end of the window, A will become the oldest entry, and will get mitigated at the end of the next window, unless 2+ entries got inserted in that window (loss = P1*P2+ = 9.6%). Thus, the total loss probability of S_1 is 35.6%.

We use Markov-Chain (Figure 16 (right)) to calculate the probability of S_0 and S_1, and use it to calculate the overall loss probability (Prob-S_0 * 0.26 + Prob-S_1 * 0.356 = 30%).

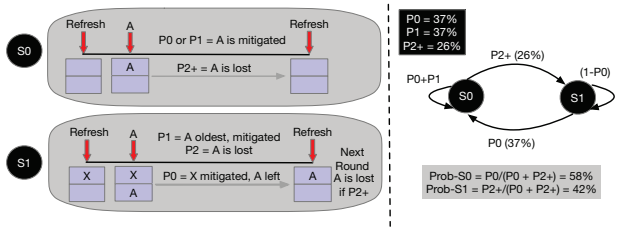


Fig. 16. (Left) Timeline and loss scenarios for starting from states S0 or S1, and (Right) Markov-Chain for calculating probability of S0 and S1.

This methodology can also determine the loss probability for an N-entry tracker, which has N starting cases (S_0, S_1, .. S_{N-1}). We estimate the loss probability of each case and use an N-state Markov Chain to calculate the total loss probability. Our artifact provides the code for modeling an N-entry tracker.

APPENDIX B: GENERALITY OF ATTACK ROUND

We explain how our analysis is general enough to capture any arbitrary attack pattern. Without loss of generality, Figure 17 shows an attack pattern that continuously activates rows A and B. Let each activation take one unit of time and tREFI period be 4 units; at refresh an arbitrarily selected row is mitigated. As the pattern contains two rows, it will concurrently have two types of attack rounds: AR_A (for Row-A) and AR_B (for Row-B). For simplicity, we assume that both A and B were refreshed at t=0. Lets assume that refresh at t=4 mitigates B, at t=8 mitigates A, and at t=12 mitigates A. We have two complete attack rounds for A, with first AR_A of size 4 and second AR_A of size 2 (note that the size is determined solely by the number of activations to a given row, without

regard to any intervening activations to other rows). Similarly, there are two attack rounds for B, the first AR_B of size 2 and the second AR_B of size 4+ (unfinished). Each activation contributes to exactly one attack round.

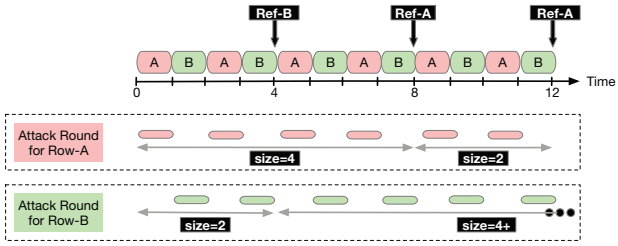


Fig. 17. A pattern consisting of two rows and a tREFI window of 4 activations. The size of an attack round of a given row is the number of activations of that row between consecutive mitigations. An attack pattern can interleave arbitrary number of rows, resulting in concurrent attack rounds.

Generalizing to Multi-Line Attacks: If a pattern contains k unique rows (interleaved arbitrarily), it will have k concurrent attack rounds (one per row). However, as shown in Section IV-C, our mitigation decision is *access-pattern independent* (does not depend on what addresses were accessed before or after or on position of attacked rows), we can analyze the pattern as containing k independent access streams (one per row), and compute probability of failure for each stream.

For PrIDE, each activation is inserted with probability p, and survives in buffer with probability $(1-L)$, so each activation has a probability $\hat{p} = p \cdot (1-L)$ of getting mitigated, independent of any other activation (not accounting for tardiness). Thus, the success of an attack with K independent streams is equivalent to an attack on a single row with the activations of all K streams. So, increasing the number of rows in an attack pattern does not increase the chance of a successful attack.

Impact of Multi-Line Attack on TTF: For our analysis, we do not consider the size of the attack-round in terms of *physical time* (total time between mitigations), but rather the *logical time* (the amount of time an attack-round spent in attacking any given row), which is determined by the number of activations in the attack round. For example, in Figure 17, the size of the first attack-round of A is 4 (even though it lasts for 8 units of physical time). As each activation contributes to exactly *one* attack-round, the sum of all logical time is equal to the total physical time, regardless of the number of attack lines in the pattern. Thus, multi-line attacks do not impact the Time-to-Failure (TTF) compared to a single-line attack.

Appendix C: Validation of Loss Probability

The loss probability is not dependent on access patterns, however, it does depend on the position of the line in the tREFI window (earlier positions are more likely to get lost). Both our analytical model and the Monte-Carlo simulation report the loss probability for the *most* vulnerable position, regardless of which addresses get accessed. Therefore, by design, the loss probability of any pattern should never exceed our estimates. To experimentally validate this claim, we evaluate the loss probability for 900 attack traces (500 traces for TressPass with 2 to 501 attack rows and 400 traces of Blacksmith with 4 to 80 attack rows, repeated 2 to 32 times, with 20 to 80 decoy rows). For each pattern, we run 1 million iterations and report only the row with the highest loss probability.

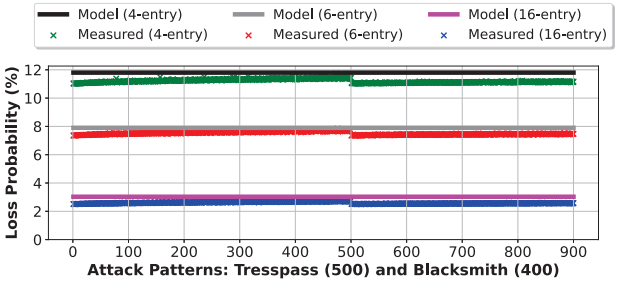


Fig. 18. Comparison of Loss Probability from our Model (Analytical and Monte-Carlo) and experimentally measured on 900 attack traces (for 4, 6, 16 entries). The measured loss probability never exceeds our estimates.

Figure 18 shows our model-estimated loss probability and measured loss-probability for representative buffer sizes of 4, 6, and 16. Across 2700 data points (3 sizes x 900 traces), the measured loss probability *never* exceeds our model estimates. For some patterns, the address moves at different positions in a tREFI window, so the measured loss probability is equal to the average position (hence lower than our worst-case position).

Appendix D: Sanity Check of Reliability Model

Our reliability model allows us to estimate TRH* using a simple closed-form solution (for example, see TRH* using Equation 8). However, the security bounds (TRH*) of PrIDE can also be derived using models used in prior works.

Saroiu and Wolman [33] describe a detailed methodology for determining the sampling rate of probabilistic Rowhammer solutions. Their *formulae* relies on a recurrence equation (see Equations 1-3 [33]) and analyzes a window of tREFW. The recurrence equation does not lend itself to a closed-form solution. We can still use this method to determine the TRH* (without Tardiness) of PrIDE for a sampling rate equal to $\hat{p} = p \cdot (1-L)$ and add Tardiness to obtain TRH*.

Table XII shows the TRH* obtained by our method and the model proposed by Saroiu-Wolman, as the number of entries (N) in the buffer varies. We also show the value for loss-probability (L), the effective mitigation probability ($\hat{p}$) and Tardiness. The TRH* produced by both models are similar, and even when there is a small divergence, our model produces a slightly pessimistic (higher) TRH*. While either model can be used to determine TRH* of PrIDE, we chose to use ours as it leads to simple and intuitive closed-form solutions.

TABLE XII
THE TRH* OF PRIDE USING OUR MODEL AND SAROIU-WOLMAN [33]

Num Entries	L	$\hat{p}$ (1/79)*(1-L)	Tardiness (NW)	PrIDE-TRH* (Our Model)	PrIDE-TRH* (Saroiu-Wolman)
Ideal	0	1.27%	0	3056	3056
1	0.63	0.47%	79	8366	8153
2	0.30	0.88%	158	4561	4520
4	0.12	1.12%	316	3787	3777
8	0.06	1.19%	632	3883	3879
16	0.03	1.23%	1262	4415	4413

REFERENCES

[1] 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," *ASPLOS-2016*.

[2] 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 *HPCA*, 2023.

[3] T. Bennett, S. Saroiu, A. Wolman, and L. Cojocar, "Panopticon: A complete in-dram rowhammer mitigation," in *Workshop on DRAM Security (DRAMSec)*, 2021.

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

[5] 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*. IEEE, 2022.

[6] 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 *IEEE Symposium on Security and Privacy*, 2020.

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

[8] D. Gruss, C. Maurice, and S. Mangard, "Rowhammer.js: A remote software-induced fault attack in javascript," in *Detection of Intrusions and Malware, and Vulnerability Assessment*, 2016.

[9] 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.

[10] 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*, 2023.

[11] A. Jaleel, S. W. Keckler, and G. Saileshwar, "Probabilistic tracker management policies for low-cost and scalable rowhammer mitigation," *arXiv:2404.16256*, 2024.

[12] 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.

[13] 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.

[14] 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.

[15] 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.

[16] 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*, 2020.

[17] 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.

[18] 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 *MICRO*, 2023.

[19] 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 *ISSCC*, 2023.

[20] 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.

[21] 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.

[22] 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.

[23] 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.

[24] K. Loughlin, S. Saroiu, A. Wolman, Y. A. Manerkar, and B. Kasikci, "Moesi-prime: preventing coherence-induced hammering in commodity workloads," in *ISCA*, 2022.

[25] J. Lowe-Power *et al.*, "The gem5 simulator: Version 20.0+," *arXiv preprint arXiv:2007.03152*, 2020.

[26] 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 *ISCA-50*, 2023.

[27] 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.

[28] 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.

[29] "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

[30] 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.

[31] 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.

[32] 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.

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

[34] 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.

[35] A. Saxena, G. Saileshwar, P. J. Nair, and M. Qureshi, "Aqua: Scalable rowhammer mitigation by quarantining aggressor rows at runtime," in *MICRO*, 2022.

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

[37] S. M. Seyedzadeh, A. K. Jones, and R. Melhem, "Mitigating wordline crosstalk using adaptive trees of counters," in *ISCA*, 2018.

[38] M. Son, H. Park, J. Ahn, and S. Yoo, "Making dram stronger against row hammering," in *Design Automation Conference*, 2017.

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

[40] 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.

[41] V. van der Veen, Y. Fratantonio, M. Lindorfer, D. Gruss, C. Maurice, G. Vigna, H. Bos, K. Razavi, and C. Giuffrida, "Drammer: Deterministic rowhammer attacks on mobile platforms," in *ACM-CCS*, 2016.

[42] 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.

[43] 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.

[44] A. G. Yağlikçi *et al.*, "Blockhammer: Preventing rowhammer at low cost by blacklisting rapidly-accessed dram rows," in *HPCA*, 2021.

[45] 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.

[46] 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.

[47] 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.

X. Artifact Appendix

A. Abstract

This artifact presents the code for PrIDE, our Rowhammer mitigation. We provide the C code for the security analysis modeling the time-to-failure for PrIDE, the probability of successful attack round on PrIDE, and the TRH it protects against. We also provide the C++ code to evaluate the performance impact of PrIDE, implemented within the gem5 [25] simulator. We provide scripts to both run the security analysis and for the performance analysis, which involves compiling the simulator, and running the baseline, PrIDE and PrIDE with RFM (RFM-40 and RFM-16). We also provide scripts to parse the results and recreate the key results in security – Figure 8, Table III, Figure 9, Table IV, Table V, Table VI, Table VIII, and Table IX, and performance evaluations – Figure 14.

B. Artifact check-list (meta-information)

- **Algorithm**: PrIDE, PrIDE+RFM mitigation.
- **Program**: 17 SPEC-2017 workload binaries using gem5 simulator and simulated with gem5's Syscall-Emulation (SE) mode.
- **Compilation**: Tested with gcc v9.4.0 with scons v3.0.5.
- **Binary**: SPEC-2017 workloads- blender, lbm, roms, gcc, mcf, cactuBSSN, xz, deepsjeng, imagick, nab, bwaves, namd, parest, leela, wrf, povray, exchange2.
- **Data set**: SPEC-2017 rate workloads with reference dataset.
- **Run-time environment**: Tested on Ubuntu 20.04 x86_64 server.
- **Hardware**: Security evaluation can be run on a laptop. Performance simulations requires many-core server. Our machine had 22 cores and 64 GB DRAM. Experiments can take 1 day ($\tilde{1}$00 gem5 runs required to reproduce the performance graph).
- **Execution**: monte-carlo, analytical model, gem5 simulations.
- **Metrics**: Loss Probability, Time-to-Fail, TRH, Normalized IPC.
- **Output**: Recreating security results (Figure 8, Table III, Figure 9, Table IV, Table V, Table VI, Table VIII, and Table IX) and performance results (Figure 14).
- **Experiments**: Instructions to run the experiments, parse results, and plot graphs are available in the README file.
- **How much disk space required (approximately)?**: About 10-20 GB excluding the SPEC2017 benchmark installation.
- **How much time is needed to prepare workflow (approximately)?**: SPEC-2017 installation takes between 2-6 hours and gem5 simulator compilation takes 30-60 minutes on a server system. Checkpointing the SPEC binaries takes about 1 day.
- **How much time is needed to complete experiments (approximately)?**: Once SPEC checkpoints are available, Approximately 1 day on a 22 core system. There are 100 parallelizable gem5 execution instances that each take about 3 to 6 hours.
- **Publicly available?**: Yes.
- **Workflow framework used?**: Monte-Carlo Simulations, Analytical Models, gem5 simulator, and SPEC-2017 benchmarks.
- **Archived (provide DOI)?**: The DOI for the artifact is https://doi.org/10.5281/zenodo.11076040. The latest code for the artifact is available at https://github.com/gururaj-s/PrIDE.

C. Description

1) How to access: The code and instructions to run the artifact are available at: https://github.com/gururaj-s/PrIDE.

2) Hardware dependencies: Security evaluations can be run on most generic linux machines (can run even on a laptop). Performance evaluations require gem5 simulations, which need to be parallelized on a system with many cores (ours has 22 cores) and large memory (ours has 64 GB DRAM) – these allow running all the 34 workloads in parallel per configuration ($\sim$6 configurations including checkpointing and baseline).

3) Software dependencies: Gcc for compilation, Perl to parse results, Python3 with matplotlib to plot graphs. Additionally all the dependencies for Gem5 itself are needed. An installed copy of SPEC CPU-2017 workloads is required.

D. Installation

The `scons` software construction tool is used to compile the gem5 simulator. The SPEC-2017 workloads must be installed separately as we cannot provide them due to SPEC's restrictive license.

E. Experiment workflow

The README provides detailed instructions required to reproduce the results from the paper. These include:

For security evaluations:

- Run the security evaluation script to perform the analysis, get the Loss Probability, TRH and Time-To-Fail, and display the corresponding tables.

For performance evaluations:

- First, compile gem5 simulator on the platform and install SPEC binaries.
- Next, create gem5 checkpoints for the workloads.
- Then, execute the simulations (baseline, PrIDE, PrIDE + RFM40, PrIDE + RFM16).
- Finally, parse the simulation results and plot the graph for normalized performance.

F. Evaluation and expected results

The artifact provides the scripts to run both the security and the performance evaluations, and display the tables/graphs. The relevant commands are provided in the artifact README. The expected results from this artifact is to recreate the key security results – Figure 8, Table III, Figure 9, Table IV, Table V, Table VI, Table VIII, and Table IX, and the key performance results – Figure 14.

G. Experiment customization

Scripts to conduct test runs before launching full checkpointing and simulation runs are provided in the artifact. Although customization is not expected, it can be done in interest of limited time or resources by changing a few parameters in the scripts in the artifact.

H. Troubleshooting Steps Useful for Debugging Perf Evals

1) Verify successful SPEC-17 benchmark native execution: Test individual SPEC-17 benchmarks after compilation, to see whether they natively run successfully. You can check this by running the run commands for individual benchmarks in the respective run directories (see https://github.com/ankurlimaye/SPECCPU2017CommandLines/blob/master/SPEC_CPU2017_Commands.pdf).

2) Verify gem5 experiment completion: In each of the configurations `output` directory, check the number of benchmark folders where the `runscript.log` has a `reached maximum instruction count` at the end of the file. If any of the files did not have this string, it would mean those experiments failed for some reason.

3) Verify stat file generation: Compare the stats generated with the ones present in the GitHub repository in the `scripts/stats_scripts/data` directory. We will also provide the raw stats (`stats.txt`) for each configuration and benchmark.