When Mitigations Backfire: Timing Channel Attacks and Defense for PRAC-Based RowHammer Mitigations

Jeonghyun Woo
The University of British Columbia
Department of Electrical and Computer Engineering
Vancouver, BC, Canada
jhwoo36@ece.ubc.ca

Joyce Qu
University of Toronto
Department of Computer Science
Toronto, ON, Canada
joyce.qu@mail.utoronto.ca

Gururaj Saileshwar
University of Toronto
Department of Computer Science
Toronto, ON, Canada
gururaj@cs.toronto.edu

Prashant J. Nair
The University of British Columbia
Department of Electrical and Computer Engineering
Vancouver, BC, Canada
prashantnair@ece.ubc.ca

Abstract

Per Row Activation Counting (PRAC) has emerged as a robust framework for mitigating RowHammer (RH) vulnerabilities in modern DRAM systems. However, we uncover a critical vulnerability: a timing channel introduced by the Alert Back-Off (ABO) protocol and Refresh Management (RFM) commands. We present *PRACLeak*, a novel attack that exploits these timing differences to leak sensitive information, such as secret keys from vulnerable AES implementations, by monitoring memory access latencies.

To counter this, we propose *Timing-Safe PRAC (TPRAC)*, a defense that eliminates PRAC-induced timing channels without compromising RH mitigation efficacy. *TPRAC* uses *Timing-Based RFMs*, issued periodically and *independent* of memory activity. It requires only a single-entry in-DRAM mitigation queue per DRAM bank and is compatible with existing DRAM standards. Our evaluations demonstrate that *TPRAC* closes timing channels while incurring only 3.4% performance overhead at the RH threshold of 1024.

CCS Concepts

• **Security and privacy → Security in hardware**.

Keywords

DRAM, RowHammer, Timing Channel, Security, PRAC, RFM

1 Introduction

Technology scaling has enabled high-capacity DRAMs [66], but also introduced security and reliability challenges such as RowHammer (RH) [44, 49]. To address this, JEDEC recently introduced Per Row Activation Counting (PRAC) in DDR5 [41], which precisely tracks row activations and provides robust RH protection. While PRAC successfully mitigates RH, this paper reveals a new critical security threat: information leakage through timing channels. Our goal is to study the extent of these timing channel risks and to propose a low-cost and secure defense that eliminates them.

The PRAC specification provides robust RH mitigation, even at ultra-low RH thresholds [10, 77, 102]. PRAC achieves this by precisely tracking row activations using its per-row counters and enabling the DRAM to request mitigation time from the memory controller via the Alert Back-Off (ABO) protocol [41]. When a row's activation counter reaches the Back-Off threshold (N_{BO}), DRAM asserts the Alert, prompting the memory controller to issue a Refresh Management (RFM) command, termed ABO-RFM, which blocks all memory requests (e.g., for 350ns) to allow RH mitigation. Alternatively, the memory controller may proactively send RFMs when any bank experiences a high number of activations [41]; we call these Activation-Based RFMs (ACB-RFMs). Although effective against RH, we observe that they can inadvertently introduce a new vulnerability–timing channels due to these mitigation actions.

Figure 1(a) shows how high row activations can create timing channels in PRAC-enabled systems. As row activation frequencies
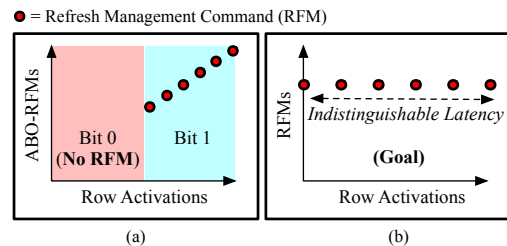

Figure 1: Timing Channels in PRAC. Alert Back-Off-triggered RFMs (ABO-RFMs) introduce activation-dependent latency, enabling timing channels. An adversary can exploit these to transmit 'Bit-0' or 'Bit-1'. Our goal is to remove this dependency and make RFMs timing independent of activations.

increase and consequently more rows surpass N_{BO}, ABO protocols are triggered more often. This, in turn, leads to an increasing number of ABO-RFMs being issued for mitigation. Each triggered ABO-RFM introduces a 350ns delay at the memory controller, uniformly increasing latency for all concurrent memory accesses to that DRAM channel. These latency variations can be exploited to transmit 'Bit-0' or 'Bit-1' in a cross-core covert channel.

While memory systems have many sources of latency variations, PRAC-based timing channels are uniquely precise, enabling exact values to be reliably transmitted across processes. For instance, a sender can activate a row exactly the $N_{BO}-X$ times; subsequently, a receiver performing precisely X additional activations on the same row can deterministically trigger an RFM due to ABO protocol (ABO-RFM). We demonstrate that this capability can be exploited to establish both covert and side channels. Thus, our goal is to prevent such timing channels by ensuring RFMs are issued independently of row activations, as shown in Figure 1(b).

This paper observes that all prior RFM-based mitigations–whether PRAC-based [10, 77, 102] or not [35, 47, 78]–issue RFMs based on DRAM row activity. Such *activity-dependent* RFMs can be exploited to create timing channels: in PRAC-based mitigations, attackers can observe a victim's row activation patterns via ABO-RFMs, while in non-PRAC mitigations, they can infer a victim's overall activation levels via Activation-Based RFMs (ACB-RFMs). We focus on timing channels arising from PRAC-based mitigations due to their expected widespread adoption in the near future.

This paper makes two key contributions. First, to demonstrate and analyze the threat posed by PRAC-induced timing channels, we introduce a new class of attacks, *PRACLeak*, which exploits timing variations caused by ABO protocol in PRAC to leak sensitive information. Second, we propose a novel defense, *Timing-Safe PRAC (TPRAC)*, which eliminates these timing channels through a proactive, *activation-independent* mitigation strategy. *TPRAC* is designed to break the link between row activation patterns and memory access latency, effectively neutralizing *PRACLeak* attacks while preserving strong RowHammer protection.

1. The *PRACLeak* Attacks: Our *PRACLeak* attacks are built on two key observations. First, PRAC-based mitigations [10, 77, 102] issue RFMs based on DRAM row activation counts. Second, the RFMs triggered by the ABO protocol (ABO-RFM) introduce latency for all concurrent memory accesses, making both their occurrence and associated delays observable system-wide. When an ABO is initiated by the DRAM, the memory controller issues an RFM All Bank (RFM_{ab}) command. RFM_{ab} halts all memory requests for a predefined period (e.g., 350ns) to allow the DRAM to perform necessary mitigations, increasing the latency of affected memory accesses. This latency impact becomes more pronounced if PRAC is configured to use 2 or 4 RFMs per ABO, extending the delay to 700 or 1400ns. These timing differences can be exploited to form a timing channel for leaking information in both cross-process and cross-VM settings. The only requirement is that the attacker and victim share a memory module. *PRACLeak* thus enables both covert and side channels, allowing attackers to establish two types of channels:

- **Activity-Based Channels**: In this channel, the sender and receiver share the same DRAM channel. The sender transmits a single bit of information over a fixed time window by either activating a row beyond N_{BO} times (to trigger an ABO-RFM) or not.

The receiver monitors the latency of its own memory accesses and decodes the transmitted bit based on the presence ('Bit-1') or absence ('Bit-0') of a latency spike.

- **Activation-Count-Based Channels**: In this channel, the sender and receiver share a single physical DRAM row. This is feasible if (1) the size of DRAM row (e.g., 8KB) exceeds the physical page size (e.g., 4KB), or (2) the memory controller uses address mapping that spreads data from a single page across multiple DRAM banks to enhance bank-level parallelism, which is common in modern systems [75, 98]. This allows two processes to occupy separate pages within the same row. Here, the receiver can leak the exact activation count of the sender. For example, if N_{BO} is 500, and the receiver observes a latency spike after 1 activation, the receiver knows the sender activated that row 499 times. Thus, the sender can transmit any values up to N_{BO} (i.e., $log_2(N_{BO})$ bits).

Using a PRAC [10] model in Ramulator2 [50, 60], we demonstrate covert channels achieving bitrates of 11–41 Kbps with activity-based channels and 39–124 Kbps with activation-count-based channels in a cross-process setting. We also demonstrate a side-channel attack on an AES T-Table implementation, leaking secret key bits in under 200 encryptions via the activation-count-based channel.

2. *TPRAC* Mitigation: *Timing-Safe PRAC (TPRAC)* eliminates PRAC-induced timing channels by replacing *activity-dependent* mitigations with an *activity-independent* strategy. It introduces *Timing-Based RFMs (TB-RFMs)*, issued at fixed time intervals and completely independent of row activation behavior. By decoupling RFMs from memory access patterns, *TPRAC* prevents attackers from inferring activity through latency variations. The time interval between consecutive TB-RFMs (TB-Window) is configured to ensure no DRAM row reaches N_{BO}, even under worst-case attack patterns (e.g., Feinting [61] or Wave [111] attacks). At each TB-RFM, *TPRAC* proactively mitigates the most heavily activated row–regardless of whether it has reached N_{BO}–using a *single-entry* mitigation queue per DRAM bank. This preemptive approach eliminates ABO events and the resulting timing channels. *TPRAC* requires no DRAM modifications and is fully compatible with existing interfaces, making it both secure and practical. For example, at the RH threshold of 1024, issuing one TB-RFM every 1.6 tREFI suffices to close timing channels with just a 3.4% performance slowdown.

Summary of Contributions:

- **Unveiling PRAC Timing Channel Vulnerabilities**: To the best of our knowledge, we are the first to reveal the timing channel vulnerability of any RowHammer mitigations. We demonstrate how timing variations introduced by the ABO protocol of PRAC can be exploited to create a new class of timing channels.

- **PRACLeak Covert-Channel Attacks**: Exploiting ABO-induced timing differences, we demonstrate covert-channel attacks achieving 39–124 Kbps when the sender and receiver share a DRAM row, and 11–41 Kbps without requiring shared rows.

- **PRACLeak Side-Channel Attacks**: Using the same timing channel, we demonstrate a side-channel attack that successfully leaks bits of a secret AES key from a victim running a vulnerable AES T-Table implementation in less than 200 encryptions.

- **TPRAC Defense**: We propose *TPRAC*, a low-cost defense that eliminates PRAC-induced timing channels by proactively issuing RFMs at fixed intervals, independent of row activations. *TPRAC* requires only a single-entry mitigation queue per bank.

2 Background and Motivation

2.1 Threat Model

We consider a realistic threat model where an adversary and a victim share a DRAM module implementing PRAC-based RowHammer (RH) mitigations [10, 77, 102]. The adversary can operate with user-level privileges and run a malicious process to induce Refresh Management (RFM) commands and exploit the resulting timing variations to establish a side or covert channel. By measuring performance metrics such as memory access latency and execution time, the adversary can infer the victim's memory access patterns, enabling them to leak sensitive information (side channel) or transmit secrets (covert channel). We assume that all users can monitor the total number of RFMs within a given time window by issuing memory requests and detecting latency spikes caused by RFMs.

2.2 The RowHammer Vulnerability

RowHammer (RH) is a read disturbance issue where rapid activations of DRAM rows (aggressor rows) accelerate charge leakage in neighboring rows (victim rows), leading to bit-flips [44, 49, 76]. The RH has worsened with continued DRAM scaling, as cells become smaller and closely packed. The RowHammer threshold (N_{RH}), the minimum number of activations required to induce bit flips, has decreased from 69.2K in 2013 [49] to nearly 4.8K in 2020 [45], and is expected to drop further with future smaller technology nodes.

RH poses a severe security threat and has been exploited in numerous ways [7, 15, 16, 26, 27, 81, 82, 96]. Example attacks include privilege escalation [90], confidential data leakage [53], degradation of ML model accuracy [31, 108], and memory performance attacks (Denial-of-Service) [37, 103]. As N_{RH} drops, RH can even occur with benign applications, making it a critical reliability concern [59].

2.3 Per Row Activation Counting (PRAC)

The latest DDR5 specification introduces Per Row Activation Counting (PRAC) to address RH [41]. PRAC employs two key mechanisms:

- **Per-Row Activation Counters**: Each DRAM row includes additional activation counter cells and sense amplifiers. On each activation, the corresponding row's counter is incremented; this requires a Read-Modify-Write operation during the precharge phase (i.e., when the row is being closed) [41]. To support this, Relevant DRAM timings (e.g., tRP and tWR) were adjusted in the PRAC specification [41], which we incorporate in our evaluation.

- **Alert Back-Off (ABO) Protocol**: When a row's counter crosses the Back-Off threshold (N_{BO}), the DRAM asserts the Alert signal to request additional mitigation time from the memory controller. In response, the memory controller permits a limited number of additional activations (ABO_{ACT}) before initiating the ABO mitigation period. During the mitigation period, the memory controller issues a predefined number of RFM All Bank (RFM_{ab}) commands, referred to as the PRAC Level (N_{mit}). Each RFM_{ab} blocks all memory requests for 350ns ($tRFM_{ab}$), allowing the DRAM to refresh *four* victim rows and *reset* its counter. A subsequent Alert can only be triggered after a specified number of activations (ABO_{Delay}), which is set equal to N_{mit}.

The PRAC specification also includes proactive Activation-Based RFMs (ACB-RFMs), known as Targeted RFMs, which are triggered

Table 1: PRAC Parameters as per JEDEC specification [41]

Parameter	Explanation	Value
N_{BO}	Back-Off Threshold	$N_{BO} < N_{RH}$
N_{mit}	Num RFMs on Alert (PRAC Level)	1, 2, or 4
ABO_{ACT}	Max. ACTs to a row from Alert to RFM	3 (up to 180ns)
ABO_{Delay}	Min. ACTs after RFM to Alert	Same as N_{mit} (1,2, or 4)
$tRFM_{ab}$	Duration of RFM All Bank (RFM_{ab})	350ns

when the number of activations of a DRAM bank hits a threshold called the Bank Activation threshold (BAT). BAT is typically set below N_{BO} (e.g., 75) to reduce or avoid frequent Alerts [41]. Additionally, the specification allows DRAM to perform extra mitigations using the slack time of refresh operations, called Targeted Refresh (TREF), similar to Target Row Refresh (TRR) [30, 40]. As we later show in Section 4.3, TREF can be leveraged to improve the performance of *TPRAC*. Table 1 shows the key PRAC parameters.

The PRAC specification provides minimal implementation details to allow DRAM vendors flexibility in their implementations. Consequently, PRAC's security and performance heavily depend on implementation decisions, particularly the design of the mitigation queue. For example, recent studies have shown that PRAC implementations using simple FIFO-based mitigation queues are vulnerable to targeted attacks [77, 102], whereas omitting a mitigation queue entirely can incur severe performance overhead [102]. Therefore, achieving both security and performance in PRAC-enabled systems necessitates careful consideration of the mitigation queue design and its associated management strategy.

2.4 Performance-Driven Timing Channels

A timing channel exploits latency differences to transmit information, serving as a *covert channel* for secret communication or a *side channel* for leaking sensitive data. Performance optimizations in modern systems often create such channels by modulating data access latencies [12, 57, 75, 97]. Mitigation typically involves reducing or disabling these optimizations.

2.5 PRAC: Security-Driven Timing Channel

PRAC-based timing channels are security-driven [41]. Unlike performance-driven timing channels, which can often be mitigated by disabling performance optimizations, disabling PRAC exposes the system to RH attacks. These channels originate from the Alert Back-Off (ABO) protocol: the DRAM asserts the Alert when a row's activation count reaches N_{BO}. In response, the memory controller issues RFMs for mitigations, and the resulting latency spikes create observable timing variations that can be exploited.

2.6 Pitfalls: ABO-RFM and ACB-RFM

Figure 2 illustrates how PRAC-based RH mitigations [10, 77, 102] rely on *activity-dependent* RFMs, such as ABO-RFM and ACB-RFM. In the case of ABO-RFM (Figure 2(a)), an attacker can repeatedly activate a DRAM row until its activation count reaches N_{BO}, triggering the ABO protocol. This results in a noticeable latency spike, which can be exploited as a timing channel. In contrast, ACB-RFM is triggered when the total number of activations in a bank crosses BAT, potentially avoiding ABO-RFMs. However, since ACB-RFMs are still tied to activation levels, an adversary can exploit them instead of ABO-RFMs to form timing channels (Figure 2(b)).
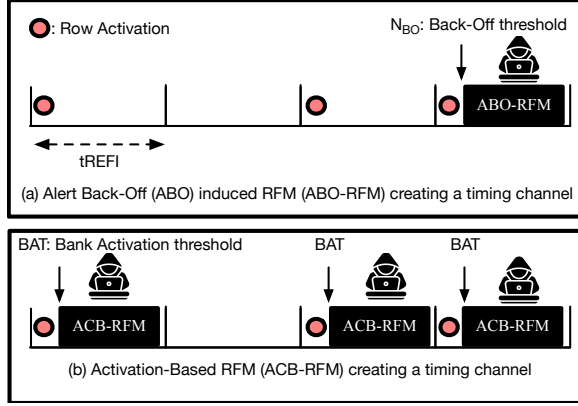
Figure 2: The activity-dependent nature of PRAC's RFMs allows attackers to exploit them. (a) Repeated activations of a DRAM row up to the Back-Off threshold (N_{BO}) trigger an RFM due to ABO (ABO-RFM). (b) Activation-Based RFMs (ACB-RFMs), which mitigate proactively at lower thresholds, can avoid ABO-RFMs but remain exploitable as attackers can still induce ACB-RFMs by activating rows within a bank.

3 Demonstrating PRAC's Timing Channels

In this section, we demonstrate timing channels inherent in PRAC. While our analysis is based on Understanding PRAC (UPRAC) [8], the presented attacks exploit fundamental features of the PRAC—the Alert Back-Off (ABO) protocol and per-row activation counters—making them applicable to other PRAC-based defenses, including QPRAC [102] and MOAT [77].

We first characterize the timing variations caused by PRAC's ABO protocol (Section 3.1). We then present the proposed *PRACLeak* attacks: first, a covert channel between a spy (receiver) and a trojan (sender) (Section 3.2), followed by a side-channel attack where a spy leaks information from a victim application (Section 3.3).

3.1 Timing Variation due to Alert Back-Off

In PRAC, when any row's activation counter hits the Back-Off threshold (N_{BO}), the DRAM asserts the Alert signal to request time for RowHammer (RH) mitigations from the memory controller. In response, the memory controller issues one or more RFM All Bank (RFM_{ab}) commands, each stalling all memory requests to the affected channel for tRFM (typically 350ns). As a result, an ABO event introduces a system-wide latency spike, which can be observed by threads running on other cores.

Because ABO occurrences depend on the activation counts of specific physical rows–which can be influenced by an adversarial thread sharing access to these rows–they can leak information about which row was accessed and how frequently it was activated.

Characterization. Figure 3 shows how an adversary can detect an ABO by observing increased memory access latency during its occurrence. We assume the attacker (e.g., a process or VM) monitors its memory access latencies on a different core in the same system as a victim, whose repeated activations trigger an ABO. The attacker accesses a different bank in the same channel and performs repeated accesses–either to the same row in an open-page setting
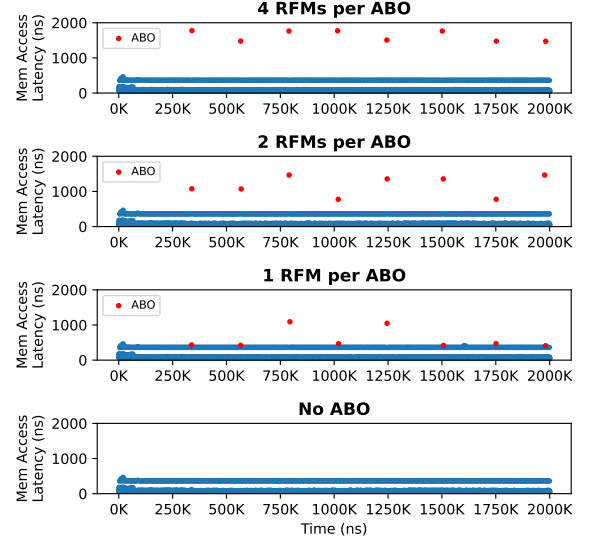

Figure 3: Timing variation for memory accesses for an attacker in the presence and absence of a concurrent Alert Back-Off (ABO) due to victim's activity.

or to different rows in a closed-page setting–to avoid increasing activation counts on its own rows and prevent self-induced ABOs.

Without a concurrent ABO from the victim, the attacker's memory access latency remains relatively stable. However, when the victim's activations trigger an ABO, the attacker experiences a notable increase in its memory accesses latency. The PRAC specification allows for a configuration number of RFMs (1, 2, or 4) per ABO. With 1 RFM per ABO, the attacker observes a latency of 545ns on average. This increases to 976ns and 1669ns with 2 and 4 RFMs, respectively, making the ABO induced by victim activity increasingly detectable.

3.2 PRACLeak Covert-Channel Attacks

Covert-channel attacks enable a trojan and a spy to communicate covertly by exploiting shared hardware resources. Our attack leverages the timing impact of the ABO on memory access latency to construct two types of covert channels: (1) an activity-based channel, where a single bit is transmitted based on the presence or absence of an ABO within a fixed time window, and (2) an activation-counter-based channel, where multiple bits are transmitted by encoding information into the timing of an ABO occurrence. We refer to these as the *PRACLeak* Covert-Channel Attacks.

(1) Activity-Based Covert Channel. In this channel, following the strategy described in Section 3.1, the receiver (spy) repeatedly accesses a single memory location in a loop. Under an open-page policy, the receiver flushes the address from the cache to ensure the row-buffer hits. Under a closed-page policy, it accesses addresses mapped to different rows within a bank to prevent activation counters from reaching N_{BO}, thereby avoiding self-induced ABOs. To transmit a 'Bit-1', the sender (trojan) activates a row in a different bank N_{BO} times, typically by alternating accesses to a pair of different rows to cause row-buffer conflicts and guarantee activations. To transmit a 'Bit-0', the sender remains idle, avoiding an ABO. When

a 'Bit-1' is transmitted, the ABO triggered by the sender causes a detectable latency spike in the receiver's memory access, while the absence of such a spike indicates a 'Bit-0'. Under a closed-page policy, the transmission period is approximately $N_{BO} \cdot tRC + tRFM$, reflecting the time required for N_{BO} activations and ABO servicing.

(2) Activation-Count-Based Covert Channel. A more sophisticated covert channel can be established if the sender and receiver share the same DRAM row. This is possible when the physical page is smaller than the DRAM row, or when successive cache lines are mapped to different banks for bank-level parallelism [75, 98], allowing both parties to access the same DRAM row from different physical pages. In this setup, the sender transmits multiple bits by encoding them into the row's activation counter. For example, within a fixed time window, the sender activates the shared row k times (where $k < N_{BO}$). The receiver then activates the same row up to N_{BO} times and observes when an ABO-induced delay occurs–specifically, after $N_{BO} - k$ activations–allowing it to infer the value k. This enables the sender to transmit $\log(N_{BO})$ bits per time window. Since both the sender and receiver perform up to N_{BO} activations sequentially, the transmission period is nearly double that of the previous attack: approximately $2 \cdot N_{BO} \cdot tRC + tRFM$.

Results. We implement these attacks in Ramulator2 [60], a trace-based DRAM simulator, using spy and trojan traces running on different CPU cores. We use the open-sourced UPRAC [8] as our PRAC implementation. While we use PRAC configuration of 4 RFMs per ABO, our bitrates are similar across 1, 2, and 4 RFMs per ABO. Table 2 shows the transmission period (i.e., time for single transmission) and bitrates for our two covert channels, evaluated with N_{BO} values ranging from 256 to 1024. At N_{BO} of 256, our activity-based covert channel achieves a transmission period of $24.1\mu s$ and a bitrate of 41.4Kbps. In contrast, our activation-count-based covert channel has a longer transmission period of $64.7~\mu s$. Still, it achieves a higher bitrate of 123.6 Kbps, as it encodes $\log(N_{BO})$ bits per period compared to the 1 bit per period in the activity-based channel. As N_{BO} increases, bitrates decrease due to longer transmission periods. The error rates are overall negligible ($< 0.1\%$).

Table 2: Covert Channel Transmission Period and bitrate

Type	N_{BO}	Transmission Period (μs)	bitrate (Kbps)
	256	24.1	41.4
Activity-Based	512	46.7	21.4
	1024	91.8	10.9
	256	64.7	123.6
Activation-Count-Based	512	128.0	70.3
	1024	257.6	38.8

3.3 PRACLeak Side-Channel Attacks

This section extends the activation-counter-based covert channel into a side-channel attack. We show how an attacker can infer a victim's memory access patterns–and ultimately leak sensitive information–by observing ABO-induced timing variations. We refer to this as the *PRACLeak* Side-Channel Attack.

Threat Model. We assume the attacker and the victim are in different processes or VMs running on different cores of the same system. They share physical DRAM rows, meaning victim secrets are co-located with attacker-controlled data. This can occur either because the DRAM row is larger than a physical page or due to physical-to-DRAM address mappings that stripe successive cache lines across banks for bank-level parallelism, causing data from different processes to map to the same row. The victim is repeatedly executed with attacker-controlled inputs, as in prior attacks [23, 51].

Attack Setup. Crypto libraries like OpenSSL and GnuPG provide an AES implementation with T-tables, known to be vulnerable to cache-based side-channel attacks that leak the secret key [3, 5, 25]. We use this implementation as our target (victim) to demonstrate our ABO-based side-channel attack. Each of the ten rounds in this AES design performs 16 secret-key-dependent memory accesses to 4 T-tables (4 accesses per table). Each table spans 16 cache lines, with each cache line mapped to a different DRAM row. The T-table lookup indices depend on the input plaintext p and the secret key k. For the first round, they are calculated as $x_i = p_i \bigoplus k_i$, where i is the byte position in p or k (i goes from 0 to 15).

We perform a chosen plaintext attack on the first AES round. The attacker provides plaintexts to the victim for encryption, keeping p_0 constant while varying other bytes p_i randomly for $i \neq 0$. This ensures that the index x_0 always accesses the same T-table cache line during the first round. The other x_i (i =1 to 15) are randomly distributed across all the cache lines. As a result, the cache line corresponding to x_0 has more accesses than the others. Concurrently, the attacker repeatedly evicts these cache lines from all cache levels during the first round–either by flushing them (e.g., using clflush) [109] when the AES library is shared read-only, or by accessing eviction sets for these cache lines in the inclusive LLC [57] or the inclusive coherence directory [107]. For simplicity, we assume the attacker flushes these addresses from the caches in parallel with the victim's execution. Consequently, the DRAM row corresponding to x_0 experiences significantly more memory accesses than the other 15 DRAM rows containing the T-table.

Attack Mechanism. The attacker ensures the victim performs n encryption iterations with different plaintexts and the same secret key. In each iteration, the attacker ensures the victim accesses T-table entries during the first round. Subsequently, the attacker measures the activation counts of different DRAM rows using the ABO-based side channel. It sequentially activates each row (R_0, R_1, ... , R_{15}) once and repeats this in a loop until one of them triggers an ABO. The first row to trigger an ABO is identified as the most frequently activated row, revealing the cache line number within the T-table that had the highest access count. This discloses 4 of the 8 bits of x_0 (i.e., the cache line index). Since p_0 is known, the attacker recovers 4 of the 8 bits of the key byte k_0. Repeating this process while fixing each p_i (i going from 0 to 15) allows the attacker to recover 64 out of 128 bits of the secret AES key.

Results. Figure 4 shows activation counts for Row-0 and other rows in an attack instance where $p_0 = 0$ and $k_0 = 0$. During the victim execution phase, the victim causes 207 activations to Row-0. In the attacker observation phase, an increase in memory access latency is observed after 49 activations to Row-0 due to an ABO. This is when the combined victim and attacker activations cross N_{BO}= 256 for Row-0, the most heavily activated row. From this, the attacker learns the top four bits of k_0, the first key byte, are 0x0.
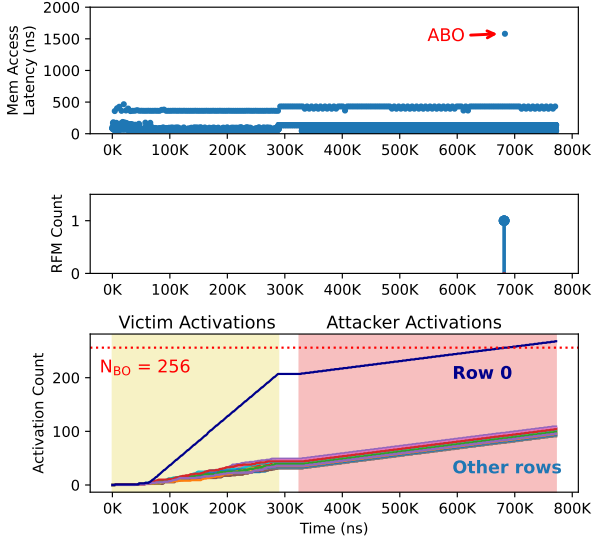
Figure 4: *PRACLeak* Side-Channel attack on AES T-tables for plaintext byte-0, $p_0 = 0$ with $k_0 = 0$. During victim activations, the Row-0 activation counts are approximately double those of the other rows. Thus, the attacker observing the high memory access latency of ABO due to activations on Row-0 learns that it is the most activated row.

Building on this, we generalize our approach to demonstrate that for all secret key bytes k_i, information can be leaked from the secret-dependent most-activated row using our *PRACLeak* side channel. Figure 5(a) shows a heatmap of the number of activations across the 16 DRAM rows containing the T-tables after the victim program runs for 200 encryptions. As k_0 varies from 0 to 256, the DRAM row with the highest number of activations shifts from Row-0 to Row-15. Figure 5(b) then shows the first row that triggers an ABO during the attacker's probing phase, along with its activation count. The total activations to this row–by both victim and attacker–sum to exactly 256, and the row index directly reveals the top 4 bits of key byte k_0. By varying the fixed plaintext byte p_i, the attacker can leak 4 of the 8 bits of each k_i, leaking 50% of the 128-bit AES key.
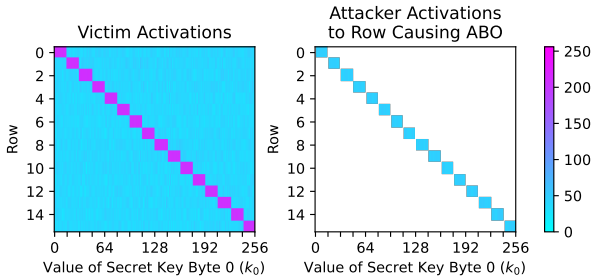

Figure 5: Side-Channel attack on AES T-tables for varying key byte-0 k_0 values, when plaintext byte-0 $p_0 = 0$. (a) Number of victim activations for DRAM rows after running 200 encryptions. (b) Number of attacker activations on row causing first ABO – index of this row leaks out the key bits.

4 Timing-Safe PRAC (TPRAC)

We propose *Timing-Safe PRAC (TPRAC)*, an *activity-independent* Refresh Management (RFM) mechanism that eliminates timing channels in PRAC-based systems. *TPRAC* employs our proposed Timing-Based RFM (TB-RFM), where the memory controller issues RFM All Bank (RFM_{ab}) commands at fixed intervals, denoted as TB-Window, independent of memory activity. This closes timing channels while maintaining performance balance.

Figure 6 illustrates how *TPRAC* mitigates PRAC-based timing channels by issuing TB-RFMs periodically at each TB-Window, configured to eliminate Alert Back-Off-triggered RFMs (ABO-RFMs). Section 4.2 discusses how to determine an appropriate TB-Window based on the worst-case attack pattern. *TPRAC* requires only a single register, the RFM Interval Register, to store TB-Window. At the end of each window, it proactively issues an RFM_{ab} (as a TB-RFM) regardless of memory activity, preventing attackers from triggering ABO-RFMs through targeted activations. *TPRAC* also disables RFM postponing [41] to prevent attacks that exploit delayed RFMs [61].
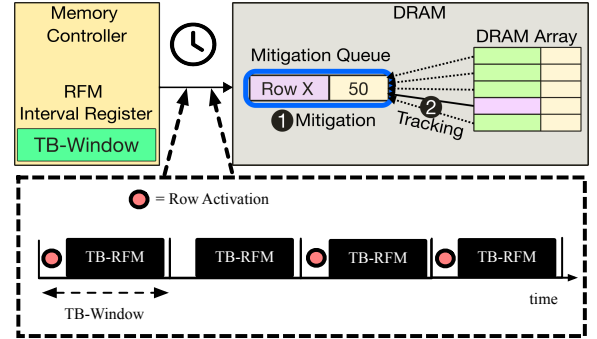
Figure 6: Overview of *TPRAC* design. It employs an activity-independent RFM mechanism (Timing-Based RFM) to eliminate timing channels in PRAC-based systems. These RFMs are issued with the fixed time windows called TB-Window.

4.1 Underlying PRAC Implementation

Our proposed TB-RFM mechanism relies on the underlying PRAC implementation to perform RowHammer (RH) mitigation during each RFM. However, the PRAC specification [41] does not define implementation details such as the mitigation queue design. Thus, we outline the potential queue designs suitable for *TPRAC*.

TPRAC mitigates the most frequently activated row in each DRAM bank during every RFM, similar to prior secure PRAC designs [10, 77, 102]. We propose a frequency-based mitigation queue with a *single-entry* per bank to achieve this practically. Each queue tracks the most highly activated row by storing its address and activation count, replacing the entry when a newly activated row exceeds the current counter (② in Figure 6). This ensures that the most activated row is always retained. Upon receiving an RFM, *TPRAC* mitigates the row in each bank's queue, regardless of whether it has crossed the Back-Off threshold (N_{BO}) (① in Figure 6).

Section 4.2.3 shows that TB-RFM, combined with this queue design, achieves security equivalent to the idealized PRAC design, UPRAC [8]. Similar queue designs are employed in secure PRAC implementations like QPRAC [102] and MOAT [77], making TB-RFM readily compatible with these designs.

4.2 Configuring the TB-Window Interval

To ensure that *TPRAC* effectively eliminates ABO-RFMs–and consequently Activation-Based RFMs (ACB-RFMs)–and prevents information leakage, we must determine an appropriate TB-Window interval for each RowHammer threshold (N_{RH}). Our analysis begins with the idealized PRAC implementation, UPRAC [8], which mitigates the most highly activated row in each DRAM bank during every RFM. We then show that *TPRAC* with our proposed single-entry mitigation queue can achieve equivalent security.

4.2.1 Worst-Case Attack Pattern and Assumptions. Following prior studies on RFM [9, 61], we adopt the Feinting [61] (also known as Wave [111]) attack as the basis for our worst-case analysis of *TPRAC*. This attack is mathematically proven to represent the worst-case access pattern for RFM-based mitigations. It proceeds in multiple rounds using a pool of decoy rows and a target row. The decoy rows help bypass RFMs, allowing the attacker to maximize activations on the target row. In each round, all rows in the pool are uniformly activated, and any mitigated rows are removed. This process continues until only the target row remains, enabling the adversary to concentrate all activations on it in the final round. For our security analysis of *TPRAC* under worst-case conditions, we make the following *conservative* assumptions:

- **Single ABO-RFM Leakage**: The adversary can leak information if even a single ABO-RFM is triggered.
- **Exclusive DRAM Access**: The adversary has exclusive access to the DRAM module used by the victim, eliminating interference or noise from other processes.
- **Full System Knowledge**: The adversary knows all details of *TPRAC* and the PRAC implementation, including the Back-Off threshold (N_{BO}), mitigation queue design, and TB-Window.
- **Precise Timing Measurement**: The adversary can accurately measure (1) the total number of RFMs, (2) the exact time at which each RFM is issued, and (3) the duration of each RFM-induced blocking period, by monitoring memory access latencies.

4.2.2 Determining TB-Window of TPRAC. The adversary aims to trigger an ABO-RFM by causing N_{BO} activations to a target row (T_{ACT}). To prevent information leakage, *TPRAC* must ensure this count stays below N_{BO}, as shown in the following equation:

$$T_{ACT} < N_{BO} \tag{1}$$

The adversary begins with an initial pool of R_1 rows. Since *TPRAC* performs one TB-RFM every TB-Window, the maximum number of row activations possible within this interval, denoted as $ACT_{TB\text{-}Window}$, is given by:

$$ACT_{TB\text{-}Window} = \frac{TB\text{-}Window}{tRC} \tag{2}$$

The number of remaining rows after the first $N-1$ attack rounds (R_N) is computed by subtracting the number of TB-RFMs issued during those rounds from the initial row pool size (R_1). Since one TB-RFM is issued for every $ACT_{TB\text{-}Window}$ activations, we have:

$$R_N = R_1 - \left\lfloor \frac{\sum_{i=1}^{N-1} R_i}{ACT_{TB\text{-}Window}} \right\rfloor, \quad \text{where } N \geq 2 \tag{3}$$

The total number of attack rounds (AR) can be computed from a given R_1 using Equation (3). Since the target row is activated once per round until the final round, and all activations in the final round ($ACT_{TB\text{-}Window}$) go to the target row, the total number of activations to the target row (T_{ACT}) is given by:

$$T_{ACT} = (AR - 1) + ACT_{TB\text{-}Window} \tag{4}$$

Lastly, we evaluate the worst-case number of activations to the target row (T_{MAX}) under two scenarios:

- **With Activation Counter Reset:** Per-row activation counters are reset at every refresh window (tREFW, typically 32ms in DDR5), as proposed in prior work [77]. Original Feinting attack shows that the worst-case occurs when R_1 equals the number of mitigations (i.e., TB-RFMs) that can be applied during the attack [61]. This gives the optimal R_1 value (OPT_{R_1}) as:

$$OPT_{R_1} = \frac{MAXACT_{tREFW}}{ACT_{TB\text{-}Window}} \tag{5}$$

where $MAXACT_{tREFW}$ (~550K) is the maximum number of activations allowed within one tREFW.

- **Without Activation Counter Reset:** Per-row activation counters are reset only when the row is mitigated via an RFM. In this case, we sweep R_1 from 1 up to its maximum possible value (128K for the DDR5 32Gb chip) to determine the value of OPT_{R_1} that maximizes the number of activations to the target row.

Figure 7 shows the theoretical maximum number of activations to the target row (T_{MAX}) across different TB-Window values, comparing scenarios with and without per-row activation counter reset at each tREFW. T_{MAX} is consistently higher without counter reset due to the larger optimal initial row pool size (OPT_{R_1}). With counter reset, OPT_{R_1} is constrained by how many TB-RFM intervals fit within a single tREFW window. For example, at TB-Window of 1 tREFI, only 8192 intervals are possible, limiting OPT_{R_1} to 8192–almost 16× smaller than the no-reset case (approximately 128K). As a result, T_{MAX} reaches 736 without counter reset, compared to 572 with reset. The impact of the reset becomes more pronounced at longer TB-Window intervals because fewer TB-RFMs fit within tREFW, further reducing OPT_{R_1}. For instance, at TB-Window of 4 tREFI, T_{MAX} reaches 3220 without reset and 2138 with reset. Conversely, at shorter intervals like 0.25 tREFI, the difference narrows, with T_{MAX} values of 118 and 105, respectively.

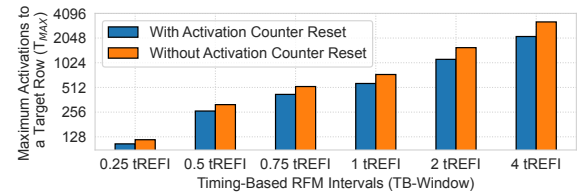**Figure 7: Theoretical maximum activations to a target row (T_{MAX}) as the TB-Window of *TPRAC* varies, shown for both with and without per-row activation counter reset at each refresh window (tREFW). Results are based on a DDR5 32Gb chip with 128K rows per bank. T_{MAX} must remain below the Back-Off threshold (N_{BO}) to avoid Alert Back-Off-triggered RFMs (ABO-RFMs) and close timing channels.**
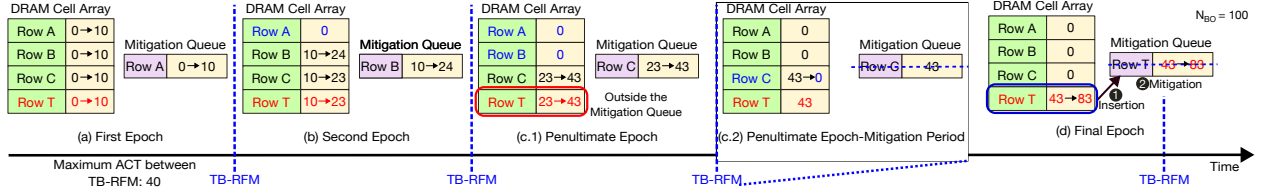
Figure 8: Example of how *TPRAC* prevents Feinting [61] or Wave [111] attack with a single-entry mitigation queue.

4.2.3 Impact of the Mitigation Queue Structure of PRAC. We demonstrate that a single-entry, frequency-based mitigation queue is sufficient to match the security guarantees of UPRAC. Figure 8 illustrates through a simple Feinting attack, where the attacker uses three decoy rows alongside a target row (Row T) to bypass TB-RFMs and build up activations on Row T. In this example, up to 40 activations are allowed within each TB-Window. During the first epoch, all four rows are activated uniformly, as shown by Figure 8(a). Row A, having the highest activation count, is tracked in the mitigation queue and then mitigated when the TB-RFM is issued. In the second epoch, the attacker continues activating Row T and the remaining decoys. Row B accumulates the most activations and is tracked and mitigated at the next TB-RFM, as shown by Figure 8(b).

Final Attack Round: At the end of the penultimate epoch (Figure 8(c.1)), both the target row (Row T) and a decoy row (Row C) have equal activation counts (i.e., 43). Since the mitigation queue holds only a single entry, only one row–Row C–is tracked and subsequently mitigated when the TB-RFM is issued, as shown by Figure 8(c.2). During this mitigation period, memory requests are blocked for 350ns, preventing the attacker from issuing further activations, including to Row T. In the final round (Figure 8(d)), the attacker focuses activations solely on Row T. As soon as Row T is activated, Row T becomes the most active row and is inserted into the mitigation queue (① in Figure 8). Before it reaches the Back-Off threshold ($N_{BO} = 100$ in this example), the next TB-RFM is issued, and Row T is mitigated (② in Figure 8), ensuring no ABO-RFM is triggered, consistent with our analysis in Section 4.2.

Notably, if the attacker had begun targeting Row T earlier, it would have entered the mitigation queue and been mitigated sooner, further reducing the attack's effectiveness. This behavior aligns with observations from the original Feinting attack [61].

Scenario 1 – Equal Activations: In this scenario, the attacker uniformly activates all rows in the initial pool across rounds, including rows that were mitigated in previous epochs. However, this strategy is suboptimal compared to the Feinting attack. Once a row is mitigated, its counter resets to zero (e.g., Row C in Figure 8(c.2)), so continuing to activate it reduces the total activations available for the target row. For instance, if both Row C and Row T are activated in Figure 8(d) rather than focusing solely on Row T, the target row receives only 20 activations instead of 40. This observation aligns with the findings from the original Feinting attack [61].

Scenario 2 – Delayed Activations: In this scenario, the attacker first activates decoy rows (e.g., Row B and Row C) to inflate their activation counts, ensuring they are mitigated when TB-RFMs are issued. The attacker then switches to intensively activating the target row (Row T) so it becomes the most activated and enters

the mitigation queue. If the same row pool size, number of decoy rows, and activation counts are used, this strategy effectively replicates the Feinting attack. However, it does not improve Feinting's effectiveness and remains secure under *TPRAC*.

Scenario 3 – Early and Aggressive Activations: In this scenario, the attacker begins by aggressively activating the target row (Row T), aiming to reach the Back-Off threshold (N_{BO}) before any mitigations occur. Row T quickly surpasses other rows in activation count and enters the mitigation queue. However, this strategy is inferior to the Feinting attack. Once Row T is mitigated by a TB-RFM, its counter resets, limiting the total number of activations to those between two TB-RFM events; this number is significantly lower than what the Feinting attack can achieve. For example, with one TB-RFM every tREFI, the attacker can induce at most 60 activations– approximately 12.3× and 9.5× fewer than the maximum derived in our analysis without (736) and with (572) counter reset, respectively (Figure 7).

Why a Single-Entry Queue is Sufficient: A single-entry, frequency-based mitigation queue is sufficient because it consistently tracks the most frequently activated row at any point in time, ensuring that the highest-risk row is mitigated during each TB-RFM. Since TB-RFMs are issued at fixed intervals, *independent* of memory access patterns, the attacker cannot manipulate mitigation timing through activation strategies. As a result, no row can accumulate enough activations to reach N_{BO}, effectively preventing all ABO-RFMs. Therefore, by combining a single-entry queue with appropriately configured TB-RFMs, *TPRAC* achieves security on par with idealized PRAC implementations (UPRAC). This design effectively thwarts a wide range of attack strategies, including balanced, delayed, and aggressive activations, ensuring robust protection even against worst-case patterns like the Feinting attack.

4.3 Designing TPRAC with Targeted Refreshes

TPRAC can leverage existing Targeted Refreshes (TREFs) to reduce mitigation overhead. Like TB-RFMs, each TREF is used to mitigate the most frequently activated row from the mitigation queue. This allows *TPRAC* to skip a scheduled TB-RFM if a TREF occurs within the same interval, improving performance without sacrificing security. For example, at N_{RH} of 512, *TPRAC* requires a TB-RFM approximately every 0.8 tREFI with counter reset, and every 0.6 tREFI without reset, to eliminate timing channels (Figure 7). If TREFs are issued once every 2 tREFI, *TPRAC* can skip a TB-RFM roughly every 2.5 tREFI (with reset) or 3.3 tREFI (without reset), reducing the total number of RFMs and improving performance.

5 Evaluation Methodology

Simulation Framework: We evaluate our design using Champ-Sim [24], a well-known trace-based, out-of-order processor simulator. We integrate ChampSim with the cycle-accurate DRAM simulator Ramulator2 [50, 60]. Table 3 summarizes our simulated system configuration, following prior work [71] that models the Intel Sunny Cove microarchitecture.

The baseline system features a 4-core out-of-order processor with a hashed perceptron branch predictor [42]. Each core has private L1 caches: a 32KB instruction cache and a 48KB data cache, with the data cache using the IP-stride prefetcher [13]. Each core also has a private 512KB L2 cache, and all cores share an 8MB last-level cache (LLC). The LLC supports 64 MSRHs per core and uses the SPP-PPF prefetcher [4] and SRRIP replacement policy [36][1]. The memory system comprises a single-channel, quad-rank DDR5 memory totaling 128GB of DRAM. The memory controller uses the Minimalist Open-Page (MOP) address mapping policy [43] and the First Ready First Come First Served (FR-FCFS) scheduler [83, 116], following prior work [9, 10, 102, 103]. The DDR5 memory is modeled as a 32Gb DDR5-8000B chip, with timing parameters and PRAC-specific adjustments based on the JEDEC specification [41].

Table 3: System Configuration

Out-Of-Order Cores	4 Cores, 4GHz, 6-issue width, 4-retire width
	352-entry ROB, hashed perceptron branch predictor [42]
L1 Instruction Cache	32KB, 8-way, 4 cycles
L1 Data Cache	48KB, 12-way, 5 cycles, IP-stride prefetcher [13]
L2 Cache	512KB, 8-way, 10 cycles
Last Level Cache (Shared)	8MB, 16-way, 20 cycles, SPP-PPF prefetcher [4], SRRIP [36]
Address Mapping	Minimalist Open-Page (MOP) [43]
Scheduling Policy	FR-FCFS [83, 116] with a cap of 4 [63]
Memory Type	32Gb DDR5-8000B
DRAM Organization	4 Bank x 8 Groups x 4 Ranks x 1 Channel
Rows Per Bank, Size	128K, 8KB
tRCD, tCL, tRAS [11]	16ns, 16ns, 16ns
tRP, tRTP, tWR, tRC	36ns, 5ns, 10ns, 52ns
tRFC, tREFI [64, 80]	410 ns, 3.9μs
tABO$_{ACT}$, tRFM$_{ab}$	180ns, 350ns

Evaluated Design: We evaluate the performance and energy impact of *TPRAC* by comparing it to a baseline PRAC-enabled DRAM without the Alert Back-Off (ABO) protocol. We extend Ramulator2 to model per-row activation counters, the ABO protocol, our proposed Timing-Based RFM (TB-RFM), and the mitigation queue design and strategy. The TB-RFM interval (TB-Window) is tuned for each RowHammer (RH) threshold (N$_{RH}$) to eliminate all ABO-triggered RFMs (ABO-RFMs). We compare *TPRAC* against the following baselines, which are insecure against timing channel attacks. As the underlying PRAC implementation, we use QPRAC [102], one of the existing secure PRAC designs [10, 77, 102]: 1) **ABO-Only**, which relies solely on the ABO protocol for RH mitigations, and 2) **ABO+ACB-RFM**, which incorporates proactive Activation-Based RFMs (ACB-RFMs) to eliminate ABO-RFMs, which is the current RFM implementation defined in the JEDEC standard [41]. For each N$_{RH}$, we configure the Bank Activation threshold (BAT) to eliminate ABO-RFMs under the worst-case Feinting [61] (or Wave [111]) attack pattern.

---
[1]We evaluated *TPRAC* with various branch predictor and prefetcher combinations in ChampSim. The result variance is within 1%, showing that our mitigation is insensitive to different microarchitectural policies.

Table 4: Workload Categorization Based on RBMPKI

RBMPKI	Workloads
High [10+]	nutch, cassandra, classification, 433.milc, cloud9, 410.bwaves, 470.lbm, 471.omnetpp, 483.xalancbmk, 519.lbm, 520.omnetpp, 649.fotonik3d, 450.soplex, 619.lbm, 429.mcf, 654.roms, 470.lbm, 483.xalancbmk, 471.omnetpp, 605.mcf, 482.sphinx3, 437.leslie3d, 627.cam4, 620.omnetpp, 628.pop2, 607.cactuBSSN, 436.cactusADM, 459.GemsFDTD
Medium [1, 10)	401.bzip2, 657.xz, 602.gcc, 473.astar, 623.xalancbmk, 464.h264ref, 481.wrf
Low [0, 1)	631.deepsjeng, 458.sjeng, 456.hmmer, 625.x264, 403.gcc, 444.namd, 603.bwaves, 456.hmmer, 464.h264ref, 638.imagick, 644.nab, 481.wrf, 600.perlbench, 621.wrf, 465.tonto, 447.dealII, 435.gromacs, 641.leela, 454.calculix, 445.gobmk, 453.povray, 416.gamess, 648.exchange2

Workloads: We evaluate 50 publicly available workloads from SPEC2006 [17], SPEC2017 [95], and CloudSuite [21] benchmarks. Workloads are categorized into three memory-intensity groups based on their row-buffer misses per kilo instructions (RBMPKI): High (H) for RBMPKI $\geq$ 10, Medium (M) for $1 \leq$ RBMPKI $<$ 10, and Low (L) for RBMPKI $<$ 1, as shown in Table 4.

We simulate four-core homogeneous workloads for SPEC2006 and SPEC2017, while each core in CloudSuite runs a distinct thread per workload. For SPEC workloads, each core executes 250 million instructions, comprising a 50M warm-up followed by 200M instructions used for performance measurement. For CloudSuite, each core executes 25M warm-up followed by 125M measured instructions.

By default, we set N$_{RH}$ to 1024 with one RFM per Alert (PRAC-1) and enable activation counter resets at every tREFW. Additionally, we analyze sensitivity to N$_{RH}$ values from 128 to 4096, PRAC levels with 1, 2, or 4 RFMs per Alert, targeted refresh rates varying from once every 1 to 4 tREFIs, and activation counter reset policies. Performance is measured using weighted speedup.

6 Results and Analysis

6.1 Empirical Security Validation

To empirically demonstrate the effectiveness of TPRAC's mitigation capability, Figure 9 shows the results of the *PRACLeak* Side-Channel attack on AES T-tables (described in Section 3.3), both with and
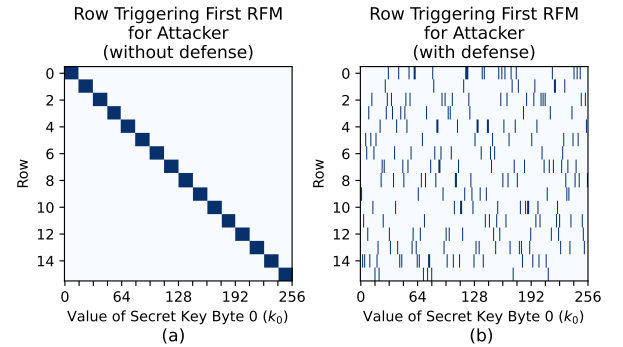

Figure 9: Row triggering the first RFM for the attacker. Without our *TPRAC* defense, the row leaks the secret key value, whereas with the defense, the triggered row is random and does not reveal any secret information.
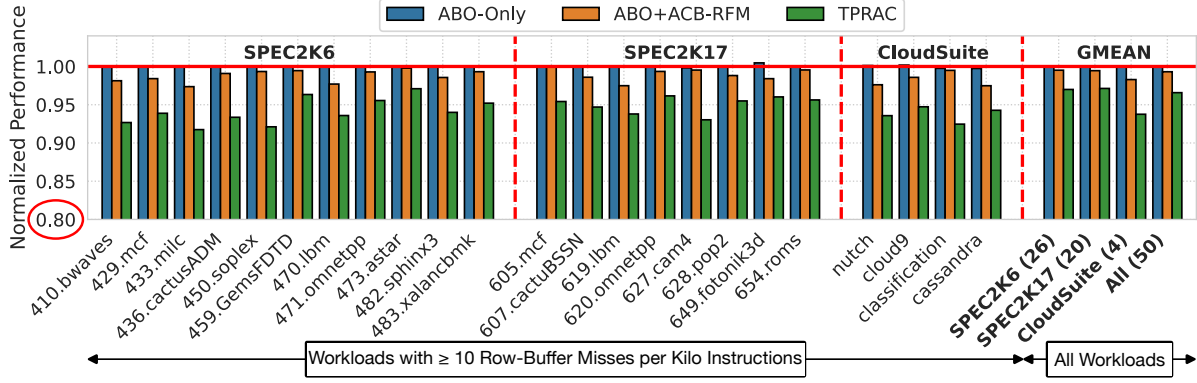
Figure 10: Normalized performance of *TPRAC* compared to insecure baselines, ABO-Only and ABO+ACB-RFM, at a RowHammer threshold of 1024. *TPRAC* incurs a slowdown of only 3.4% on average due to periodic Timing-Based RFMs (TB-RFMs). In contrast, ABO+ACB-RFM and ABO-Only exhibit slowdowns of 0.7% and near zero, respectively, due to their lower RFM frequency resulting from their activity-dependent design. However, this activity-dependent RFM method is not secure.

without our *TPRAC* defense. The figure shows that the DRAM row that triggers the first RFM during the attacker's probing phase reveals the most heavily activated row during the victim execution, which is dependent on the secret key.

In Figure 9(a), without the defense, there is a strong correlation between the value of key byte k_0 and the row triggering the first RFM, clearly leaking k_0. In contrast, Figure 9(b) shows that with *TPRAC*, the row triggering the first RFM appears random, leaking no information to the attacker. This is due to two reasons: (1) *TPRAC*'s periodic Timing-Based RFMs (TB-RFMs) proactively mitigate the most activated rows, fully preventing ABO-RFMs; and (2) even if the TB-RFM interval (TB-Window) is misconfigured, the attacker cannot distinguish TB-RFMs from ABO-RFMs, as both incur the same 350ns latency. As a result, no secret information is leaked through memory latency variations.

6.2 Performance Overhead

Figure 10 shows the performance of *TPRAC* at a RowHammer threshold (N_{RH}) of 1024, compared to two insecure baselines: ABO-Only and ABO+ACB-RFM. Performance is normalized to a PRAC-enabled DDR5 system without the ABO protocol. *TPRAC* incurs only a 3.4% average slowdown, as issuing one Timing-Based RFM (TB-RFM) every 1.6 tREFI (6.2μs) is sufficient to eliminate all ABO-RFMs. The primary source of overhead is reduced DRAM bandwidth–each TB-RFM block all banks for 350ns (tRFM$_{ab}$) every 6.2μs, leading to a maximum DRAM bandwidth loss of 5.6% $\left(\frac{tRFM_{ab}}{TB\text{-}Window}\right)$. As a result, several memory-intensive workloads experience slowdowns near or slightly above 6%.

Another source of slowdown is reduced row-buffer locality. Since each TB-RFM requires closing all open rows to issue an RFM$_{ab}$, row-buffer hit rates may decline. On average, *TPRAC* increases row-buffer misses by only 0.5%, with negligible impact on most workloads. However, in *433.milc*, a 2.3% increase in misses, combined with bandwidth loss, results in the highest slowdown of 8.3%.

ABO+ACB-RFM incurs only a 0.7% slowdown, as it triggers Activation-Based RFMs (ACB-RFMs) only when per-bank activations reach the Bank Activation threshold (BAT), minimizing the

bandwidth loss. ABO-Only shows almost no slowdown because ABO-RFMs are rare at N_{RH} of 1024. These results are consistent with recent works [10, 77, 102]. However, both ABO-Only and ABO+ACB-RFM remain vulnerable to timing channel attacks due to their *activity-dependent* mitigation strategies.

6.3 Sensitivity to PRAC Levels

Figure 11 compares the performance of *TPRAC* and insecure baselines, ABO-Only and ABO+ACT-RFM, as the PRAC level–the number of RFMs per ABO–varies at N_{RH} of 1024. While higher PRAC level allows DRAM to mitigate more rows per ABO, they also increase DRAM blocking time, which can impact both performance and security when ABO-RFM occurs. The results show that PRAC level variation has no impact on the performance of *TPRAC* and ABO+ACB-RFM, since both designs eliminate all ABO-RFMs. *TPRAC* achieves this through periodic Timing-Based RFMs (TB-RFMs), while ABO+ACB-RFM uses Activation-Based RFMs (ACB-RFMs). Across all PRAC levels, *TPRAC* and ABO+ACB-RFM incur slowdowns of 3.4% and 0.7%, respectively. ABO-Only experiences almost no ABO-RFMs at this threshold, resulting in near-zero slowdown.

Figure 11: Performance comparison of *TPRAC* and insecure baselines, ABO-Only and ABO+ACB-RFM, across different PRAC levels at a RowHammer threshold of 1024. Performance remains consistent across PRAC levels. *TPRAC* and ABO+ACB-RFM incur slowdowns of 3.4% and 0.7%, respectively, due to proactive RFMs reducing DRAM bandwidth. In contrast, ABO-Only shows almost no overhead, as ABO-RFMs are rare in benign applications at this threshold.

6.4 Sensitivity to Targeted Refreshes

Figure 12 shows the performance of *TPRAC* as the frequency of Targeted Refreshes (TREFs) varies from once every four tREFI to once per tREFI at a RowHammer threshold (N_{RH}) of 1024. Increasing TREF frequency improves performance since it allows *TPRAC* to skip more Timing-Based RFMs (TB-RFMs). Without TREF, *TPRAC* incurs a 3.4% slowdown. The slowdown drops to 2.4%, 2.0%, and 1.4% with one TREF every 4, 3, and 2 tREFIs, respectively. At one TREF per tREFI, *TPRAC* incurs no slowdown, as TREFs fully replace TB-RFMs, eliminating ABO-RFMs and preventing timing channels. As discussed in Section 6.5, this approach enhances the practicality of *TPRAC* at lower N_{RH} by reducing performance slowdowns.
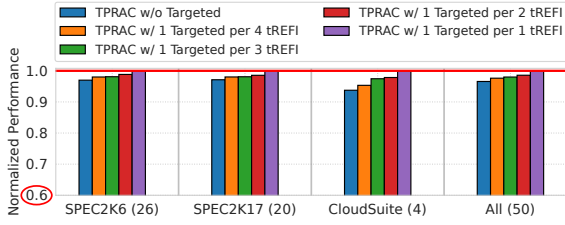

Figure 12: Normalized performance of *TPRAC* as the frequency of Targeted Refreshes (TREFs) varies at N_{RH} of 1024. More frequent TREFs reduce slowdowns by reducing the need for Timing-Based RFMs. *TPRAC* incurs slowdowns of 3.4%, 2.4%, 2.0%, and 1.4% with one TREF per 4, 3, and 2 tREFI, respectively, and no overhead with one TREF per tREFI.

6.5 Sensitivity to RowHammer Threshold

Figure 13 shows the performance of *TPRAC* as N_{RH} varies from 128 to 4096. At N_{RH} of 4096, 2048, and 1024, *TPRAC* incurs slowdowns of 0.6%, 1.6%, and 3.4%, respectively. Even at an ultra-low N_{RH} of 512, it maintains a moderate overhead of 6.5%. However, with lower N_{RH} of 256 and 128, the slowdown rises to 14.1% and 22.6%, respectively, due to 4.3× and 6.4× more frequent TB-RFMs needed to eliminate ABO-RFMs and prevent timing channel attacks.

Co-designing *TPRAC* with Targeted Refreshes (TREFs) improves performance across all evaluated N_{RH} values, as it can skip some
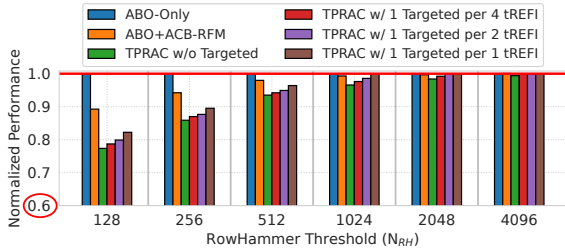

Figure 13: Performance comparison of *TPRAC* and insecure baselines, ABO-Only and ABO+ACB-RFM, as N_{RH} varies. *TPRAC* incurs 3.4%, 1.6%, and 0.6% slowdown at N_{RH} of 1024, 2048, and 4096, respectively. However, slowdowns increase to 6.5%, 14.1%, and 22.6% for N_{RH} of 512, 256, and 128, respectively, due to more frequent Timing-Based RFMs (TB-RFMs). Co-designing *TPRAC* with Targeted Refreshes improves performance by allowing some TB-RFMs to be skipped.

TB-RFMs by leveraging TREFs. At N_{RH} of 256, this reduce the slowdown from 14.1% (Without TREF) to 13.0%, 12.3%, and 10.5% with one TREF every 4, 2, and 1 TREF, respectively.

ABO+ACB-RFM shows similar trends, with slowdowns increasing at lower N_{RH} due to more frequent ACB-RFMs issued to eliminate ABO-RFMs. Its overhead remains lower than *TPRAC*, incurring just 0.7% at N_{RH} of 1024 and 10.7% at 128. However, it remains vulnerable to timing channel attacks, and the vulnerability worsens as N_{RH} decreases because more frequent ACB-RFMs increase the chance of information leakage. ABO-Only incurs negligible slowdowns across all thresholds, since ABO-RFMs are rare in benign applications. Nonetheless, like ABO+ACB-RFM, it is insecure against timing channels, with leakage becoming more pronounced at lower N_{RH}, where fewer activations are needed to trigger ABO-RFMs.

6.6 Sensitivity to Activation Counter Reset

Figure 14 shows the performance of *TPRAC* with and without activation counter reset as N_{RH} varies from 128 to 4096. At N_{RH} of 1024 and above, the impact of counter reset is negligible, with performance differences under 1%. However, at ultra-low N_{RH} ($N_{RH} \leq$ 512), resetting counters every tREFW improves performance. For instance, at N_{RH} of 128 without TREF, *TPRAC* without counter reset incurs a 73.9% slowdown, compared to 77.4% with reset, resulting in a 3.4% performance improvement. This improvement arises from a lower TB-RFM frequency, as counter resets reduce the attacker's initial row pool size (see Section 4.2). With fewer TB-RFMs, DRAM suffers less bandwidth loss, resulting in better overall performance.
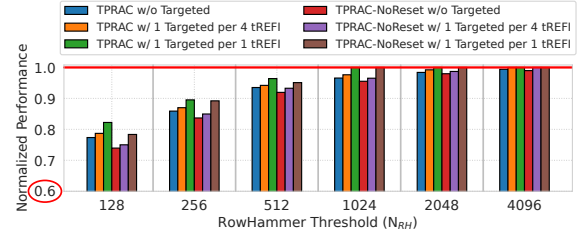

Figure 14: Normalized performance of *TPRAC* with and without activation counter reset as N_{RH} varies from 128 to 4096. Resetting counters at each tREFW has negligible impact (< 1%) at $N_{RH} \geq$ 1024, but offers noticeable performance benefits at lower thresholds due to reduced TB-RFM frequency.

6.7 Energy Overhead

Table 5 presents the energy overhead of *TPRAC*. Our TB-RFM introduces two main sources of overhead. First, each TB-RFM triggers mitigations for the most heavily activated row in the mitigation queue, consisting of *four* victim refreshes and *one* aggressor activation to reset the counter, totaling *five* additional activations per TB-RFM. Second, TB-RFMs increase overall execution time, contributing to what we refer to as non-mitigation energy overhead.

At N_{RH} of 1024, *TPRAC* incurs an average energy overhead of 7.4%, primarily due to increased execution time. As N_{RH} decreases, more frequent TB-RFMs are required, increasing the overhead. For example, at N_{RH} of 4096, TB-RFMs are issued approximately once every seven TREF, leading to only 1% overhead. In contrast, at ultra-low N_{RH} of 128, TB-RFMs are issued every $1\mu s$−nearly 29× more often−resulting in significant 44.3% overhead.

Notably, the JEDEC DDR5 specification defines the minimum Bank Activation threshold (BAT) for ACB-RFM as 16 [41], which allows issuing one ACB-RFM every $0.8\mu s$. This rate is similar to the $1\mu s$ TB-RFM interval at N_{RH} of 128. As a result, the worst-case energy overhead of *TPRAC* aligns with what is already permissible in the standard. Future work can explore *TPRAC* to reduce energy impact at ultra-low N_{RH}.

Table 5: Energy Overhead of *TPRAC*

N_{RH}	Mitigation (RFM)	Non-Mitigation (Execution Time)	Total
128	18.1%	26.2%	44.3%
256	11.3%	14.8%	26.1%
512	5.9%	4.5%	10.4%
1024	**2%**	**5.4%**	**7.4%**
2048	1%	1.6%	2.6%
4096	0.45%	0.55%	1%

6.8 Storage Overhead

TPRAC introduces minimal storage overhead by requiring only a single register per memory controller to store the interval for issuing periodic Timing-Based RFMs (TB-RFMs). This RFM interval register uses 24 bits (3 bytes), which is sufficient to represent intervals up to approximately half of the tREFW duration.

7 Alternative Timing Channel Mitigations

7.1 Obfuscation-Based Defenses

An alternative to mitigating PRAC-induced timing channels is to obfuscate the attacker by injecting random noise into memory access latency. The key idea is to introduce timing delays that resemble legitimate ABO-RFMs (350ns), making it harder for attackers to infer sensitive information. This can be achieved by having either the memory controller issue random RFM_{ab}s or the DRAM probabilistically assert the Alert signals. While this approach does not eliminate ABO-RFMs like *TPRAC*, it is effective against simple channels that rely solely on memory access latency variations.

An attacker could develop more sophisticated strategies beyond simple latency monitoring. For example, they might profile RFM count distributions over extended periods (e.g., hours or days), collecting data from windows with and without memory activity. The attacker could attempt to infer transmitted bits or secret-dependent behavior by training statistical or machine learning classifiers on this data. If random RFMs are injected with 50% probability per tREFI, the attacker may typically observe around 4096 RFMs within a tREFW window during idle periods and higher counts during active ones. In extreme cases, observing zero RFMs can definitively indicate a Bit-0 (covert channel) or absence of victim activity (side channel), while counts exceeding 8192 confirm a Bit-1 or victim activity. Nonetheless, the distribution overlap introduced by randomization reduces the reliability of such inferences.

This probabilistic approach offers a flexible trade-off between security and performance. While it does not eliminate all information leakage, it reduces attacker precision, making it suitable in scenarios where occasional leakage is acceptable. It is also appealing for ultra-low N_{RH} values, where *TPRAC* incurs higher performance overhead

due to frequent TB-RFMs. Future work will further explore how such techniques can complement or enhance *TPRAC*.

7.2 Bank Partitioning with Per-Bank RFM

DRAM bank partitioning, where each bank is exclusively assigned to a specific user or process (e.g., per-VM), can provide spatial isolation and prevent inter-bank interference [58, 92]. While this can block some PRAC-based timing channels, such as our activation-count-based channel, it is insufficient to prevent all channels, including our activity-based channel. This is because the PRAC specification only allows memory controllers to issue an RFM All Bank (RFM_{ab}), which blocks the entire DRAM channel for 350ns [41]. Consequently, an attacker in a separate bank on the same DRAM module can still observe latency spikes and infer victim activity.

One way to close these timing channels under bank partitioning is to extend the ABO protocol to support a fine-grained Per-Bank RFM (RFM_{pb}) command, which mitigates a specific bank without stalling others. However, this requires changes to the current PRAC specification, such as enabling DRAM to report to the memory controller which bank triggered the Alert signal. In contrast, *TPRAC* eliminates all PRAC-induced timing channels without requiring such modifications or bank partitioning.

RFM_{pb} could also benefit *TPRAC* by allowing it to issue Per-Bank TB-RFM, further reducing DRAM bandwidth loss. We leave the design and evaluation of this extension to future work[2].

7.3 Hiding RFM-Induced Latency Variations

Another approach to mitigating PRAC's timing channels is to hide or eliminate the observable latency of RowHammer (RH) mitigation caused by Alert Back-Off-triggered RFMs (ABO-RFMs). For instance, this can be done by increasing the frequency of periodic retention refreshes and borrowing time from them to perform necessary mitigations, similar to prior Target Row Refresh (TRR). Instead of issuing *TPRAC*'s Timing-Based RFMs (TB-RFMs), this method uses additional refreshes to track and mitigate the most frequently activated rows using an in-DRAM mitigation queue and mitigate them, following a mitigation strategy of *TPRAC*.

However, this approach has key limitations: (1) it requires changes to JEDEC specifications, (2) incurs higher performance overhead due to longer blocking time (410ns for Refresh vs. 350ns for RFM), (3) must be conservatively configured for worst-case N_{RH} due to aging and spatial variation [72, 112], and (4) complicates DRAM refresh logic by repurposing refreshes for both retention and RH mitigation. In contrast, *TPRAC* uses JEDEC-defined RFM commands, offering a more efficient and standards-compliant solution.

8 Related Work

8.1 Timing Side Channels

DRAM Timing Channels. DRAMA [75] proposes a side channel due to timing variation between row-buffer hits and conflicts. While this leaks the last row or bank accessed, our PRAC-based attacks can leak the history of row accesses, leaking the activation counts for any row accessed in a given time window.

---

[2]QPRAC [102] also proposes to incorporate RFM_{pb} as part of the ABO protocol to address Denial-of-Service risks. In contrast, our focus is on closing timing channels.

Other works [18, 92, 99, 100, 115] discuss side channels due to memory-scheduling decisions in shared memory controllers, and propose spatial [92, 100] or temporal partitioning [99], or memory traffic shaping [18, 115] to prevent leakage. Such defenses cannot mitigate all PRAC-based timing channels, as the RFM_{ab} command used for mitigation in PRAC stalls the entire DRAM channel, causing delays for all memory accesses despite such defenses. Our solution, TPRAC, closes all PRAC-based timing channels.

Security-Metadata based Timing Channels. Security metadata can introduce timing side channels. MetaLeak [14] showed that Intel SGX's Merkle trees leak timing information. For instance, victim accesses trigger tree traversals that slow an attacker's traversals. Similarly, split counters [84, 85, 106] expose a victim's access counts via shared counter cache lines and overflow behavior. Our attacks also leverage activity-based and counter-based channels, but target PRAC-based row-activation counters. Unlike integrity trees and counter-based encryption [114], which are now obsolete in new variants of SGX (SGX-v2) [33], PRAC is added to the DDR5 DRAM specification and is likely to be widely adopted. Thus, our attacks are more widely applicable.

Timing Channels Due to Performance Optimizations. Most timing channels are due to timing variations introduced by performance optimizations. Among these, cache timing channels [3, 28, 57, 109] have been the most well-explored, with the root cause being timing variations due to cache hits and misses. Recently, cache performance optimizations like prefetchers [12, 93, 97] and way-predictors [55] have also been shown to cause timing variations and side channels. Moreover, processor front-end optimizations like branch prediction [19] and speculative execution [51, 56], critical for performance, have also been shown to be vulnerable to side channels. Unlike these side channels, which can be mitigated by disabling performance optimizations for sensitive computations (e.g., using fences or uncached memory), our PRAC-based side channels cannot be mitigated by disabling activation counters, as that can lead to vulnerability to RowHammer.

8.2 Alternative RowHammer Defenses

A. TRR-Based Defenses: TRR-based defenses mitigate RowHammer (RH) by borrowing time from periodic refreshes to perform mitigation, either probabilistically [32, 48] or based on counters tracking frequently activated rows [30]. While these approaches avoid additional latency and are generally immune to timing channels, they have been shown to be ineffective against modern RH attacks [22, 30, 34, 38, 39, 52]. In contrast, TPRAC remains secure against RH attacks while also eliminating timing channels.

B. RFM-Based In-DRAM Defenses: These in-DRAM defenses perform RH mitigations using Activation-Based RFMs (ACB-RFMs). Upon receiving an RFM, some approaches apply mitigation probabilistically [35, 78], while others use tracking structures to determine which rows to mitigate [47, 61]. Although effective against RH, these techniques are vulnerable to timing channels because they rely on activity-dependent ACB-RFMs. In contrast, TPRAC employs activity-independent Timing-Based RFMs (TB-RFMs) that are both secure against RH and eliminate timing leakage.

C. Defenses with DRAM Chip and Interface Modifications: Several prior works propose redesigning the DRAM interface or

chip internals to support concurrent mitigations or refreshes during activations [62, 110]. Because they avoid introducing explicit mitigation delays, these designs are resistant to timing channels. However, they require substantial modifications to both DRAM chips and the interface protocol, making deployment difficult. In contrast, TPRAC requires no such modifications and achieves timing-channel prevention based on the PRAC specification.

D. RFM-Based Host-Side Defenses: Host-side RH defenses mitigate attacks by issuing Directed RFM (DRFM) commands [41] to refresh neighboring rows, either probabilistically [44, 46, 49, 94, 113] or using trackers [6, 54, 73, 74, 79, 88, 91]. Tracking-based defenses monitor potential aggressors and issue DRFMs when activity crosses a threshold; this activity-dependent strategy introduces timing leakage. Probabilistic approaches issue DRFMs with a fixed probability on each activation, reducing susceptibility to timing channels, but failing to protect against RH at low N_{RH} [35].

E. Alternative Mitigative Actions: Row migration [29, 86, 87, 89, 101, 104] and access throttling techniques [9, 105] mitigate RH by relocating or slowing access to frequently activated rows. However, these methods introduce noticeable latency variations, making them susceptible to timing channels. ECC-based defenses [2, 20, 65, 67–70] provide partial defense for RH but do not address timing leakage. In contrast, TPRAC is built on PRAC, preserving its RowHammer protection while eliminating timing channels.

9 Conclusions

This paper uncovers a critical security vulnerability in PRAC-based RowHammer mitigations, stemming from timing channels introduced by the Alert Back-Off mechanism and Refresh Management (RFM) commands. We present an attack called *PRACLeak*, which exploits these timing channels to create covert and side channels. This enables an adversary sharing a DRAM row with a victim to leak sensitive information – such as secret keys from vulnerable AES implementations – by monitoring memory access latencies.

To address this issue, we propose Timing-Safe PRAC (*TPRAC*), an activity-independent RFM mechanism that eliminates timing channels without compromising RowHammer mitigation. *TPRAC* introduces Timing-Based RFMs that are issued periodically and independently of access patterns to eliminate timing leakage. Our evaluation shows that *TPRAC* achieves robust security using only a single-entry per-bank mitigation queue and incurs just 3.4% performance overhead at a RowHammer threshold of 1024.

Appendix A   Artifact Appendix

A.1   Abstract

This artifact supports two key aspects of the paper: (1) the proposed *PRACLeak* covert and side channel attacks, which exploit timing variations introduced by PRAC's mitigation actions, and (2) our defense, *TPRAC*, which uses activity-independent Timing-Based RFMs (TB-RFMs) to eliminate PRAC-induced timing channels.

For the *PRACLeak* evaluations, we provide the Ramulator2 [60] source code with modifications to support `clflush` along with required traces for experiments. We also include Bash and Python scripts to generate victim traces for AES T-Table encryption using the Ramulator trace generator, as well as the corresponding sender and receiver traces. Additional scripts are provided to run these traces in Ramulator2 and to reproduce the results shown in Figure 3 through Figure 5 and Figure 9.

For the *TPRAC* evaluations, we provide Python scripts for the security analysis, allowing users to evaluate the maximum number of activations to a row as the TB-RFM interval (TB-Window) varies and regenerate Figure 7. For performance evaluations, we include the C++ implementation of the integrated ChampSim and Ramulator2 framework, with the *TPRAC* implementation within Ramulator2. We also provide all evaluated workload traces, along with Bash and Python scripts to run experiments and automate result collation and plot generation (Figure 10 to Figure 14).

A.2   Artifact Check-List (Meta-Information)

A.2.1   *PRACLeak Evaluations.*

- **Program:** C++ implementations of Ramulator2 with modifications to support `clflush`. Bash scripts and Python scripts to generate program traces, run experiments, and produce plots.
- **Compilation:** Tested with g++ 11.4.0; should also compile with any C++20-compliant compiler.
- **Run-time environment:** Tested on Ubuntu 22.04; should run on any Linux distribution with a valid Python3 interpreter.
- **Hardware:** Any modern desktop/laptop suffices.
- **Output:** ABO timing variation: Figure 3; *PRACLeak* on AES T-Table Encryption: Figure 4, Figure 5, Figure 9
- **Experiments:** Instructions for running experiments and parsing results are available in the provided README file.
- **How much disk space required (approximately)?:** ∼ 20 GB.
- **How much time is needed to prepare workflow (approximately)?:** Under 5 minutes to install the dependencies.
- **How much time is needed to complete experiments (approximately)?:** ∼ 1 hour.
- **Publicly available?:** Yes. GitHub: https://github.com/STAR-Laboratory/PRAC_TC_ISCA25. Traces: https://doi.org/10.5281/zenodo.15104055
- **Archived (provide DOI)?:** https://doi.org/10.5281/zenodo.15104037

A.2.2   *TPRAC Evaluations.*

- **Program:** C++ implementation of the integrated ChampSim–Ramulator2 framework, including the *TPRAC* implementation within Ramulator2. Accompanied by Bash and Python scripts for running experiments, aggregating results, and generating plots.
- **Compilation:** We highly recommend using g++ 12 or newer, as it significantly reduces the compilation time of ChampSim binaries. Tested with g++ 11 and 12; should also compile with any C++20-compliant compiler.

- **Run-time environment:** We recommend using a modern Linux distribution with support for C++20. For example, Ubuntu 22.04 or later is recommended if you prefer Ubuntu. This artifact has been tested on Ubuntu 22.04 and Rocky Linux 9.4.
- **Metrics:** Weighted Speedup.
- **Output:** *TPRAC* security analysis: Figure 7; *TPRAC* performance results: Figure 10 to Figure 14.
- **Experiments:** Instructions for running experiments and parsing results are available in the provided README file.
- **How much disk space required (approximately)?:** ≈ 40GB.
- **How much time is needed to prepare workflow (approximately)?:** Under an hour to install the dependencies, download the traces, and build all ChampSim binaries.
- **How much time is needed to complete experiments (approximately)?:**
  - ≈ 1 day (on a cluster with 1000 cores) and ≈ 2 days (on an Intel Xeon CPU with 40 cores and 128GB memory) for the security analysis (Figure 7) and main performance result (Figure 10).
  - ≈ 2 days (on a cluster with 1000 cores) and ≈ 1 week (on an Intel Xeon CPU with 40 cores and 128GB memory) for all performance experiments (Figure 10 to Figure 14).
- **Publicly available?:** Yes. GitHub: https://github.com/STAR-Laboratory/PRAC_TC_ISCA25. Traces: https://doi.org/10.5281/zenodo.15104055
- **Archived (provide DOI)?:** https://doi.org/10.5281/zenodo.15104037

A.3   Description

A.3.1   *How to access.* The artifact is available at https://github.com/STAR-Laboratory/PRAC_TC_ISCA25.

A.3.2   *Hardware Recommendations.*

- ***PRACLeak* Evaluation:** Any modern desktop/laptop should suffice. A laptop with a 2-core CPU and 16GB of memory can perform *PRACLeak* analysis within ∼ 1 hour.
- ***TPRAC* Evaluation:** We strongly recommend using Slurm with a cluster capable of running bulk experiments to accelerate evaluations. If using a personal server, we recommend a machine with at least 40 hardware threads with 128GB of memory to run all evaluations in a reasonable time.

A.3.3   *Software Requirements.*

- g++ with C++20 support (tested with g++ 11 and 12). We highly recommend using g++ 12 or newer, as it significantly reduces the compilation time of ChampSim binaries.
- Python3 (tested with version 3.9 and 3.10).

A.3.4   *Traces.*

- ***PRACLeak* Evaluation:** Our program generates 256 traces for the AES T-Table Encryption program, as well as other supportive traces for covert and side channel evaluations. Traces are available for download at https://doi.org/10.5281/zenodo.15104055
- ***TPRAC* Evaluation:** We use 50 traces from SPEC2006, SPEC2017, and CloudSuite. Traces are available for download at https://doi.org/10.5281/zenodo.15104055.

A.4   Installation and Experiment Workflow

First, clone the GitHub repository:

git clone https://github.com/STAR-Laboratory/PRAC_TC_ISCA25

A.4.1 PRACLeak Evaluation. No additional setup is required if dependencies are satisfied. To start the experiment:

```
cd PRAC_TC_ISCA25/PRACLeak
./run_artifact.sh
```

Alternatively, to avoid regenerating traces and use the provided sample data:

```
cd PRAC_TC_ISCA25/PRACLeak
./run_artifact.sh --use_sample
```

A.4.2 TPRAC Evaluations. 1. Setup required library paths:

```
cd PRAC_TC_ISCA25/TPRAC
source setup_lib_path.sh
```

2. Configure the following parameters in PRAC_TC_ISCA25/TPRAC/run_artifact.sh:

- Using Slurm:
  - SLRUM_PART_NAME: Partition name for Slurm jobs.
  - SLRUM_PART_DEF_MEM: Default memory size for jobs (recommended: ≥ 6GB).
  - MAX_SLRUM_JOBS: Maximum number of Slurm jobs to submit.
- Using a Personal Server:
  - PERSONAL_RUN_THREADS: Number of parallel threads to use for simulations.

3. Run the following commands to install dependencies, build ChampSim and Ramulator2, and execute simulations. If using a personal server, we recommend first running the security analysis (Figure 7) and main performance experiment (Figure 10) and reviewing the results before proceeding with the full set of experiments.

3.1. Running the security analysis (Figure 7) and main performance experiment (Figure 10):
Using Slurm:

```
cd PRAC_TC_ISCA25/TPRAC
./run_artifact.sh --method slurm --artifact main
```

Using a Personal Server:

```
cd PRAC_TC_ISCA25/TPRAC
./run_artifact.sh --method personal --artifact main
```

3.2. Running the security analysis (Figure 7) and all performance experiments (Figure 10 to Figure 14):
Using Slurm:

```
cd PRAC_TC_ISCA25/TPRAC
./run_artifact.sh --method slurm --artifact all
```

Using a Personal Server:

```
cd PRAC_TC_ISCA25/TPRAC
./run_artifact.sh --method personal --artifact all
```

A.5 Evaluation and Expected Results

A.5.1 PRACLeak Evaluations. We provide run_artifact.sh to demonstrate *PRACLeak* through 5 experiments. Scripts to run them can be found in PRAC_TC_ISCA25/PRACLeak/scripts/sim_scripts,

including run_latency.sh, run_side_channel.sh, run_covert_channel.sh, run_aes_no_defense.sh, and run_aes_with_defense.sh. After the experiments, raw data will be stored in ./PRACLeak/results/raw.

Moreover, we provide plot_all_figures.sh in PRAC_TC_ISCA25/PRACLeak to generate Figure 3 through Figure 5 and Figure 9. Generated plots will be stored as PDF files in PRAC_TC_ISCA25/PRACLeak/results/plots. Scripts for plotting individual figures can be found in PRAC_TC_ISCA25/PRACLeak/scripts/plot_scripts.

A.5.2 TPRAC Evaluations. After completing the security analysis and performance experiments using run_artifact.sh, the results and plots can be regenerated with the provided scripts. Specifically, the artifact includes the plot_main_figure.sh and plot_all_figures.sh files in the PRAC_TC_ISCA25/TPRAC directory. These scripts collate the results (obtained as CSV files in PRAC_TC_ISCA25/TPRAC/results/csvs) and generate the plots (obtained as PDF files in PRAC_TC_ISCA25/TPRAC/results/plots). The plot_main_figure.sh script regenerates Figure 10, while the plot_all_figures.sh script generates Figure 10 to Figure 14. Additionally, we provide scripts to collate results (generate_csv_fig#.py) and generate plots (plot_fig#.py or plot.ipynb) for each experiment in the PRAC_TC_ISCA25/TPRAC/plot_scripts directory. Sample result files and plots are available in the PRAC_TC_ISCA25/TPRAC/results/sample_results directory.

A.6 Experiment Customization: *TPRAC* Performance Evaluations

We offer easy configuration options for the following parameters: 1) the evaluated *TPRAC* mechanisms, 2) the tested PRAC levels, 3) the evaluated RowHammer Thresholds, and 4) the simulation duration (minimum number of instructions per core during experiments). These parameters can be customized in the PRAC_TC_ISCA25/TPRAC/sim_scripts/run_config_fig#.py files:

- *TPRAC* configurations: mitigation_list.
- PRAC levels: PRAC_levels.
- RowHammer Thresholds: NRH_lists.
- Simulation duration: NUM_EXPECTED_INSTS.

A.7 Methodology

Submission, reviewing and badging methodology:

- https://www.acm.org/publications/policies/artifact-review-and-badging-current
- https://cTuning.org/ae

References

[1] [n. d.]. UBC Advanced Research Computing, "UBC ARC Sockeye." UBC Advanced Research Computing, 2019, doi: 10.14288/SOCKEYE. ([n. d.]).
[2] Ali Asgari Khoshouyeh, Florian Geissler, Syed Qutub, Michael Paulitsch, Prashant Nair, and Karthik Pattabiraman. 2023. Structural Coding: A Low-Cost Scheme to Protect CNNs from Large-Granularity Memory Faults. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis* (Denver, CO, USA) *(SC '23)*. Association for Computing Machinery, New York, NY, USA, Article 85, 17 pages. https://doi.org/10.1145/3581784.3607084
[3] Daniel J. Bernstein. 2005. *Cache-timing attacks on AES*. Technical Report.

[4] Eshan Bhatia, Gino Chacon, Seth Pugsley, Elvira Teran, Paul V. Gratz, and Daniel A. Jiménez. 2019. Perceptron-Based Prefetch Filtering. In *2019 ACM/IEEE 46th Annual International Symposium on Computer Architecture (ISCA)*. 1–13.

[5] Joseph Bonneau and Ilya Mironov. 2006. Cache-collision timing attacks against AES. In *CHES*.

[6] F. Nisa Bostanci, ISmail Emir Yüksel, Ataberk Olgun, Konstantinos Kanellopoulos, Yahya Can Tuğrul, A. Giray Yağlıçi, Mohammad Sadrosadati, and Onur Mutlu. 2024. CoMeT: Count-Min-Sketch-based Row Tracking to Mitigate RowHammer at Low Cost. In *2024 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*. 593–612. https://doi.org/10.1109/HPCA57654.2024.00050

[7] Kunbei Cai, Md Hafizul Islam Chowdhuryy, Zhenkai Zhang, and Fan Yao. 2024. DeepVenom: Persistent DNN Backdoors Exploiting Transient Weight Perturbations in Memories. In *2024 IEEE Symposium on Security and Privacy (SP)*. 2067–2085. https://doi.org/10.1109/SP54263.2024.00223

[8] Oğuzhan Canpolat, A Giray Yağlıkçı, Geraldo F Oliveira, Ataberk Olgun, Oğuz Ergin, and Onur Mutlu. 2024. Understanding the Security Benefits and Overheads of Emerging Industry Solutions to DRAM Read Disturbance. In *Workshop on DRAM Security (DRAMSec)*.

[9] Oğuzhan Canpolat, A. Giray Yağlıkçı, Ataberk Olgun, Ismail Emir Yuksel, Yahya Can Tuğrul, Konstantinos Kanellopoulos, Oğuz Ergin, and Onur Mutlu. 2024. BreakHammer: Enhancing RowHammer Mitigations by Carefully Throttling Suspect Threads. In *2024 57th IEEE/ACM International Symposium on Microarchitecture (MICRO)*. 915–934. https://doi.org/10.1109/MICRO61859.2024.00072

[10] Oğuzhan Canpolat, A. Giray Yağlıkçı, Geraldo F. Oliveira, Ataberk Olgun, Nisa Bostancı, Ismail Emir Yuksel, Haocong Luo, Oğuz Ergin, and Onur Mutlu. 2025. Chronus: Understanding and Securing the Cutting-Edge Industry Solutions to DRAM Read Disturbance. In *2025 IEEE International Symposium on High Performance Computer Architecture (HPCA)*. 887–905. https://doi.org/10.1109/HPCA61900.2025.00071

[11] Kevin K. Chang, Prashant J. Nair, Donghyuk Lee, Saugata Ghose, Moinuddin K. Qureshi, and Onur Mutlu. 2016. Low-Cost Inter-Linked Subarrays (LISA): Enabling fast inter-subarray data movement in DRAM. In *2016 IEEE International Symposium on High Performance Computer Architecture (HPCA)*. 568–580. https://doi.org/10.1109/HPCA.2016.7446095

[12] Boru Chen, Yingchen Wang, Pradyumna Shome, Christopher W Fletcher, David Kohlbrenner, Riccardo Paccagnella, and Daniel Genkin. 2024. GoFetch: Breaking constant-time cryptographic implementations using data memory-dependent prefetchers. In *Proc. USENIX Secur. Symp.* 1–21.

[13] Yun Chen, Lingfeng Pei, and Trevor E. Carlson. 2023. AfterImage: Leaking Control Flow Data and Tracking Load Operations via the Hardware Prefetcher. In *Proceedings of the 28th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 2* (Vancouver, BC, Canada) *(ASPLOS 2023)*. Association for Computing Machinery, New York, NY, USA, 16–32. https://doi.org/10.1145/3575693.3575719

[14] Md Hafizul Islam Chowdhuryy, Hao Zheng, and Fan Yao. 2024. MetaLeak: Uncovering Side Channels in Secure Processor Architectures Exploiting Metadata. In *2024 ACM/IEEE 51st Annual International Symposium on Computer Architecture (ISCA)*. IEEE, 693–707.

[15] Zachary Coalson, Jeonghyun Woo, Shiyang Chen, Yu Sun, Lishan Yang, Prashant Nair, Bo Fang, and Sanghyun Hong. 2024. PrisonBreak: Jailbreaking Large Language Models with Fewer Than Twenty-Five Targeted Bit-flips. arXiv:2412.07192 [cs.CR] https://arxiv.org/abs/2412.07192

[16] Lucian Cojocar, Kaveh Razavi, Cristiano Giuffrida, and Herbert Bos. 2019. Exploiting Correcting Codes: On the Effectiveness of ECC Memory Against Rowhammer Attacks. In *2019 IEEE Symposium on Security and Privacy (SP)*. 55–71. https://doi.org/10.1109/SP.2019.00089

[17] Standard Performance Evaluation Corporation. 2006. SPEC CPU2006 Benchmark Suite. http://www.spec.org/cpu2006/

[18] Peter W Deutsch, Yuheng Yang, Thomas Bourgeat, Jules Drean, Joel S Emer, and Mengjia Yan. 2022. DAGguise: mitigating memory timing side channels. In *Proceedings of the 27th ACM International Conference on Architectural Support for Programming Languages and Operating Systems*. 329–343.

[19] Dmitry Evtyushkin, Ryan Riley, Nael CSE Abu-Ghazaleh, ECE, and Dmitry Ponomarev. 2018. Branchscope: A new side-channel attack on directional branch predictor. *ACM SIGPLAN Notices* 53, 2 (2018), 693–707.

[20] Ali Fakhrzadehgan, Yale N Patt, Prashant J Nair, and Moinuddin K Qureshi. 2022. SafeGuard: Reducing the Security Risk from Row-Hammer via Low-Cost Integrity Protection. In *HPCA*. IEEE.

[21] Michael Ferdman, Almutaz Adileh, Onur Kocberber, Stavros Volos, Mohammad Alisafaee, Djordje Jevdjic, Cansu Kaynak, Adrian Daniel Popescu, Anastasia Ailamaki, and Babak Falsafi. 2012. Clearing the clouds: a study of emerging scale-out workloads on modern hardware. *SIGPLAN Not.* 47, 4 (March 2012), 37–48. https://doi.org/10.1145/2248487.2150982

[22] Pietro Frigo, Emanuele Vannacc, Hasan Hassan, Victor van der Veen, Onur Mutlu, Cristiano Giuffrida, Herbert Bos, and Kaveh Razavi. 2020. TRRespass:

Exploiting the Many Sides of Target Row Refresh. In *2020 IEEE Symposium on Security and Privacy (SP)*. 747–762. https://doi.org/10.1109/SP40000.2020.00090

[23] Stefan Gast, Jonas Juffinger, Martin Schwarzl, Gururaj Saileshwar, Andreas Kogler, Simone Franza, Markus Köstl, and Daniel Gruss. 2023. Squip: Exploiting the scheduler queue contention side channel. In *2023 IEEE Symposium on Security and Privacy (SP)*. IEEE, 2256–2272.

[24] Nathan Gober, Gino Chacon, Lei Wang, Paul V. Gratz, Daniel A. Jimenez, Elvira Teran, Seth Pugsley, and Jinchun Kim. 2022. The Championship Simulator: Architectural Simulation for Education and Competition. arXiv:2210.14324 [cs.AR] https://arxiv.org/abs/2210.14324

[25] Daniel Gruss, Julian Lettner, Felix Schuster, Olya Ohrimenko, Istvan Haller, and Manuel Costa. 2017. Strong and Efficient Cache Side-Channel Protection using Hardware Transactional Memory. In *USENIX Security*.

[26] Daniel Gruss, Moritz Lipp, Michael Schwarz, Daniel Genkin, Jonas Juffinger, Sioli O'Connell, Wolfgang Schoechl, and Yuval Yarom. 2018. Another Flip in the Wall of Rowhammer Defenses. In *2018 IEEE Symposium on Security and Privacy (SP)*. 245–261. https://doi.org/10.1109/SP.2018.00031

[27] Daniel Gruss, Clémentine Maurice, and Stefan Mangard. 2016. Rowhammer.js: A Remote Software-Induced Fault Attack in JavaScript. In *Proceedings of the 13th International Conference on Detection of Intrusions and Malware, and Vulnerability Assessment - Volume 9721* (San Sebastián, Spain) *(DIMVA 2016)*. Springer-Verlag, Berlin, Heidelberg, 300–321. https://doi.org/10.1007/978-3-319-40667-1_15

[28] Daniel Gruss, Clémentine Maurice, Klaus Wagner, and Stefan Mangard. 2016. Flush+ Flush: a fast and stealthy cache attack. In *DIMVA*.

[29] Hasan Hassan, Minesh Patel, Jeremie S Kim, A Giray Yaglikci, Nandita Vijaykumar, Nika Mansouri Ghiasi, Saugata Ghose, and Onur Mutlu. 2019. Crow: A low-cost substrate for improving dram performance, energy efficiency, and reliability. In *Proceedings of the 46th International Symposium on Computer Architecture*. 129–142.

[30] Hasan Hassan, Yahya Can Tugrul, Jeremie S Kim, Victor Van der Veen, Kaveh Razavi, and Onur Mutlu. 2021. 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*. 1198–1213.

[31] Sanghyun Hong, Pietro Frigo, Yigitcan Kaya, Cristiano Giuffrida, and Tudor Dumitras. 2019. Terminal Brain Damage: Exposing the Graceless Degradation in Deep Neural Networks Under Hardware Fault Attacks. In *28th USENIX Security Symposium (USENIX Security 19)*. USENIX Association, Santa Clara, CA, 497–514. https://www.usenix.org/conference/usenixsecurity19/presentation/hong

[32] Seungki Hong, Dongha Kim, Jaehyung Lee, Reum Oh, Changsik Yoo, Sangjoon Hwang, and Jooyoung Lee. 2023. Dsac: Low-cost rowhammer mitigation using in-dram stochastic and approximate counting algorithm. *arXiv:2302.03591* (2023).

[33] Intel. 2022. What does SGX 2.0 (scalable SGX) sacrifice? Github. https://github.com/intel/linux-sgx/issues/899

[34] Aamer Jaleel, Stephen W. Keckler, and Gururaj Saileshwar. 2024. Probabilistic Tracker Management Policies for Low-Cost and Scalable Rowhammer Mitigation. *arXiv:2404.16256* (2024).

[35] Aamer Jaleel, Gururaj Saileshwar, Stephen W. Keckler, and Moinuddin Qureshi. 2024. PrIDE: Achieving Secure Rowhammer Mitigation with Low-Cost In-DRAM Trackers. In *2024 ACM/IEEE 51st Annual International Symposium on Computer Architecture (ISCA)*. 1157–1172. https://doi.org/10.1109/ISCA59077.2024.00087

[36] Aamer Jaleel, Kevin B. Theobald, Simon C. Steely, and Joel Emer. 2010. High performance cache replacement using re-reference interval prediction (RRIP). *SIGARCH Comput. Archit. News* 38, 3 (June 2010), 60–71. https://doi.org/10.1145/1816038.1815971

[37] Yeongjin Jang, Jaehyuk Lee, Sangho Lee, and Taesoo Kim. 2017. SGX-Bomb: Locking Down the Processor via Rowhammer Attack. In *Proceedings of the 2nd Workshop on System Software for Trusted Execution* (Shanghai, China) *(SysTEX'17)*. Association for Computing Machinery, New York, NY, USA, Article 5, 6 pages. https://doi.org/10.1145/3152701.3152709

[38] Patrick Jattke, Victor Van Der Veen, Pietro Frigo, Stijn Gunter, and Kaveh Razavi. 2022. BLACKSMITH: Scalable Rowhammering in the Frequency Domain. In *2022 IEEE Symposium on Security and Privacy (SP)*. 716–734. https://doi.org/10.1109/SP46214.2022.9833772

[39] Patrick Jattke, Max Wipfli, Flavien Solt, Michele Marazzi, Matej Bölcskei, and Kaveh Razavi. 2024. ZenHammer: Rowhammer Attacks on AMD Zen-based Platforms. In *33rd USENIX Security Symposium (USENIX Security 24)*. USENIX Association, Philadelphia, PA, 1615–1633. https://www.usenix.org/conference/usenixsecurity24/presentation/jattke

[40] JEDEC. 2017. DDR4 SDRAM standard (JESD79-4B). (2017).

[41] JEDEC. 2024. *JESD79-5C*. https://www.jedec.org/document_search?search_api_views_fulltext=jesd79-5c.

[42] D.A. Jimenez and C. Lin. 2001. Dynamic branch prediction with perceptrons. In *Proceedings HPCA Seventh International Symposium on High-Performance Computer Architecture*. 197–206. https://doi.org/10.1109/HPCA.2001.903263

[43] Dimitris Kaseridis, Jeffrey Stuecheli, and Lizy Kurian John. 2011. Minimalist open-page: a DRAM page-mode scheduling policy for the many-core era. In

Proceedings of the 44th Annual IEEE/ACM International Symposium on Microarchitecture (Porto Alegre, Brazil) *(MICRO-44)*. Association for Computing Machinery, New York, NY, USA, 24–35. https://doi.org/10.1145/2155620.2155624

[44] Dae-Hyun Kim, Prashant J. Nair, and Moinuddin K. Qureshi. 2015. Architectural Support for Mitigating Row Hammering in DRAM Memories. *IEEE Computer Architecture Letters* 14, 1 (2015), 9–12. https://doi.org/10.1109/LCA.2014.2332177

[45] Jeremie S. Kim, Minesh Patel, A. Giray Yağlıkçı, Hasan Hassan, Roknoddin Azizi, Lois Orosa, and Onur Mutlu. 2020. Revisiting RowHammer: an experimental analysis of modern DRAM devices and mitigation techniques. In *Proceedings of the ACM/IEEE 47th Annual International Symposium on Computer Architecture* (Virtual Event) *(ISCA '20)*. IEEE Press, 638–651. https://doi.org/10.1109/ISCA45697.2020.00059

[46] Kwangrae Kim, Jeonghyun Woo, Junsu Kim, and Ki-Seok Chung. 2021. Hammer-Filter: Robust Protection and Low Hardware Overhead Method for RowHammer. In *2021 IEEE 39th International Conference on Computer Design (ICCD)*. 212–219. https://doi.org/10.1109/ICCD53106.2021.00043

[47] Michael Jaemin Kim, Jaehyun Park, Yeonhong Park, Wanju Doh, Namhoon Kim, Tae Jun Ham, Jae W. Lee, and Jung Ho Ahn. 2022. Mithril: Cooperative Row Hammer Protection on Commodity DRAM Leveraging Managed Refresh. In *2022 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*. 1156–1169. https://doi.org/10.1109/HPCA53966.2022.00088

[48] Woongrae Kim, Chulmoon Jung, Seongnyuh Yoo, Duckhwa Hong, Jeongjin Hwang, Jungmin Yoon, Ohyong Jung, Joonwoo Choi, Sanga Hyun, Mankeun Kang, Sangho Lee, Dohong Kim, Sanghyun Ku, Donhyun Choi, Nogeun Joo, Sangwoo Yoon, Junseok Noh, Byeongyong Go, Cheolhoe Kim, Sunil Hwang, Mihyun Hwang, Seol-Min Yi, Hyungmin Kim, Sanghyuk Heo, Yeonsu Jang, Kyoungchul Jang, Shinho Chu, Yoonna Oh, Kwidong Kim, Junghyun Kim, Soohwan Kim, Jeongtae Hwang, Sangil Park, Junphyo Lee, Inchul Jeong, Joohwan Cho, and Jonghwan Kim. 2023. 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)*. 1–3. https://doi.org/10.1109/ISSCC42615.2023.10067805

[49] Yoongu Kim, Ross Daly, Jeremie Kim, Chris Fallin, Ji Hye Lee, Donghyuk Lee, Chris Wilkerson, Konrad Lai, and Onur Mutlu. 2014. Flipping bits in memory without accessing them: an experimental study of DRAM disturbance errors. *SIGARCH Comput. Archit. News* 42, 3 (June 2014), 361–372. https://doi.org/10.1145/2678373.2665726

[50] Yoongu Kim, Weikun Yang, and Onur Mutlu. 2015. Ramulator: A fast and extensible DRAM simulator. *IEEE Computer architecture letters* 15, 1 (2015), 45–49.

[51] Paul Kocher, Jann Horn, Anders Fogh, Daniel Genkin, Daniel Gruss, Werner Haas, Mike Hamburg, Moritz Lipp, Stefan Mangard, Thomas Prescher, et al. 2019. Spectre Attacks: Exploiting Speculative Execution. In *2019 IEEE Symposium on Security and Privacy (SP)*. IEEE, 1–19.

[52] Andreas Kogler, Jonas Juffinger, Salman Qazi, Yoongu Kim, Moritz Lipp, Nicolas Boichat, Eric Shiu, Mattias Nissler, and Daniel Gruss. 2022. Half-Double: Hammering From the Next Row Over. In *31st USENIX Security Symposium (USENIX Security 22)*. USENIX Association, Boston, MA, 3807–3824. https://www.usenix.org/conference/usenixsecurity22/presentation/kogler-half-double

[53] Andrew Kwong, Daniel Genkin, Daniel Gruss, and Yuval Yarom. 2020. RAM-Bleed: Reading Bits in Memory Without Accessing Them. In *2020 IEEE Symposium on Security and Privacy (SP)*. 695–711. https://doi.org/10.1109/SP40000.2020.00020

[54] Eojin Lee, Ingab Kang, Sukhan Lee, G Edward Suh, and Jung Ho Ahn. 2019. TWiCe: preventing row-hammering by exploiting time window counters. In *Proceedings of the 46th International Symposium on Computer Architecture*. 385–396. https://doi.org/10.1145/3307650.3322232

[55] Moritz Lipp, Vedad Hadžić, Michael Schwarz, Arthur Perais, Clémentine Maurice, and Daniel Gruss. 2020. Take A Way: Exploring the Security Implications of AMD's Cache Way Predictors. In *AsiaCCS*.

[56] Moritz Lipp, Michael Schwarz, Daniel Gruss, Thomas Prescher, Werner Haas, Anders Fogh, Jann Horn, Stefan Mangard, Paul Kocher, Daniel Genkin, Yuval Yarom, and Mike Hamburg. 2018. Meltdown: Reading kernel memory from user space. In *USENIX Security*.

[57] Fangfei Liu, Yuval Yarom, Qian Ge, Gernot Heiser, and Ruby B Lee. 2015. Last-level cache side-channel attacks are practical. In *2015 IEEE symposium on security and privacy*. IEEE, 605–622.

[58] Kevin Loughlin, Jonah Rosenblum, Stefan Saroiu, Alec Wolman, Dimitrios Skarlatos, and Baris Kasikci. 2023. Siloz: Leveraging DRAM Isolation Domains to Prevent Inter-VM Rowhammer. In *Proceedings of the 29th Symposium on Operating Systems Principles*. 417–433.

[59] Kevin Loughlin, Stefan Saroiu, Alec Wolman, Yatin A Manerkar, and Baris Kasikci. 2022. MOESI-prime: preventing coherence-induced hammering in commodity workloads. In *ISCA*.

[60] Haocong Luo, Yahya Can Tuğrul, F. Nisa Bostancı, Ataberk Olgun, A. Giray Yağlıkçı, , and Onur Mutlu. 2023. Ramulator 2.0: A Modern, Modular, and Extensible DRAM Simulator.

[61] Michele Marazzi, Patrick Jattke, Flavien Solt, and Kaveh Razavi. 2022. Protrr: Principled yet optimal in-dram target row refresh. In *IEEE Symposium on Security and Privacy (SP)*. 735–753.

[62] Michele Marazzi, Flavien Solt, Patrick Jattke, Kubo Takashi, and Kaveh Razavi. 2023. REGA: Scalable Rowhammer Mitigation with Refresh-Generating Activations. In *IEEE Symposium on Security and Privacy (SP)*. IEEE.

[63] Onur Mutlu and Thomas Moscibroda. 2007. Stall-Time Fair Memory Access Scheduling for Chip Multiprocessors. In *40th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO 2007)*. 146–160. https://doi.org/10.1109/MICRO.2007.21

[64] Prashant Nair, Chia-Chen Chou, and Moinuddin K. Qureshi. 2013. A case for Refresh Pausing in DRAM memory systems. In *2013 IEEE 19th International Symposium on High Performance Computer Architecture (HPCA)*. 627–638. https://doi.org/10.1109/HPCA.2013.6522355

[65] Prashant J. Nair, Bahar Asgari, and Moinuddin K. Qureshi. 2019. SuDoku: Tolerating High-Rate of Transient Failures for Enabling Scalable STTRAM. In *2019 49th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*. 388–400. https://doi.org/10.1109/DSN.2019.00048

[66] Prashant J. Nair, Dae-Hyun Kim, and Moinuddin K. Qureshi. 2013. ArchShield: architectural framework for assisting DRAM scaling by tolerating high error rates. In *Proceedings of the 40th Annual International Symposium on Computer Architecture* (Tel-Aviv, Israel) *(ISCA '13)*. Association for Computing Machinery, New York, NY, USA, 72–83. https://doi.org/10.1145/2485922.2485929

[67] Prashant J. Nair, David A. Roberts, and Moinuddin K. Qureshi. 2014. Citadel: Efficiently Protecting Stacked Memory from Large Granularity Failures. In *2014 47th Annual IEEE/ACM International Symposium on Microarchitecture*. 51–62. https://doi.org/10.1109/MICRO.2014.57

[68] Prashant J. Nair, David A. Roberts, and Moinuddin K. Qureshi. 2015. FaultSim: A Fast, Configurable Memory-Reliability Simulator for Conventional and 3D-Stacked Systems. *ACM Trans. Archit. Code Optim.* 12, 4, Article 44 (Dec. 2015), 24 pages. https://doi.org/10.1145/2831234

[69] Prashant J. Nair, David A. Roberts, and Moinuddin K. Qureshi. 2016. Citadel: Efficiently Protecting Stacked Memory from TSV and Large Granularity Failures. *ACM Trans. Archit. Code Optim.* 12, 4, Article 49 (Jan. 2016), 24 pages. https://doi.org/10.1145/2840807

[70] Prashant J. Nair, Vilas Sridharan, and Moinuddin K. Qureshi. 2016. XED: exposing on-die error detection information for strong memory reliability. In *Proceedings of the 43rd International Symposium on Computer Architecture* (Seoul, Republic of Korea) *(ISCA '16)*. IEEE Press, 341–353. https://doi.org/10.1109/ISCA.2016.38

[71] Agustín Navarro-Torres, Biswabandan Panda, Jesús Alastruey-Benedé, Pablo Ibáñez, Víctor Viñals-Yúfera, and Alberto Ros. 2022. Berti: an Accurate Local-Delta Data Prefetcher. In *2022 55th IEEE/ACM International Symposium on Microarchitecture (MICRO)*. 975–991. https://doi.org/10.1109/MICRO56248.2022.00072

[72] Ataberk Olgun, Majd Osseiran, A. Giray Yağlıkçı, Yahya Can Tuğrul, Haocong Luo, Steve Rhyner, Behzad Salami, Juan Gomez Luna, and Onur Mutlu. 2024. Read Disturbance in High Bandwidth Memory: A Detailed Experimental Study on HBM2 DRAM Chips. In *2024 54th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*. 75–89. https://doi.org/10.1109/DSN58291.2024.00022

[73] Ataberk Olgun, Yahya Can Tugrul, Nisa Bostanci, Ismail Emir Yuksel, Haocong Luo, Steve Rhyner, Abdullah Giray Yaglikci, Geraldo F. Oliveira, and Onur Mutlu. 2024. ABACuS: All-Bank Activation Counters for Scalable and Low Overhead RowHammer Mitigation. In *33rd USENIX Security Symposium (USENIX Security 24)*. USENIX Association, Philadelphia, PA, 1579–1596. https://www.usenix.org/conference/usenixsecurity24/presentation/olgun

[74] Yeonhong Park, Woosuk Kwon, Eojin Lee, Tae Jun Ham, Jung Ho Ahn, and Jae W Lee. 2020. Graphene: Strong yet Lightweight Row Hammer Protection. In *MICRO*. IEEE, 1–13.

[75] Peter Pessl, Daniel Gruss, Clémentine Maurice, Michael Schwarz, and Stefan Mangard. 2016. DRAMA: Exploiting DRAM Addressing for Cross-CPU Attacks. In *25th USENIX Security Symposium (USENIX Security 16)*. USENIX Association, Austin, TX, 565–581. https://www.usenix.org/conference/usenixsecurity16/technical-sessions/presentation/pessl

[76] Moinuddin Qureshi. 2025. AutoRFM: Scaling Low-Cost in-DRAM Trackers to Ultra-Low Rowhammer Thresholds. In *2025 IEEE International Symposium on High Performance Computer Architecture (HPCA)*. 991–1004. https://doi.org/10.1109/HPCA61900.2025.00078

[77] Moinuddin Qureshi and Salman Qazi. 2025. MOAT: Securely Mitigating Rowhammer with Per-Row Activation Counters. In *Proceedings of the 30th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 1* (Rotterdam, Netherlands) *(ASPLOS '25)*. Association for Computing Machinery, New York, NY, USA, 698–714. https://doi.org/10.1145/3669940.3707278

[78] Moinuddin Qureshi, Salman Qazi, and Aamer Jaleel. 2024. MINT: Securely Mitigating Rowhammer with a Minimalist in-DRAM Tracker. In *2024 57th IEEE/ACM International Symposium on Microarchitecture (MICRO)*. 899–914.

https://doi.org/10.1109/MICRO61859.2024.00071

[79] Moinuddin Qureshi, Aditya Rohan, Gururaj Saileshwar, and Prashant J Nair. 2022. Hydra: enabling low-overhead mitigation of row-hammer at ultra-low thresholds via hybrid tracking. In *ISCA*.

[80] Moinuddin K. Qureshi, Dae-Hyun Kim, Samira Khan, Prashant J. Nair, and Onur Mutlu. 2015. AVATAR: A Variable-Retention-Time (VRT) Aware Refresh for DRAM Systems. In *2015 45th Annual IEEE/IFIP International Conference on Dependable Systems and Networks*. 427–437. https://doi.org/10.1109/DSN.2015.58

[81] Adnan Siraj Rakin, Md Hafizul Islam Chowdhuryy, Fan Yao, and Deliang Fan. 2022. DeepSteal: Advanced Model Extractions Leveraging Efficient Weight Stealing in Memories. In *2022 IEEE Symposium on Security and Privacy (SP)*. 1157–1174. https://doi.org/10.1109/SP46214.2022.9833743

[82] Kaveh Razavi, Ben Gras, Erik Bosman, Bart Preneel, Cristiano Giuffrida, and Herbert Bos. 2016. Flip Feng Shui: Hammering a Needle in the Software Stack. In *25th USENIX Security Symposium (USENIX Security 16)*. USENIX Association, Austin, TX, 1–18. https://www.usenix.org/conference/usenixsecurity16/technical-sessions/presentation/razavi

[83] S. Rixner, W.J. Dally, U.J. Kapasi, P. Mattson, and J.D. Owens. 2000. Memory access scheduling. In *Proceedings of 27th International Symposium on Computer Architecture (IEEE Cat. No.RS00201)*. 128–138. https://doi.org/10.1145/339647.339668

[84] Gururaj Saileshwar, Prashant J. Nair, Prakash Ramrakhyani, Wendy Elsasser, Jose A. Joao, and Moinuddin K. Qureshi. 2018. Morphable Counters: Enabling Compact Integrity Trees For Low-Overhead Secure Memories. In *2018 51st Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*. 416–427. https://doi.org/10.1109/MICRO.2018.00041

[85] Gururaj Saileshwar, Prashant J Nair, Prakash Ramrakhyani, Wendy Elsasser, and Moinuddin K Qureshi. 2018. Synergy: Rethinking secure-memory design for error-correcting memories. In *2018 IEEE International Symposium on High Performance Computer Architecture (HPCA)*. IEEE, 454–465.

[86] Gururaj Saileshwar, Bolin Wang, Moinuddin Qureshi, and Prashant J Nair. 2022. Randomized row-swap: mitigating Row Hammer by breaking spatial correlation between aggressor and victim rows. In *ASPLOS*.

[87] Anish Saxena, Saurav Mathur, and Moinuddin Qureshi. 2024. Rubix: Reducing the Overhead of Secure Rowhammer Mitigations via Randomized Line-to-Row Mapping. In *Proceedings of the 29th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 2* (La Jolla, CA, USA) *(ASPLOS '24)*. Association for Computing Machinery, New York, NY, USA, 1014–1028. https://doi.org/10.1145/3620665.3640404

[88] Anish Saxena and Moinuddin Qureshi. 2024. START: Scalable Tracking for any Rowhammer Threshold. In *2024 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*. 578–592. https://doi.org/10.1109/HPCA57654.2024.00049

[89] Anish Saxena, Gururaj Saileshwar, Prashant J Nair, and Moinuddin Qureshi. 2022. Aqua: Scalable rowhammer mitigation by quarantining aggressor rows at runtime. In *MICRO*.

[90] Mark Seaborn and Thomas Dullien. 2015. Exploiting the DRAM rowhammer bug to gain kernel privileges. *Black Hat* 15 (2015), 71.

[91] Seyed Mohammad Seyedzadeh, Alex K Jones, and Rami Melhem. 2018. Mitigating wordline crosstalk using adaptive trees of counters. In *ISCA*.

[92] Ali Shafiee, Akhila Gundu, Manjunath Shevgoor, Rajeev Balasubramonian, and Mohit Tiwari. 2015. Avoiding information leakage in the memory controller with fixed service policies. In *Proceedings of the 48th International Symposium on Microarchitecture*. 89–101.

[93] Youngjoo Shin, Hyung Chan Kim, Dokeun Kwon, Ji Hoon Jeong, and Junbeom Hur. 2018. Unveiling hardware-based data prefetcher, a hidden source of information leakage. In *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*. 131–145.

[94] Mungyu Son, Hyunsun Park, Junwhan Ahn, and Sungjoo Yoo. 2017. Making DRAM stronger against row hammering. In *Design Automation Conference*.

[95] SPEC2017 [n. d.]. SPEC CPU2017 Benchmark Suite. Standard Performance Evaluation Corporation. http://www.spec.org/cpu2017/

[96] Victor van der Veen, Yanick Fratantonio, Martina Lindorfer, Daniel Gruss, Clementine Maurice, Giovanni Vigna, Herbert Bos, Kaveh Razavi, and Cristiano Giuffrida. 2016. Drammer: Deterministic Rowhammer Attacks on Mobile Platforms. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security* (Vienna, Austria) *(CCS '16)*. Association for Computing Machinery, New York, NY, USA, 1675–1689. https://doi.org/10.1145/2976749.2978406

[97] Jose Rodrigo Sanchez Vicarte, Michael Flanders, Riccardo Paccagnella, Grant Garrett-Grossman, Adam Morrison, Christopher W Fletcher, and David Kohlbrenner. 2022. Augury: Using data memory-dependent prefetchers to leak data at rest. In *2022 IEEE Symposium on Security and Privacy (SP)*. IEEE, 1491–1505.

[98] Minghua Wang, Zhi Zhang, Yueqiang Cheng, and Surya Nepal. 2020. DRAMDig: a knowledge-assisted tool to uncover DRAM address mapping. In *Proceedings of the 57th ACM/EDAC/IEEE Design Automation Conference* (Virtual Event, USA) *(DAC '20)*. IEEE Press, Article 89, 6 pages.

[99] Yao Wang, Andrew Ferraiuolo, and G. Edward Suh. 2014. Timing channel protection for a shared memory controller. In *2014 IEEE 20th International Symposium on High Performance Computer Architecture (HPCA)*. 225–236. https://doi.org/10.1109/HPCA.2014.6835934

[100] Yao Wang, Benjamin Wu, and G. Edward Suh. 2017. Secure Dynamic Memory Scheduling Against Timing Channel Attacks. In *2017 IEEE International Symposium on High Performance Computer Architecture (HPCA)*. 301–312. https://doi.org/10.1109/HPCA.2017.27

[101] Minbok Wi, Jaehyun Park, Seoyoung Ko, Michael Jaemin Kim, Nam Sung Kim, Eojin Lee, and Jung Ho Ahn. 2023. SHADOW: Preventing Row Hammer in DRAM with Intra-Subarray Row Shuffling. In *HPCA*.

[102] Jeonghyun Woo, Shaopeng Chris Lin, Prashant J. Nair, Aamer Jaleel, and Gururaj Saileshwar. 2025. QPRAC: Towards Secure and Practical PRAC-based Rowhammer Mitigation using Priority Queues. In *2025 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*. 1021–1037. https://doi.org/10.1109/HPCA61900.2025.00080

[103] Jeonghyun Woo and Prashant J. Nair. 2025. DAPPER: A Performance-Attack-Resilient Tracker for RowHammer Defense. In *2025 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*. 1005–1020. https://doi.org/10.1109/HPCA61900.2025.00079

[104] Jeonghyun Woo, Gururaj Saileshwar, and Prashant J Nair. 2023. Scalable and Secure Row-Swap: Efficient and Safe Row Hammer Mitigation in Memory Systems. In *HPCA*.

[105] A. Giray Yağlıkçı, Minesh Patel, Jeremie S. Kim, Roknoddin Azizi, Ataberk Olgun, Lois Orosa, Hasan Hassan, Jisung Park, Konstantinos Kanellopoulos, Taha Shahroodi, Saugata Ghose, and Onur Mutlu. 2021. BlockHammer: Preventing RowHammer at Low Cost by Blacklisting Rapidly-Accessed DRAM Rows. In *2021 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*. 345–358. https://doi.org/10.1109/HPCA51647.2021.00037

[106] Chenyu Yan, D. Englender, M. Prvulovic, B. Rogers, and Yan Solihin. 2006. Improving Cost, Performance, and Security of Memory Encryption and Authentication. In *33rd International Symposium on Computer Architecture (ISCA'06)*. 179–190. https://doi.org/10.1109/ISCA.2006.22

[107] Mengjia Yan, Read Sprabery, Bhargava Gopireddy, Christopher Fletcher, Roy Campbell, and Josep Torrellas. 2019. Attack directories, not caches: Side channel attacks in a non-inclusive world. In *2019 IEEE Symposium on Security and Privacy (SP)*. IEEE, 888–904.

[108] Fan Yao, Adnan Siraj Rakin, and Deliang Fan. 2020. DeepHammer: Depleting the Intelligence of Deep Neural Networks through Targeted Chain of Bit Flips. In *29th USENIX Security Symposium (USENIX Security 20)*. USENIX Association, 1463–1480. https://www.usenix.org/conference/usenixsecurity20/presentation/yao

[109] Yuval Yarom and Katrina Falkner. 2014. FLUSH+ RELOAD: A high resolution, low noise, l3 cache Side-Channel attack. In *23rd USENIX security symposium (USENIX Security)*. 719–732.

[110] A. Giray Yağlıkçı, Ataberk Olgun, Minesh Patel, Haocong Luo, Hasan Hassan, Lois Orosa, Oğuz Ergin, and Onur Mutlu. 2022. HiRA: Hidden Row Activation for Reducing Refresh Latency of Off-the-Shelf DRAM Chips. In *2022 55th IEEE/ACM International Symposium on Microarchitecture (MICRO)*. 815–834. https://doi.org/10.1109/MICRO56248.2022.00062

[111] Abdullah Giray Yağlıkçı, Jeremie S. Kim, Fabrice Devaux, and Onur Mutlu. 2021. Security Analysis of the Silver Bullet Technique for RowHammer Prevention. arXiv:2106.07084 [cs.CR] https://arxiv.org/abs/2106.07084

[112] Abdullah Giray Yağlıkçı, Yahya Can Tuğrul, Geraldo F. Oliveira, İsmail Emir Yüksel, Ataberk Olgun, Haocong Luo, and Onur Mutlu. 2024. Spatial Variation-Aware Read Disturbance Defenses: Experimental Analysis of Real DRAM Chips and Implications on Future Solutions. In *2024 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*. 560–577. https://doi.org/10.1109/HPCA57654.2024.00048

[113] Jung Min You and Joon-Sung Yang. 2019. MRLoc: Mitigating Row-hammering based on memory Locality. In *2019 56th ACM/IEEE Design Automation Conference (DAC)*. IEEE, 1–6.

[114] Vinson Young, Prashant J. Nair, and Moinuddin K. Qureshi. 2015. DEUCE: Write-Efficient Encryption for Non-Volatile Memories. In *Proceedings of the Twentieth International Conference on Architectural Support for Programming Languages and Operating Systems* (Istanbul, Turkey) *(ASPLOS '15)*. Association for Computing Machinery, New York, NY, USA, 33–44. https://doi.org/10.1145/2694344.2694387

[115] Yanqi Zhou, Sameer Wagh, Prateek Mittal, and David Wentzlaff. 2017. Camouflage: Memory traffic shaping to mitigate timing attacks. In *2017 IEEE International Symposium on High Performance Computer Architecture (HPCA)*. IEEE, 337–348.

[116] William K Zuravleff and Timothy Robinson. 1997. Controller for a synchronous DRAM that maximizes throughput by allowing memory requests and commands to be issued out of order. US Patent 5,630,096.