DISS. ETH NO. 30484

PRINCIPLED DRAM SECURITY AGAINST ROWHAMMER ATTACKS

A thesis submitted to attain the degree of

DOCTOR OF SCIENCES
(Dr. sc. ETH Zurich)

presented by

MICHELE MARAZZI
M.Sc. in Biomedical Engineering, Politecnico di Milano

born on 02.05.1993

accepted on the recommendation of

Prof. Dr. Kaveh Razavi
Prof. Dr. Moinuddin Qureshi
Prof. Dr. Jung Ho Ahn

2024

Diss. ETH No. 30484
TIK-Schriftenreihe-Nr. 218

To my family, my partner, and my friends.

ABSTRACT

The security of a system is fractioned into the guarantees of the multiple hardware devices that it relies on. DRAM is pivotal to today's systems, yet its guarantees against sophisticated Rowhammer attacks are uncertain and undisclosed. Instead, the industry's answer to Rowhammer has been security-by-obscurity, quickly proven to be a failure by researchers. It is therefore unclear if, differently from these results, in-DRAM mitigations can provide security against Rowhammer once designed with principled security guarantees. Designing in-DRAM Rowhammer mitigations is complex due to the devices' synchronous nature and rigorous timings. As well, their scalability towards the possible worsening of Rowhammer in future devices is a crucial factor. On top of these challenges, DRAM vendors do not disclose the internal architecture of their devices, making the deployability of mitigations uncertain and their design based on assumptions. Meanwhile, the first high-end RISC-V CPU recently became available; however, no existing research has studied the feasibility of Rowhammer on this new emerging architecture.

In this thesis, we demonstrate that the RISC-V ecosystem is also affected by Rowhammer by triggering bit flips on this architecture for the first time. While DRAM vendors have failed to secure DDR4 devices with TRR, we prove that this is possible with our principled in-DRAM Rowhammer mitigation. As future devices might suffer from low Rowhammer thresholds and a high blast radius, we modify the internal DRAM architecture to protect against such cases. Our design is based on the collaboration with a minor DRAM vendor and the existing literature, and as such, its applicability to commodity devices is unclear. Therefore, to fill the long-lasting gap between industry and research, we image and reverse engineer DRAM devices from the three major vendors.

SOMMARIO

La sicurezza di un sistema è frazionata nelle garanzie dei molteplici dispositivi hardware su cui si basa. La memoria DRAM è fondamentale per i sistemi odierni, tuttavia le sue garanzie contro sofisticati attacchi Rowhammer sono incerte e non rese pubbliche. Invece, la risposta dell'industria a Rowhammer è stata la sicurezza tramite segretezza, rapidamente dimostrata essere un fallimento dai ricercatori. È quindi incerto se, diversamente da questi risultati, le protezioni in-DRAM possano fornire sicurezza contro Rowhammer una volta che esse siano progettate con un approccio alla sicurezza basato su principi. Progettare protezioni in-DRAM contro Rowhammer è complesso a causa della natura sincrona dei dispositivi e dei loro specifici tempi di operazione. Inoltre, la loro scalabilitá rispetto ad un possibile peggioramento della vulnerabilitá Rowhammer è un fattore cruciale. Sfortunatamente, i produttori di DRAM non rendono pubblica l'architettura interna dei loro dispositivi. Questo comporta che i design delle protezioni siano basate su ipotesi, e rende incerta la possibilitá di implementarle. Recentemente, la prima CPU RISC-V di fascia alta è diventata disponibile; tuttavia, nessuno studio ha fin ora valutato la fattibilitá di Rowhammer su questa nuova architettura.

In questa tesi, dimostriamo che anche l'ecosistema RISC-V è affetto da Rowhammer, generando bit flips su questa architettura per la prima volta. Sebbene i produttori di DRAM non siano riusciti a mettere in sicurezza i dispositivi DDR4 con TRR, dimostriamo che ció è possibile con la nostra protezione per Rowhammer basata su principi. Poichè i futuri dispositivi potranno essere molto piú vulnerabili a Rowhammer, abbiamo modificato l'architettura interna della DRAM per proteggere contro tali casi. Il nostro design è basato sulla collaborazione con un produttore di DRAM minore e sulla letteratura esistente. Perció, la sua applicabilitá a dispositivi piú comuni non è chiara. Per colmare il divario tra l'industria e la ricerca, facciamo ingegneria inversa di dispositivi DRAM dei tre principali produttori dopo averne acquisito immagini.

PUBLICATIONS

I base this dissertation on the papers published in workshop and conference proceedings here presented.

ProTRR: Principled yet optimal in-DRAM target row refresh

Michele Marazzi, Patrick Jattke, Flavien Solt, Kaveh Razavi.

43rd IEEE Symposium on Security and Privacy (SP 2022), San Francisco, CA, USA, May 22–26, 2022.

In this work, I designed ProTRR and ProMG, and performed most of the evaluation. I wrote most of the paper and produced all of the figures.

REGA: Scalable Rowhammer Mitigation with Refresh-Generating Activations

Michele Marazzi, Flavien Solt, Patrick Jattke, Kubo Takashi, Kaveh Razavi.

44th IEEE Symposium on Security and Privacy (SP 2023), San Francisco, CA, USA, May 21–25, 2023.

In this work, I designed REGA and performed most of the evaluation. I wrote most of the paper and produced all of the figures.

HiFi-DRAM: Enabling High-fidelity DRAM Research by Uncovering Sense Amplifiers with IC Imaging

Michele Marazzi, Tristan Sachsenweger, Flavien Solt, Peng Zeng, Kubo Takashi, Maksym Yarema, Kaveh Razavi.

51st IEEE/ACM International Symposium on Computer Architecture (ISCA), Buenos Aires, Argentina, June 29 - July 3, 2024.

In this work, I designed the project scope and performed most of the evaluation. I wrote most of the paper and produced all of the figures.

RISC-H: Rowhammer Attacks on RISC-V

Michele Marazzi, Kaveh Razavi.

4th Workshop on DRAM Security (DRAMSec) co-located with ISCA 2024, Buenos Aires, Argentina, June 29, 2024.

In this work, I designed and implemented the experiments, and performed the evaluation. I wrote the paper and produced all of the figures.

The publications listed in what follows have been part of my research as PhD student, however they are not included in this dissertation.

PayRide: Secure Transport e-Ticketing with Untrusted Smartphone Location

Michele Marazzi, Patrick Jattke, Jason Zibung, Kaveh Razavi.

21st Conference on Detection of Intrusions and Malware & Vulnerability Assessment (DIMVA 2024), Lausanne, Switzerland, July 17-19, 2024.

In this work, I designed and verified PayRide, and performed some of the evaluation for FreeRide. I wrote most of the paper and produced all of the figures.

ZenHammer: Rowhammer Attacks on AMD Zen-based Platforms

Patrick Jattke, Max Wipfli, Flavien Solt, Michele Marazzi, Kaveh Razavi.

33rd USENIX Security Symposium (USENIX Security 2024), Philadelphia, PA, USA, August 14-16, 2024.

BLASTER: Characterizing the Blast Radius of Rowhammer

Zhenrong Lang, Patrick Jattke, Michele Marazzi, Kaveh Razavi.

3rd Workshop on DRAM Security (DRAMSec) co-located with ISCA 2023, Online, June 17, 2023.

ACKNOWLEDGMENTS

Deciding to start a PhD is hard. It requires the motivation and passion to extend your studies beyond a Master's degree, along with the readiness to push the boundaries of science and create new knowledge. I have been lucky in this, thanks to my family, who raised me in an environment that encouraged cultivating my passions and taught me the joy of learning and the value of hard work. Successfully completing a PhD is an achievement that is very hard to accomplish without the support of many great people. This support can take many forms, such as excellent advisory, help with projects, and strong psychological encouragement. I have been very lucky in this regard as well, having been granted all of these.

For these reasons, I would like to thank all the people who, directly or indirectly, helped me reach this achievement.

First, I would like to thank my entire family. My grandmothers, who endured my absence from home but were always ready to prepare amazing Italian meals during the holidays. My parents, who taught me the importance of learning and hard work, supported me throughout my studies, encouraged me to always do my best, and were always there for me through difficult times. My brother, who was the first to suggest that I pursue a PhD, helped me navigate the complex world of universities, and always pushed me to achieve something greater. We share great memories. My girlfriend and adventure companion, who never stopped supporting me when I struggled, always understood my needs during intense work periods, and never ceased to provide love and care. Thanks to all of you —— this journey would have been impossible without your guidance, support, and love.

I would like to thank all my friends from Modena, with whom I have, unfortunately, spent less time than I would have liked, but who were always there for me during tough times. In particular, Enrico Giuliani, Alberto Barchi, and Giulia Guaitoli. I would also like to thank my friends from Zurich, with whom I also wish I had spent more time. In particular, Rémy Mercenier and Emiliano Casalini, with whom I shared a great deal of time on bouldering walls.

I would also like to thank the teachers who, in my younger years, distilled in me the joy of learning and doing things properly: Roberto Zanasi, Anna Maria Prandini, and Giovanna Ghittoni.

Then, I must thank my amazing supervisor, Kaveh Razavi. Kaveh has been the perfect advisor. He understood and envisioned projects related to complex topics that would have long term impact. He took the time to teach me a lot: how to do research, how to write papers, what matters and what not. He also taught me how to present my work in a comprehensible way, and importantly, how to enjoy these moments. Regardless of how many times I knocked on his door (we count these on a per-hour basis), he always found time to discuss things with me.

I am also deeply grateful to the entire COMSEC team, who provided me with great psychological support, invaluable help with projects, and who shared and understood the difficult moments. Special thanks to Flavien Solt, for his partnership in maintaining an impressively messy office, bad jokes, and crazy rants, but also great scientific discussions. To Patrick Jattke, for his ever-positive outlook and invaluable help in writing papers (very) late into the night —— it was great to attend our first conference together. To Finn de Ridder, for our open-minded yet critical discussions. To Katharina Ceesay-Seitz, for bringing more sanity to the group and for her amazing cakes. To Johannes Wikner, for his insights into the complexities of CPUs and the time we spent bouldering. To Silvan Niederer and Sandro Rüegge, for their bright observations and the energy they brought to the group. And to Carmine Rizzi and Jiahui Xu, who, despite being from a different group, were always ready to chat and have fun with us.

Thanks to Maksym Yarema for his clear explanations of imaging technologies. I would also like to thank the students who contributed to some of my projects: Tristan Ballantyne and Jason Zibung. Special thanks to Kubo Takashi, who offered immense help in understanding the intricate details of DRAM internal architecture and always provided a perfect balance of politeness and clarity in his explanations.

I would like to thank Stefan Saroiu for his contributions, both direct and indirect, to the world of Rowhammer mitigations, as well as for his interest and feedback on my work. I am also grateful to Moinuddin Qureshi and Jung Ho Ahn for their valuable discussions and feedback on my research. Thanks to Christophe Deleuze for organizing CSAW ARC Europe, which provided an excellent opportunity to share and discuss ideas. Thanks to Ralf Sasse for his input on Tamarin and Peng Zeng for her assistance with

imaging chips. Thanks to Mariano Graziano for advising me to have my first meeting with Kaveh, and to Beat Futterknecht for his amazing support to both me and the COMSEC group. Thanks as well to Edoardo Talotti for his help with the organization of internal systems.

Finally, a special thanks goes to ETH Zurich for providing an amazing environment, from its top-notch infrastructure to the incredible resources available to us.

Thank you all.

CONTENTS

1

INTRODUCTION

The security of systems necessarily relies on the guarantees of the underlying hardware. Most systems are modular to some degree, allowing the use of a complex variety of hardware devices in a scalable way. That is, a system can be configured based on the required performance, power limitations, or available budget. This modularity has enabled different manufacturers to focus on specific hardware parts, like CPUs or memory components, as long as their products follow shared standards. Furthermore, this division has resulted in specialized manufacturers who are able to push technological improvements year by year. As a consequence, device security guarantees have to be entrusted to the manufacturers, losing the possibility of a global system security overview.

In the age of globalization, manufacturers are incentivized to create cheap and high-performance products, often leaving security guarantees uncertain and undisclosed. This raises many questions. Are these hardware devices affected by security weaknesses? What is the difficulty of patching possible security vulnerabilities? Do companies acknowledge or know about the existence of these vulnerabilities? In this thesis, we first focus on the security of DRAM, and we seek to answer the following question:

> **Leading Research Question:** Can DRAM be made secure against Rowhammer and in a scalable way?

On the security of DRAM. DRAM is a widely used type of main memory in computing systems. It allows systems to rely on large and fast memory at a cheap price. To achieve this, DRAM relies on storing memory on capacitors and exploiting a highly hierarchical internal architecture based on rows and columns. Because of the nature of capacitors, the stored data is slowly lost due to charge leakage. The possible memory corruption is avoided by periodically restoring charge in the entire device. Unfortunately, the push for denser cheap memory has resulted in the DRAM vulnerability known as Rowhammer. With Rowhammer, the capacitors leakage of *victim rows* is increased by accessing nearby data in *aggressor rows*. When

data is read enough times, known as the Rowhammer threshold, bit flips occur in victim rows that are within a blast radius from the aggressor row. Currently, the blast radius is estimated to be ± 2 rows, but it is expected to increase. Rowhammer breaks the expected guarantees of isolation of DRAM, and as the data of aggressor and victim rows can belong to different security domains, this has severe security repercussions. For example, unprivileged software can cause bit flips in kernel memory, leading to privilege escalation.

Rowhammer was initially found by Intel in 2012 [1, 2] on DDR3 devices and later made public by researchers in 2014 [3]. However, a very similar effect is discussed in patents already in 2002 [4]. In response, DRAM vendors marketed DDR4 as Rowhammer-free, without any disclosure of the deployed mitigations. Researchers showed that this security-by-obscurity approach was a failure [5, 6]. The deployed mitigations, that became known as Target Row Refresh (TRR), were identified as insecure and could be bypassed by using complex attack patterns [7]. Because attacks on DDR4 devices require complex and high-performance memory controllers and CPUs, researchers' efforts have mostly been focused on generating bit flips from Intel [5, 7–10] and AMD devices [11].

Meanwhile, the first RISC-V high-end CPU became recently available, supporting DDR4 DIMMs for the first time. Our first research question is the feasibility of Rowhammer on RISC-V systems. No study exists yet, and it is therefore unclear if it is possible to perform Rowhammer on this architecture, or if the success of Rowhammer requires complex and mature devices. Unlike Intel and AMD CPUs, which have benefited from years of design improvements, RISC-V CPUs are in their early stages.

Research Question 1. Is the RISC-V ecosystem affected by Rowhammer?

We answer this question with *RISC-H* in Chapter 2, demonstrating Rowhammer bit flips triggered by a RISC-V CPU for the first time. RISC-H is the result of complex analysis on the memory subsystem of the RISC-V CPU. In particular, we identify a memory bottleneck that makes Rowhammer attacks unsuccessful. After we carefully characterize it, we are able to generate specific Rowhammer patterns that achieve the required high performance. Then, we discover that ordering the attack patterns by fencing or pointer chasing is too expensive on this CPU, prohibiting successful Rowhammer attacks. Such ordering is a requirement for the success of the attack and further makes the RISC-V CPU appear safe. Instead, we devise a

novel method to order memory requests that is based on surgically inserted delays. With RISC-H, we show that the RISC-V ecosystem is also affected by Rowhammer, urging for the deployment of secure mitigations.

Most of the literature has focused on mitigations designed for the memory controller. However, Rowhammer is a DRAM problem and there is no indication that the division and fragmentation between CPU and DRAM vendors is an approach that would result in security against Rowhammer. Further, the undisclosed DRAM internals and security guarantees make the design and deployment of CPU-based mitigations a challenge. Our next research question examines whether DRAM can be secured with an in-DRAM Rowhammer mitigation based on the current technology. Currently, DRAM vendors rely on TRR mitigations that have been exposed as insecure by researchers. We seek to understand if TRR is flawed by design, or if it can be designed in a secure way. Moreover, given the recent DDR5 protocol, we study the possible security enhancement provided by the new standard.

Research Question 2. Can TRR provide security in current and future DRAM devices?

In Chapter 3, we present our Rowhammer mitigation *ProTRR*. We demonstrate that it is possible to implement TRR in a secure way already on DDR4 devices. Then, we show that the recent RFM addition on DDR5 [12] allows us to strongly enhance device security. Through mathematical proofs, we design ProTRR to be secure and optimal in terms of counters for a given Rowhammer threshold. We further prove what is the optimal attack that a malicious user can perform against ProTRR. Our results are valid for cases where perfect counting is used, such as in the latest DDR5 standard that includes PRAC [13].

Due to the synchronous nature of DRAM, the mitigative actions of TRR are periodic. As we showed in our work, this can become a challenge for ProTRR in case future devices become highly vulnerable. In particular, the current trend for Rowhammer indicates the possibility that future technologies will have extremely low Rowhammer thresholds and high blast radiuses. With our third research question, we seek to understand if the internal DRAM architecture can be modified to accommodate a Rowhammer mitigation with security against high degrees of Rowhammer vulnerability.

Research Question 3. Is it possible to secure DRAM with an in-DRAM Rowhammer mitigation that scales with the increase of the blast diameter and Rowhammer vulnarability?

We show that this is possible with our mitigation *REGA* in Chapter 4. By designing a new internal DRAM architecture, we are able to provide security against Rowhammer even for thresholds under 1 K and the blast radius up to 4. We designed REGA to perform refreshes in parallel to row activations, therefore eliminating the periodicity constrain of TRR mechanisms. REGA does not require expensive counters, and instead relies on the addition of new sense amplifiers in the circuitry that multiplex shared DRAM elements. We verify the feasibility and reliability of our architecture through analog electric simulations. Because no reliable DRAM model exists, we collaborated with a DRAM vendor (Zentel Japan) to design a new one that is modern and reliable (REM), which we further open source. REM is the first accurate modern DRAM model available for research. With REM, we discover that the literature on DRAM architecture appears to be outdated. First, the only other modern DRAM model available is severely optimistic about its electrical values. Second, the circuitry used on real chips by Zentel Japan includes an overdriving component not considered in research.

We designed REGA based on the collaboration with a minor DRAM vendor, but its applicability to commodity vendors remains unclear. With our fourth question, we therefore wish to understand how REGA and previous DRAM research relate to modern commodity devices.

Research Question 4. Is current DRAM research applicable and accurate on real modern DDR4 and DDR5 devices?

In Chapter 5, with our work *HiFi-DRAM*, we image and reverse engineer DDR4 and DDR5 devices from the three major DRAM vendors. By analyzing ten years of research, we discover fundamental inaccuracies that have been shared across DRAM research. First, the commonly assumed DRAM circuitry has been replaced in half of the studied chips. Second, no existing analog DRAM model correctly captures the devices characteristics. Third, the overhead estimations presented in papers are often severely inaccurate. We make all our extracted data open source in the hope of enabling more rigorous future DRAM research.

By answering all these research questions, we demonstrate that a principled approach to DRAM security is fundamental to mitigate Rowhammer and to reliably study its feasibility. With careful characterization, we were able to trigger Rowhammer bit flips on a novel architecture that would otherwise appear secure. Then, by following rigorous security principles, we designed mitigations even for technologies considered flawed, such as TRR on DDR4, and expanded DRAM security into the long-term future. Moreover, the accuracy of DRAM research and its viability was severely weakened by questions that research had previously left unanswered. By reverse engineering modern DRAM devices, we were able to fill that void and to enable more accurate research in the future.

2

RISC-H: ROWHAMMER ATTACKS ON RISC-V

The first high-end RISC-V CPU with DDR4 support has been released just a few months ago. There are currently no Rowhammer studies on RISC-V devices and it is unclear whether it is possible to compromise systems on these newer architectures. With RISC-H, we aim to fill this gap by overcoming a number of challenges: first, the DRAM functions of the memory controller are not disclosed, which we reverse engineer via the bank-conflict side channel. Second, we discover that hammering many rows achieves a significantly low activation throughput, making Rowhammer unsuccessful. We determine that this low performance is caused by a contention in the memory subsystem when aggressor rows share certain physical address bits and slow ordering instructions. To address this challenge, we leverage different column addresses to reduce contention, and we rely on a novel approach for ordering memory accesses by inserting surgical delays in the access patterns. Combining these insights, our new Rowhammer attack, called RISC-H, can trigger the first DDR4 bit flips from a RISC-V CPU. These results show that the RISC-V ecosystem is similarly affected by Rowhammer and further highlights the need for effective mitigations.

2.1 INTRODUCTION

Rowhammer on modern DRAM has mostly been successful on high-end complex CPUs from Intel [5, 7–10] and AMD [11]. These mature products, resulting from years of improvements, render instruction execution fast and the memory controllers complex. RISC-V CPUs are comparably in their early stage and it is unclear whether it is possible to trigger Rowhammer bit flips from these platforms. Our results on the first RISC-V processor with DDR4 support [14] shows a very low activation throughput as well as expensive ordering instructions, impeding Rowhammer attacks. This paper shows how the careful selection of physical addresses and the insertion of surgical delays for ordering allows ordered activations with high throughput, enabling the first successful Rowhammer attack on RISC-V.

DRAM functions. Rowhammer causes disturbance errors on *victim* rows that are in close proximity of an *attacker* row. Memory controllers map physical addresses to DRAM banks and rows using proprietary functions, exploiting DRAM parallelism to increase performance. Hence, the success of a Rowhammer attack relies on reverse engineering these functions. In RISC-H, we use the bank-conflict side channel [15] to reverse engineer the bank and row functions. We discover that our target RISC-V CPU employs a linear mapping instead of the common XOR-based functions of Intel and AMD CPUs for bank addressing [11, 15].

Activation throughput. A key aspect for a successful Rowhammer attacks is a high activation rate generated by the memory controller to both induce bit flips and to bypass deployed mitigations [8, 11, 16, 17]. On the Sophon CPU [14], we discover that subsequent accesses to certain memory accesses are surprisingly slow. Our characterization shows that this is caused by a contention in the memory subsystem when memory addresses share columns bits. Consequently, we are able to increase the memory activation rate by distributing subsequent memory requests among different columns.

Memory ordering. Memory requests are reordered by the memory controller to reduce the number of generated activations. First, this lowers the effective activation throughput, required for Rowhammer. Second, DDR4 devices deploy Target Row Refresh (TRR) as a Rowhammer mitigation [5, 6, 18, 19], which can be bypassed by activating aggressor rows in complex patterns [5, 7]. To prevent the memory controller from reodering these patterns and making them ineffective, researchers employ fencing instructions [7] or pointer chasing [9]. We show that both these options significantly reduce the activation rate on the RISC-V CPU. Instead, we devise a novel approach to order memory requests. We make the key observation that memory ordering can be achieved by carefully delaying the requests. By exploiting the row buffer hit as a side channel, we are able to identify the right amount of delay, which we induce via NOP instructions.

We combine our insights to build RISC-H, the first Rowhammer attack on RISC-V. RISC-H is able to obtain 841 bit flips in 6 h of fuzzing on a supported DIMM. These bit flips are highly repeatable (on average, 74% of the times), enabling reliable Rowhammer exploitation.

Contributions. The following summarizes our contributions:

1. We reverse engineer the proprietary DRAM functions of the Sophon memory controller.

2. We identify, characterize, and describe how to avoid a new memory bottleneck that severely slows down the activation rate.
3. We devise and demonstrate a novel method to efficiently order memory requests by surgically-inserted delays.
4. We demonstrate Rowhammer bit flips on RISC-V for the first time.

2.2 BACKGROUND

In this section, we introduce DRAM (§2.2.1), Rowhammer (§2.2.2), and Rowhammer-required CPU primitives (§2.2.3).

2.2.1 *DRAM*

DRAM devices are used as main memory in current high-end computing systems. These devices provides fast, dense and cheap memory. The DDRx protocol [20] describes how the CPU memory controller (MC) can access DRAM, with chips typically mounted on Dual In-line Memory Modules (DIMMs, Fig. 2.1). Currently, the majority of DIMMs are DDR4 devices, with the new DDR5 standard [12] being released a few years ago.

DRAM Hierarchy. The MC needs to respect a logical hierarchy to access memory. Internally in a DRAM chip, memory is obtained as densely packed capacitors distributed across different DRAM banks [21, 22]. Each bank has multiple rows, and each row has many data columns. When the MC accesses memory, it needs to specify the bank, the row and the column associated to the data. Prior to accessing the specific column, the MC needs to issue an activation (ACT) to the row [23]. Each bank can only have one row activated at a time. To deactivate an active row in a bank, the MC issues a precharge (PRE). Once the row has been activated, the MC can finally read or write data by specifying a column. Further reads/writes to columns of an activated row do not require additional activations and are described as *row buffer hits*. Due to the internal DRAM circuitry, activating and precharging rows is slow. For this reason, MCs will typically try to optimize the order of generated memory requests such that ACTs and PREs are reduced. This reordering is done by using a memory request buffer.

DRAM Capacitors. DRAM is based on capacitors, which are elements capable of storing charge. A single capacitor allows to encode a single bit of data via the stored charge. Due to the manufacturing capabilities of integrated

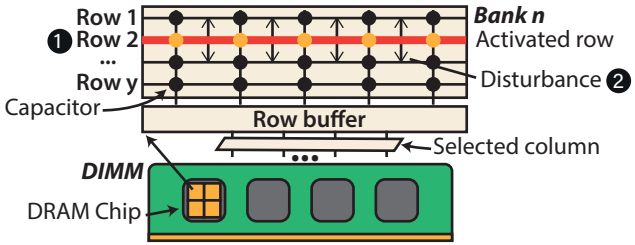

FIG. 2.1: **DRAM Architecture and Rowhammer.** Memory is organized in many rows in different banks. A row ACT **(1)** causes disturbance to nearby data **(2)**.

circuits (ICs), these capacitors are extremely small and compact [21]. This compactness is kept by the highly hierarchical internal structure, which enables memory chips with large addressability and cheap price. Because capacitors leak charge, MCs need to send a refresh command (REF) every tREFi (7.8 us on DDR4) such that the DRAM chip has time to restore rows to their full values. An entire chip is fully restored after 8192 refreshes (tREFW, 64 ms on DDR4). Without these refreshes, stored data would get corrupted.

2.2.2 *Rowhammer*

The highly-packed capacitors require very dense circuitry to provide the market with cheaply-produced memory. Unfortunately, such IC scaling has come with drawbacks in terms of memory reliability. In 2014, researchers demonstrated that aggressor-row activations can have an effect on nearby victim-rows on DDR3 devices [3]. The effect, known as Rowhammer, causes an increased charge leakage that can corrupt victim data without directly accessing it. To trigger Rowhammer, aggressor rows that are physically nearby a victim row are activated repeatedly for a large number of times, known as *Rowhammer threshold* (e.g., 50 K), before the victim row is refreshed by a REF command. This vulnerability has been subsequently exploited in many different ways from different attack vectors [9, 10, 24–27] and deeply characterized [17, 28–34]. As a response, DRAM vendors have deployed Rowhammer mitigations known as Target Row Refresh (TRR) on DDR4 devices. TRR implements Rowhammer detection mechanisms, which are followed by the refresh of the victim row [5, 6]. On DDR4 devices, TRR

has been shown to be flawed when advance row activation patterns are used [7]. Research efforts have provided industry many different Rowhammer mitigations based on alternative approaches [16, 22, 23, 35–44].

2.2.3 *CPU Primitives*

To reliably perform Rowhammer on DDR4 devices from different vendors, researchers have relied on two key CPU primitives. First, the activation rate (i.e., how many ACTs per second the MC issues) is maximized [8, 11, 16, 17]. Second, the aggressor rows that are activated are based on complex patterns to bypass TRR [6, 7]. To ensure the row activation order, researchers either used pointer chasing or CPU fencing instructions [7, 9, 11].

2.3 OVERVIEW

With our research, we aim to successfully flip DRAM bits via Rowhammer on a RISC-V CPU for the first time. To achieve this goal, we face multiple challenges.

First, the MC maps physical addresses to specific banks and rows in a way that is not disclosed. This mapping is fundamental to perform Rowhammer, as the aggressor and victim rows need to be placed in physical proximity.

Challenge 1. Reverse engineer the MC DRAM functions to link physical addresses to DRAM addresses.

We solve challenge 1 in Section 2.4 by using the bank conflict side channel [15]. For this purpose, we implemented a multi-thread counter to measure time.

A high activation rate by the MC is a key Rowhammer primitive. We seek to understand if the RISC-V CPU is capable of generating a high activation throughput while allowing for complex row activations patterns.

Challenge 2. Maximize the DRAM activation rate (i.e., ACT/s) without losing aggressors pattern generality.

We address this challenge in Section 2.5. We identify a memory bottleneck that substantially slows down the ACTs rate. In particular, subsequent memory accesses that share particular bits of the physical address create

Processor		Cores		DRAM	
Vendor	Sophon	Vendor	T-Head	Vendor	X
Model	SG2042	Model	C920	Memory	8 GB
Memory	DDR4	Frequency	2GHz	Total Banks	16
System Cache	64MB	L2 cache	1MB	Rows/Bank	64 K
Core Clusters	16	Cores/clusters	4	Production Yr.	2018

Tbl. 2.1: Hardware used in RISC-H.

contention on the memory subsystem. By distributing temporally-close memory accesses across different columns, we are able to heavily increase the activation throughput.

The last challenge is to keep a high row activation rate without losing control over the order of row activations. As the RISC-V CPU might not be as complex as Intel and AMD counterparts, we identified that fencing and other common ways to hold activation order intact have a huge toll on the activation throughput.

Challenge 3. Maintain the order of row activations, without relying on fencing or pointer chasing.

We assume that the MC scheduling policy is based on some variation of the First-Ready First-Come-First-Serve (FR-FCFS) strategies as these are considered standard [45–48], and we solve challenge 3 by carefully delaying memory requests, forcing activation ordering. By exploiting the row hit as a side channel, we prove that our approach is keeping the intended activation order.

By combining all these aspects, we are able to produce Rowhammer bit flips in a fast and reproducible way, obtaining 841 bit flips in 6 hours of fuzzing on a supported DIMM.

2.4 REVERSE ENGINEERING OF DRAM FUNCTIONS

We report the configuration of our system in Tbl. 2.1. We use the well-known bank conflict side channel to determine the DRAM functions [15, 49, 50]. As this is a timing side channel, we require an accurate and precise method to measure time. RISC-V provides an instruction called `rdcycle`, which is supposed to return the number of cycles executed by the CPU. However,

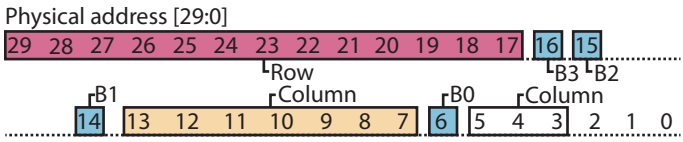

FIG. 2.2: **Reverse engineered DRAM Functions.** The memory controller applies a linear mapping of the bits 6, 14, 15, and 16 to the DRAM banks.

the RISC-V instruction `rdcycle` returns a constant value on our system. We build a counting thread, similar to previous work [51], obtaining a timer resolution of 4.85 ns.

Results. We use the recently released tool AMDRE [11] to reverse engineer the DRAM functions allocating a GiB super page. We adapt AMDRE to rely on shared memory as a counter and to use the RISC-V `fence` instruction. Further, we modify it to use average instead of the minimum and to use more repetitions to reduce noise. We report the results in Fig. 2.2. In the RISC-V CPU under evaluation, the mapping functions are linear. This differs from the complex XOR-based functions reported for Intel and AMD [11, 15]. We identify column bits by exploiting the row buffer hit, accessing couples of addresses of the same bank where one bit differs. We categorize the bit as column if the access is fast and classify the remaining as rows bits.

2.5 MAXIMIZING THE ACTIVATION RATE

A key aspect of successful Rowhammer bit flips is a high ACT rate generated by the memory controller. Given the DDR4 standard, the minimum time between different ACTs to the same bank is given by `tRC` [20] which for our system is 46.5 ns. We seek to understand if the RISC-V CPU is capable of generating such a high amount of activations per second.

Baseline performance. To evaluate the activation throughput of the CPU, we measure the time to access an array of 256 different rows that target the same bank. The rows are different, hence we do not require pointer chasing or fencing to avoid memory requests reordering. Such reordering would inflate the throughput by reducing the number of activations, exploiting row buffer hits. We discover that the average access time per row is severely slow, around 210 ns. We repeat the experiment to understand if there is a dependency to the number of accessed rows. The access time saturates to 210 ns with a high number of accessed rows, while it is slightly faster

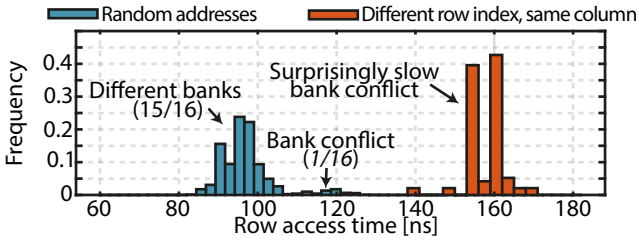

FIG. 2.3: **Histogram of access times.** We report the row access time for couple of random addresses and for addresses that differ only for the row index.

(180 ns/ACT) when very few rows are accessed (e.g., 12 rows). All these values are much higher than the tRC of the system, making a Rowhammer attack unlikely to succeed.

We speculate that the design of the CPU might not be as optimized as for Intel and AMD devices, resulting in contention on a particular microarchitectural resource. Specifically, we formulate the hypothesis that this contention is address-dependent, for example, due to accessing internal cache blocks, other sub-blocks, or due to the internal MC design. We devise the following experiments to investigate this effect, in which we use contiguous memory obtained from a GiB super page.

Identifying the memory bottleneck — first experiment. Our aim is to identify if two subsequent memory requests have resource contention, and if this contention can be avoided by varying their addresses. To this end, we first generate addresses that only differ for the row index. Given the previous results, we expect their combined access time to result in a bank conflict. We show the results in Fig. 2.3, which further includes the histogram of the bank-conflict side channel previously used. Surprisingly, the access time of different rows is always slower than the bank-conflict time. Note that the access time is slightly lower than in the previous experiment (160 ns compared to 180 ns), as we directly dereference registers for a more controlled experiment instead of accessing an array in a loop.

Identifying the memory bottleneck — second experiment. The results from AMDRE are generated by using random couples of addresses. These random couples cause bank conflicts, yet their access time is faster than our experiment. Therefore, as we are now using almost identical addresses, we wish to see if particular parts of them are causing contention that increases the slowdown. In what follows, instead of changing only the row index,

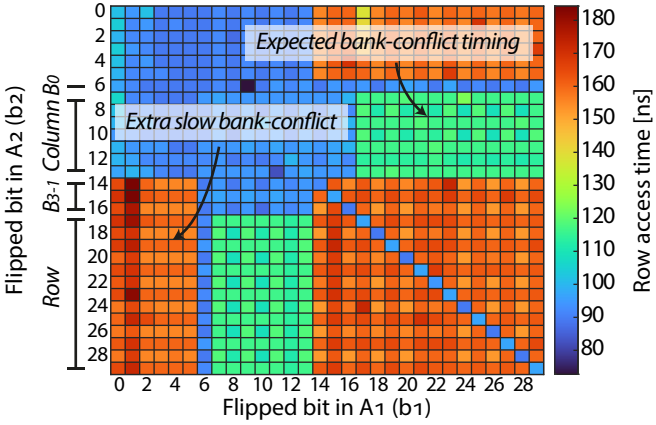

FIG. 2.4: **Time access experiment.** We access two addresses (A_1, A_2), where $A_1 = base \oplus b_1$ and $A_2 = base \oplus b_2$, and we report the row access time.

we vary each address bit at the time. Starting from a base address, we measure the combined access time of two addresses, A_1 and A_2, where $A_1 = base \oplus b_1$ and $A_2 = base \oplus b_2$. We test all combinations of b_1 and b_2 between $b_0 - b_{29}$, and report the results in Fig. 2.4. From the results we identify three categories of access speed: (i) fast, (ii) semi-slow, and (iii) slow. Semi-slow timings correspond to the bank-conflict found during the DRAM function reverse engineering. Instead, slow timings correspond to the "extra" slow access.

Address dependency of the bottleneck. By flipping a bank bit (e.g., b_{14}) only in one address, we would expect the timing to be always fast, as A_1 and A_2 would target different banks. Instead, in many of the combinations the access time is slow. By varying bits between $b_7 - b_{13}$, different bank accesses result (as expected) in fast memory accesses. By varying row bits (e.g., b_{19}) and any bit between $b_7 - b_{13}$, we obtain the expected bank conflict timing. We conclude that subsequent accesses of addresses with identical bits $b_7 - b_{13}$ are severely slowed down. Coincidentally, these bits correspond to the column bits of a row. We now investigate how many different columns, in an access loop, are required to obtain a high ACT rate.

Columns dependency. We distribute from 1 to 128 different columns to 256 different rows accessed in a loop. For each setup, we report the average access time *per row* in Fig. 2.5. After a few columns, the access time saturates close to the protocol limit (`tRC`). For simplicity, in the remainder of the paper,

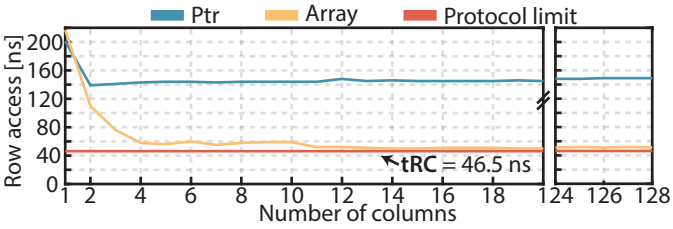

FIG. 2.5: **Average access time when using different columns.** Pointer chasing (Ptr) access time is severely slower compared to Array access (Array). When multiple columns are used, the access time of Array becomes close to tRC.

we will always linearly distribute the columns of accessed memory region across all possibilities (i.e., 128). The reader should note that this is required *independently* from the accessed rows (i.e., if the pattern is accessing the same row or different ones).

2.6 ENFORCING MEMORY REQUESTS ORDER

We now analyze the impact of enforcing memory requests order on the activation rate.

Effect of memory requests reordering. MCs try to reduce activations by reordering memory requests that would target the same row. For Rowhammer attacks, this has two damaging effects. First, the *effective* ACT throughput will be reduced, as rows will be kept open longer than required. Second, state-of-the-art Rowhammer patterns are complex and require precise ordering of aggressor rows to fool the deployed TRR mechanisms [11]. If requests are reordered, these pattern would get scrambled, reducing the possibility of Rowhammer success.

Speed of pointer chasing and fence. The two main ways to order memory requests is to use pointer chasing and the `fence` instruction. We now evaluate their speed on our RISC-V CPU, by measuring the average access time of 256 addresses. Accesses divided by `fence` have an average speed of 210 ns/ACT, regardless of the number of columns used. Pointer chasing increases its speed when temporally-close accesses are distributed among different columns, however, the access is still severely slow saturating at around 145 ns/ACT (Fig. 2.5).

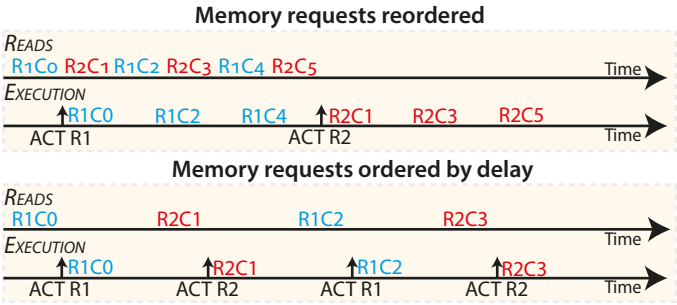

FIG. 2.6: **Memory requests ordering.** In the first case, no ordering is enforced and different memory requests are merged in single activations. In the second case, delayed memory requests induce multiple activations.

In Section 2.5, we observed that the system is capable of reaching high access speed (55 ns/ACT). Therefore, we cannot rely on `fence` and pointer chasing as strategies for preserving the order of memory requests. With our novel delayed memory requests, we now demonstrate how it is possible to reach high speed while ensuring ordering.

2.6.1 *Memory ordering via delayed accesses*

We assume the MC scheduling policy to be based on a variation of First-Ready First-Come First-Serve (FR-FCFS), as these are considered standard strategies [45–48]. When these policies are employed, the memory request buffer will be used to batch requests that go to the same rows, incrementing row hit and decreasing the number of ACTs (Fig. 2.6) [45].

We aim to enforce memory ordering by delaying memory requests. If the request buffer does not (yet) contain requests that can be merged, it should eventually perform the activation. Likewise, if a request has been in the queue for a long time (i.e., old request), it should be prioritized over new memory requests to avoid starvation [46]. We explain this concept in Fig. 2.6.

We perform our technique by placing NOPs in between the different requests. As the CPU does not expose any relevant performance counters (e.g., issued activations), we must rely on a carefully crafted experiment to validate our method and to understand for how long the requests should be delayed for.

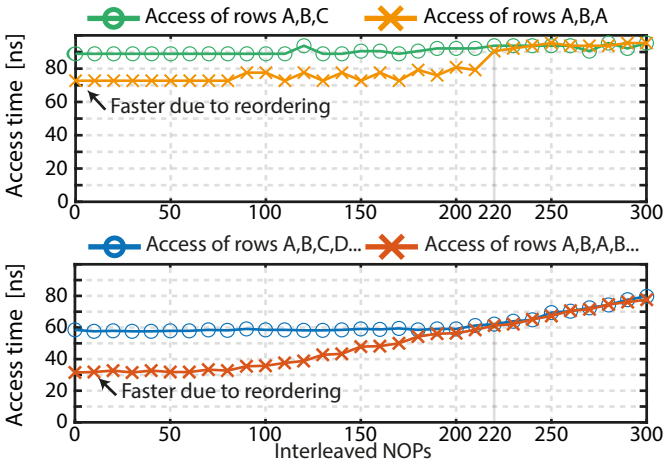

FIG. 2.7: **Delayed access experiment.** We measure 3 ACTs (A,B,C) and compare it to 3 requests that can be sped-up by reordering (A,B,A). When enough NOPs are inserted, the MC does not reorder accesses anymore. We repeat the experiment with 16 different rows compared to the access pattern (A,B)×8.

NOPs-delayed requests experiment. An access pattern with 3 different rows (e.g., `A-B-C`) forces the MC to issue 3 ACTs. Instead, a pattern that targets the same row twice (e.g., `A-B-A`) allows the MC to reorder requests, resulting in only 2 ACTs and a row buffer hit. We perform an experiment to measure this difference. In Fig. 2.7, we report the average time per memory request of the two cases. The pattern `A-B-A` is evidently faster due to memory reordering. If our hypothesis is correct, the pattern `A-B-A` should generate 3 ACTs when memory requests are delayed long enough. We interleave the accesses with a varying number of NOPs, from 0 to 300, and report the results in Fig. 2.7. When around 220 NOPs are inserted, the pattern `A-B-A` starts behaving as expected.

Now, we generalize the experiment and measure the effect when *multiple* reordering are possible, as this is the typical Rowhammer case. We repeat the experiment by accessing 16 different rows (e.g., `A-B-C-D...`) compared to eight times the couple `A-B`. As previously found, around 220 NOPs the behavior converges to the expected timing (Fig. 2.7).

Finally, we evaluate the overhead of our technique. Repeating the experiment of Fig. 2.5, we measure the average access time for a loop of different

rows with 220 NOPs interleaved. The result is an average slow down of only 3 ns per access, which shows that our new ordering technique has very low overhead and is viable to perform Rowhammer on the RISC-V CPU.

2.7 RISC-H

We combine all the previous observations and results to perform Rowhammer on the first high-end RISC-V CPU. We now explain the setup and the results.

Setup. We allocate 1 GiB of memory as transparent huge pages (2 MiB) and verify that all the pages result in bank conflict. This is expected, as the highest bank bit is the 16th (i.e., lower than 2 MiB, Fig. 2.2). Then, we initialize the 1 GiB of memory with 32 different row patterns of 8 bytes, selected by hashing the memory address. This allows faster execution times compared to the performance toll of using randomized memory to check for bit flips. The patterns are based on classic repetitions of `0xAA`, `0xEE`, `0x00`, `0xFF`, and variations.

We perform Rowhammer by generating non-uniform patterns, as they represent state-of-the-art [7, 11]. We do not use Blacksmith as a dependency (asmjit) is not yet ported to RISC-V, but instead rely on our own implementation called RISC-H. All memory requests are interleaved by 220 NOPs and linearly distributed across 128 columns. We fuzz each pattern for a duration of $4 \times tREFW$. After fuzzing, we check for bit flips for only the addresses that correspond to the same targeted bank and whose rows are nearby the aggressors. This further improves the performance of RISC-H. Once we identify a bit flip, we test its repeatability by testing the same pattern 10 times for double the time. We further test hammering by using `fence`, pointer chasing, and without any ordering. All the experiments take place in a controlled environment at 23° C.

Results. We discover that on the tested DIMM, we are able to trigger bit flips with double-sided patterns (i.e., frequency-based patterns are not necessary) of which we report the results. With our complete approach to RISC-H, we obtain 841 unique bit flips in only 6 h of fuzzing. The first bit flip occurs within one minute after the fuzzing is started (i.e., after data has been initialized to DRAM). Bit flips have a high repeatability, with an average success rate of 7.4 out of 10 times.

We confirm that we did not obtain bit flips when the ordering was absent (i.e., no NOPs), or when it was enforced by `fence` or by pointer chasing. In conclusion, our delayed-request ordering is necessary to get Rowhammer bit flips on the CPU.

Limitations and future work. We evaluated RISC-H only on one DIMM. This is due to the extremely limited memory support provided by the CPU. We tested 85 DDR4 DIMMs present in our lab, of which only one resulted in the system booting. We tried to clone the SPD values of the working DIMM to different modules with the same timings and geometry, and we further tried 22 SODIMMs connected via an adapter. Unfortunately, this also did not result in booting. Future studies should assess RISC-H on multiple devices in case the DIMM support will be extended.

2.8 CONCLUSION

With RISC-H, we proved that Rowhammer bit flips are possible on the first high-end RISC-V CPU. To perform Rowhammer, we had to overcome multiple challenges. First, we identified the undisclosed DRAM functions by exploiting the bank-conflict side channel. Then, we discovered a memory bottleneck related to the address of subsequent memory requests, which made the ACT rate insufficient to trigger bit flips. As a solution, we distributed the activations across the many row columns. Lastly, we devised a novel technique to enforce memory requests order with high performance. By carefully delaying accesses with NOP instructions, we enforce row activations without relying on slow fence instructions or pointer chasing. To the best of our knowledge, we are the first to demonstrate bit flips on a RISC-V CPU.

3

PROTRR: PRINCIPLED YET OPTIMAL IN-DRAM TARGET ROW REFRESH

The DRAM substrate is becoming increasingly more vulnerable to Rowhammer as we move to smaller technology nodes. We introduce PROTRR, the first principled in-DRAM Target Row Refresh mitigation with formal security guarantees and low bounds on overhead. Unlike existing proposals that require changes to the memory controllers, the in-DRAM nature of PROTRR enables its seamless integration. However, this means that PROTRR must respect the synchronous nature of the DRAM protocol, which limits the number of DRAM rows that can be protected at any given time. To overcome this challenge, PROTRR *proactively* refreshes each row that is most likely to observe bit flips in the future. While this strategy catches the rows that are hammered the most, some others may still *fly under the radar*. We use this observation to construct FEINTING, a new Rowhammer attack that we formally prove to be optimal in this setting. We then configure PROTRR to be secure against FEINTING. To achieve this, PROTRR should keep track of accesses to each row, which is prohibitively expensive to implement in hardware. Instead, PROTRR uses a new frequent item counting scheme that leverages FEINTING to provide a provably optimal yet flexible trade-off between the tolerated DRAM vulnerability, the number of counters, and the number of additional refreshes. Our extensive evaluation using an ASIC implementation of PROTRR and cycle-accurate simulation shows that PROTRR can provide principled protection for current and future DRAM technologies with a negligible performance, power, and area impact. PROTRR is fully compatible with DDR4 and the new Refresh Management (RFM) extension in DDR5.

3.1 INTRODUCTION

Despite numerous mitigation attempts under Target Row Refresh (TRR), Rowhammer is still an unsolved problem in practice [5–7], threatening systems security in many different scenarios [9, 24, 25, 49, 52–58]. Existing proposals attempt to mitigate Rowhammer in the memory controller [3, 41,

42, 59–61], but CPU vendors have little incentive to introduce expensive mitigations for a problem in the products of DRAM vendors. The natural place to fix Rowhammer is inside DRAM itself, but mitigations with strong security guarantees are currently lacking.

We present PROTRR, the first principled in-DRAM Rowhammer mitigation that is secure against FEINTING, a novel Rowhammer attack that we mathematically prove to be optimal. PROTRR uses the bounds given by FEINTING in the design of a new frequent item counting scheme, called PROMG (Proactive Misra-Gries), with a provably optimal yet flexible trade-off between the number of required counters and additional refreshes. Our extensive evaluation of PROTRR using an ASIC implementation and cycle-accurate simulation shows the feasibility of principled in-DRAM Rowhammer protection for current and future DRAM technologies.

Rowhammer. In their seminal work, Kim et al. [3] showed that by repeatedly activating a DRAM row (i.e., aggressor), it is possible to flip bits in its adjacent rows (i.e., victims) before these rows have a chance to be refreshed as part of the background DRAM refresh operation. This effect is present in most DDR3 devices and has only worsened in DDR4 devices deployed on more recent systems [5–7, 17]. In essence, Rowhammer is compromising the isolation of data on DRAM. A plethora of attacks followed, showing that it is possible to abuse these bit flips to escalate privileges [24, 55, 56], compromise browsers [9, 52–54], break into co-located virtual machines in the cloud [25, 49], and even attack servers over the network [57, 58]. These attacks highlight the urgent need for strong mitigations against Rowhammer.

Mitigations. Originally, two practical countermeasures were believed to stop Rowhammer: doubling the DRAM's refresh rate and error-correcting code (ECC) DRAM. Unfortunately, neither can fully protect systems [16, 62]. There are also proposals to mitigate Rowhammer in software [16, 56, 63, 64], but these solutions have security and performance issues [52, 55, 65]. To mitigate Rowhammer in hardware, previous work mostly proposes to modify the memory controller to detect potential aggressors and refresh their victims [3, 41, 42, 59, 61]. Unfortunately, due to their substantial cost, CPU vendors are reluctant to deploy these mitigations given the promise of Rowhammer-free devices by the DRAM vendors [66, 67]. However, without carefully analyzing the security implications of performing TRRs inside DRAM, there will be gaps in the protection, as evident in recent work [5–

7, 9, 41]. These gaps will only worsen with the increasing Rowhammer vulnerability in newer DRAM generations with smaller technology nodes.

FEINTING. In this paper, we advocate for a principled approach for designing secure in-DRAM mitigations. In-DRAM mitigations allow for seamless system integration, but they need to strictly adhere to the synchronous DRAM timing specifications defined in the DDRx standard [12, 20]. For example, a DRAM refresh command cannot suddenly take longer when the system is under attack. This means that any in-DRAM mitigation can only protect a handful of victim rows at any given point in time. Consequently, even with an ideal in-DRAM TRR scheme that always protects rows that are hammered the most, an attacker can use *decoy* rows to slowly increase the number of times a victim is hammered without it ever being subject to the mitigation. We use this observation to construct FEINTING, a novel Rowhammer attack that we mathematically prove to be optimal against an ideal in-DRAM mitigation. FEINTING enables us to calculate strict bounds on the degree of Rowhammer vulnerability that can be tolerated on any compliant DDR4 device and future DDR5 devices that use Refresh Management (RFM), a new extension that is primarily introduced in the DDR5 standard to address Rowhammer [12]. To the best of our knowledge, this is the first work to define and calculate these crucial bounds.

PROTRR. Counting the activations of each row for an ideal in-DRAM mitigation is too expensive to implement in hardware. Existing frequent item counting schemes can reduce the number of necessary counters when frequent items need to be identified over an arbitrary sequence of row activations [41]. Unfortunately, these schemes are unsuitable for in-DRAM TRR which needs to proactively protect target rows based on the information that is available at short intervals. We develop Principled yet Optimal Target Row Refresh (PROTRR), a new in-DRAM Rowhammer mitigation that we prove is both secure and optimal in this setting. PROTRR makes use of a new frequent item counting scheme, called PROMG, that adapts FEINTING to right-size Misra-Gries summaries [68] for secure in-DRAM operation. Our calculations show that the insights from FEINTING enable PROTRR to significantly reduce the required number of counters with slight changes to the Rowhammer tolerance. This property provides PROTRR with an unprecedented flexibility: depending on the degree of Rowhammer vulnerability, a DRAM vendor can decide how to balance the number of counters and in-DRAM refreshes for keeping its DRAM devices secure. Furthermore, we provide a proof that PROTRR is optimal in terms of counters and the required refreshes at any given configuration; fixing the number of

refreshes, any in-DRAM mitigation that uses fewer counters than PROTRR will be vulnerable to Rowhammer. Similarly, fixing the number of counters, any in-DRAM mitigation that uses fewer refreshes will also be vulnerable.

Our extensive evaluation using an ASIC implementation and cycle-accurate simulation shows that PROTRR provides principled protection with a negligible performance, area, and power impact. For example, PROTRR can protect a DDR5 device where bits flip after only 3,200 activations, with less than 0.2% performance overhead, while increasing the area by 1.78% and energy consumption of DRAM by 2.35%.

Contributions. We make the following contributions:

1. The construction of FEINTING and a mathematical proof of its optimality against an ideal in-DRAM TRR.
2. The design of PROTRR, a principled in-DRAM TRR that is secure against FEINTING while providing a provably-optimal yet flexible trade-off between the required counters and refreshes.
3. A comprehensive evaluation of PROTRR using (i) an ASIC implementation in a popular 12 nm technology for measuring its area and power requirements in DDR4 and DDR5 devices, and (ii) cycle-accurate simulation for measuring its performance overhead when using the recently introduced RFM extension in DDR5.

3.2 BACKGROUND

We briefly discuss the architecture and operation of a DRAM device (§3.2.1) before discussing the Rowhammer vulnerability (§3.2.2). We then introduce the current proposals for mitigating Rowhammer and discuss their limitations (§3.2.3). We kindly refer the reader to Tbl. 3.4 (Appendix 3.12) for a summary of all symbols introduced in this and following sections.

3.2.1 *DRAM architecture*

The architecture of DRAM and its basic operation is depicted in Fig. 3.1. Like most memory devices, a principal abstraction in DRAM is the association of data with its address. A DRAM address traverses a hierarchy, starting with a *channel* and continuing to a specific connected DRAM device. Once a device is selected, the data address is further used to identify a *rank* and then a specific *bank* within that rank (Fig. 3.1a). Each bank is a matrix

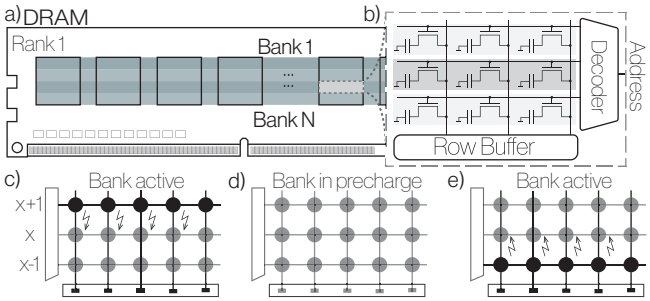

FIG. 3.1: **DRAM architecture and relevant DRAM operations.** (a) the rank/bank hierarchy in a DRAM device, (b) row addressing after rank/bank selection, (c) activating a row $X+1$ in a bank using `ACT` to bring its content to the row buffer, (d) deactivating the row in the row buffer using `PRE`, (e) activating another row $X-1$. Repeated activation of rows $X+1$ and $X-1$ can potentially trigger Rowhammer bit flips in row X.

of cells that stores information using a capacitor (Fig. 3.1b). When data has to be read or written, its associated row has to be activated using the DRAM `ACTIVATE` (`ACT`) command, which connects the row to the *row buffer* (Fig. 3.1c), making the bank *active*. To deactivate a bank, the DRAM `PRECHARGE` (`PRE`) command is used. The memory controller can decide when to send the `PRE` command based on a policy. With a closed-page policy, the memory controller sends the `PRE` command right after or with the DRAM access. In contrast, with an open-page policy, the memory controller can delay the `PRE` command. Internally and transparently to the outside world, banks can further be divided into subarrays [69]. Each subarray has its own local row buffer, which is connected to the bank's row buffer. Subarrays allow for parallelization of certain DRAM operations such as the `REFRESH` (`REF`) command. Because of the physical nature of capacitors, their charge constantly leaks. To preserve their value, the CPU's memory controller periodically sends `REF` commands to DRAM, which triggers an internal refresh mechanism. Each issued `REF` only covers a fraction of the addresses. The JEDEC DRAM standard requires each row to be refreshed at least once in a `tREFW` and the memory controller to issue `REF`s at intervals defined by `tREFI` [12, 20]. As an example, if `tREFW` equals 64 ms and `tREFI` equals 7.8125 µs, the memory controller needs to send a total of 8192 `REF` commands in a `tREFW`.

3.2.2 *Rowhammer*

Thanks to continuous improvements in process technology, we observe an increased DRAM chip density each year. Unfortunately, this comes at a reliability cost [70]. As DRAM rows get closer to each other, their electrical isolation gets compromised. Rowhammer is an attack based on repeated row activations [3] that causes cells in nearby rows to leak charge and eventually change their stored values (i.e., bits flip). The row with repeated activations is commonly referred to as the *aggressor* row. The repeated activations of an aggressor row affect its neighboring rows, which are commonly referred to as *victim* rows. A variant of this attack where a victim row is sandwiched between two aggressor rows, known as *double-sided Rowhammer*, is depicted in Fig. 3.1c-e. Recently, it has been shown that an aggressor row can influence victims that are two rows apart from the aggressor [71]. This means that in certain DRAM devices, an aggressor can have a *blast diameter* (B) of 4, affecting up to four victim rows.

Seaborn [24] showed for the first time that Rowhammer bit flips could severely compromise security by building a native privilege-escalation exploit. Plenty of other attacks followed [15, 72–78], where researchers showed that it is possible to use these bit flips to compromise browsers [52–54], cloud virtual machines [25, 49], mobile phones [55, 56] and even remote machines over the network [57, 58].

3.2.3 *Rowhammer mitigations*

In response to these attacks, many solutions have attempted to mitigate Rowhammer in software or hardware. The ones implemented in software, usually inside the operating system's kernel, try to detect aggressor accesses and refresh their victims [16], isolating sensitive data from bit flips [56, 63, 64], or using certain pages to store sensitive information [79]. Unfortunately, these solutions require adoption by operating systems, which has not happened to date. They are also often vulnerable to more advanced attacks [26, 55, 65].

At the hardware level, Rowhammer can be mitigated either in the CPU's memory controller or inside the DRAM itself. Over the years, there have been many proposals by academia to modify the memory controller to detect aggressor rows either deterministically [41, 42, 59, 60, 80] or probabilistically [3, 60] and to refresh their victims under the Target Row Refresh

(TRR) scheme. Except for a low-cost solution that was briefly adopted by Intel [3, 5, 81], the remaining ones require extensive modifications to the CPU's memory controller with non-trivial area or performance overhead. As a result, they have not seen any adoption [5]. It is unlikely that all CPU vendors will deploy an expensive mitigation to fix a problem that is in the products of DRAM vendors. Perhaps, the only enabled mitigation in the CPU is the memory controller-based Error-Correction Code (ECC) in server systems. This covers only a fraction of existing computer systems that use DRAM, and even then, ECC does not provide an adequate level of protection against Rowhammer attacks [10, 62].

Rowhammer is a DRAM vulnerability, and arguably the best place to address it is inside the DRAM itself. In fact, this is exactly what DRAM vendors have done [66, 67]. Unfortunately, these in-DRAM TRR mitigations are undocumented and lack formal security guarantees. Recent work shows that there are indeed gaps in currently deployed mitigations and slight changes to existing Rowhammer patterns result in bit flips to resurface [5–7]. The only existing academic work on in-DRAM TRR [82] similarly suffers from slightly more advanced patterns [41]. Hence, we urgently need an in-DRAM TRR mechanism with formal security guarantees. In this paper, we show not only that this is possible, but it can be done in a way that is optimal in terms of the number of required counters and the introduced refresh overhead.

3.3 THREAT MODEL

We consider a DRAM device that is affected by the Rowhammer vulnerability. At the time of this writing, Rowhammer is present in all recent DRAM technologies [7, 17]. We assume that bits start to flip after R_{thresh} cumulative accesses to aggressor rows and that each aggressor row can influence up to B victim rows. We assume an adversary that is capable of sending requests to the DRAM device either through local code execution [10, 24–26, 49, 55, 56], from the Web [9, 52–54], or even over the network [57, 58] through a CPU that deploys a memory controller that is compliant with the respective DRAM standard [12, 20]. The aim of the adversary is to craft an access pattern that triggers Rowhammer bit flips to compromise the system by ensuring that a victim is hammered at least R_{thresh} times. Our mitigation should provide a formal guarantee that no row can be hammered R_{thresh} times before it is protected by TRR.

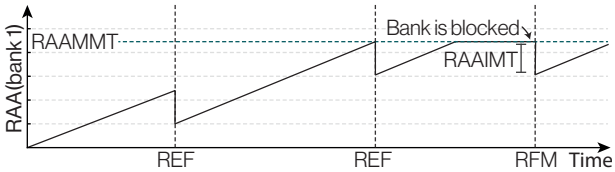

FIG. 3.2: **RFM example.** Activations are sent to the same bank, increasing RAA. At each REF, RAA is decremented by RAAIMT. Once the RAA reaches RAAMMT, the bank does not accept any ACTs anymore. In this case, issuing a RFM can reduce the counter by RAAIMT to unblock it before the next REF, which will also reduce the RAA counter value.

3.4 REFRESH MANAGEMENT IN DDR5

Recent (LP)DDR4 devices internally perform TRRs on potential victim rows, whenever they receive REF commands [5]. In theory, it is possible to perform TRRs during the execution of other DRAM commands such as ACT or read/write. However, as these commands are latency-critical, it would adversely affect the performance. As such, the REF is shared between regular refreshes and TRRs. Consequently, TRRs are scarcely performed and can only refresh a limited number of rows each time. Performing multiple TRRs overloads the REF command, and moving to smaller technology nodes with increasing Rowhammer vulnerability [17] only exacerbates this problem. As a remedy, the DDR5 standard [12] introduces a new DRAM command called Refresh Management (RFM) that provides additional time for TRRs.

RFM mechanisms. An RFM command either targets the same bank address in each bankgroup (RFMsb) or all banks (RFMab). Each bank has a counter called *Rolling Accumulated ACT* (RAA) that tracks the number of received ACTs. Once RAA reaches a maximum value defined as *RAA Maximum Management Threshold* (RAAMMT), no more ACTs are accepted by the bank until the RAA counter is decremented. There are two possibilities to decrement this counter: RFM and REF commands. Every time an RFM is received, the target banks' RAA is reduced by the value set in the *Initial Management Threshold* (RAAIMT). Instead, REF reduces RAA either by 0.5× or 1× of the RAAIMT, depending on the value of the MR59 OP[7:6] DRAM register. Fig. 3.2 summarizes these concepts with an example. In the current DDR5 standard, valid values for RAAIMT range from 32 to 80, in steps of 8. Since the RFM command can be postponed by the memory controller, in practice RAAIMT defines the average number of activations received by a bank before an RFM

is issued. Instead, RAAMMT = $m \times$ RAAIMT defines the maximum number of activations before an RFM or a REF must be issued, where m is an integer between 3 and 6 set by the DRAM. This gives the memory controller flexibility for scheduling RFM and REF commands as long as a bank's RAA count remains below RAAMMT.

3.5 Feinting

As stated in Section 3.2, the design of a secure and working in-DRAM TRR is still an open problem. The operations of such a mitigation are fundamentally different from those implemented inside the memory controller. In particular, (i) the points at which TRRs can be performed in a tREFW are limited, and (ii) only a small number of rows can be refreshed at each point.

In other words, performing in-DRAM TRR means occasionally refreshing a bounded number of rows. Therefore, to successfully protect against Rowhammer, the mitigation has to use the available TRRs effectively. Given these conditions, the only way to implement a secure mitigation is to *proactively* refresh rows. To provide deterministic guarantees, a proactive TRR scheme must keep track of row activations. This can be achieved by storing a list of victims or aggressors. Additionally, we define a Rowhammer mitigation to be *proactive* if (i) rows are refreshed without using a fixed hammering threshold, and (ii) the TRR mechanism is triggered periodically. In a proactive mitigation, every time the mechanism is triggered (*TRR event*), the most hammered V victim rows (*TRR volume*) are refreshed. Because this happens periodically, we consider two consecutive TRR events to be interleaved by T activations (*interval*).

In this section, we consider an ideal TRR scheme, TRR_{ideal}, which has a hammer counter for each victim row. The victim row's counter increases every time one of its aggressor rows is activated, TRRed or refreshed by the regular REF. The victim row's counter is reset to zero every time the victim row is activated, TRRed, or refreshed by the regular REF. For clarity, we define REF_I as the refresh where a specific row is regularly refreshed (i.e., not TRRed). In Section 3.6, we show how we can relax these requirements to build an in-DRAM TRR scheme that is both counter- and TRR-optimal while providing the same guarantees as TRR_{ideal}.

3.5.1 *Security analysis of TRR_{ideal}*

Any proactive TRR mitigation can protect up to a specific degree of Rowhammer vulnerability (R_{thresh}). In an ideal proactive mitigation with unlimited counters, this limit depends on V, T and B. Selecting V and T (B is technology-dependent), there exists a maximum count ($Hammer_{max}$) that a victim row can reach before getting refreshed either by REF_I or TRR.

Definition 1 (Victim hammering)**.** *A victim row $\tilde{x}$ is hammered each time one of its aggressor rows $\tilde{r}$ is activated (i.e., $\tilde{x}$ is one of the $B/2$ rows on each side of $\tilde{r}$). We denote by $x(\alpha)$ the hammer count of row $\tilde{x}$ after the α-th* ACT *of the attack. $x(\alpha)$ becomes zero every time $\tilde{x}$ is subject to REF_I, TRR, or an activation.*

Definition 2 (Rowhammer attack)**.** *We define a Rowhammer attack $\mathcal{A}$ on a victim $\tilde{x}$, as a finite sequence of L_{attk} activations to a bank's rows. $\mathcal{A}$ is successful against $\tilde{x}$ iff $\exists \alpha \geq 1 \mid x(\alpha) \geq R_{thresh}$. We denote by $\mathscr{A}$ the set of all attacks, which is the set of finite sequences over $[\![1, N_{rows}]\!]$, with N_{rows} being the number of rows in a bank.*

Definition 3 (Optimal Rowhammer attack)**.** *For a given $(V, T, B, mitigation)$, we define $Hammer_{max} = \max_{\mathscr{A}} \max_{1 \leq x \leq N_{rows}} \max_{\alpha \geq 1}[x(\alpha)]$. An attack $\mathcal{A} \in \mathscr{A}$ is optimal against a victim $\tilde{x}$ iff $\mathcal{A}$ reaches $Hammer_{max}$.*

Following, we express the security requirement for TRR_{ideal}:

Requirement (Security of TRR_{ideal})**.** *For a given DRAM technology (B, R_{thresh}) and configuration (V, T), TRR_{ideal} is secure if $Hammer_{max} < R_{thresh}$.*

Identifying $Hammer_{max}$ corresponds to finding the *optimal Rowhammer attack* against the mitigation. In what follows, we present and prove the best attack against TRR_{ideal}.

Assumptions. In our analysis, (i) we consider a memory controller with a closed-page policy (i.e., no bank collisions are required to induce a PRECHARGE); (ii) if during a TRR event, more than V rows have the same highest count, we consider an attacker that is able to influence which are refreshed; (iii) we assume an attacker that knows when the rows are refreshed by the REF_I — including the victim $\tilde{x}$. These assumptions constitute the worst possible conditions for the defender.

Without TRR, all the activations in a tREFW (L_{tREFW}) can be used against the victim. However, this approach would quickly fail against a proactive

mitigation: the mechanism would refresh the victim at the first TRR event, as the victim row would have the highest count. We will demonstrate that by using a specific activation pattern, the TRR event will never refresh the target victim before its REF_I. Moreover, we will show how this pattern can be used to build the best possible attack against TRR_{ideal}, which we refer to as *FEINTING*[1].

Decoy rows. Given a target victim row $\tilde{x}$, the attacker aims at activating the aggressor rows while protecting the victim from refreshes. During a TRR event, the only case where $\tilde{x}$ is not refreshed is if there are at least V different victim rows (decoys) with a greater or equal hammer count. When this happens, we say that the victim "survives" the TRR event.

Definition 4 (Conditions for victim survival)**.** *A victim row $\tilde{x}$ is not refreshed during a TRR event, after activation α, iff there exist V distinct rows $\tilde{d}_1 ... \tilde{d}_V$, each different from $\tilde{x}$, such that $\min_j[d_j(\alpha)] \geq x(\alpha)$. We refer to the rows $\tilde{d}_1..\tilde{d}_V$ as "decoys".*

Every time a victim is hammered, its counter is incremented by one. Given Definition 4, it follows that decoy rows must be incremented concurrently. Unfortunately for the attacker, when decoys are refreshed, their counters reset to zero and become lower than the victim's count. At the next event, different decoys will have to be higher or equal to the victim count for the victim to survives again. Generally, to survive n TRR events, a victim needs a total of $n \times V$ decoys. Note that each time an aggressor row is activated, it influences up to B victim rows. That is, for a single aggressor row activation, B decoys are hammered and their counters increase by one.

Problem formalization. This condition creates an optimization problem: before a TRR event, part of the activations should be used to hammer the victim and the remaining to hammer the decoy rows. However, if too many activations target the victim, the decoys cannot protect it from being refreshed. On the opposite, if just a few activations hammer the victim, it will reach a lower hammer count than possible since the extra activations used for the decoys are "wasted" (i.e., not used against the victim). Hence, the number of decoys and their hammer count should be minimized. We formalize this problem as follows: *Considering all activations in an attack (L_{attk}), then $L_{attk} - k$ activations must be used to build and maintain a set of decoys. The remaining $\mathbf{k}$ activations can be used for hammering the victim row and thus should be maximized.*

1 FEINTING refers to maneuvers that distract or mislead the opponent.

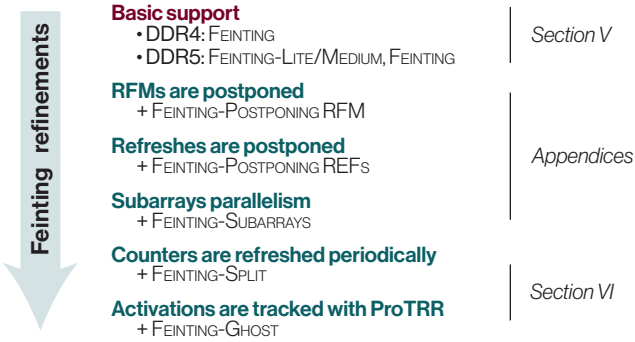

FIG. 3.3: **Overview of FEINTING variations. The final attack is a combination of the listed refinements, depending on the DDR technology.**

We solve this problem by answering the following questions:

1. What is the optimal hammer ratio between the different rows? [*optimal distribution*]
2. How many times should the rows be hammered in each step? [*optimal intensity*]
3. How many TRR events should the attack last? [*optimal duration*]

Answering these questions will lead us to the FEINTING attack. We start by obtaining FEINTING for DDR4 devices before adapting it to handle RFM on DDR5. In Section 3.6, we will adapt FEINTING to securely design PROTRR, and we will discuss how FEINTING can be further refined to handle protocol optimizations such as `REF` and `RFM` postponing, and certain DRAM architectural optimizations such as subarray parallelism (Appendix 3.12: §3.12.1, §3.12.2, and §3.12.3). Fig. 3.3 provides a summary of these different FEINTING variations.

3.5.2 *FEINTING on DDR4*

We consider an attack that lasts n TRR events (*intervals*). In the last TRR event, the victim can be refreshed (as the attack ends), so no further decoy is needed. Thus the minimum number of rows hammered in the attack is $D_T = (n-1) \times V + 1$, i.e. $(n-1) \times V$ decoy rows plus the victim row.

Generalizing, the victim row can be seen as the last decoy that is refreshed. We refer by $D(\alpha)$ to the number of decoy rows that have not been refreshed yet before the activation α. We define $\tilde{d_i}$ as the i-th decoy, and α_i as the moment it is refreshed.

Theorem 1 (Optimal distribution and intensity). *For a generic TRR event $i \in [\![1, n]\!]$ happening after activation α_i, with $D(\alpha_i)$ decoys $(\tilde{d_1}...\tilde{d}_{D_T-(i-1)\times V})$, an attack $\mathcal{A}$ can only be optimal if all decoys' hammer count $(d_1(\alpha_i)...d_{D_T-(i-1)\times V}(\alpha_i))$ is the same.*

► *Intuition.* To maximize $\boldsymbol{k}$ (the activations that hammer the victim), we must minimize the total activations used to hammer decoys during the attack. Decoys should not be hammered more than the victim because this is unnecessary for the victim to survive. Likewise, a decoy that is hammered insufficient times is useless for the victim's survival. Practically, this translates to *steps* in which all the decoys and the victim increase their hammer counts together and in unison, as shown in Fig. 3.4.

► *Proof.* First, we prove that no decoy should be refreshed with a hammer count higher than the victim $\tilde{x}$. We consider any TRR event i, after an activation α_i, in which a decoy $\tilde{d_i}$ is refreshed. We define Δ as the difference of hammer counts between decoy and victim: $d_i(\alpha_i) = y + \Delta$ for $x(\alpha_i) = y$. Given Definition 4, the victim already survives if $d_i(\alpha_i) = x(\alpha_i)$. This means that Δ hammerings are wasted by not spreading them equally over all remaining $D(\alpha_i)$ rows. In other words, the victim can survive with a count of $x(\alpha_i) = y + \frac{\Delta}{D(\alpha_i)}$, which creates a better attack.

Similarly, we now prove that it is not optimal to have decoys hammered less than the one refreshed at the TRR event. We consider any TRR event i (after activation α_i) in which a decoy $\tilde{d_i}$ is refreshed. In this case, Δ is the difference of hammer counts between a lower decoy ($\tilde{d_l}$) and $\tilde{d_i}$, with $d_i(\alpha_i) = y + \Delta$ for $d_l(\alpha_i) = y$. Decoy $\tilde{d_i}$ is refreshed with an excess of hammer counts: $\tilde{x}$ would have already survived with $d_i(\alpha_i) = x_i(\alpha_i) = y + \Delta'$, where $\Delta' = \frac{\Delta\times(D(\alpha_i)-1)}{D(\alpha_i)}$. The extra hammers $(\Delta - \Delta')$ are wasted, as they could have been used to hammer decoy d_l, which has to be hammered to make the victim survive in a future interval. Concluding, the optimal distribution and intensity minimizes the difference between all decoys and the victim by hammering them in steps and in each step, in unison.

Algorithm 3.1: **The pseudocode for FEINTING on DDR4.**

```
1 nr_intervals = 8192
2 A_T = nr_intervals*166 // T=166
3 nr_decoys = nr_intervals*V
4 aggressors = GetDifferentRows(nr_decoys/B)
5 for ACT = 1 ; ACT ≤ A_T ; ACT++ do
6 |   ACTIVATE GetLeastActivated(aggressors)
7 |   if ACT%T is 0 then // TRR event
  |   |   // remove the TRRed aggressors
8 |   |   RemoveHighest(aggressors, V/B)
```

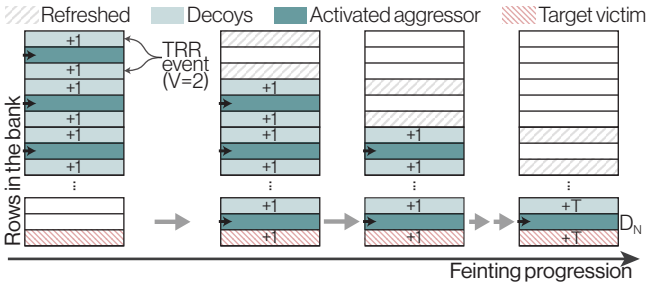

FIG. 3.4: **FEINTING strategy.** As the attack progresses, decoys get refreshed. In the last round, only the target victim (D_N) is left to be refreshed and all the activations are used against that victim, hammering it T times.

Theorem 2 (Optimal duration)**.** *Given n TRR events happening in a* `tREFW`*, an attack $\mathcal{A}$ is optimal if, given $\mathcal{A}$, $D_T = (n-1) \times V + 1$ and $L_{attk} = L_{tREFW}$.*

► *Intuition.* The last intervals of two attacks of different lengths are equivalent. In the last interval, in both cases, only one row survives (the victim), while in the previous interval, there were $V+1$ rows alive (the decoys and the victim), and so on. In other words, the longer attack extends the shorter attack by more intervals. An attacker can use these extra intervals to hammer the victim and the necessary decoys. As a result, using a fewer number of intervals only leads to a lower $Hammer_{max}$.

► *Proof.* Independently from the attack duration, the victim is refreshed after the last interval. Thus, according to Theorem 1 attacks of lengths n_1

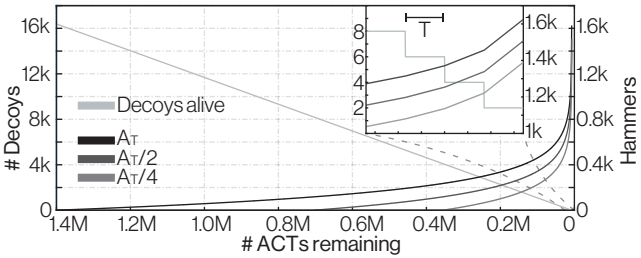

FIG. 3.5: **Different durations of FEINTING.** Example for DDR4, $\{V; B\} = 2$.

and n_2 (with $n_1 < n_2$) will share the same pattern for the corresponding last interval. As such, they will also share the previous intervals (i.e., $n_1 - 1$ and $n_2 - 1$) and so on, until the first $n_2 - n_1$ intervals of the longer attack. We now prove that having $n_2 - n_1 \geq 1$ is beneficial for the attack of length n_2.

Consider an attack that requires $D_T^{(1)}$ rows hammered and lasts n intervals. Adding one interval at the beginning results in hammering each row by $\Delta\epsilon = \frac{B \times T}{D_T^{(1)} + V}$ more and consequently must have V more decoys. However, because $B \times T > V$, it is beneficial for the attack to have this extra interval. Generally, for j intervals added, the victim row is increased by $\Delta\epsilon_{tot}(j) = \sum_{\phi=0}^{\phi=j-1} \frac{B \times T}{D_T - \phi \times V}$, where $D_T = D_T^{(1)} + j \times V$. Thus, the optimal duration of an attack is the maximum number of activations in a `tREFW` (i.e., L_{tREFW}), from which follows $L_{attk} = L_{tREFW}$. This covers all n TRR events in a `tREFW`, which means having $D_T = (n-1) \times V + 1$ decoys. Note that for simplicity, we consider that the available $B \times T$ hammering in each interval can be used flexibly for any victim without loss of generality. In reality, when $D(\alpha) < B$ (last interval(s)), rows are hammered at maximum T times per interval which is what is considered for all the plots, evaluation, and calculations.

FEINTING. To summarize, the optimal attack (FEINTING) is an attack that starts immediately after the victim row has been refreshed internally. The attack lasts a `tREFW` (for a total of L_{tREFW} activations), where $D(\alpha)$ rows are alternately hammered once. As TRR events happen, $D(\alpha)$ decreases, up to having only the victim row in the last interval. The number of aggressors needed is $\frac{D_T}{B}$, each associated with unique B decoys. Algorithm 3.1 presents the implementation of FEINTING according to these three theorems, and Fig. 3.5 shows how the increased duration of the attack allows for a higher

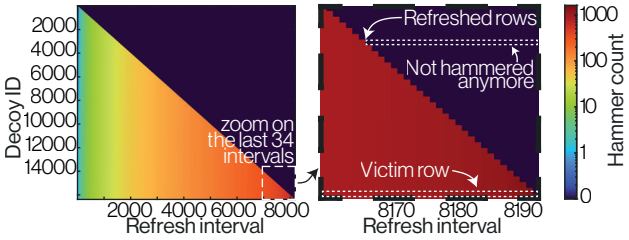

FIG. 3.6: **Impact of FEINTING against TRR$_{ideal}$.** Decoys' hammer count over time. After a decoy has been refreshed, it is never hammered again.

$Hammer_{max}$. Fig. 3.6 shows the hammer count of the victim in FEINTING over one tREFW.

Number of TRR events. In DDR4, a REF is sent every tREFI and may trigger a TRR event [5]. The distance d identifies after how many REFs one triggers a TRR event. This means that the total number of TRR events is $\frac{8192}{d}$ *regardless of the number of activations used in a tREFW*. In contrast, DDR5 introduces the new RFM command (Section 3.4), which, depending on the number of activations, allows for a higher number of TRR events. Next, we look at the impact of RFM on FEINTING.

3.5.3 *FEINTING on DDR5*

We adapt the previous theorems to DDR5 devices. In DDR5, TRR events happen for both REF and RFM. For this reason, while keeping the previous definitions, we specify T as follows. We define T_{REF} as the number of activations between two refreshes that perform TRR, and T_{RFM} as RAAMMT (=RAAIMT$\times m$). We first consider a simple memory controller that generates RFM commands every RAAIMT activations (i.e., $m = 1$). Later, in Section 3.5.4, we relax this assumption to consider postponing RFMs (i.e., $m > 1$).

TRR events on DDR5. We calculate the minimum number of possible TRR events generated on a DDR5 device during a tREFW. This leads to the minimum number of decoys needed to perform FEINTING. Per DDR5 standard [12], a register in the device indicates whether every REF or every second REF, a TRR happens (i.e., $d = 1$ or 2). For simplicity, we denote by REF_{TRR} the REFs that do TRRs. Depending on d there are 8192 or 4096 REF_{TRRs} in a tREFW. These are the minimum numbers of TRR events that

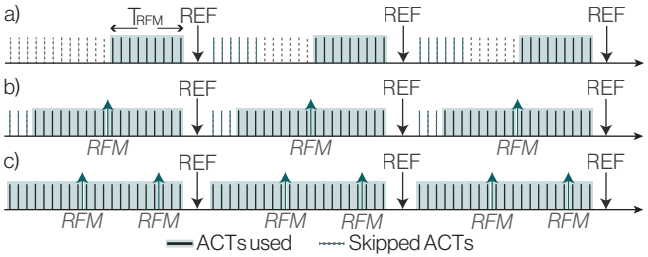

FIG. 3.7: **Different FEINTING strategies on DDR5.** a) FEINTING-Lite, b) FEINTING-Medium, and c) FEINTING.

happen in a `tREFW`, without including `RFMs`. With FEINTING-Lite, we show how an attacker can perform FEINTING without ever inducing an `RFM`.

FEINTING-Lite. In DDR5, `tREFW` is 32 ms by default, which leads to $T_{REF} = 83$ ($d = 1$). Instead, the maximum value of T_{RFM} is 80. For FEINTING-Lite and the other variants to be introduced later, we always consider an optimized memory controller that does not send an `RFM` if the next command is a REF_{TRR}. For this reason, T_{RFM} activations can always be sent between two REF_{TRR} without causing an `RFM`: as the `RAA` counter becomes `RAAMMT`, it is immediately set to zero with a REF_{TRR}. The FEINTING attack is reproducible without variations by skipping $T_{REF} - T_{RFM}$ activations every REF_{TRR} (Fig. 3.7a): we refer to such attack as FEINTING-Lite. Because we have already proven FEINTING to be optimal, *this is the optimal attack if no `RFM` command is triggered.*

FEINTING-Medium. If multiple blocks of T_{RFM} activations can fit between two REF_{TRR}, it is straightforward to prove that FEINTING-Lite can be improved by using the complete $T_{REF} - (T_{REF}\ mod\ T_{RFM})$ activations between two REF_{TRR}. T_{REF} can be segmented into blocks of T_{RFM} activations as shown in Fig. 3.7b. These additional blocks increase the number of intervals used for the attack in a `tREFW`. In the case of FEINTING-Lite, exactly 8192 (or 4096) intervals are used for the attack, each of T_{RFM} activations. In FEINTING-Medium, each additional block performs T_{RFM} activations and requires V (additional) decoys: exactly as if FEINTING-Lite lasted longer. Because of Theorem 2, this strategy improves the attack. In FEINTING-Medium, between two REF_{TRR}, the remaining ($T_{REF}\ mod\ T_{RFM}$) *extra activations* are skipped. FEINTING-Medium is the optimal attack if the remaining extra activations are not used. In the last step, we analyze if it can ever be beneficial for the attacker to use these extra activations.

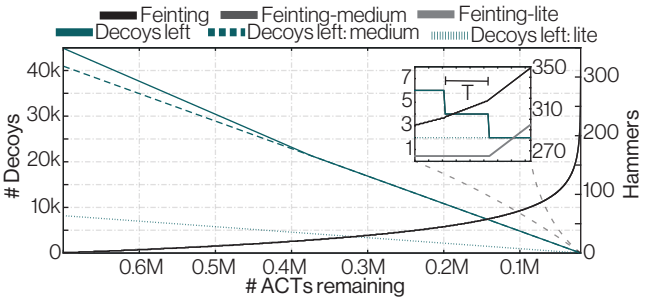

FIG. 3.8: **Different FEINTING strategies for DDR5.** Example for $\{V; B\} = 2$.

FEINTING. Starting from FEINTING-Medium, we evaluate if the attack can be improved by causing *some* extra RFMs using the remaining extra activations (T_{REF} *mod* T_{RFM}) between two REF_{TRR}. There is a cost attached when using these extra activations: every extra RFM triggered increases the number of decoys needed by V. The attacker needs to use activations to hammer these additional decoys. Unfortunately, these additional decoys are less impactful than the others since they add fewer activations to the attack. This leads to the following question:

Considering "FEINTING-Medium", when is it optimal for an attacker to use the extra activations that cause RFM?

Theorem 3 (Optimal number of extra RFMs)**.** *If using extra activations ceases to be beneficial for an attacker, then it can never become beneficial again in the same attack.*

Corollary. *If using extra activations at the beginning of the attack is not useful, then it will never be.*

▶ *Intuition.* Extra activations will trigger more TRR events, requiring more decoys to be hammered during the attack. As time passes, these decoys must be hammered (Theorem 1) until they are finally refreshed by the extra RFM. This can be seen as an expense for the attacker. From an attacker's point of view, it is less expensive to trigger the extra RFM earlier, so that the accumulated cost of hammering these decoys is lower.

▶ *Proof.* For simplicity, let us assume that only one full T_{RFM} fits between two REF_{TRR}. We consider two cases that are identical up to REF_{TRR} interval

$i-1$ with victim count $x(\alpha_{i-1}) = y$. Case (1): the attacker uses $\epsilon = T_{REF} - T_{RFM}$ extra activations in the interval i. Case (2): the attacker skips interval i as using extra activations is not useful, and then, in the next interval $i+1$, these ϵ activations become useful and are used for the attack. We now prove that case (2) is impossible. We start by evaluating the victim hammer count in the two cases, summing the different contributions:

$$x^{(1)}(\alpha_{i+1}) = y + \frac{T_{RFM} \times B}{D(\alpha_i)+V} + \frac{\epsilon \times B}{D(\alpha_i)} + \frac{T_{RFM} \times B}{D(\alpha_i)-V}$$

$$x^{(2)}(\alpha_{i+1}) = y + \frac{T_{RFM} \times B}{D(\alpha_i)+V} + \frac{T_{RFM} \times B}{D(\alpha_i)} + \frac{\epsilon \times B}{D(\alpha_i)-V}$$

We can evaluate when $x^{(1)}(\alpha_{i+1}) > x^{(2)}(\alpha_{i+1})$. This results in $T_{RFM} > \epsilon$ which is always true. Therefore, case (2) can never be more optimal than case (1). This means that is not possible that using the extra activations ceases to be useful in one interval and becomes useful again in a later interval. Likewise, if case (1) was not useful, case (2) would also not be useful. By induction, it cannot become useful in the future: if i is not useful and $i+1$ is not useful, $i+2$ will also not be useful, and so on. Concluding, the attacker can calculate when to stop inducing extra RFMs, deriving the best possible FEINTING. Fig. 3.8 shows the effectiveness of different FEINTING strategies on DDR5. These results show that while FEINTING-Medium improves the attack compared to FEINTING-Lite, in the case of $\{V; B\} = 2$ the improvement of the last optimization does not result in a higher $Hammer_{max}$.

3.5.4 *FEINTING on DDR5 with RFM postponing*

More sophisticated memory controllers may issue RFM commands irregularly, i.e., not always precisely after RAAIMT activations. However, it must never be after $T_{RFM} = m \times$ RAAIMT (i.e., RAAMMT) activations. In case that $T_{RFM} > T_{REF}$, FEINTING can be improved if we assume that the attacker can influence the scheduling of RFM commands. The idea is to leverage extra activations gained by postponing RFMs to build blocks of RAAIMT activations. This causes the RAA counter to increase quickly, and at some point, the memory controller will have to issue multiple, previously postponed RFM commands. It is optimal for the attacker if the L_{tREFW} activations are equally distributed over intervals of size RAAIMT, similarly as for FEINTING-Medium. In the last few intervals, postponed RFMs can be sent after the tREFW, as such, allowing the attacker to further increase the count of the

decoys (needed for REFs) and victim in these intervals without causing RFMs. Furthermore, in this setting, the attacker requires fewer decoys since fewer RFMs are issued during the attack. We refer to Appendix 3.12 for more details.

3.6 PROTRR

An ideal TRR mechanism (TRR_{ideal}) requires a large amount of storage. For example, a single-rank module with 16 banks/rank and 16 bit row addresses needs in total 14 MiB ($R_{thresh} = 5$ K). Mitigations deployed in the memory controller can use known optimized data structures to detect when a potential victim row reaches a specific threshold. Once this happens, these mitigations can delay the execution of normal DRAM operations to refresh this victim row [41, 42, 59, 80]. As already explained (Section 3.5), it is not possible for in-DRAM mitigations to delay DRAM requests due to the synchronous nature of the DRAM protocol.

Park et al. [41] use Misra-Gries summaries [68] that provide deterministic guarantees of finding the most frequently activated (aggressor) rows [83]. Misra-Gries summaries are proven to be optimal in the number of counters they need for detecting frequent items. Unfortunately, these summaries cannot be directly applied to the in-DRAM setting. First, Misra-Gries provides guarantees of finding frequent items occurring more than a fixed threshold in a stream with a specific length. However, an in-DRAM mitigation must protect V rows with the highest count at *any TRR event* without a fixed threshold. It is unclear how many counters are necessary to provide similar guarantees in PROTRR. Second, in a proactive in-DRAM setting, the counters of refreshed rows must reset while processing the stream, which is not considered in Misra-Gries.

Our proposed in-DRAM Rowhammer mitigation, PROTRR, uses a new frequent item counting scheme for in-DRAM operation, called PROMG (Proactive Misra-Gries). PROMG operates similarly to the Misra-Gries scheme, but is designed to function in the in-DRAM scenario. In the followings, we show how PROMG is similarly optimal in the number of required counters by leveraging the bounds given by FEINTING. Furthermore, we show how PROMG enables PROTRR to provide an optimal trade-off between the number of required counters and additional refreshes – given a DRAM device with a specific R_{thresh}.

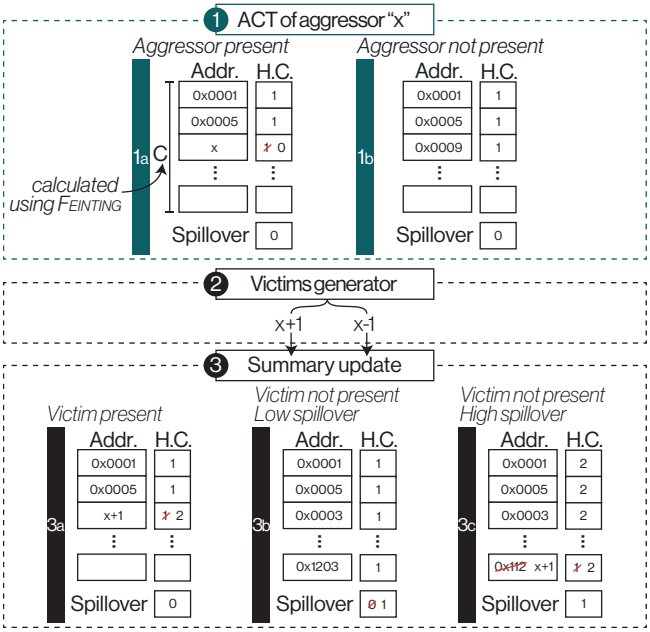

FIG. 3.9: **Victim counting in ProTRR.** Once a row is activated (1), if its address is contained in the summary it is pruned (1a). Then, the aggressor blast diameter is considered (2, for e.g. $B = 2$) identifying the victim rows. The victim rows are compared with the summary's content, which is updated accordingly (3a, 3b, 3c).

3.6.1 *Design of ProTRR*

ProMG is a proactive version of Misra-Gries summaries with two crucial differences. First, ProMG needs a different number of counters than the original Misra-Gries since it needs to make proactive decisions. We later show how FEINTING can be used to right-size ProMG summaries. Second, ProMG supports pruning entries from its summaries.

Similar to Misra-Gries, a ProMG summary is a table of $<$ ID, count $>$ pairs and a *spillover* counter. Conceptually, the spillover counter represents the upper bound of counts for all rows that are currently not in the summary. For every input, its ID is compared with all existing table entries; if there is a match, the associated counter is increased. Otherwise, the spillover value is compared with the lowest counter, and if the former is equal to or

higher than the latter, the new input replaces that entry and its counter is increased. If every entry has a higher count than the spillover, the spillover is increased. Unlike Misra-Gries, in PROMG, a row that is either activated or refreshed is pruned from the summary, and its victim rows are treated as summary inputs.

Fig. 3.9 shows how PROTRR makes use of PROMG. On each activation, PROTRR updates its summary accordingly by incrementing counters that are associated with victim rows of the activated row. At each TRR event, PROTRR refreshes the V rows with the highest counters in the summary.

Right-sizing the PROMG summary in PROTRR. In the original Misra-Gries scheme, given C counters and an input stream of size L, any entry occurring more than $\frac{L}{C+1}$ times will be included in the summary [41]. In contrast, PROTRR uses PROMG to make proactive decisions without reaching a threshold. To do this securely, we need to find the right number of PROMG counters for PROTRR to be secure against FEINTING. Furthermore, every row will be refreshed in a `tREFW` which we also leverage in PROTRR to ensure that the counters do not grow unbounded. To do this securely, however, we have to adjust the bounds given by FEINTING. We now prove theorems that shows how PROTRR right-sizes PROMG considering these observations.

Theorem 4 (FEINTING optimality against PROTRR)**.** *If the amount of TRR events in an attack is n, given PROTRR with $C = (n-1) \times V + 1$ counters in the summary (excluding the spillover), FEINTING is the optimal attack against PROTRR.*

Corollary. *Given $Hammer_{max}$ obtained with FEINTING for fixed (V, B and n TRR events) and considering PROTRR with $C = (n-1) \times V + 1$ counters (excluding the spillover), PROTRR protects any device less vulnerable than $Hammer_{max}$, i.e., where the Rowhammer threshold $R_{thresh} > Hammer_{max}$.*

▶ *Proof.* Given that $C = (n-1) \times V + 1$, PROTRR behaves exactly like an ideal counter against FEINTING. Therefore, an attacker is able to reach $Hammer_{max}$ as described earlier. We now prove that an attacker forcing the replacement of rows in the summary due to the limited number of counters does not increase $Hammer_{max}$. A replacement happens if a row $\tilde{d}_s$ that is not in the summary is hammered, and the spillover is equal or higher than the minimum count of the summary (row $\tilde{d}_t$). The replacement increases the counter that now refers to $\tilde{d}_s$. The effect on the attack is equivalent as if $\tilde{d}_t$ had been hammered, since for the victim to survive, it does not matter

which decoy is TRRed. Note that the replacement can only happen if more than C decoys have already been hammered; otherwise, $\tilde{d}_s$ is added to the summary. Moreover, because $C = D_T$, all the decoys necessary for the attack have already been hammered. Therefore, these replacements cannot improve the attack.

Resetting. Over time, the counters can grow unbounded, thus requiring unlimited storage to avoid overflows. This does not reflect reality where every row is refreshed at least once in a `tREFW`. To handle this, ProTRR resets the entire summary once every `tREFW`. The refresh of a given row, however, is not necessarily synchronized with the summary reset. This loss of information about the past `tREFW` allows an attacker to perform Feinting across a reset, thus changing the supported R_{thresh}. We address this in Theorem 5.

Theorem 5 (Non-linearity of Feinting)**.** *In the presence of a summary reset, two independent and shorter back-to-back Feinting result in a higher $Hammer_{max}$ than a longer one.*

▶ *Intuition.* Feinting starts after the victim row has been regularly refreshed (REF_I) to maximize the activations available for the attack (L_{tREFW}). However, during the attack, the summary could reset, leading to an information loss that can be exploited to increase $Hammer_{max}$. For example, two attacks of (each) 4096 intervals require half of the decoys than one attack lasting 8192 intervals but using the same number of activations, allowing the victim to be hammered more.

▶ *Proof.* We define the baseline as case (1): Feinting lasting n intervals, never crossing a summary reset. The number of times the victim will be hammered by the end of these intervals is denoted by $x^{(1)}(\alpha_n)$. In case (2), we consider a summary reset happening σ intervals after Feinting has started (with $\sigma < n - 1$). Once the summary resets, it becomes empty, and a new Feinting can be initialized, lasting the remaining $i = n - \sigma$ intervals. The cumulative number of times the victim is hammered, after n intervals is $x^{(2)}(\alpha_n)$. We compare these two cases. In case (2), the number of hammers to the victim is obtained by two different contributions, the first attack (σ intervals) and the second attack ($n - \sigma$ intervals): $x^{(2)}(\alpha_n) = \sum_{\phi=1}^{\sigma} \frac{B \times T}{\phi \times V + 1} + \sum_{\phi=0}^{i-1} \frac{B \times T}{1 + \phi \times V}$. Instead, case (1) consists of only one attack: $x^{(1)}(\alpha_n) = \sum_{\phi=0}^{n-1} \frac{B \times T}{1 + \phi \times V}$. The second part of case (2) overlaps with the start of case (1), i.e., their contributions are equal — a direct consequence of

Theorem 2. The first part of case (2) is larger than the sum of the σ last terms in case (1), which proves the non-linearity.

Corollary (FEINTING-Split)**.** *Given a summary reset every* `tREFW`*, two balanced ($\sigma = \frac{n}{2}$), independent back-to-back FEINTING attacks are optimal.*

We proved that if $\sigma < n - 1$, it is always better for the attacker to have two distinct and independent FEINTING. Now we prove that the optimal condition for the attacker is when there are two equally long attacks. We start by showing the effect of moving an interval from the attack's second part (i.e., last $n - \sigma$ intervals) to the first part (i.e., first σ intervals). The reader will remember that the sum of the two intervals is fixed by n. Moving an interval from the second to the first part is beneficial for the attacker when $\frac{B \times T}{\sigma \times V + 1} \geq \frac{B \times T}{1 + i \times V}$ leading to $i \geq \sigma$. Given that $i = n - \sigma$, it follows that the best case for the attacker is when $\sigma = \frac{n}{2}$. Because PROTRR implements summary refresh, we have to consider this adaptation of FEINTING, which we refer to as FEINTING-Split, when right-sizing the PROMG summary. Before finalizing FEINTING-PROTRR, we add flexibility to PROTRR.

3.6.2 *Optimality and Flexibility*

Depending on the DRAM technology, a vendor may afford a maximum number of TRR events (N) to be performed in a `tREFW` and a certain number of counters (C) to keep track of victim rows. We design PROTRR to be flexible: given any pair of (N, C), the maximum vulnerability protected can be obtained using FEINTING. A DRAM vendor, knowing the R_{thresh} for its own devices, can decide to change N or C as needed. Furthermore, we show that for any given (N, C, R_{thresh}), PROTRR is optimal: there exists no other deterministic in-DRAM TRR that is secure against FEINTING with a smaller number of TRR events than N. Similarly, for a given R_{thresh} and N, the number of counters C is optimal. We first show how PROTRR achieves flexibility and optimality for N, and then we discuss the same for C.

Flexible and optimal TRR events. The bounds given by FEINTING enable vendors to calculate the required TRR events (N) in a `tREFW` for a device-specific R_{thresh}. The following theorem shows that N is optimal for a given R_{thresh}.

Theorem 6 (TRR events optimality)**.** *For a supported R_{thresh}, PROTRR is optimal in the number of TRR events needed.*

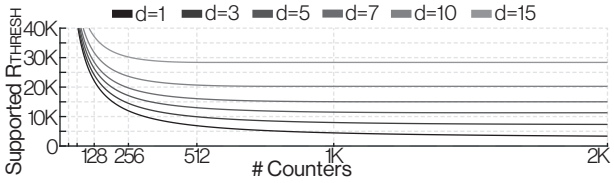

Fig. 3.10: **Flexibility** in ProTRR. Example for DDR4 (B=2, V=2, `tREFW` =64 ms). For a fixed storage, TRR distance (d) can be used as trade off.

To defend $R_{thresh} = Hammer_{max} + 1$ against Feinting, the device requires at least $\frac{D_T - 1}{V} + 1$ TRR events in a `tREFW`. If a smaller number of TRRs are employed, then the decoys for Feinting will be fewer, and $Hammer_{max}$ will exceed R_{thresh}. Hence, the number of TRR events is optimal. This feature of ProTRR provides it with flexibility on the number of TRR events. We can reduce the number of TRR events if a device has a high R_{thresh}. In practice, a manufacturer can tune the number of TRR events using the distance d (Section 3.5.2). This enables configurability of ProTRR according to the DRAM vendors' needs. Fig. 3.10 shows how ProTRR can support devices with different R_{thresh} by appropriately choosing d. We now show how ProTRR provides further flexibility in the number of required counters.

Flexible and optimal number of counters. For a given R_{thresh}, Feinting gives us the optimal number of TRR events. It follows that D_T counters are needed. Given that Misra-Gries summaries are space-optimal [83], using D_T counters will be optimal against Feinting. For more flexibility, we show how ProTRR can reduce this number of counters with a slight increase of R_{thresh}.

Feinting-Ghost. We adapt Feinting to handle cases where ProTRR has a limited storage, providing a trade-off between the supported R_{thresh} and the number of counters in the summary. With reduced storage, an attacker engaged in Feinting can create ghost decoys by first saturating the number of counters. Theorem 7 proves the optimal number of decoys for this modified attack.

Theorem 7 (Feinting-Ghost optimality). *For ProTRR with $C < (n-1) \times V + 1$ counters, where n is the number of TRR events in a `tREFW`, Feinting-Ghost with $C + 1$ decoys is optimal.*

▶ *Proof.* We assume $C < D_T$ and prove that $C + 1$ is the maximum number of decoys needed. After C decoys are hammered, the summary is full, and

the next (new) hammered decoy turns the spillover counter to one. Now, the rows that are not in the summary are considered already hammered once (i.e., *ghost* decoys) – thus reducing the number of hammers for maintaining them. Likewise, after the next $C + 1$ hammers, each row will be considered hammered twice, and so on. This condition persists until the number of decoys is C. From this point on, all hammers target rows present in the summary, and the attack is the same as the original FEINTING.

Theorem 8 (Counters optimality)**.** *For a supported R_{thresh}, given a number of TRR events, PROTRR is counter-optimal.*

▶ *Proof.* If we remove one counter (i.e., $C - 1$), there would be a ghost decoy for which an attacker does not need to waste activations until there are only $C - 1$ alive decoys left. These extra activations could be used to further increase the victim (and decoys) to exceed R_{thresh}. Hence, the number of counters needed in PROTRR is optimal. Fig. 3.10 shows how this allows PROTRR to massively reduce the number of counters needed, marginally increasing R_{thresh} in most settings.

FEINTING-PROTRR. Summarizing, the optimal attack against PROTRR is the adaptation of FEINTING given two new conditions: summary reset and limited number of counters. We define this attack as FEINTING-PROTRR, which is the implementation of FEINTING-Split, where each part is performing FEINTING-Ghost. We consider $Hammer_{max}$ achieved by FEINTING-PROTRR in different settings in our evaluation in Section 3.7.

3.6.3 *Implementation of PROTRR*

We implemented PROTRR in a popular 12 nm ASIC technology, to confirm its feasibility. In our evaluation (Section 3.7), we assessed the supported vulnerability for the number of counters implemented in current mitigations. Our design, depicted in Fig. 3.11, uses a *decoder* logic (❶) to distribute simple micro-operations over several clock cycles. The *entries update logic* (❷) performs the summary update and, depending on the given micro-operation (❸): removes a row after it has been refreshed *(REF request)*, increases the counters of a victim *(Blast request)*, or resets the summary *(Clear request)*. Within the same cycle, two parallel combinational circuits *(min/max reduction*, ❹a and ❹b*)* determine the rows with the lowest and highest counts for the next summary update (❺a and ❺b). We implemented the summary as a

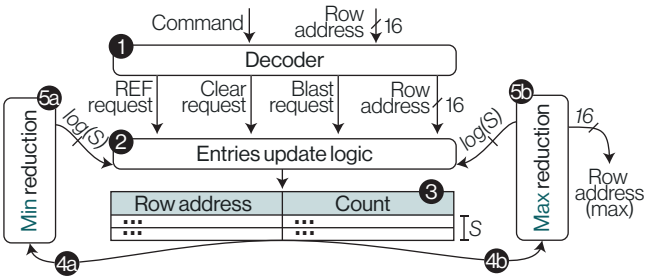

FIG. 3.11: **PROTRR's ASIC design**. Schematic of PROTRR's mechanisms.

standard cell memory to get simultaneous access to all its elements for the reductions.

Integration and placement. PROTRR can replace existing counter-based, in-DRAM TRR schemes [6, 84, 85]. Typically, control logic (excluding array decoders, Fig. 3.1) is placed in the center of the DRAM chip, while the rest of the area is devoted to the DRAM cell blocks [86–90]. Instead, for LPDDR devices, the control logic is placed on an edge pad. We received confirmation from a DRAM vendor that the TRR mechanism is placed in the peripheral logic which is part of the control logic. They also confirmed that it is feasible to implement 2K counters in this area in an older technology than the one used by PROTRR. While this is enough for almost all settings we considered in Section 3.7, more recent process technologies are capable of implementing more counters if needed.

3.7 EVALUATION

In this section, we present an extensive evaluation of PROTRR. We consider three key aspects that we assess for both DDR4 and DDR5: the impact on performance, storage requirements, and energy consumption. We show that PROTRR is lightweight, incurs negligible energy and performance overhead, and is practical for real-world deployments.

In §3.7.1, we show PROTRR's flexibility in supporting different device constraints with a varying number of required counters. To estimate the performance and energy overhead, we run the SPEC2017 benchmark suite [91], as described in §3.7.2. We run the benchmarks using full system simulation, allowing us to evaluate the impact of PROTRR under real-world conditions.

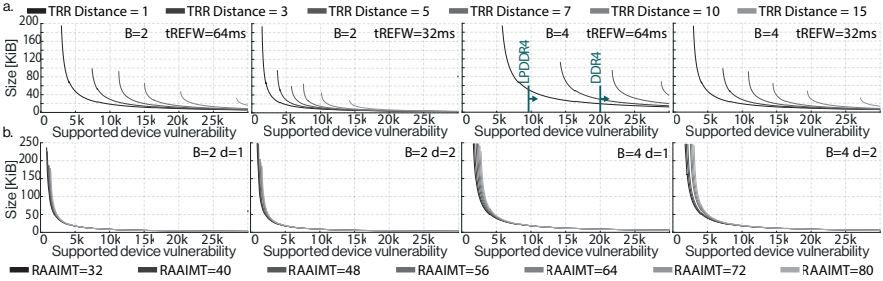

FIG. 3.12: **Storage size of PROTRR, per-chip values.** The green arrows indicates the worst device vulnerability taken from Kim et al. [17]. **a. DDR4 storage size.** $V = 2$. **b. DDR5 storage size.** $m = 6$, $V = 2$. The results for DDR4 with `tREFW` = 32 ms also apply for DDR5 without RFM support.

Additionally, even though PROTRR provides formal guarantees, we verified its implementation against state-of-the-art Rowhammer fuzzers [5, 7] and FEINTING (§3.7.4). We provide a confirmation of PROTRR's feasibility with our ASIC implementation (§3.7.3), and lastly, we test FEINTING against real DDR4 devices (§3.7.5). We point out that PROTRR is the first Rowhammer mitigation that is compatible with the latest DDR standard (DDR5), and this is the first work that evaluates the impact of `RFM`.

3.7.1 *Storage size and supported vulnerability*

The required storage of PROTRR is derived from the number of banks (N_{banks}) and the size of each summary. A summary contains entries ($S_{entries}$), each consisting of a row address and a counter. We consider 16-bit addresses, and $\log_2(Hammer_{max})$ bits for the counter. The total size in bits is $N_{banks} \times S_{entries} \times (16 + \lceil \log_2(Hammer_{max}) \rceil)$.

Fig. 3.12 presents the storage size of different DDR4 and DDR5 settings based on the geometries given in Tbl. 3.1. These figures show the required size per rank to support varying levels of device vulnerability to Rowhammer in different setups. The blast diameter of 4 incorporates devices subject to the recently discovered *half-double* attack [71]. These results illustrate how storage can flexibly be traded-off by a higher refresh rate, a lower TRR distance, or RFM postponing in DDR5.

DDR4. We consider a `tREFW` of 64 ms and 32 ms with a TRR volume of 2, for blast diameters of 2 and 4 (Fig. 3.12a). We also indicate the highest

CPU *(OoO)*		Memory Controller		DRAM	DDR4 2933	DDR5 4800
Cores	8	Channels	2	Ranks	1	1
CPU Freq.	3 GHz	Page Policy	Open Page	Bankgroups	4	8
L1D/I Cache	32 KiB	Scheduling	FR-FCFS	Banks/Group	4	2
L2 Cache	256 KiB	Queue Structure	Per Bank	Banks/Rank	16	16
L3 Cache	8 MiB	Total Capacity	16 GB	Rows/Bank	64 K	64 K

TBL. 3.1: Hardware settings and DRAM geometries of our gem5 simulations.

vulnerability degree as reported in previous work [17]. We make two observations using these results: (i) The TRR distance has a significant impact on the supported vulnerability. Due to the lack of RFM support in DDR4, this suggests that a TRR distance of one is required for newer process technologies. For example, LPDDR4 devices with 64 ms of `tREFW` can only be protected against the half-double attack with a TRR distance of one. (ii) A high blast diameter impacts the level of protection offered. In Appendix 3.12.6, we extend the results and present the same analysis for a TRR volume of 4. When a TRR volume of 4 is supported, PROTRR can mitigate much lower Rowhammer thresholds.

DDR5. Fig. 3.12b shows the required storage size for DDR5 for the worst possible case with `RFM` postponing of 6. We refer to Fig. 3.20 (Appendix 3.12) for more details. We make the following observations: (i) Thanks to the `RFM` extension, PROTRR can protect DDR5 devices with drastically lower Rowhammer thresholds. (ii) Lowering `RAAIMT` only marginally increases the offered protection, suggesting that the current set of possibilities in the latest JEDEC standard [12] is suboptimal. (iii) All the possible setups can protect against the most recently discovered half-double patterns.

3.7.2 *Performance and energy overhead*

Methodology. We evaluate PROTRR on the *SPEC®2017* [91] benchmark suite to assess its performance and energy overhead in real-world workloads. We follow the benchmark's guidelines and run each benchmark with eight parallel copies (i.e., number of cores) to maximize the simulated load. We use *gem5* [92], a cycle-accurate hardware simulator, in conjunction with *DRAMsim3* [93], a cycle-accurate memory controller. We implemented PROTRR in DRAMsim3, and due to the lack of publicly available DDR5

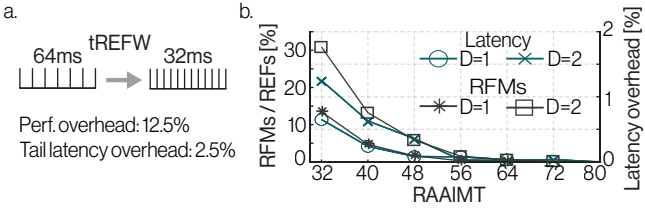

FIG. 3.13: **Average performance impact on DDR4 and DDR5. (a) DDR4.** Performance and tail latency overhead with `tREFW` = 32 ms. **(b) DDR5.** Left: percentage of `RFM`, relative to `REFs` in a `tREFW`. Right: tail latency overhead.

simulators, added DDR5 support to DRAMsim3, including the new `RFM` command. For benchmarking, we use the full system simulation mode of gem5 to run Ubuntu 20.04 with the Linux kernel 5.4.49. We follow the SMARTS methodology [94] to obtain 20 equally-spaced checkpoints, each running 10 M instructions, for a total of 200 M instructions in line with previous work [17, 42].

The simulated hardware setup is listed in Tbl. 3.1. The results are relative to a baseline, which was obtained by running the benchmarks without any active mitigation. The simulations consider varying (a) TRR volumes, (b) TRR distances, and (c) `tREFW` durations. As recommended by JEDEC [95] to help against Rowhammer, we assume that the `REFs` cannot be postponed. We still show in Section 3.8 how PROTRR can support postponing `REFs`. We configure the memory controller to immediately send an `RFMsb` upon reaching `RAAIMT`, which is the worst-case scenario for performance. Note that our performance and (dynamic) power measurements are independent of R_{thresh}. A vendor should select the correct TRR distance, `tREFW` (in case of DDR4), and `RAAIMT` (in case of DDR5) according to the R_{thresh} for their device. In contrast, the implementation-dependent area and static power overhead depend on R_{thresh}, which we report in §3.7.3 using our ASIC implementation.

Performance. In DDR4, TRRs happen only during `REF` without any performance overhead. However, as discussed, a default `tREFW` of 64 ms may not provide adequate protection with low Rowhammer thresholds. For this reason, we evaluated the impact of changing `tREFW` to 32 ms (Fig. 3.13a). This not only reduces the time window available for an attack but also increases the frequency of internal TRRs. The result is an average CPI

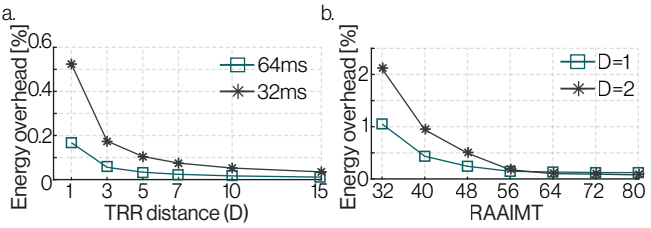

FIG. 3.14: **Average energy impact on DDR4 and DDR5. (a) DDR4.** Energy overhead of TRRs performed during REF. **(b) DDR5.** Energy overhead due to TRRs performed during REF and RFM.

(cycles-per-instruction) overhead of 12.5% while increasing the tail latency of DRAM accesses by 2.5%[2].

In DDR5, TRR events still happen during REF, but, if required, the new RFM command is sent, potentially introducing overhead. To analyze the RFM's impact, we tested all possible combinations of RAAIMT and TRR distances. In all scenarios, the performance overhead is always negligible, never exceeding 0.2%. To better understand the impact of RFM, we present the percentage of RFM compared to REF commands and the increasing tail latency with varying RAAIMT and TRR distance in Fig. 3.13b (for more details, see Appendix 3.12). We make two observations: (i) For small RAAIMT numbers, we require a substantial number of RFM commands (30.87% increase compared to the baseline REF in the worst case). These RFM commands, however, do not alter the instruction throughput (i.e., CPI) due to the parallelism offered by the out-of-order CPU cores and bank-level parallelism offered by RFM. In DDR4, the REF is a *per-rank* command, blocking the entire rank and substantially increasing the overhead when moving from tREFW of 64 ms to 32 ms. (ii) While CPI (i.e., instruction throughput) remains mostly unaffected, RFM does increase the tail latency of DRAM accesses (1.25% in the worst case).

Energy. We analyze the energy impact of the additional refreshes during TRR events. For each benchmark, we calculate the energy consumption as a sum of the device's plain energy, the energy of the TRRs performed during REF commands, and the energy consumed by RFM commands. To estimate the energy of these extra TRR refreshes, we calculate the energy required to refresh a single row and multiply it by the volume.

2 Considering DRAM accesses that take longer than 200 cycles.

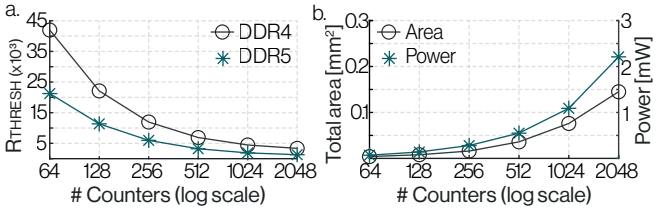

FIG. 3.15: **PROTRR feasibility, per-bank values. (a)** Required number of counters for different R_{thresh} in DDR4 (`tREFW` $= 64ms$, $d = 1$) and DDR5 ($T_{RFM} = 32$, $d = 1$). **(b)** ProTRR ASIC costs in terms of total area and power consumption.

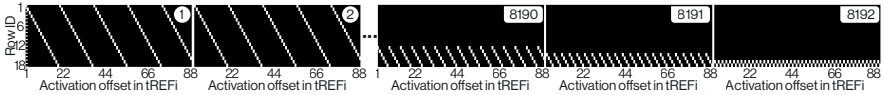

FIG. 3.16: **Trace of FEINTING-Ghost against DDR4 samples with 16 counters. The attack duration is 8192 `tREFI`.**

Fig. 3.14a reports the energy overhead of PROTRR in DDR4 for a `tREFW` of 64 ms and 32 ms with varying TRR distance between 1 and 15. Fig. 3.14b shows the energy overhead of DDR5 for different `RAAIMT` and the two possible TRR distances. We make the following observations: (i) The energy overhead in DDR4 is always below 0.6% of the device's total energy. (ii) The energy overhead in DDR5 is generally higher than for DDR4 due to the additional TRRs. However, this is still relatively small and at 2.11% in the worst case. (iii) In DDR5, given the same number of activations, for a TRR distance of 2, a higher number of `RFM` must be sent to compensate, increasing the energy overhead.

3.7.3 *Feasibility*

We implemented PROTRR in ASIC, using a popular 12 nm technology and the Synopsys Design Compiler. Fig. 3.15b reports the total area required and power consumption and Fig. 3.15a shows the R_{thresh} that PROTRR can protect for the number of counters. As results show, having more counters than 1024 does not substantially increase security; therefore, we consider it as the worst-case scenario. We designed the ASIC such that all updates (including lookups) are faster than the time between two consecutive `ACTs`,

allowing PROTRR to execute in parallel. In particular, the operations require V cycles during a refresh, and B+1 cycles during an ACT. We considered 45 ns as the minimum time between two activations, as previously reported [41].

Static power. Although previous work showed that the energy consumption for the mitigation logic is negligible [41, 42], we evaluated it for completeness. The overhead for 1024 counters is at maximum 17.44 mW for 16 banks, obtaining a total of 139.52 mW for 8 chips. This is in line with previously reported values [42]. For a baseline static consumption of 3 W/8 GiB [96], this leads to 4.65% static power overhead. However, given the current technology and consumer DDR4 chips, 512 counters are enough to ensure protection in the worst cases, leading to 2.35% static power overhead.

Area. Chips area depends on process technology, fabrication, and the array size. For our analysis, we consider a common density for DDR4 devices, 0.247 GB/mm^2 [97]. For a chip that uses 16 banks and 1024 counters per bank, this leads to a maximum area overhead of 3.7%. Unfortunately, currently deployed TRR mechanisms are kept secret and there is no open DRAM implementation that can integrate PROTRR. For this reason, to further confirm the feasibility of PROTRR, we contacted a DRAM manufacturer. We obtained confirmation that (i) up to 2K counters have already been deployed in the past, and (ii) given PROTRR's specifications (dimension, as obtained from results), it is reasonable to deploy it.

3.7.4 *Correctness*

We tested PROTRR against FEINTING to check its implementation by running PROTRR in DRAMsim3 with memory traces. In all the cases with a correct configuration, PROTRR could withstand FEINTING. Instead, in cases where PROTRR was improperly configured, FEINTING could successfully trigger bit flips. We also generated traces from two state-of-the-art Rowhammer fuzzers [5, 7] and executed them against PROTRR for three days without observing any bit flip.

3.7.5 *FEINTING on real devices*

FEINTING assumes a mitigation that counts every activation with an adequate number of counters. Existing TRR schemes are not ideal and may

DIMM	Mf. Date (yy-ww)	Size (GiB)	Freq. (MHz)	Geom. #R., #B.	Best Params.	Bit Flips Observed
$\mathbb{D}_0$	20-03	8	2666	1, 16	2048, 9, 1	✔
$\mathbb{D}_1$	20-06	32	2666	2, 16	2048, 9, 3	✔
$\mathbb{D}_2$	20-10	8	2400	1, 16	8192, 9, 4	✔

TBL. 3.2: **Results of FEINTING on three DDR4 devices.** We report the best attack's parameters (**Best Params.**) as: attacks duration (in `tREFIs`), TRR distance, and number of victim hammer repetitions.

employ multiple concurrent mechanisms to catch aggressors, some completely different from PROTRR [6]. However, we were still interested to see if FEINTING is able to generate bit flips on devices with a deployed counter-based mitigation. To evaluate this, we acquired three DDR4 devices from the same manufacturer previously reported to use a counter-based mitigation [6] (see Tbl. 3.2).

We conducted our experiments on an Intel i7-8700K running on Linux with kernel 4.15.0. We adapted FEINTING based on insights from [6] as follows: we assumed that counters could track at most 16 rows (i.e., 18 decoys needed as part of FEINTING-Ghost), and systematically tested different attack durations (2048× up to 32768× `tREFI`) as a row could be refreshed multiple times in a `tREFW`. We tested different TRR distances (1 up to 9) and victim hammer repetitions (1 to 4) while assuming 5 hammering repetitions for decoys.

In Tbl. 3.2, we show the results of running FEINTING-Ghost on our acquired DDR4 devices. An attack trace can be seen in Fig. 3.16, where the duration is 8192× `tREFI`. Our results show that with minor adaptations, we could successfully trigger bit flips on all three devices using FEINTING. Further, we can see that an attack duration shorter than a `tREFW` and hammering the victim fewer times (e.g., one time for $\mathbb{D}_0$) can be beneficial because sampling may happen only at specific times as reported in previous work [5, 7]. We tested Blacksmith [7] on the same devices, which could trigger bit flips on all of them, while TRRespass [5] failed to obtain bit flips on DIMM $\mathbb{D}_0$.

3.8 DISCUSSION

We now discuss how PROTRR can (i) be adapted to handle postponing and pulling-in of refresh commands, (ii) obtain better bounds by using subarray parallelism, and (iii) generalize to other, yet unknown, Rowhammer effects.

Postponing and pulling-in of REFs. The DDRx standard gives some flexibility in terms of REFs by allowing REF postponing and pulling-in. Attackers can exploit postponing to maximize TRR-free REFs, which reduces the number of decoys needed for both DDR4 and DDR5. For DDR4, the victim can be hammered more often than before, but for DDR5 nothing changes due to RFM still being sent. A more detailed analysis of REF postponing/pulling-in is given in Appendix 3.12.1.

Subarray parallelism. Subarrays enable a bank to refresh multiple rows at each REF. PROTRR can potentially leverage this to perform more TRRs when necessary. We provide a detailed description of how FEINTING can be adapted to subarray parallelism in Appendix 3.12.3. In summary, each bank can perform multiple TRRs at the same time, effectively increasing V. However, the additional TRRs cannot target any row as each subarray can still only refresh V rows at any given TRR event. An adapted FEINTING can exploit this limitation by reducing the number of required decoys to create the optimal attack.

Generalization. FEINTING provides the basis to configure PROTRR to protect against Rowhammer. The only (implicit) requirement for FEINTING is knowing the interaction between an aggressor and its victims. In the case of the standard Rowhammer attack, which we originally considered, an aggressor activation interacts with its direct neighboring victim rows. With new Rowhammer effects, FEINTING should be adjusted to consider new interactions between aggressors and victims. We discuss two of these cases next.

During the development of PROTRR, the half-double pattern was disclosed by Google researchers [71]. To also protect against it, we only needed to consider that on certain devices, an aggressor activation can also interact with victim rows that are more than one row apart (i.e., up to B rows). Currently, a rigorous characterization of the half-double effect is missing. For this reason, we assumed the worst-case in the design of PROTRR, i.e., the same effect on every row in the blast diameter. Once this relationship is better understood, future research can adapt FEINTING accordingly to derive the optimal version of PROTRR when $B > 2$.

Another concurrent discovery shows that rows that are kept active can also influence adjacent rows [32]. PROTRR can easily generalize to this case as well. The only new requirement is to increase the counter for victims of the (aggressor) row that remains active. It remains unclear, however, whether simply keeping a row active is more effective than using the time for additional hammering which should be characterized more rigorously in the future.

3.9 SECURITY ANALYSIS OF EXISTING SCHEMES

We first discuss a general limitation when mitigating Rowhammer outside of DRAM. We then present our security analysis of state-of-the-art hardware mitigations, which resulted in the discovery of novel vulnerabilities in four earlier proposed schemes [35, 41, 59, 80].

Internal row remapping. Previous work has observed that bits can flip in rows that are not adjacent to an aggressor [3, 57, 58]. This is due to internal row remapping that does not necessarily map logically-adjacent to physically-adjacent rows inside the DRAM device [98]. This is a major limitation of all existing Rowhammer mitigations that are outside of DRAM, both hardware and software [3, 16, 41, 56, 57, 59–61, 63, 64, 80] except Blockhammer [42]. PARA [3] explicitly requires this remapping information to be communicated from the DRAM to the CPU, which has never been implemented. In contrast, the in-DRAM nature of PROTRR allows it to use the correct row mapping that is known by the DRAM chip only.

CBT [80] and CAT-TWO [59]. Both mitigations reset their table after a `tREFW` period. If within this period, an aggressor row reaches R_{thresh} activations, its neighbors are refreshed. An attacker can, however, activate an aggressor $R_{thresh} - 1$ times immediately before and after the `tREFW`, violating the guarantees provided by the mitigation. A second issue concerns CAT-TWO only: it employs trees of counters distributed among the rows of a full rank. However, these trees are blind to victim rows that share aggressor rows across different trees. By hammering each aggressor for $R_{thresh} - 1$ times, the victim can exceed the threshold.

Graphene [41]. Refreshing a row with TRR has a similar effect like an `ACT`, which is used while hammering rows. As a consequence, an attacker could exploit TRRs to hammer rows. While this could be easy to fix, it is not taken

	Mitigation	Scalability		Security			Support		Integration		
		Flex.	Opt.	Det.	FP	Vuln.	DDR4	DDR5	OS	CPU	DRAM
HW-based	**PROTRR**	●■	●	●	●	—	●◇	●	○	○	●
	Blockhammer [42]	○□	○	●	●	—	●◇	○	○	●	○
	CBT [80]	○□	○	●	○	●	●◆	○	○	●	○
	CAT-TWO [59]	○□	○	●	○	●	●◆	○	○	●	○
	Graphene [41]	◐□	●	●	●	◐	●◆	○	○	●	○
	MRLoc [60]	○■	○	○	○	◐	●◆	○	○	●	○
	Panopticon [35]	○■	○	◐	○	◐	●◆	○	○	◐	◐
	PARA [3]	○■	○	○	○	—	●◆	○	○	●	○
	PRoHIT [82]	○■	○	○	○	●	●◇	○	○	○	●
	TWiCe [61]	○□	○	●	●	◐	●◆	○	○	●	○
SW-based	ALIS [57]	—	—	●	○	◐	—	—	●	○	○
	ANVIL [16]	—	—	●	○	●	—	—	●	○	○
	CATT [63]	—	—	●	○	◐	—	—	●	○	○
	GuardION [56]	—	—	●	○	●	—	—	●	○	○
	ZebRAM [64]	—	—	●	○	◐	—	—	●	○	○

TBL. 3.3: **Rowhammer mitigations in hardware and software.**

into account in the current design of Graphene. We discuss how PROTRR securely handles TRRs in Appendix 3.12.4.

Panopticon [35]. Concurrent to our work, Panopticon is a new in-DRAM mitigation against Rowhammer that relies on per-row counters stored in DRAM and uses the ALERT mechanism to request more time (from the memory controller) to TRR victim rows that reached a threshold. While storing counters inside DRAM itself is cheap, it is insecure as they can similarly be affected by Rowhammer bit flips. Furthermore, overloading the ALERT mechanism has multiple undesirable implications. First, not all devices may support ALERT as it is optional according to the standard, and the PHY-level errors causing them are very rare. Second, ALERT is a signal that blocks the whole device, likely causing significant performance degradation. Finally, it is unclear how the memory controller can tell the difference between a real ALERT (to retry commands) and one due to Rowhammer activity. If counters in DRAM can be secured (e.g., with a strong ECC), PROTRR can use these counters to provide a better alternative.

3.10 RELATED WORK

We summarize existing work on Rowhammer mitigations in Tbl. 3.3 and compare the following properties: (i) scalability, i.e., the optimality of re-

source allocation; (ii) security, i.e., the strength of the provided security guarantees; (iii) support, i.e., the mitigation's supported DRAM standards; (iv) and integration, i.e., the solution's required integration effort.

For **scalability**, we consider if mitigations optimally use counters and refreshes (Opt.); and if these resources can flexibly be traded-off with each other (Flex., ●). For flexibility, we further analyze if the mitigation's required storage size is always the same (□), hence is more wasteful, or scales with the system's connected devices (■). The **security** category includes the mitigations' guarantees, which are either deterministic (Det., ●) or probabilistic (○). Deterministic mitigations provide a stronger guarantee against bit flips. Further, we consider if a mitigation provides a formal proof (FP) for its design (●). Lastly, we highlight those mitigations for which we (or existing work) revealed vulnerabilities (Vuln.), and we distinguish between minor issues (◐) and fundamental flaws in the design (●). An extensive **support** of different DRAM standards (DDR4, DDR5) is essential to ensure practicality and widespread adoption. We further analyze whether mitigations require changes to the DRAM protocol (◆) or not (◇). Finally, we consider the system **integration** effort by describing which components need to be modified. Minimizing the effort is critical for real-world adoption as indirectly affected manufacturers (i.e., CPU/OS vendors) may not be willing to implement complex solutions.

Scalability. Only two mitigations (PROTRR and Graphene) are optimal w.r.t. counters and refresh requirements. PROTRR is the only solution that can flexibly trade-off storage with additional refreshes. PROTRR, ProHIT and Panopticon are the only mitigations that have counters in-DRAM, i.e., their required storage scales per connected device. Panopticon's storage is flexible as the counter table uses DRAM. PARA is completely stateless and does not require any storage. Similarly, MRLoc has negligible storage requirements. All other hardware-based mitigations are implemented in the memory controller; hence vendors need to provision enough storage for the system's maximum supported DRAM size.

Security. Few mitigations provide formal security guarantees for protection against Rowhammer attacks. We denote mitigations without known vulnerabilities by "—". Based on our security analysis (Section 3.9) and previous work [41], most of the hardware-based mitigations suffer from vulnerabilities. PARA's security is probabilistic, and to protect modern devices the overhead can be substantial [17]. Instead, all software mitigations provide a partial protection because of blindness to internal row remapping, and to

newer Rowhammer variants like half-double [71]. Previous work has also shown design-level flaws in ANVIL [26, 55, 98] and GuardION [78].

Support. None of the existing hardware-based mitigations are DDR5-ready, except PROTRR, which considers the new RFM extension introduced in the DDR5 standard [12]. Software-based mitigations are agnostic to the DDR technology. PROTRR, ProHIT, and Blockhammer are the only three mitigations that do not require changing the DRAM protocol. TWiCe and Graphene require adding new DRAM commands for refreshing rows adjacent to the aggressors, and PARA requires communicating the mapping of internal rows to the CPU. All other mitigations implicitly assume that there exists a DRAM command for refreshing a specific row – which currently does not exist.

Integration. Our comparison shows that all hardware-based solutions require modifications to the CPU (e.g., memory controller), except for PRoHIT, and PROTRR, which can be fully implemented in-DRAM. PRoHIT is vulnerable to specific patterns [41]. Panopticon [35] requires the CPU's memory controller to handle the ALERT signal gracefully, and as discussed in Section 3.9, some of its security aspects remain unclear. Instead, PROTRR is the only solution with deterministic and formal security guarantees. Software-based solutions are often integrated into the operating system's kernel. None has seen widespread adoption so far.

3.11 CONCLUSION

We introduced PROTRR, the first in-DRAM Rowhammer mitigation with formal security guarantees, for which we also proved that it is optimal in terms of storage and refresh overhead for any given DRAM technology. PROTRR is secure against FEINTING, the best possible attack we have formally constructed against a perfect in-DRAM TRR. Moreover, we used insights from FEINTING to provide a flexible trade-off between needed storage and refreshes given a DRAM device with a certain degree of vulnerability to Rowhammer. PROTRR is compatible with DDR4 and leverages the recent RFM extension in DDR5 to support future devices that are more susceptible to Rowhammer. We evaluated PROTRR's space, performance, and power overhead using an ASIC implementation and cycle-accurate simulation. In summary, PROTRR can protect current and future devices while requiring minimal storage and incurring negligible power and performance overhead.

3.12 APPENDIX

3.12.1 *Impact of REF postponing and pulling-in: FEINTING-PostponingREFs*

The DDR4 and DDR5 standards [12, 20] allow the memory controller to postpone some REF commands (e.g., under heavy DRAM activity) or to pull in a number REF commands (e.g., under idle DRAM activity). With the standard refresh rate in DDR4, up to 8 REF commands can be postponed or pulled in, and the maximum distance between two consecutive refreshes can be up to 9× tREFI. Likewise, for DDR5 devices, up to 4 REF commands can be postponed or pulled in, and the maximum distance between two consecutive refreshes can be up to 5× tREFI. The JEDEC consortium recommended to disable REF postponing and pulling-in to reduce the impact on in-DRAM Rowhammer mitigations [95]. Nonetheless, we show how PROTRR can securely support REF postponing and pulling-in by slighting modifying FEINTING.

Postponing and pulling-in are a relaxation of *when* REF commands need to be sent to a DRAM device. That said, even with postponing and pulling-in, a certain number of REF commands needs to be sent to the DRAM device in a tREFW. Given that the structure of the FEINTING is agnostic to when REF commands are issued, the only remaining question is whether it enables using fewer decoys. In both DDR4 and DDR5, at the end of the tREFW, the attacker can abuse postponing to maximize the number of tREFIs without any REF commands, which we indicate by P_{max} (1 when there is no postponing).

This configuration has two implications. First, $(P_{max} - 1) \times V$ fewer decoys are needed compared to the original FEINTING for both DDR4 and DDR5. Second, for DDR4, an attacker can continuously hammer the victim for $P_{max} \times$ tREFI. For DDR5, if RAAIMT is between 32 and 64, this does not change the number of times the victim can be hammered other than what is allowed by RFM postponing (as discussed in Section 3.5.4). When RAAIMT is set to 72 or 80, however, the distance between groups of RFM commands can be higher than those of postponed REF commands: $6 \times$ RAAIMT $> 5 \times T_{REF}$. Where, in the default tREFW of DDR5 (32 ms), T_{REF} is equal to 83. In these cases, equally as in DDR4, the last round of the attack is extended.

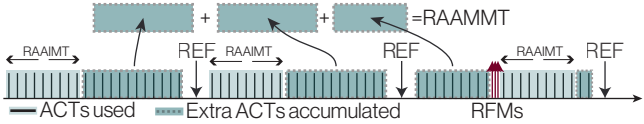

FIG. 3.17: **FEINTING for large T_{RFM}.** As the attacker sends activations, `REF` can reduce `RAA` accounting only for `RAAIMT` activations. The exceeding value eventually reaches `RAAMMT`, and m RFMs are sent by the controller.

3.12.2 *RFM postponing: FEINTING-PostponingRFMs*

We now consider a more advanced memory controller that postpones `RFM` commands. This means that `RFM` commands do not have to be sent exactly after `RAAIMT` activations. Instead, the controller has the flexibility to choose a better scheduling. As explained in Section 3.4, the only requirement set by the standard [12] is that `RAA` can never exceed $T_{RFM} = m \times$ `RAAIMT`. In other words, `RFM` commands can be postponed up to m times. In a real scenario, it is very hard for an attacker to influence the way `RFM` commands are scheduled. Nonetheless, we assume the most favorable scheduling from an attacker's perspective. Depending on T_{RFM}, there are two possibilities: (i) $T_{RFM} \leq T_{REF}$, where the same as in an earlier section (§3.5.3) applies and nothing changes from an attacker's point of view; and (ii) $T_{RFM} > T_{REF}$, where FEINTING can further be improved. We now analyze case (ii) considering an attacker who is able to precisely influence the scheduling of `RFM` commands.

FEINTING for large T_{RFM}. Postponing RFM commands enables a lucky attacker to avoid the situation of costly `RFM` commands triggered due to only a few extra activations, such as the case in Fig. 3.7c, Section 3.5.3. Instead, RFM postponing can be used to create blocks of `RAAIMT` activations similar to FEINTING-Medium (Fig. 3.7b, Section 3.5.3). This postponing, however, causes the `RAA` counter to increase, and at some point, the memory controller will have to issue `RFM` commands. Overall, two phenomena are happening simultaneously: a slow accumulation of extra activations that results in a series of `RFM`, and a fast increase of `RAA` that is reduced immediately upon refresh as shown in Fig. 3.17. However, given that our L_{tREFW} activations are now equally distributed over intervals of size `RAAIMT`, this is the optimal scenario for the attacker similar to FEINTING-Medium.

An exception are the last few intervals of FEINTING in this scenario. The last `RFM` commands can potentially be sent after the `tREFW`, de facto removing

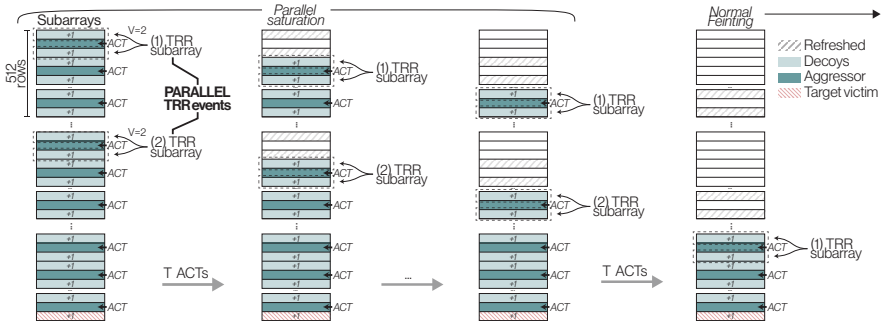

FIG. 3.18: FEINTING against subarray parallelism.

them from the mitigation. These activations can freely be used to increase the count of the last decoys (needed for REF) and the victim. We consider RFM postponing when configuring PROTRR as discussed in Section 3.6 and our evaluation in Section 3.7.

3.12.3 *FEINTING against subarray parallelism: FEINTING-Subarrays*

In the following, we describe how FEINTING can be adapted for subarray TRR parallelism. To obtain the highest $Hammer_{max}$, the choice of aggressor rows used for FEINTING must be optimized. We define S as the maximum number of subarrays refreshed at each TRR event. We consider a TRR mechanism that refreshes the highest rows from S different subarrays, resulting in a volume of $S \times V$. The number of rows in each subarray is R_{sb}, typically 512 [69] resulting in 128 subarrays per bank.

Theorem 9 (Optimal aggressor distribution for subarray parallelism)**.** *In the case of subarray TRR parallelism, the aggressor rows have to be distributed equally over all the subarrays in a bank to maximize* $Hammer_{max}$.

► *Proof.* The maximum number of decoys in a subarray is given by $R_{sb} \times B/(B+1)$ (remember that B is the blast diameter of an aggressor). If only one subarray is used for FEINTING, all the rows involved would be refreshed after $T_{alive} = R_{sb}/V \times B/(B+1)$. If the aggressors are distributed over a number of subarrays lower or equal to S, the same result would apply as all the rows would be refreshed in parallel. Moreover, as the number of subarrays in the attack increases up to S, the share of activations used for the victim is reduced as this only increases the number of necessary decoys,

lowering the final $Hammer_{max}$. Instead, targeting a number of subarrays higher than S means that the parallel refresh will be saturated, and some subarrays will be skipped (Fig. 3.18). Similar to FEINTING without subarray parallelism, we assume that for equal counters, the attacker can control that the victim row to have the lowest refresh priority. That is, a subarray is never picked for a refresh if at least S different subarrays exist with the same maximum row count. Because of FEINTING, all the rows are equally often activated. Considering targeting $S+1$ subarrays and using all the possible decoys, it would mean that in the first T_{alive} TRR events, the S *decoy* subarrays are completely refreshed, and in the last T_{alive} event, the rows from the victim subarrays are refreshed. In the same way as the original FEINTING, any other distribution of activations either induces a refresh on the victim subarray or is a loss because a decoy is refreshed with a higher activation count that could have otherwise been used for the victim. Therefore, the distribution of rows across subarrays should still follow the original theorems for FEINTING (see Section 3.5). To exploit the saturation as much as possible, FEINTING must be performed using all available subarrays. Fig. 3.18 shows the structure of FEINTING considering subarrays parallelism.

3.12.4 *Impact of TRR Events*

The TRR mechanism itself performs an activation when refreshing a row. This effect should be considered when deriving $Hammer_{max}$. In our study, the maximum number of activations L_{tREFW} is increased by the times TRR is performed and the TRR volume: $L'_{tREFW} = L_{tREFW} \times (1 + V/T)$. Moreover, because every T activations, V more TRRs are sent, the effective TRR interval T' is calculated as $T' = T + V$.

3.12.5 *Double-sided Rowhammer versus FEINTING*

Double-sided Rowhammer is a technique to hammer a victim row, where both its directly adjacent rows are alternatingly activated. In PROTRR, this technique is avoided as it is not beneficial for the attacker. To model the defender's worst case, we assume a closed-page policy for the DRAM device. This means that a row is automatically precharged after activating it. In other words, in an interval of T activations, a victim row can be hammered T times by accessing only one aggressor. This is the same amount of activations that can be achieved with double-sided Rowhammer

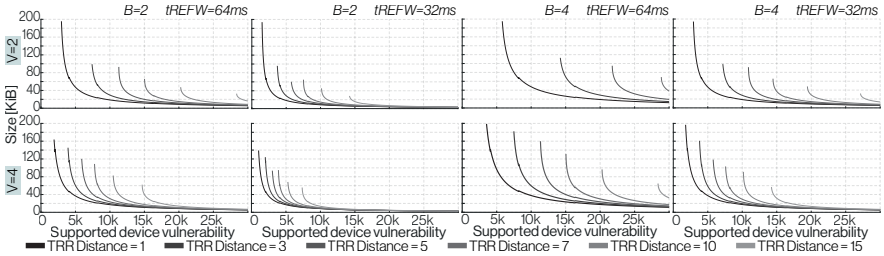

FIG. 3.19: The storage size for different possible setups and degrees of vulnerability in DDR4. The first line considers volumes of 2 and the second volumes of 4. Setups with `tREFW` of 64 ms 32 ms are alternated to blast diameters of 2 and 4.

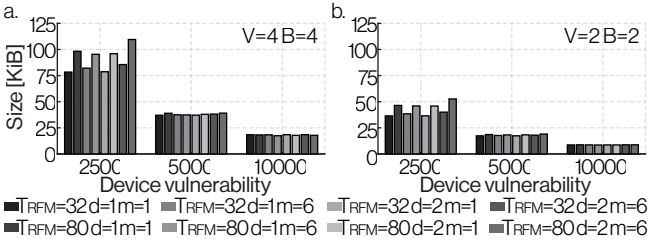

FIG. 3.20: Required storage size on DDR5 for various possible setups.

but with the difference that the total number of victims affected is higher, and as such, the total number of generated decoys. Consequently, for the same number of TRR events where the victim is not refreshed, a higher number of activations can be used against the victim, resulting in a higher $Hammer_{max}$.

3.12.6 *Extra figures*

Extended storage size analysis. Fig. 3.19 and Fig. 3.20 show the storage size required for PROTRR in different settings, including a volume of 4 at each TRR event. Fig. 3.21 reports the maximum vulnerabilities that can be protected, in various DDR5 configurations.

Details of increased RFM sent. Fig. 3.22 shows the increase in RFM sent and the increased tail latency for individual SPEC benchmarks.

Symbol	Description	Ref. (§)
tREFI	Duration of a refresh interval (tREFW/8192) in μs.	3.2.1
tREFW	Duration of a REF window, e.g. 64 ms (DDR4).	3.2.1
B	No. of rows affected by an aggressor (e.g., 2 or 4).	3.3
R_{thresh}	Number of hammer required to trigger a bit flip.	3.3
L_{attk}	Number of total activations of an attack.	3.5.1
m	The value of the MR59 register in OP[7:6].	3.4
RAA	Rolling Accumulated ACT.	3.4
RAAMMT	Maximum Management Threshold (RAAIMT $\times$ m).	3.4
RAAIMT	Initial Management Threshold.	3.4
V	TRR volume: no. of rows refreshed at every TRR event.	3.5
T	No. of ACTs in between of two consecutive REFs.	3.5
$\mathcal{A}$	Rowhammer attack: a sequence of row activations	3.5.1
$Hammer_{max}$	Max. hammer count a victim can reach before refresh.	3.5.1
L_{tREFW}	Total number of activations in a tREFW.	3.5.1
D_T	No. of rows used in FEINTING (decoys and victims).	3.5.2
$D(\alpha)$	No. of decoys that have not been refreshed at ACT a.	3.5.2
d	Distance of TRR events expressed in REFs.	3.5.2
T_{REF}	Number of activations between two consecutive REFs that perform a TRR.	3.5.3
T_{RFM}	Number of activations between two consecutive RFMs.	3.5.3
N	Maximum number of TRR events in a tREFW.	3.6.2
C	No. of counters used to track victim rows.	3.6.2
$S_{entries}$	Number of counters in a PROTRR summary.	3.7.1
N_{banks}	No. of banks of the system.	3.7.1
P_{max}	The max. number of tREFIs without any REFs.	3.8

TBL. 3.4: Overview of used symbols.

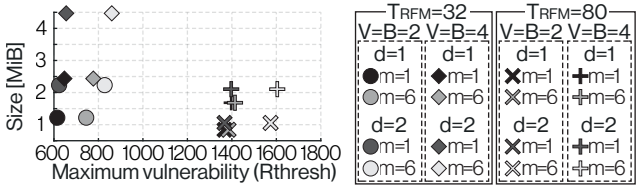

FIG. 3.21: Maximum vulnerability supported in DDR5.

Symbols. In Tbl. 3.4, we present an overview of symbols with references to the section where they were introduced.

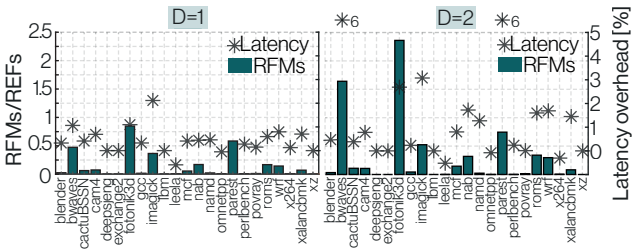

FIG. 3.22: RFMs sent relative to REFs and the increase in the tail latency for all SPEC benchmarks. D={2;1}.

4

REGA: SCALABLE ROWHAMMER MITIGATION WITH REFRESH-GENERATING ACTIVATIONS

Mitigating Rowhammer requires performing additional refresh operations to recharge DRAM rows before bits start to flip. These refreshes are scarce and can only happen periodically, impeding the design of effective mitigations as newer DRAM substrates become more vulnerable to Rowhammer, and more "victim" rows are affected by a single "aggressor" row.

We introduce REGA, the first in-DRAM mechanism that can generate extra refresh operations each time a row is activated. Since row activations are the sole cause of Rowhammer, these extra refreshes become available as soon as the DRAM device faces Rowhammer-inducing activations. Refresh operations are traditionally performed using sense amplifiers. Sense amplifiers, however, are also in charge of handling the read and write operations. Consequently, the sense amplifiers cannot be used for refreshing rows during data transfers. To enable refresh operations in parallel to data transfers, REGA uses additional low-overhead buffering sense amplifiers for the sole purpose of data transfers. REGA can then use the original sense amplifiers for parallel refresh operations of other rows during row activations.

The refreshes generated by REGA enable the design of simple and scalable in-DRAM mitigations with strong security guarantees. As an example, we build REGA_{M}, the first deterministic in-DRAM mitigation that scales to small Rowhammer thresholds while remaining agnostic to the number of victims per aggressor. REGA_{M} has a constant 2.1% area overhead, and can protect DDR5 devices with Rowhammer thresholds as small as 261, 517, and 1029 with 23.9%, 11.5%, and 4.7% more power, and 3.7%, 0.8% and 0% performance overhead.

4.1 INTRODUCTION

Rowhammer has been a moving target when it comes to mitigations. Worsening Rowhammer thresholds in newer DRAM devices have enabled new access patterns that bypass all deployed in-DRAM mitigations [7]. New

Rowhammer effects such as half-double further challenge the design of secure mitigations as they increase the number of potential victim rows for a single aggressor row [34]. The next generation of Rowhammer mitigations will require exceedingly more refresh operations to protect potential victim rows. Unfortunately, the current DRAM architecture provides limited capabilities for additional refreshes.

Since Rowhammer is triggered by DRAM row activations, the only way to scale the number of required refresh operations is during the activations themselves. We present a new in-DRAM mechanism called REfresh-Generating Activation (REGA). REGA time-multiplexes existing DRAM resources using additional lightweight elements, enabling parallel refresh operations while a row is being activated. These parallel refreshes allow for simple, effective, and scalable in-DRAM mitigations against current and future Rowhammer attacks while respecting the respective standards [12, 20].

Rowhammer attacks. Rowhammer is part of a broader class of reliability issues known as disturbance errors. Kim et al. [3] showed that Rowhammer affects most DRAM devices in production settings. To trigger Rowhammer, an aggressor row in DRAM must be accessed repeatedly. If this happens frequently enough, bits start to flip in the neighboring victim rows. Follow-up research showed that these reliability errors can compromise systems in a variety of scenarios, most notably, to escalate privileges [24, 26, 34], compromise the browser [9, 52–54], phones [55, 56, 78], clouds [10, 25, 49, 62], and across the network [57, 58]. Given the severity of these attacks, numerous mitigations have been devised by both academia and industry.

Mitigations. Most research in academia proposes to modify the CPU's memory controller (MC) to track aggressor rows, for blocking them before they cause bits to flip in their victims [42], or to send preventive refreshes to their victim rows [3, 40, 41, 59–61, 80]. In contrast, in recent years, industry has adopted mitigations that solely operate inside DRAM given the high cost of mitigating Rowhammer inside the CPU [5]. These mitigations track aggressor rows and issue preventive refreshes, also known as Target Row Refresh (TRR), to potential victim rows. Recent academic work provides a formal foundation for designing secure in-DRAM TRR mitigations [23].

New effects and new patterns. The additional refreshes that can be generated inside DRAM are scarce, and generating them from the CPU negatively impacts the performance [40]. This forces the mitigations to keep track of aggressor rows to utilize these precious refreshes only when necessary.

Tracking aggressors requires assumptions on the behavior of Rowhammer, which is constantly changing with newer technology nodes: the number of required aggressor accesses is rapidly dropping [17], while the number of affected victim rows for a given aggressor is increasing [99]. Researchers were quick to show that these new effects enable new access patterns that evade the mitigations on newer devices [5, 7, 34]. Learning from these experiences, the next generation of Rowhammer mitigations must not tailor their design towards specific (known) behaviors of Rowhammer. The question is how to make this possible given the limited refreshing capability in the current DRAM architecture.

REGA. To cleanly decouple the mitigation mechanism (i.e., additional refreshes) from Rowhammer-dependent policies, we should minimize or completely eradicate the state that is necessary for tracking aggressors. Our new in-DRAM mechanism, REGA, makes this possible by allowing parallel refresh operations whenever DRAM receives an activate command to access a row. Because row activations are the sole cause of triggering Rowhammer, these parallel refreshes are immediately available for mitigating Rowhammer without the need for tracking aggressors. As an example, we have built a simple and scalable in-DRAM mitigation on top of REGA, called REGA_{M}, that sequentially refreshes all rows in a DRAM sub-array that receives activate commands.

Refresh-generating activations. Modifying DRAM to enable parallel refresh operations is non-trivial due to its highly-optimized architecture and stringent timing requirements dictated by the respective DDRx standards [12, 20]. Without altering the dense DRAM mats (where the data is stored), REGA enables refresh-generating row activations by decoupling row refreshing and data transfer operations performed by the DRAM sense amplifiers. REGA achieves this using a second set of low-overhead buffering sense amplifiers for the sole purpose of supporting data transfers, while the original sense amplifiers are used for parallel refresh operations.

Operating the bitline wires inside the DRAM mats with the additional sense amplifiers complicates the timing of the internal DRAM signals. To ensure the correct operation of REGA, we developed a new accurate DRAM model in collaboration with a DRAM vendor. On top of showing the correctness of REGA in this model, we show that the time between two row activations is sufficient for REGA to perform a single parallel refresh on all 21 DDR4 and 16 DDR5 devices in our test pool. We can further scale

REGA to refresh multiple rows during a single activation by increasing the time a row must remain active, defined in the standard as `tRAS`.

Our evaluation using an ASIC implementation of the mitigations on top of REGA, and an analog and cycle-accurate simulation of REGA itself, demonstrates a constant 2.1% area overhead, independent of the degree of Rowhammer vulnerability. The performance and power overhead of REGA depends on the number of refreshes it needs per activation to protect DRAM devices with varying degree of Rowhammer vulnerability. As an example, REGA_{M} protects DDR5 devices with Rowhammer thresholds as small as 261, 517, and 1029 with 23.9%, 11.5%, and 4.7% more power, and 3.7%, 0.8% and 0% performance overhead; regardless of the number of victims per aggressor. These results show that REGA is the first mitigation that can comfortably scale to Rowhammer thresholds that are up to 36× smaller than previously reported (i.e., 9.8 K [17]). Without relying on other mitigations (e.g., ECC), we estimate that REGA can comfortably provide Rowhammer protection for another 10 years. Furthermore, REGA can be scaled to even lower thresholds using the same circuitry by increasing `tRAS` if necessary.

Contributions. The following summarizes our contributions:

1. We present REGA, the first in-DRAM mechanism that can scale the number of required refreshes with activations.
2. We develop a new, accurate model of DRAM internals in collaboration with a DRAM vendor. We use this model to ensure the correctness of REGA. To enable future research on DRAM architecture, we release this model as open source to the community: `https://comsec.ethz.ch/rega`.
3. We design and implement a new in-DRAM mitigation on top of REGA, called REGA_{M}, that can protect against current and future Rowhammer attacks.
4. We evaluate the correctness, performance, area, and power impact of REGA and REGA_{M} using an ASIC implementation, SPICE and cycle-accurate simulations. REGA_{M} has a constant minimal area impact while its performance and power overhead scales to very small Rowhammer thresholds.

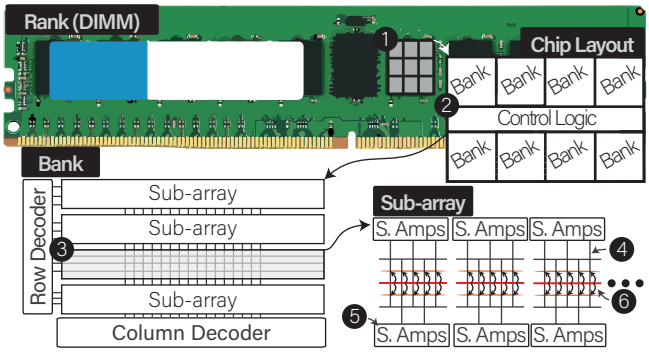

FIG. 4.1: **DRAM.** Example of multiple DRAM chips on a DIMM (❶). The chip layout is divided into banks (❷) and includes a control logic pad. Each bank is comprised of multiple sub-arrays (❸), in turn composed of mats (❹). Each row spans across an entire sub-array. Each mat is surrounded by bitline sense amplifiers for accessing data inside the mat (❺). Rowhammer affects victim rows adjacent to a repeatedly accessed aggressor row inside a sub-array (❻).

4.2 BACKGROUND AND MOTIVATION

We describe DRAM's organization and operation (§4.2.1), discuss Rowhammer attacks and defenses (§4.2.2), and the ongoing Rowhammer trends with DRAM technology scaling and what they entail for future defenses (§4.2.3).

4.2.1 *Organization and Operation of DRAM*

Logical Organization. The DDR*x* standard [12, 20] describes the logical organization and operation of DRAM devices attached to an external memory controller (typically, in the CPU). Multiple DRAM devices are usually operated simultaneously by the memory controller in a dual in-line memory module (DIMM), as shown in Fig. 4.1. The memory controller can access DRAM by providing an address, which specifies a DRAM bank, a row inside that bank, and a column inside that row that identifies a single byte of data. Internally and abstracted away from the memory controller, each bank is organized into multiple sub-arrays. Each sub-array contains multiple DRAM mats arranged in rows and columns. Each DRAM row consists of capacitors (or cells), each storing one bit of information as an electrical

charge. To access a specific column, the entire DRAM row is first selected by a row decoder in the sub-array. The charges in the capacitors of the selected row are then detected by sense amplifiers, collectively known as the row buffer. A column decoder then selects the sense amplifiers associated with the requested column.

The DDR Protocol. To access data in DRAM, the memory controller must first activate the row by sending an `ACTIVATE` (`ACT`) command to a specific DRAM bank. After sending an `ACT`, the memory controller must wait `tRCD` for reliable logical values to become present in the sense amplifiers. Once this happens, the MC can send a `READ` or `WRITE` command to a selected column. The data will then be read from or written to the DRAM bus after `tCL` or `tCWL`, respectively. DRAM allows one row of a bank to be activated at a time. Thus, if a different row is requested, the memory controller must first deactivate the currently activated row by issuing a `PRECHARGE` (`PRE`) command. The `PRE` is considered complete after waiting for `tRP`, and the new row of choice can now be activated by sending an `ACT` for that row. Independently of the DRAM operations, a row can be precharged only after a minimum of `tRAS` has passed since its activation.

As discussed, DRAM saves data on capacitors that leak charge over time. Therefore, to prevent data corruption, the capacitors' charge is periodically restored via the `REFRESH` command (`REF`). `REF` has to be sent on average every `tREFI` to keep data integrity. DRAM handles `REF` as multiple row activations where an incremental index defines a subset of rows that are activated and precharged simultaneously. We will provide more information on the organization and operation of DRAM when describing our accurate DRAM model in Section 4.5.3.

4.2.2 *Rowhammer*

As shown in Fig. 4.1, electrical interferences generated by activating a row (aggressor row) can accelerate charge leakage in physically adjacent rows (victim rows). If the victim rows' charges leak fast enough, i.e., before these rows are refreshed by a `REF`, DRAM can no longer detect the values that were previously stored in these cells, and bits start to flip. This type of crosstalk was of concern already in 2002 [4] and was publicly demonstrated for the first time in 2014 [3]. Shortly after, security researchers showed many critical attacks based on this effect, now famously known as Rowhammer.

Attacks. Rowhammer allows an attacker to compromise the integrity of data stored in adjacent rows that are assumed not be accessible to the attacker. Rowhammer attacks often go through three stages: templating, memory massaging, and exploitation [25]. First, the attacker identifies vulnerable memory locations (templating). The attacker then forces the system to store security-sensitive information in these locations (memory massaging) and retriggers Rowhammer to corrupt the security-sensitive information (exploitation).

Previous research has shown that Rowhammer can be used for local privilege escalation [24, 26, 34, 55, 56], compromising co-located cloud virtual machines [10, 25, 49], web browsers [9, 52–54] and even machines across the network [57, 58]. It has recently even been shown that Rowhammer can increase the strength of Spectre attacks [100].

Defenses. Common mechanisms to mitigate Rowhammer include preventive refreshes to victim rows, which are broadly known as Target Row Refresh (TRR) [5], isolation of sensitive data, and error-correcting codes (ECC). Isolation of sensitive data requires software changes, making it challenging to deploy in practice. ECC complicates Rowhammer attacks but does not prevent them [62]. Furthermore, the increasing density of bit flips in newer technology nodes makes secure ECC schemes rather expensive [6]. As a result, TRR is the only specific mitigation that is currently deployed for tackling Rowhammer.

TRR can be implemented in software [16, 56, 57, 63, 64], in the CPU's MC [3, 40–42, 44, 59–61, 80], and inside DRAM [23, 35, 39, 82]. Mitigating Rowhammer is costly, making its deployment unattractive for software and CPU vendors. Consequently, TRR is currently deployed inside the component affected by Rowhammer, which is DRAM itself. Unfortunately, recent work shows that all existing implementations of TRR in DDR4 DRAM devices are vulnerable to special access patterns [7, 34]. This is due to the limited availability of additional refresh operations to perform TRR in the current DRAM architecture and the increasing cost of tracking potential aggressor rows for using these refresh operations judiciously. New Rowhammer effects due to technology scaling further accentuate the cost of future mitigations.

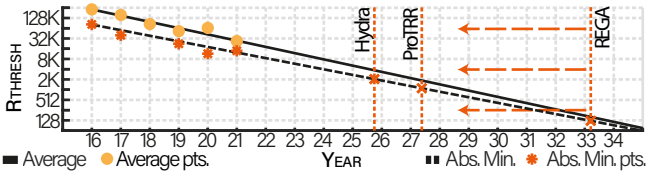

FIG. 4.2: **Trend of R_{thresh} from 2016 to 2035 (B=4).** The curve represents the minimum number of activations to induce bit flips, both for the absolute minimum values and for the average. Vertical orange lines report the minimum R_{thresh} for ProTRR [23], Hydra [40] and our mitigation, REGA. For REGA, the minimum depends on its configuration, as later discussed in the paper. The fitting is based on previous work [6, 87]. We refer the reader to Appendix 4.12.1 for details about the mitigations' threshold and the estimation methodology.

4.2.3 *The Impact of Technology Scaling on Rowhammer*

There are two Rowhammer trends as DRAM moves to smaller technology nodes: the worsening Rowhammer vulnerability and the Rowhammer impact of a single aggressor. As we show, these factors have an heavy impact on existing Rowhammer mitigations.

The worsening Rowhammer thresholds. The original Rowhammer study with DDR3 DRAM [3] measured the minimum number of activations required to induce bit flips (i.e., R_{thresh}) to be in the order of 139 K to 284 K. A more recent study showed that the R_{thresh} is dropping with each new generation of DRAM devices, reaching as few as 9.6 K activations [17]. Fig. 4.2 shows that R_{thresh} has dropped almost 15 times since Rowhammer was first demonstrated and is posed to drop even further with future generations of DRAM devices. Fig. 4.2 further shows that proposed mitigations can only cope for a few more years with the dropping R_{thresh}. This is due to the fact that they need to keep a non-trivial amount of state to use the available row refreshes without significantly sacrifying performance. We show that our proposed mitigation, REGA, provides *better scalability* against Rowhammer in future devices by reducing the cost of refreshes inside DRAM.

The increasing impact of a single aggressor. An aggressor row impacts the victim rows directly adjacent to it. This means that the aggressor row has a blast diameter (B) of 2 victim rows. Recent work [34] shows that the blast diameter has increased to 4 in more recent DRAM devices. The blast diameter is posed to further increase, and we are likely to see de-

vices with blast diameters of 6 or even higher based on a recent JEDEC specification [12]. None of the existing mitigations has so far considered blast diameters of 6 or above. We show that the refreshes generated by our new in-DRAM mechanism enable the construction of mitigations that are *independent of the blast diameter*.

4.3 THREAT MODEL

We assume that the CPU's MC complies with the respective DRAM standard. We assume the software to be untrusted: the attacker can send ACT commands to target DRAM rows through native code execution [10, 24–26, 55, 56], a browser tab (e.g., running JavaScript) [9, 52, 53], or over the network [57, 58] to trigger Rowhammer bit flips that compromise the target system. We assume that DRAM is vulnerable to Rowhammer and that bits start to flip if (aggressor) rows receive a certain number of ACT commands before the standard refresh mechanism in a tREFW refreshes their victims. All recent DRAM devices are reported to be vulnerable to Rowhammer [7, 17]. While some Rowhammer effects are known (e.g., half-double [34]), we assume more will be discovered over time, and our proposed mitigation should be able to cope with that.

4.4 OVERVIEW

Our goal is to design a new mechanism that can scale the number of additional refreshes as needed to defend against current and future Rowhammer attacks. We first overview the requirements for this new mechanism, called REGA, before discussing the challenges in designing it.

4.4.1 *Requirements*

We define our requirements (**R1**–**R3**) around scalability, (forward) security, and practicality.

Refresh scaling. The DDR4 protocol only allows issuing extra refreshes during REFs [95]. The DDR5 protocol improves this by introducing the RFM command [12, 23], which can periodically send additional refreshes to help mitigate Rowhammer. However, its periodic nature requires mitigations to

keep a state (e.g., counters) to decide which victims to refresh. As discussed before (Section 4.2.3), maintaining this state becomes more expensive with smaller technology nodes. Further, shortening the RFM period to allow for more refreshes would negatively impact performance. Instead, an ideal mechanism should allow scaling REFs as needed, up to generating an extra REF, or more, for each ACT received by the DRAM device.

Requirement (R1). REGA should allow scaling of extra REFs as needed.

Forward security. Deployed mitigations do not decouple mechanisms from policies. This impedes their adaptability to new Rowhammer effects, such as the rapidly dropping R_{thresh} with smaller technology nodes [17] and the half-double effect [34]. The main reason is the need for careful state management in hardware due to the scarcity of extra REFs. Hence, as a second requirement, we define the possibility of configuring simple (stateless) policies to mitigate current and future Rowhammer effects.

Requirement (R2). REGA should decouple its mitigation mechanism from policies to counter current and future Rowhammer effects.

Practicality. An in-DRAM solution must meet two requirements to make its deployment practical. First, the new REF mechanism should have a small impact on current and future technology nodes. Requiring more area with smaller technology nodes will reduce the benefits of using the smaller technology nodes in the first place. Moreover, the DRAM mats should not be changed since they are the most important and highly optimized building block of today's DRAM chips. Second, a new mitigation should operate within the bounds defined by the DDR standards [12, 20] as protocol changes are a multi-year effort requiring consensus among all involved parties.

Requirement (R3). REGA should have a minimal impact on the area and layout of the DRAM while operating inside the bounds of the respective DDR standard.

Next, we discuss how fulfilling these requirements introduces challenges in the design and implementation of REGA, and summarize how we address these challenges in the rest of the paper.

4.4.2 *Challenges*

REGA requires changing certain DRAM elements to achieve low-cost and scalable refresh generation. To ensure that our changes do not compromise the device's functionality, it is paramount to use an accurate DRAM model. Currently, such a model does not exist, which brings us to our first challenge:

> **Challenge (C1).** Deriving an accurate model of DRAM internals that represents modern DRAM devices.

We address this challenge in Section 4.5 by explaining the details of the internal DRAM architecture and presenting REGA Model (REM), an accurate DRAM model that we developed in collaboration with Zentel Japan. Furthermore, we show that REM captures timings and internal signal propagations more precisely than the previous state-of-the-art DRAM model. Equipped with REM, we seek to understand and modify the DRAM architecture to fulfill requirements **R1** and **R3**.

So far, extra refreshes have been generated only in response to `REF` or `RFM` commands, which are periodic and do not directly scale with a decreasing R_{thresh}. Hence, to fulfill **R1**, the DRAM chip must be able to generate these extra refreshes as part of `ACTs`. To fulfill **R3**, these extra refreshes should not change the DRAM timings, as defined in the respective standards [12, 20]. Consequently, we propose generating these refreshes in parallel with `ACT` commands.

> **Challenge (C2).** Generating refreshes in parallel with `ACTs`.

In Section 4.6, we discuss possible placement options for REGA and explain involved timings, leading us to its final design. Because we need direct control of rows, REGA is deployed next to the mats. To perform extra refreshes in parallel with every `ACT`, REGA must finish before a `PRE`. We describe the additional circuitry to achieve this capability.

REGA's implementation needs to time-multiplex wires in the mats, which implies that more operations must be performed during `tRAS`. Through experiments conducted on real DDR4 and DDR5 devices in Section 4.7, we show that this is already feasible in today's existing DRAM designs. To protect devices with ultra-low R_{thresh}, REGA requires increasing the `tRAS` further.

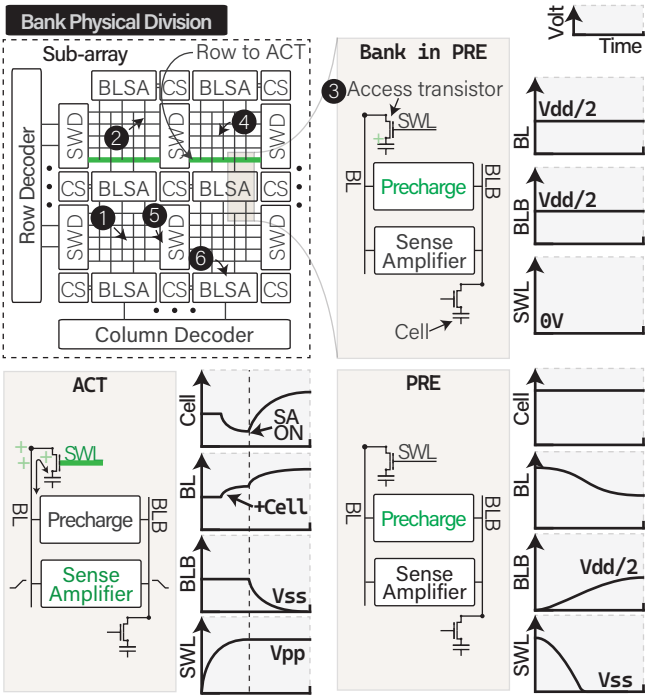

FIG. 4.3: **DRAM's internal physical organization.** Top left: DRAM bank composition. Top right: voltage levels when a bank is in the precharged state. Bottom left: voltage levels when a row is activated. Bottom right: voltage levels when going from an activated to a precharged state. The voltage levels under these different states are discussed in Section 4.5.2.

With REGA in place, we address **R2** by building a secure, configurable, and future-proof mitigation against Rowhammer.

> **Challenge (C3).** Building a secure and configurable mitigation on top of REGA.

In Section 4.8, we present REGA_M, a new deterministic mitigation on top of REGA that protects against all known and **future** Rowhammer attacks by relying on the sole assumption that Rowhammer is induced by activations. REGA_M is secure against new Rowhammer effects by design, and its power consumption can be configured depending on the R_{thresh}.

4.5 ACCURATE MODELING OF DRAM

We now describe the details of DRAM's internal physical organization (§4.5.1) and focus on the sense amplifier's operation (§4.5.2). Then, we present our novel DRAM model and compare it with the state-of-the-art DRAM model [101] (§4.5.3). A summary of all abbreviations and symbols introduced in this section can be found in Appendix 4.12.

4.5.1 *Physical Organization*

In DRAM, data is saved in cells as a full charge (`Vdd`) or an empty charge (`Vss`). Depending on the encoding, either one of the two charge levels can correspond to a logical one (`1b`) [102]. Physically, cells are organized in compact, hierarchical, and matrix-based structures as shown in Fig. 4.3. These structures (*mats*) are composed of 512×1024 cells (❶), and each cell is connected to a column bitline (❷), via an *access transistor* (❸) [87, 102, 103]. Along a mat's row, all access transistors share the same control line, the *sub-word line* (SWL) (❹), which is raised upon row activation by the *sub-word line driver* (SWD, ❺) [104, 105]. Functionally, the SWL corresponds to the logical row selector. A single row address uses multiple mats. Multiple rows, typically 512, form a sub-array and multiple sub-arrays form a bank [106, 107].

The bitlines are connected to bitline sense amplifiers (BLSAs, ❻) placed on top or on the bottom of the mat in an alternating manner. Each BLSA is connected to two bitlines, one coming from the mat above (BL) and one from the mat below it (BLB). During the sensing operation, one bitline transmits information, and the other is used as a reference voltage. This *open bitline* design reduces crosstalk between bitlines and improves space efficiency. Along a mat's row, each cell has a unique bitline, but along a column, all cells share the same bitline.

Activation process. After the memory controller sends an `ACT`, the associated SWLs are raised by the SWDs. All the cells of the row get connected to the BLSAs, which are subsequently activated. The BLSAs read and amplify the stored data; as they read it, they also recharge the cells. After the sensing operation stabilizes to a logical value, data can be read (or written) by specifying a column. The column's data is obtained by connecting one byte from each mat to the local I/O (LIO). The LIO precedes the global I/O

(GIO), and other data manipulation, such as further sensing operation (I/O sense amplifier), serialization, and I/O line drivers [108].

4.5.2 *Sense Amplifier*

We now describe the bitline sense amplifier and its operation. In DRAM, the cell capacitance is in the order of femtofarads (fF) [109], which allows for high-density memories. However, because of the low capacitance, sense amplifiers are needed to amplify the cell's value to become interpretable. Moreover, sense amplifiers are also used to restore the capacitors' charge. A sense amplifier is a circuit with two inputs: BL and BLB (Fig. 4.4center). When the sense amplifier is activated, it *senses* the voltage difference between BL and BLB, amplifies it, and obtains its rail-to-rail output. That is, if $V(\text{BL}) > V(\text{BLB})$ for voltage V, then BL is raised to `Vdd` and BLB is brought to `Vss` (whose voltage levels are given in Appendix 4.12.3). These values are then held or *latched*. As described before, only BL or BLB will be connected to a cell.

Charge sharing. Before the first `ACT` reaches a bank, the bank is in a precharged state (Fig. 4.3, "Bank in `PRE`"). Consequently, all SWLs are low, and all BLSAs are `OFF`. A precharge circuit keeps the bitlines' voltage at `Vdd`/2. After an `ACT` is received (Fig. 4.3, "`ACT`"), the precharge is turned off, allowing the bitline's voltage to change. Then, the corresponding SWLs are raised to connect the cells to the bitlines.

For a given BLSA, when the access transistor is activated the bitline voltage varies depending on the charge stored in the cell. As the other bitline connected to the BLSA remains at `Vdd`/2, the difference between BL and BLB can be amplified by activating the sense amplifier (Fig. 4.4-center, signals `S`/`S#`). Because the bitline has a high parasitic capacitance, the cell's value is lost during this operation, which is called *charge sharing*. The sensing operation amplifies the signal to enable retrieving its logical value, and restores the charges in the cells.

Precharge. When `PRE` is received (Fig. 4.3, "`PRE`"), the SWLs' voltage is lowered, thus disconnecting the cells. Now the BLSAs are driving the bitlines to either `Vdd` or `Vss`. At this point, if another row were to get activated, the values in this other row would get corrupted [110]. To avoid this, after having turned the sense amplifier to `OFF`, the precharge circuit (Fig. 4.4left) brings the bitlines back to `Vdd`/2.

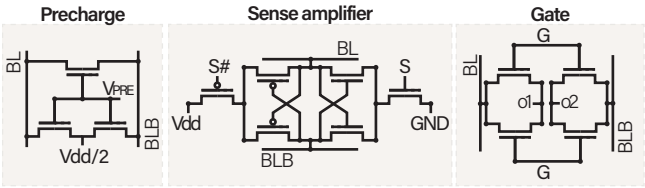

FIG. 4.4: **Relevant electronic circuits.** Left: the precharge circuit is used to bring the bitline to the reference voltage (`Vdd/2`). Center: the sense amplifier circuit is used to sense and amplify the charge inside the cell's capacitor and recharge it when needed. Right: the gate circuit is used as a multiplexer in the design of REGA, as discussed in Section 4.6.

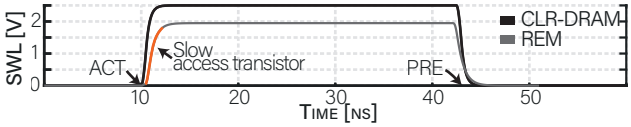

FIG. 4.5: **DRAM model comparison (SWL).** CLR-DRAM inaccurately assumes an optimistic rise of the SWL. This results in faster (de-)activation of the access transistor.

Timings and parasitics. The operations we described are delayed by parasitic elements of the lines (i.e., resistance and capacitance). For example, the SWL's parasitics slow down the (de-)activation of the access transistors. The bitline's parasitics reduce the sensed voltage, slow down the signal propagation, and increase the time required for the precharge operation. These effects are the key elements for a correct and realistic DRAM simulation. Yet, no accurate and up-to-date circuit description based on real devices exists today.

We collaborated with a DRAM vendor to overcome this limitation by designing an accurate model called the REGA Model (REM), using details from a real DRAM design. Next, we provide additional details about REM and show that it captures DRAM details that existing models [101, 104] do not. We open source REM to enable further research on DRAM architecture and its security in the following URL: `https://comsec.ethz.ch/rega`.

Property	DIFF	IN	Explanation
Transistors sizes		✔	Inaccurate ratios
Voltage sources	✔		Simplified sources
Control voltages	✔		Simplified control
Line parasitics		✔	Low parasitics
SWL drivers		✔	Not modeled
LIO & MIO loads		✔	Not modeled

TBL. 4.1: **Identified issues in CL-DRAM compared to REM.** Summary of differences (DIFF) and inaccuracies (IN) of CL-DRAM compared to REM.

4.5.3 *REM*

In the following, we present REM and compare it with the state-of-the-art model described in CLR-DRAM [101]. In CLR-DRAM, the authors propose DRAM architecture modifications to improve performance. To derive our accurate model, REM, the DRAM vendor provided us with real physical values obtained from DDR4 devices and the circuit we use in this work. REM and CLR-DRAM differ substantially.

Overall, accurately describing DRAM requires knowing the (i) transistor dimensions, (ii) source voltages, (iii) control voltages, and (iv) line parasitics. Tbl. 4.1 summarizes our findings. We divide our comparison with CLR-DRAM into inaccuracies (**IN**) and circuit differences (**DIFF**). Inaccuracies are fundamental as they affect the simulation's validity. Differences in the circuit may not always be critical as they could emerge from alternative but correct DRAM topologies. Lax timing constraints are easier to comply with, and the resulting power consumption will generally be underestimated. More importantly, an inaccurate model does not have enough fidelity to confirm the feasibility of proposed DRAM modifications. We now describe the differences between REM and CLR-DRAM.

Sub-word line (IN). A major inaccuracy of CLR-DRAM is the SWL's characterization. First, this model considers SWL as a single element instead of a transmission line. Second, the model underestimates the line resistance and parasitic capacitance. Lastly, the SWDs are not modeled at all, heavily impacting the reliability. We observed that SWL's rising time, illustrated in Fig. 4.5, is one of the slowest in the circuit when modeled accurately. A slow SWL delays the activation of the access transistor, which further delays the sense amplifier's activation. Evaluating the SWL's *actual* timing is essential to assess the feasibility of performing REGA's parallel refresh within

tRAS. Likewise, the SWL timing affects most architectural modifications as it affects the precharging and activation speed.

Voltages (DIFF). CLR-DRAM considers only two voltage sources in the circuit: 1.2 V and 2.5 V. Our model describes voltages controlled by 1 V, 1.1 V, 1.4 V, 1.5 V and 2.5 V sources (see Section 4.12.3). Modeling incorrect voltages may affect switching speeds, noise robustness, and power consumption. Our bitlines are referenced to a maximum of 1 V, which makes the sensing operation more difficult yet realistic.

Sense amplifier's transistor ratios (IN). Transistors ratios characterize the speed of operation of the sense amplifiers, reflecting in noise sensitivities and timings. For the sense amplifiers, CLR-DRAM uses overly optimistic ratios and optimistic absolute transistor sizes. In other cases, CLR-DRAM uses pessimistic ratios.

Data path (IN). We modeled the local and global I/O lines, which all previous models omitted. LIO and GIO lines act as a load to the BLSA during read/write operations.

Sense amplifier circuit (DIFF). So far, literature has kept adopting the textbook DRAM model [104] where the BLSAs are supplied with a single voltage Vdd. However, in reality modern DRAM implementations may differ, with BLSAs that can be overdriven [107, 111, 112]. The DRAM vendor confirmed that they use overdriven BLSAs, and we designed our DRAM model to implement overdriven BLSAs as described by them. Such BLSA can switch between the common cell high voltage of 1 V and a higher (overdrive) voltage of 1.4 V. The goal of the overdrive voltage is to accelerate the first phase of sense amplification, while the lower 1 V, used in the second phase, saves energy and avoids overcharging the cell.

4.6 REGA

Equipped with REM, we proceed to the design of REGA. Parallelizing an additional DRAM operation next to an ongoing one has (to the best of our knowledge) not yet been explored. We provide a high-level description of how REGA achieves this in §4.6.1, before discussing implementation details and internal signal timing control in §4.6.2 and §4.6.3, respectively. We also discuss supporting multiple refreshes per ACT in §4.6.4.

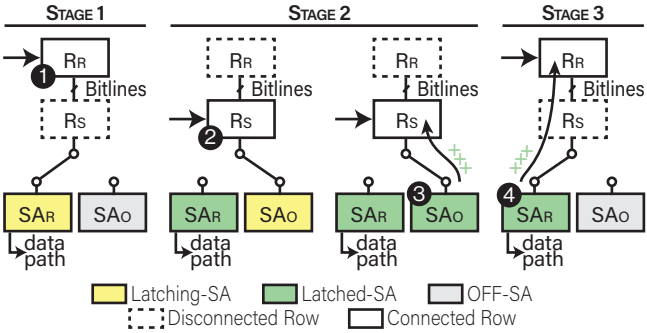

Fig. 4.6: **REGA's concept.** Row requested (R_R) is activated and REGA sense amplifier (SA_R) starts latching its charge (❶). While SA_R latches the value, R_R and SA_R are disconnected from the bitline. Row shadow (R_S) and the original sense amplifier (SA_0) are activated (❷). R_S is recharged by SA_0 (❸). SA_R and R_R are connected, and R_R is recharged (❹).

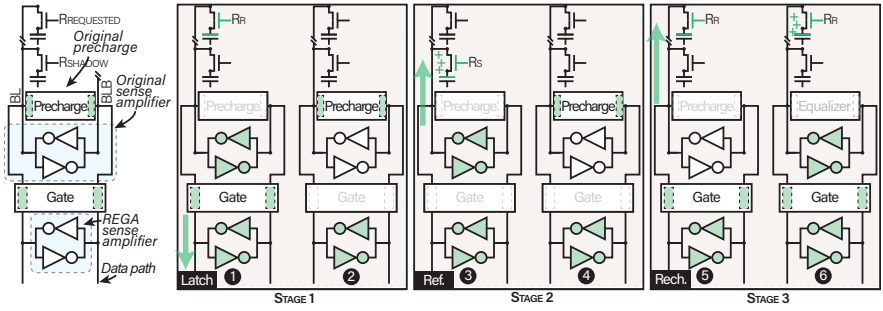

Fig. 4.7: **REGA's design.** The DRAM receives an ACT for R_R. After turning OFF the precharge and connecting R_R, SA_R starts latching supported by SA_0 (❶). After SA_R has started amplifying the voltage, the gate and R_R are disconnected, the bitline is precharged, and SA_0 is turned OFF (❷). SA_R remains active to support reads and writes. When the bitline has been precharged, R_S is connected, and SA_0 starts refreshing it (❸). After R_S has been refreshed and R_S is closed, the bitline is precharged (❹). R_R and SA_R are connected to the bitline (❺). Lastly, SA_0 is turned ON to support the refresh operation on R_R (❻).

4.6.1 *High-Level Operation*

To parallelize DRAM operations, one obvious direction is doubling the bitlines to make cells in different rows accessible at the same time. Unfortu-

nately, this introduces a significant per-cell area overhead in the otherwise highly-optimized DRAM mats. This means that to remain area-efficient, REGA must multiplex the bitlines during the parallel operations.

Time-multiplexing the bitline. The most effective scenario for a Rowhammer attack is one that maximizes the number of activations by simply alternating ACT and PRE on an aggressor row [23, 41]. This means that REGA should generate the necessary refresh either during ACT or PRE. In the same bank, the time between a PRE and an ACT is tRP, typically 13.75 ns. As discussed in Section 4.5, DRAM requires this time to bring the bitlines back to the reference voltage. Therefore, we cannot use tRP for our parallel operation.

The time between an ACT and a PRE is defined as tRAS, which is at minimum 32 ns. The row activation at the beginning of tRAS also uses the bitlines. However, since tRAS is relatively long, we can leverage it to multiplex the bitlines for our parallel operation in REGA.

Reads and writes. During bitline multiplexing, the memory controller will send either READ or WRITE. As described in Section 4.5, these commands can be sent tRCD after an activation, which is less than tRAS. REGA should comply, allowing reads and writes while refreshing an additional row in parallel.

Shadow refresh. REGA allows normal DRAM operations on a requested row while simultaneously enabling refreshing another row. We denote the requested row by R_R and the shadow row by R_S (i.e., the row that gets refreshed in parallel). One may propose using the bitline sense amplifiers to first read data from R_R (as part of ACT) and then to refresh R_S. However, after R_R has been activated, the memory controller can read from/write to any column at any given time. Therefore, there should be two sets of sense amplifiers: the original sense amplifiers (SA_0) to perform refresh operations, and a new set of sense amplifiers, called the REGA sense amplifiers (SA_R), to hold the values that could be read (or written). In practice, SA_R acts as a buffer to the original SA_0.

Fig. 4.6 shows the high-level operation of REGA. To enable time-multiplexing of the bitlines, we make a key observation that for the sense amplifiers to start their mechanism, **charge sharing** is the only required operation (❶). This step provides the logical values that can be read by the memory controller in time with a tRCD. After the charge sharing phase, the bitlines can be disconnected and connected to a different row and sense amplifiers (❷). When the second set of sense amplifiers is connected, REGA refreshes

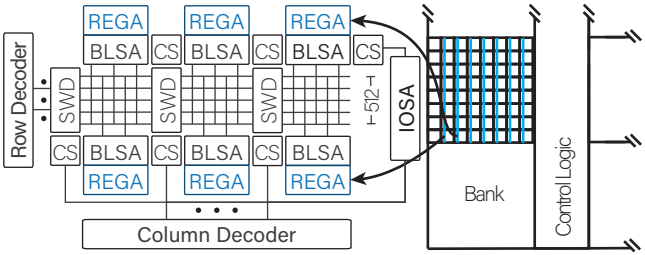

FIG. 4.8: **REGA's Internal Deployment.** REGA is deployed next to the original bitline sense amplifier (BLSA).

the shadow row (❸). After the shadow refresh, the R_R is reconnected (❹), allowing any operation conforming to the standard to be executed identically to what is possible after a normal ACT on R_R. In other words, REGA does not require changing the DRAM standard, hence satisfying **R3**. The time available to perform both refreshes is tRAS; after this, the memory controller can send a PRE. If tRAS is preceded by a read, the value can be read by the sense amplifier that had latched the logical value. If tRAS is preceded by a write, the new value will replace the one latched by SA_R, later used to finish refreshing R_R.

4.6.2 *Low-Level Operation*

REGA requires replicating existing elements in the sub-array's circuit as shown in Fig. 4.8. In particular, additional sense amplifiers and transmission gates (Fig. 4.4right) are required. No modification to the data path is necessary. We now describe REGA's operation for a single sense amplifier, as the same applies to all of them. We assume that the device receives an ACT for R_R, which may or may not be part of an attack, and we assume that the shadow refresh targets R_S. REGA's complete design for a single cell is in Fig. 4.7.

Circuit's basics. For each bitline, two sets of sense amplifiers are used: the already present, "original" sense amplifier (SA_0), and our addition, the REGA sense amplifier (SA_R). SA_R latches the requested row's value and serves as an I/O buffer. SA_0 refreshes the shadow and requested rows. REGA uses the existing circuit to precharge the bitline.

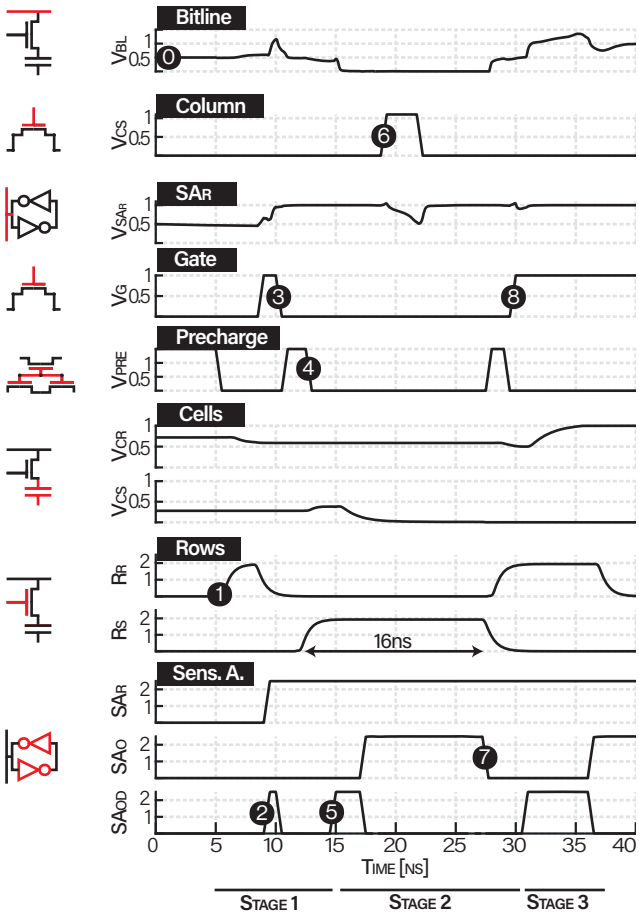

FIG. 4.9: **Timing of REGA.** Initially, the bitline is precharged (❶). An ACT is received for R_R, and its word line is risen (❶). After the charge sharing operation, SA_{0D} helps SA_R latching the cell's value (❷). After SA_R has started the latching operation, the gate is opened to disconnect SA_R from the bitline (❸). Now, the bitline is brought back to the reference voltage via the precharge circuit (❹). R_S is activated, and SA_0 and SA_{0D} perform a normal refresh operation (❺,❼). Parallel to R_S refresh, a read operation occurs (❻) by reading from SA_R. Once R_S has been refreshed, SA_0 is turned OFF (❼). After a brief precharge operation, the gate is closed (❽), and R_R is refreshed by the combined operation of SA_R and the original sense amplifier.

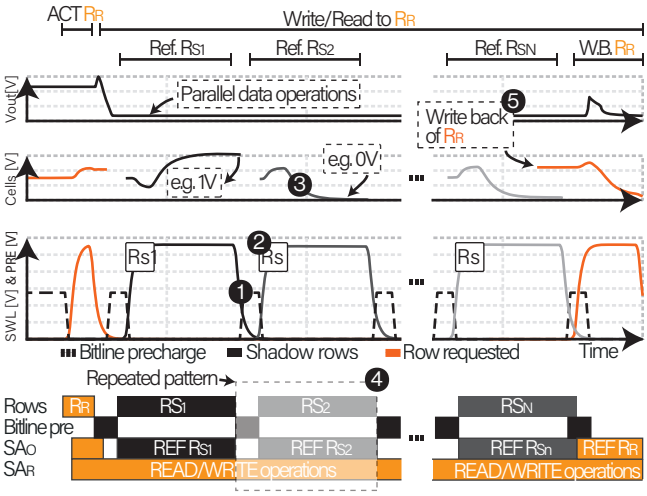

FIG. 4.10: **REGA Performing Multiple Refreshes.** The bitline is precharged (❶), after which the target shadow row is activated (❷), and SA_0 is used to perform its refresh (❸). The same can be repeated for more shadow rows (❹), and lastly, the process can be completed by restoring R_R (❺).

Stage 1: Logical ACT. In Stage 1 (Fig. 4.6), REGA latches the logical values from the requested row. The latching mechanism is first started by SA_0 and quickly latched by SA_R (Fig. 4.7, Stage 1.1). This combined operation allows SA_R to be smaller than SA_0, thereby keeping area overhead and power consumption low. To correctly multiplex the bitline, a transmission gate is required between the two sense amplifiers, which prevents SA_0's refresh mechanism from corrupting SA_R's logical value when the signal V_G is low. Therefore, before activating R_S, two events are necessary: turning OFF the original sense amplifier SA_0 and precharging the bitline (Fig. 4.7, Stage 1.2).

Stage 2: Parallel REF. In Stage 2 (Fig. 4.6), REGA performs a parallel refresh to R_S. First R_S is activated, then the standard charge sharing and recharging, as described in Section 4.5, happen via SA_0 (Fig. 4.7, Stage 2.1). Before Stage 3 can be started, SA_0 must be turned OFF, and the bitline must be precharged. This precharge is needed because the SA_R might lose the latched value due to the high-charge parasitics of the bitline.

Stage 3: Logical REF. In the last stage, REGA recharges R_R with the up-to-date logical value. This is necessary, as the cell had partially lost its value during the charge sharing operation of Stage 1. First, the transmission gate

is turned ON, connecting SA_0 to the bitlines (Fig. 4.7, Stage 3.1). Then, SA_R is activated to assist in the refresh of R_R (Fig. 4.7, Stage 3.2).

4.6.3 *Detailed Signal Timings*

We now analyze the detailed signal timings. Our design considers the worst-case scenario, where the R_R and R_S cells have opposite values and with a minimum charge. The timings are obtained using REM simulations as shown in Fig. 4.9. In this accurate description, we use SA_0 and SA_{0D} to refer to the activation of the original sense amplifier's supply, either with the common (1 V) or overdriven (1.4 V) voltage.

Stage 1: Logical ACT. In the initial state, the bitline is precharged (Fig. 4.9, ❶). The memory controller sends an ACT to the row R_R. This causes the precharge signal to be de-asserted and R_R to be connected to the bitline while R_R's SWL is set high (❶). With this last operation, charge sharing begins, inducing a voltage variation along the bitline. After the bitline has received the capacitor's charge, SA_{0D} is activated (SA_{0D} is set high) to help SA_R latch the value (❷) with the transmission gate active. Shortly after, SA_R is activated and starts latching the logical value of R_R. SA_{0D} is turned OFF, and the transmission gate is opened (V_G is set low, ❸).

At this point, the bitline is precharged for the required time (❹). SA_R will latch the logic value before a tRCD. This operation is very fast because SA_R does not have the load of the bitline, which is disconnected via the transmission gate. Before the end of tRAS, read and write operations go to SA_R.

Stage 2: Parallel REF. This stage follows the standard charge sharing and recharging but targets R_S (❹-❻). After this activation (used as a refresh) of R_S is over, R_S is disconnected from the bitline, and SA_0 is turned OFF (❼). Then, the bitline is precharged. As previously noted, this precharge is needed because in the next stage, SA_R will be connected to the bitline.

SA_R now holds a value that must be stored back in R_R. However, if the parasitics of bitline held an opposite value, connecting the SA_R might corrupt its value. This depends on SA_R's transistors sizes, on its power supply, and on the bitline parasitics. Considering the technology of our collaborating DRAM vendor, the precharge operation is fast, and the SA_R's transistors are big enough to reliably keep their value.

Supporting other DRAM technologies. We provide solutions for cases where the SA_R or the precharge circuit must be very weak. First, we simulated a weaker precharge circuit and determined that it is not required to bring the bitline exactly to Vdd/2. Given our values for the SA_R, we found that a margin of at least 60 mV is tolerated. Second, we tested a weaker SA_R with transistor widths halved. This situation can be overcome by controlling the gate circuity separately for the two bitlines (BL and BLB). In particular, in Stage 3 only the gate that connects BL can be connected. This improves the reliability of SA_R to hold its value once connected to charged bitlines. If the assisted refresh is needed in Stage 3, SA_0 can initially be kept OFF while only BLB is precharged. Then, BLB's gate is connected, and SA_0 finishes the refresh operation.

Stage 3: Logical REF. R_R is activated, and the gate is turned ON (❽). After a short time, SA_{0D} is turned ON to assist the row refresh. To compensate for the short refresh time, SA_{0D} is held overdriven longer than usual before switching to SA_0.

All the discussed timings need to consider the rising time and the delays due to parasitics. For the sake of simplicity, we did not include them in this description. They are, however, used in our simulations and included in our model. All voltage levels are summarized in Tbl. 4.6.

4.6.4 *Parallel Refresh of Multiple Rows*

As we discuss in Section 4.8, multiple parallel refreshes at every activation enable protecting devices with very small Rowhammer thresholds. REGA can perform more than one refresh in parallel to a row activation without changing its circuit. Parallel refreshes involve repeating some key operations, illustrated in Fig. 4.10 for shadow rows R_{S1} to R_{SV}. For each shadow row, REGA interleaves bitline precharging with Stage 2 (Parallel REF). First, the bitline is precharged (❶). Then, the target shadow row is activated (❷), and SA_0 is used to perform its refresh (❸). Then, these operations can be repeated for a different shadow row (❹), or the process can be completed by (re)storing the charge in the requested row (❺).

In the next section, we show that on recent DDR4 and DDR5 devices, the minimum tRAS of 32 ns suffices for refreshing a single shadow row ($V = 1$). Performing multiple refreshes requires more time, which a DRAM device can do by extending tRAS from its minimum of 32 ns. To refresh V shadow

Algorithm 4.1: **Experiment for measuring the slack in tRAS.**

```
Output : Number of corruptions C for all rows
S ← PickRandomRows(1000)
for t ← {8, 16, 24, 32} do // tRAS in ns
    for R ← S do
        Fill R with a random data pattern
        Send ACT-PRE to R with tRAS = t
        Read R
        C_R^t ← number of corruptions in R
        if C_R^t == 0 then
            // If it does not corrupt with a low tRAS,
            // neither it will with a higher tRAS
            Remove R from S
return C
```

rows on top of R_R, REGA requires a tRAS of $32 + (V - 1) \times (17.5)$ ns which we evaluate in Section 4.9.

4.7 IMPACT OF REGA ON tras

For a single shadow refresh ($V = 1$), REGA needs 16 ns (Fig. 4.9). We introduce two experiments showing that (i) tRAS has sufficient slack to implement REGA in today's devices, and (ii) reducing tRAS does not negatively affect data retention. We present our experimental platforms in §4.7.1, and the experimental methodologies and results in Sections 4.7.2 and 4.7.3. We discuss how tRAS can be configured for values higher than 32 ns allowing multiple parallel refresh operations in §4.7.4.

4.7.1 *Experimental Platforms*

We measure the slack in tRAS on PCs to obtain minimal tRAS values. To measure the impact of reducing tRAS on data retention, we rely on an FPGA platform which provides us with precise timing for DRAM commands.

PC configurations. We use Intel Core i7-8700K (DDR4) and Intel Core i7-12700K (DDR5) machines for PC-based experiments. Our mainboards

(Appendix 4.12.4) allow accurately setting DRAM timings, such as tRAS. We use a SO-DIMM-to-UDIMM extender to connect our SO-DIMMs to these machines, which limits their speed to 2666 MHz. We use MemTest86 Pro (version 9.4) for testing the DIMMs under reduced tRAS.

FPGA details. Our platform is based on a Zynq ZCU104 FPGA running Antmicro's Rowhammer tester [113]. It supports off-the-shelf DDR4 SO-DIMMs, and allows issuing DRAM commands directly to the memory device. Further, we have a DRAM heating infrastructure with which we can keep the DRAM device's temperature up to 100 °C.

Test devices. Our DRAM test pool consists of 21 DDR4 SO-DIMMs and 16 DDR5 UDIMMs, all off-the-shelf DRAM devices, covering all major DRAM manufacturers and varying in size and frequency. For more details about our test devices and the experimental platforms, we kindly refer to Appendix 4.12.4.

4.7.2 *Experiment 1: Slack in tRAS*

In this first experiment, we investigate the slack in the default tRAS of existing devices. We achieve this by sending ACT-PRE sequences with a reduced tRAS value and using corruptions as an indicator for failures caused by a too small tRAS value.

We describe our experiment in Algorithm 4.1. As temperature might affect the DRAM's operation, we perform this experiment under normal room temperature (25 °C) and the maximum temperature specified by the JEDEC standard (85 °C). Because precise timing is essential, we executed this experiment using our FPGA platform on all our SO-DIMMs. The results for the devices where we observed any corruptions are given in Tbl. 4.2. We observe that even with an extremely reduced tRAS of 8 ns, we could only observe corruptions on 3 of 21 tested DDR4 devices. S_{16} reports fewer corruptions with a higher temperature. This is because higher temperatures can lower the threshold of access transistors, allowing for faster data restoration in some cases (i.e., fewer corruptions). For a tRAS of 16 ns, the value we require for REGA with $V = 1$, we never observed corruptions.

We do not have precise control over DRAM commands on a PC. However, to verify that our results hold for UDIMMs, we reconfigured the tRAS of these devices in the PC's BIOS/UEFI to 16 ns. We then ran a full pass of all

DIMM	Room temp.		Max. (85 °C)	
/tRAS	8	16	8	16
S_6	7	0	1,747	0
S_{13}	1,413	0	2,840	0
S_{16}	2,937	0	1,315	0

TBL. 4.2: **Results of the free tRAS slack experiment.** We report the no. of corruptions on all SO-DIMMs and UDIMMs in our test pool. We omit devices without any observed corruptions.

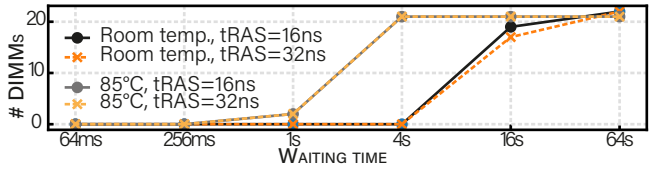

FIG. 4.11: **Retention time experiment.** Time required for corruptions to occur on the DDR4 SO-DIMMs under different tRAS settings and temperatures.

16 MemTest86 [114] tests on all our test devices (including the SO-DIMMs) to check for any corruptions that might be caused by the lower tRAS. We confirm having observed no corruption over all tests in any of our DDR4 and DDR5 devices.

4.7.3 *Experiment 2: Impact of a Reduced tRAS on Data Retention*

This experiment's goal is to show that reducing tRAS does not negatively affect the data retention time even though less time is given to the ACT command. In other words, reducing tRAS should not lead to data corruption due to retention errors. To analyze this, we construct an experiment where we refresh rows varying tRAS values and use wait periods of different lengths to see the impact of tRAS on data retention.

We precisely describe our experiment in Algorithm 4.2, which we repeat for the two different temperature levels for all DDR4 SO-DIMMs in our test pool. As the JEDEC standard requires a data retention time of 64 ms, we do not expect to see any retention failures for this waiting period. Fig. 4.11 reports the number of DIMMs in which we observed corruptions, for different values of tRAS and after a waiting period. The data clearly shows that for up to 256 ms, there are no data corruptions: the first corruptions

start to appear after a waiting time of 1 s. This is significantly more than the 64 ms that JEDEC specifies in the DDR4 standard. Furthermore, there is no clear difference between the retention profiles of devices with different tRAS values. Hence, we conclude that it is safe to reduce the tRAS value to 16 ns, or more precisely, use the available slack to perform REGA operations.

4.7.4 *Configuring tRAS*

DDR*x* devices use an SPD chip to inform the memory controller which timings to use [115–117]. The content of the SPD chip is standardized by JEDEC [118, 119] and intended to allow departing from the timings described in the DDR standard for future devices.

The SPD chip includes tRAS, therefore devices deploying REGA can set the necessary tRAS value. For DDR4 devices, the tRAS in the SPD can be set up to 512 ns, making REGA with high V (e.g., 8) already deployable. On DDR5 devices, the tRAS on the SPD can be set up to 65.5 ns, making V values higher than 2 not immediately compatible with the current SPD standard. As we discuss in Section 4.8, $V > 2$ enable protection for devices with R_{thresh} smaller than 517, estimated to occur in around 7 years from now. We hence recommend future revisions to the JEDEC SPD standard to allow the SPD chip to set higher tRAS values as necessary. In Section 4.9 we evaluate the performance overhead introduced by increasing tRAS beyond 32 ns.

4.8 REGA$_M$

We show how we can leverage refreshes generated by REGA to design new Rowhammer mitigations by demonstrating a blast-independent, fully in-DRAM, stand-alone mitigation called REGA$_M$. REGA$_M$ is simple, deterministic, and configurable based on the device's R_{thresh}.

4.8.1 *Design*

Aggressor rows only affect victims inside the same sub-array since sub-arrays are physically separated from each other by sense amplifiers. Given a sub-array, REGA$_M$ cyclically refreshes all its rows as it receives activations.

Algorithm 4.2: **Evaluation of the impact of tRAS on the data retention time.**

Output : Number of corruptions C for all rows

```
1  S ← PickConsecutiveRows(1000)
2  for R ← S do
3  |  Fill R with a random data pattern;
4  for t ← {8, 16} do // tRAS in ns
5  |  for m ← {64k, 16k, 4k, 1k, 256, 64} do // waiting time in ms
6  |  |  C_R^t ← 0;
7  |  |  for r ← 0 to 5 by 1 do // reps. account for VRT
8  |  |  |  for R ← S do
9  |  |  |  |  Send PRE-ACT with tRAS = t ns // ≙ refresh
10 |  |  |  Wait by sending NOPs for m ms;
11 |  |  |  for R ← S do
12 |  |  |  |  Read R;
13 |  |  |  |  C_R^t ← C_R^t + number of corruptions in R;
14 |  |  |  if C_R^t == 0 then
   |  |  |  |  // m causes no corruptions
   |  |  |  |  //  ⟹  any m' < m will not cause corruptions
15 |  |  |  |  break;
16 |  |  |  C_R^t ← C_R^t / 5 // Average over the 5 repetitions
17 return C
```

In particular, REGA$_M$ refreshes V different rows in a sub-array every time it receives T activations. T and V are parameters that can be dynamically configured through a freely available register inside SPD to account for devices with different Rowhammer thresholds and the discovery of new Rowhammer effects. We show in Section 4.9 that REGA$_M$ outperforms the state-of-the-art in-DRAM mitigation when it comes to known Rowhammer effects. Furthermore, since REGA$_M$ refreshes all rows in a sub-array that is receiving activations, it also provides strong protection against new (yet unknown) Rowhammer effects. As an example, after this paper was submitted, the latest JEDEC standard extended the maximum supported blast diameter to 6 [12]. REGA$_M$ is immediately capable of protecting devices in these scenarios. To easily integrate normal refresh operations, refreshes targeting a particular sub-array will use REGA$_M$'s row index as a target, and then increment it.

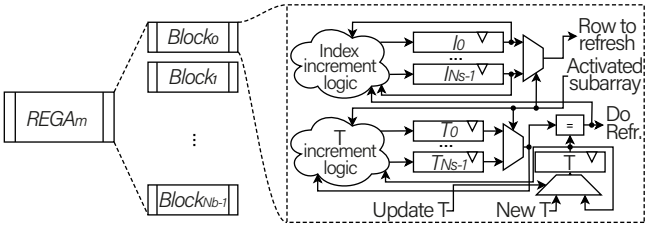

FIG. 4.12: **Overview of the REGA$_M$ implementation.**

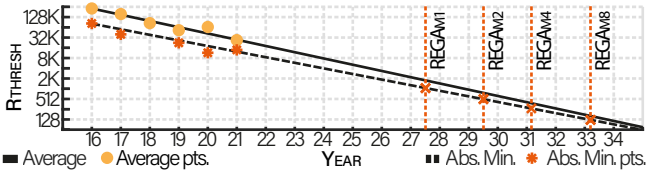

FIG. 4.13: **Trend of R_{thresh} from 2016 to 2035.** In orange is reported the minimum R_{thresh} for REGA$_{M1}$, REGA$_{M2}$, REGA$_{M4}$ and REGA$_{M8}$.

4.8.2 *ASIC implementation*

We propose to implement REGA$_M$ in the CMOS area of the DRAM chip, similarly to our previous work [23]. Fig. 4.12 provides an overview of REGA$_M$'s ASIC implementation. REGA$_M$ is made of N_b identical blocks, where N_b is the number of banks. Each block is essentially composed of N_s indices I_i, where N_s is the number of sub-arrays inside a bank and each I_i is a 9-bit register pointing to the next row to be refreshed in the corresponding sub-array. Additionally, each block contains N_s registers T_i, which are duty-cycling the refresh commands, i.e., to ensure that REGA$_M$ sends a refresh every T activations received by a sub-array. Whenever an activation affects the i-th sub-array, if $T_i = T - 1$, then V refreshes are sent to I_i, T_i is reset and I_i is incremented by V, wrapping around the maximum row index.

4.8.3 *Security*

REGA$_M$ can protect a device for a minimum R_{thresh} depending on its configuration. To obtain R_{thresh}, we consider the worst-case scenarios happening before and after a victim row is refreshed. For a typical sub-array size of

512 rows, a victim row is refreshed after a maximum of $\frac{512}{V} \times T$ activations, during which its aggressor rows can repeatedly be activated. Of these activations, $\frac{512}{V}$ will perform extra refreshes. Therefore, because the SWL is raised twice, these will hammer the victim twice. The last extra-refresh operation is an exception, as the victim will be refreshed after SWL is raised once. Instead, $\frac{512}{V} \times (T-1)$ activations will not perform extra refreshes, hammering the victim only once. Lastly, refreshed rows will also hammer the victim if inside its blast diameter B. Depending on the victim's position inside the refreshed group, the refreshed rows will either be a hammering baseline for the next iteration or will hammer before the victim's refresh. In either case, this will result in a total of B hammerings (i.e., row activations). A victim will always have a hammer baseline of 1 due to the SWL raise after its refresh. Therefore, a victim row can be hammered at most $\frac{512}{V} \times (T+1) + B$ times. We further validated REGA_{M} using state-of-the-art Rowhammer fuzzers [5, 7] without observing any bit flips.

In certain DRAM chips, rows that are kept active for longer periods could increase the Rowhammer vulnerability [32]. This behavior still requires extensive characterization to conclude whether it can provide an additional benefit to an attacker. If necessary, REGA_{M} can completely eliminate this effect with a minor variation. Because REGA adds buffering sense amplifiers, R_{R} does not need to stay active during writes and reads. REGA_{M} write back operation can then be performed during a PRE, as it only requires 12 ns.

Fig. 4.13 reports REGA_{M} thresholds for $V = 1, 2, 4, 8$, respectively as $\text{REGA}_{\text{M1-8}}$, offering Rowhammer protection for devices up to 10 years from now.

4.9 EVALUATION

We now evaluate the key aspects of REGA and the mitigations $\text{REGA}_{\text{M1-8}}$. Results for REGA and the ASIC implementation are shared by all the mitigations. Results that are V-specific are indicated as REGA_{MX} ($V = x$).

First, we analyze the die's area overhead due to the extra REGA circuitry and the ASIC implementation, and we report the static power consumption (§4.9.1). Second, we evaluate the circuit's reliability based on 160 K analog Monte Carlo simulations (§4.9.2). Third, we estimate the power, energy and performance overhead of $\text{REGA}_{\text{M1-8}}$ by running cycle-accurate simulations (§4.9.3).

We compare the overheads of $\text{REGA}_{\text{M1-8}}$ with ProTRR [23], which is the state-of-the-art in-DRAM Rowhammer mitigation. We synthesized REGA_{M}'s ASIC and ProTRR using a 12 nm technology with Synopsys Design Compiler 2021. In our evaluations, we consider $B = 2$ for the classical Rowhammer effect [3], $B = 4$ for the recently introduced half-double effect [99], $B = 6$ that has been added in the latest JEDEC standard [12] and $B = 8$ for future DRAM technologies where an aggressor row can cause bit flips in four victim rows on each side. We regard R_{thresh} of interest to be lower than 4 K.

4.9.1 *Area Overhead*

The area in the DRAM chip is generally limited, and the vendors aim to maximize the area for the mats. Given that, we must ensure that REGA's implementation is practical, i.e., it consumes a reasonably small amount of die area.

Methodology. Depending on the DRAM design and technology, the ratio between the sensing circuit and the chip's die can vary between 8% and 15% [87], averaging 11.5%. The sensing circuit includes column selectors, precharge circuits, and BLSAs. According to the sense amplifier layout reported by our collaborating DRAM vendor, the BLSAs occupy 60% of the sensing area. REGA adds small buffering sense amplifiers and transmission gates, and for simplicity, we consider their length to be equivalent to the original BLSAs. More precisely, REGA requires eight additional transistors per each BLSA: four that are 1/6-th of the BLSA's transistors width and another four that are 1/8-th of the BLSA's transistors width.

To conservatively evaluate the area overhead of REGA, we assume no available free space to place our extra transistors. Instead, because each BLSA is formed by four transistors, we propose placing them in two groups following the BLSAs. Consequently, REGA's area overhead can be calculated as follows: $60\% \times (\frac{1}{6} + \frac{1}{8}) \times 11.5\%$ resulting in, on average, only 2% area overhead. Our ASIC implementation of REGA_{M} incurs as little as 0.06% area overhead.

Comparison with ProTRR. In Fig. 4.14, we show the total area overhead of REGA_{M} compared to ProTRR for DDR4 and DDR5. As REGA_{M} does not use any counters, its area overhead is independent of the Rowhammer threshold.

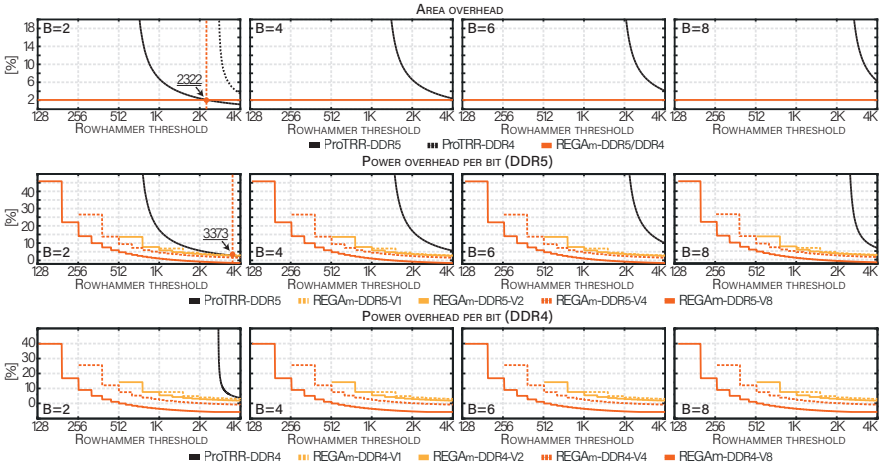

FIG. 4.14: **Area and power overhead of $REGA_{M1\text{-}8}$. (i.) Area overhead.** Overhead for 32 banks and blast diameters of 2, 4, 6, and 8. **(ii.) Power overhead per bit (DDR5/4).** Power overhead related to a bit for a fixed chip area. In orange, the point of crossing between $REGA_M$ and ProTRR is highlighted. Because $REGA_M$ is blast independent, its evaluation for different *Bs* is identical. The only difference is a R_{thresh} shift of 2 between each panel (see Section 4.8.3).

In DDR4, ProTRR struggles for Rowhammer thresholds lower than 4 K ($B = 2$) or has no protection ($B > 2$). For DDR5 devices, $REGA_M$'s area is lower for Rowhammer thresholds lower than 1156 ($B = 2$) and any threshold lower than 4 K ($B > 2$). As thresholds approach 1 K, ProTRR can no longer protect these devices ($B = 4, 6, 8$).

Static power overhead. The extra sense amplifier circuit uses CMOS technology and incurs a negligible static power overhead. For 32 banks, $REGA_M$ has a static power consumption of 0.015%, for a baseline of 3 W [23]. This is significantly less than ProTRR, which requires overheads between 4.65% and 2.35% for 16 banks.

4.9.2 *Circuit Reliability*

Methodology. We used LTspice [120] to simulate REGA using REM. We ran 40 K Monte Carlo simulations while introducing random variations of ±5% in the transistor's dimension and the line parasitics. As previously

CPU		Memory Controller		DRAM	DDR4	DDR5
Sched. Type	OoO	#Channels	2	Freq (GHz)	2.9	4.8
# Cores	8	Page Policy	Open	Ranks	1	1
Freq. (GHz)	3	Scheduler	FR-FCFS	Bankgroups	4	8
L1 (KiB)	2x32	Queue Type	Per Bank	Banks/group	4	2
L2 (KiB)	256	Capacity (GiB)	16	Banks/rank	16	16
				Rows/bank	64 K	64 K

TBL. 4.3: **Gem5 system configuration.** For results including L3 (8 MiB shared), see Appendix 4.12.9

done for DRAM [82, 101, 121–124], we modeled the transistors using the 22 nm predictive transistor model (PTM) [125]. Based on indications from our DRAM collaborator, we set the DRAM cells' charge to the minimum that is still considered correct. We repeated the simulations for $V = 1, 2, 4, 8$.

Results. The circuit reported 100% reliability on all 160 K Monte Carlo simulations for $\text{REGA}_{\text{M1-8}}$. We consider the circuit reliable if all capacitors are recharged to the correct value. To evaluate the worst-case scenario, we tested the combinations where the sequence of cells to recharge have opposite values.

4.9.3 *Performance, energy and power overhead*

We benchmark SPEC®2017 [91] with a cycle-accurate simulator, evaluating the energy and power overhead of the extra row refreshes, and the performance overhead due to the corresponding `tRAS` extension. Before presenting our results, we briefly describe our methodology and simulation setup. Because power presents the main design factor for devices, we refer the reader to §4.12.5 for the energy overhead results.

Gem5 simulation. We configured gem5 [92] as described in Tbl. 4.3 to do a full-system simulation of Ubuntu (Linux kernel 5.4) based on an 8-core out-of-order CPU. Like previous work [23], we modeled the DRAM subsystem using DRAMsim3 [93] as it allows for a higher accuracy than the DRAM model included in gem5. Per SPEC®2017's recommendation, we ran multiple copies in parallel, equal to the number of cores (i.e., eight in our configuration). This maximizes the workload and, as such, increases the load on the memory subsystem. For each benchmark, we obtained 20

equally-spaced checkpoints (SMARTS methodology [94, 126]) and we ran each checkpoint for 10 M instructions.

Methodology for energy and power overhead. We evaluated the energy overhead on top of a regular ACT operation for $V = 1, 2, 4, 8$ extra refreshes. We considered the worst case where a cell needs to be *fully* recharged. Simulating REGA using LTspice [120], averaging over 50 Monte Carlo simulations, showed a per-ACT energy overhead of 38%, 95%, 223% and 479%, respectively for $V = 1, 2, 4, 8$. For each benchmark, we extracted the average total energy consumed and the energy consumed due to activations, repeated for DDR4 and DDR5. Depending on T and V, we obtained the overhead with respect to the baseline ($V = 0$). We used this energy to derive power consumption and power overhead, by using the individual CPU time simulated for each benchmark and each checkpoint. We then calculated the energy and power overhead per bit by considering a fixed die size available, as we do not expect manufacturers to increase the die area freely.

Power overhead. REGA_{M} performs refreshes every T activations, which depends on the Rowhammer threshold and on V. A high value of T substantially decreases power consumption. For example, if a device has a Rowhammer threshold of 4 K, REGA_{M1} can be activated every 6 activations, incurring 6 times less energy overhead and consequently less power overhead. Fig. 4.14 shows the average power overhead per bit depending on the Rowhammer threshold, compared with ProTRR. In almost all cases of interest, REGA_{M} has a lower power consumption compared to ProTRR. As an example, to protect DDR5 devices with R_{thresh} of 1027 and $B = 2$, REGA_{M} requires only 7% extra power per bit, while ProTRR needs 18%. In the case of extended tRAS and high values of T, the power is reduced with respect to the baseline (negative overhead). This is due to a reduced amount of activations sent for a given time, and is reflected in a performance overhead (see next). We further provide REGA_{M}'s absolute power overhead (i.e., not relative to the area) in Appendix 4.12.7.

Performance overhead. REGA does not require timing changes when refreshing a single row ($V = 1$). Therefore, it does not introduce any performance overhead. If REGA refreshes $V > 1$ rows, tRAS must be extended.

We repeat the measurements for $V = 2, 4, 8$ and for the baseline ($V = 0$). In Fig. 4.15 we report the average performance overhead relative to the baseline. A detailed overview can be found in Appendix 4.12. ProTRR has a negligible performance overhead, however, for low thresholds it becomes infeasible due to its area overhead or it cannot provide a sufficient

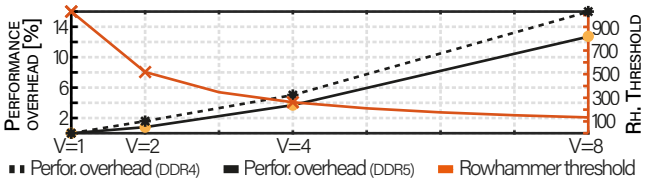

FIG. 4.15: **Performance overhead.** Average performance overhead for V=1, 2, 4, 8 for SPEC®2017 on DDR4 and DDR5.

protection. REGA$_M$ offers a protection with 0% performance overhead for the thresholds that are covered by ProTRR, and it extends this protection to thresholds significantly lower by adding a performance overhead due to extending tRAS (e.g., 3.7% for $R_{thresh} = 261$). In Appendix 4.12.9, we report detailed results when L3 is included. Generally, the addition of L3 reduces the performance overhead (e.g., 1.9% compared to 3.7% for $R_{thresh} = 261$).

4.10 RELATED WORK

In the following, we summarize and compare existing Rowhammer mitigations to REGA$_M$. In Tbl. 4.4, we provide an overview of the existing mitigations, comparing them for their security properties (**Security**), their deployment location, their compatibility to the DDRx standard (**Comp.**), their approach (**Concepts**), and if they were evaluated on the latest DDR standard (**Eval.**). We differentiate between agreement (●), and disagreement (○); for positive (●) and negative properties (●). Not applicable properties are denoted by "–".

Security. First, 15 of our 19 considered mitigations are deterministic (**Det.**), which is favorable due to stronger security guarantees. Other mitigations use probabilistic decisions such as MRLoc, ProHIT, and PARA. Second, REGA$_M$ is the first proposed mitigation that can protect against new (unknown) Rowhammer effects without modification (**Eff.**) such as the recently discovered half-double effect or even higher blast diameters [12, 34]. Third, some mitigations suffer from known vulnerabilities (**Vuln.**) as shown by existing work [23, 41] and in §4.12.8. We only considered insecurities in the original design, not arising from new effects. As previously exposed [127], in-CPU mitigations that rely on refreshes are either vulnerable or incompatible. For a refresh to be secure, the internal DRAM topology must be known. This way, the mitigation can keep track of the rows that are hammered

	Yr	Security			Comp.		Eval.	Concepts			
		Eff.	Vuln.	Det.	DDR4/5		DDR5	Ct.	Pr.	Is.	B.I.
DRAM											
REGA$_{m1-m2}$	'22	●	○	●	●	●	●	○	○	○	●
REGA$_{m4-m8}$	'22	●	○	●	●	○	●	○	○	○	●
Mithril [39]	'22	○	●	●	○	○	●	●	○	○	○
ProTRR [23]	'22	○	○	●	●	●	●	●	○	○	○
Panopticon [35]	'21	○	○	●	●	●	○	●	○	○	○
ProHIT [82]	'17	○	●	○	●	●	○	○	●	○	○
Memory controller											
Hydra [40]	'22	○	○	●	○	○	○	●	○	○	○
Row-Swap [44]	'22	○	○	○	●	●	○	●	●	●	○
BlockH. [42]	'21	○	○	●	●	●	○	●	○	○	○
CAT-TWO [59]	'20	○	●	●	○	○	○	●	○	○	○
Graphene [41]	'20	○	●	●	○	○	○	●	○	○	○
TWiCe [61]	'19	○	●	●	○	○	○	●	○	○	○
MRLoc [60]	'19	○	●	○	○	○	○	○	●	○	○
CBT [80]	'16	○	●	●	○	○	○	●	○	○	○
PARA [3]	'14	○	○	○	●	●	○	○	●	○	○
Software											
ALIS [57]	'18	○	○	●	–	–	○	○	○	●	○
GuardION [55]	'18	○	●	●	–	–	○	○	○	●	○
ZebRAM [64]	'18	○	●	●	–	–	○	○	○	●	○
CATT [63]	'17	○	●	●	–	–	○	○	○	●	○
ANVIL [16]	'16	○	●	●	–	–	○	●	○	○	○

TBL. 4.4: **Overview of Rowhammer mitigations.**

by extra refreshes. Unfortunately this has never been addressed by the standard, leading to new attacks to surface [34]. We considered a mitigation to be vulnerable if they do not mention this effect, or do not require the topology to be known.

Location. The majority of existing mitigations (9 out of 19) need changes in the CPU's memory controller. Software-based mitigations involving the operating system have also been proposed. Notably, the focus has moved since 2019 from software- towards memory controller-based mitigations and, more recently (2021+), to fully in-DRAM mitigations.

Compatibility. Most mitigations target the **DDR4** standard, though many require changes to the protocol, for example, by requiring a new (refresh) command or the internal row layout. We consider mitigations to be non-

compliant, if in the original publication they required standard modifications for the evaluated protocol. We report Row-Swap to be compatible with the standard, however, as confirmed with the authors, the time delay for the operations should be roughly twice what is used in the paper. Only the more recent mitigations from 2022, namely REGA, ProTRR, and Mithril, are evaluated for **DDR5**. We ignore software-based mitigations, as they are by design independent of the DDRx standard. We considered mitigations that rely on the knowledge of internal row mapping to be incompatible but secure. Currently, $\text{REGA}_{\text{M4-M8}}$ are non-compliant with the DDR5's SPD specification, which limits the compatible volume to $V = 1, 2$.

Concepts. Most mitigations employ counters (**Cnt.**) to keep track of aggressors or victims. Those who do not employ counters use other data structures such as queues (e.g., ProHIT and MRLoc) or are entirely stateless (e.g., PARA). As it is generally difficult to precisely track row activations from software, 4 out of 5 proposed mitigations use isolation (**Isol.**) to protect against Rowhammer. ANVIL is an exception to this trend: it uses performance counters to track row activations. We consider PARA, ProHIT, MRLoc, and also Row-Swap's random swapping of rows to be based on probabilities (**Prob.**).

REGA_{M} is the only blast-diameter independent (**B.I.**) mitigation. The design of all other mitigations heavily relies on the considered diameter. This gives REGA_{M} the flexibility to scale with the increasing blast diameter.

4.11 CONCLUSION

We presented REGA, a new in-DRAM mechanism that can scale the number of refreshes with activations. REGA uses buffering sense amplifiers to time-multiplex DRAM bitlines so that refreshes can occur in parallel to standard DRAM operations. We demonstrated the correctness of REGA's circuit using a new accurate DRAM model that we developed in collaboration with a DRAM vendor. REGA enables simple yet powerful mitigations against current and future Rowhammer attacks. We built a deterministic and scalable mitigation on top of REGA, called REGA_{M}, and evaluated its area, power and performance impact. REGA is the first in-DRAM mitigation that scales to small Rowhammer thresholds with a constant small area overhead (2.1%) and power and performance overhead dependent on the Rowhammer threshold. As an example, REGA scales to $R_{thresh} = 517$ with 11.5% power and 0.8% performance overhead.

4.12 APPENDIX

4.12.1 *Rowhammer trend*

The fitting of the curves is based on the minimum Rowhammer thresholds reported in previous works [6, 87]. We now briefly report the methodology and fitting results.

Average curve. For each DRAM vendor, we calculated the average R_{thresh} in each year. Then, we averaged across vendors for the same year. The resulting points are reported in Fig. 4.2, which also includes a fitted curve using an exponential function (i.e., $a \times e^{b \times x}$) obtained using MATLAB® 2020 automatic fitting tool (R-square=0.96).

Absolute minimum curve. For each year, we considered the absolute minimum across all vendors. In the dataset, the year 2018 included only a single point, which we considered to be an overly optimistic outlier. For this reason, we removed it from the computation, as it would have skewed the minimum curve to be overly optimistic. Like above, the points used for the fitting are reported in Fig. 4.2, and fitted with an exponential function (i.e., $a \times e^{b \times x}$) obtained using MATLAB® 2020 automatic fitting tool (R-square=0.98).

The figure reports the minimum thresholds supported by the mitigations. However, the Rowhammer threshold is defined differently depending on the publication. In this work and others [23, 39, 41], R_{thresh} is the minimum number of activations to have bit flips. A R_{thresh} of 1024 with $B = 4$ could be reached by 4 different aggressors, each activated $1024/4 = 256$ times. Other mitigations [40, 44] refer to R_{thresh} relatively to aggressors. In those cases, the threshold is the number of times *every* aggressor in the blast diameter needs to be activated to induce bit flips. For example, Hydra [40] targets a threshold of 500 with $B = 4$. Because each row can be activated 500 times, and each victim has 4 different aggressors, this corresponds to a R_{thresh} equal to 2000.

4.12.2 *Abbreviations*

In Tbl. 4.5, we summarize the abbreviations and symbols we used throughout this work.

Abbrv./Symb.	Description	Ref. (§)
BLSA	Bitline Sense Amplifier	4.5.1
DIMM	Dual-Inline Memory Module	4.2.1
ECC	Error-Correcting Codes	4.2.2
GIO	Global I/O	4.5.1
LIO	Local I/O	4.5.1
MC	Memory Controller	4.10
MC	Monte Carlo	4.9
REM	REGA (DRAM) Model	4.5.3
REGA	Refresh-Generating Activations	4.4
RFM	Refresh Management (DDR5 Extension)	4.4
SPD	Serial Presence Detect	4.4
SWD	Sub-Word Line Driver	4.5.1
SWL	Sub-Word Line	4.5.1
TRR	Target Row Refresh	4.2.2
B	Blast Diameter	4.2.3
T	Period of REGA_{M} parallel refreshes	4.8
V	No. of shadows rows refreshed in parallel	4.6.2
`ACT`	A DRAM activation command	4.2.1
`PRE`	A DRAM precharge command	4.2.1
`tRAS`	Min. Row Address Strobe: `ACT`-to-`PRE` delay	4.2.1
R_{thresh}	`ACT`s req. to trigger bit flips	4.2.3

TBL. 4.5: **Abbreviations & Symbols.** A summary of abbreviations and symbols we used in our work with a brief description and reference to the section where it has been introduced.

4.12.3 *Voltages used in REM*

The voltage levels of our REM are listed in Tbl. 4.6.

Parameter	Voltage (V)	Description
`Vpp`	2.5	SWD power supply and overdrive control
`Vss`	0.0	Ground
`Vperi`	1.1	Peripherical circuitry voltage
`Vdd`	1.0	DRAM core voltage and cell's high value
`Vpre`	1.5	Control voltage of the precharge circuit
`Vod`	1.4	Overdrive voltage

TBL. 4.6: **Voltage levels used by REM.**

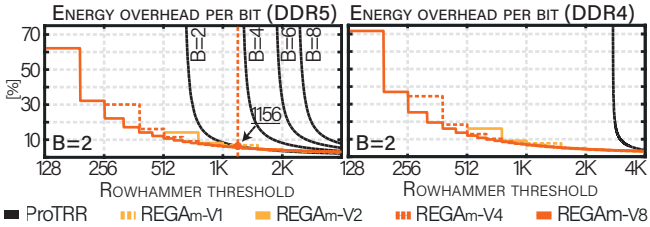

FIG. 4.16: **Energy overhead per bit of REGA$_{m1-8}$.** Energy overhead related to a bit for a fixed chip area. In orange, the point of crossing between REGA$_M$ and ProTRR is highlighted. Because REGA$_M$ is blast independent, its evaluation for different Bs is identical and in this figure grouped in a single plot. The only difference is a shift in R_{thresh} as described before (Section 4.8.3).

4.12.4 *Experiment Platform & Devices*

In the following, we provide more details on our PC-based test platform and the test devices of our DRAM test pool.

PC. We use Intel Core i7-8700K ("Coffee Lake") machines for DDR4 experiments, equipped with ASUS ROG STRIX Z930-E mainboards. We use Intel Core i7-12700K ("Alder Lake") machines for DDR5 experiments, equipped with Gigabyte Z690 AORUS PRO mainboards.

DDR4/5 Test Devices. In Tbl. 4.7, we list all the DDR4 and DDR5 DIMMs in our DRAM test pool.

4.12.5 *Energy overhead*

Results energy overhead. Fig. 4.16 shows the average energy overhead per bit depending on the Rowhammer threshold, compared with ProTRR. Given the recent half-double effect, REGA$_M$ is always convenient for DDR4 and DDR5 for the threshold in scope. For $B = 2$, REGA$_M$ is comparable to ProTRR (DDR5) and convenient in current DDR4 technologies.

DIMM	DRAM Manuf.	Mf. Date (yy-ww)	Freq. (MHz)	Size (GiB)	Geom. (#R, #B)	tRAS (ns)
S_0	SK Hynix	22-31 †	2133	8	1, 16	33.000
S_1	Micron	20-41	2400	16	1, 16	29.125
S_2	Micron	20-48	3200	8	1, 16	26.250
S_3	Samsung	20-47	2666	8	1, 16	32.000
S_4	Samsung	20-52	2666	4	1, 8	32.000
S_5	Micron	22-31 †	3200	16	1, 16	32.000
S_6	Samsung	20-44	2133	4	1, 16	33.000
S_7	Micron	20-45	2400	8	1, 16	32.000
S_8	Nanya	20-43	2400	8	1, 16	32.000
S_9	SK Hynix	22-31 †	2400	16	2, 16	32.000
S_{10}	Samsung	19-34	2666	8	1, 16	32.000
S_{11}	*n/a*	22-31 †	2666	8	1, 16	32.000
S_{13}	Samsung	22-31 †	2666	16	2, 16	29.250
S_{14}	Micron	22-22	2666	8	2, 16	32.000
S_{15}	Micron	22-21	2666	16	2, 16	32.000
S_{16}	Samsung	22-31 †	3200	8	1, 16	32.000
S_{17}	Micron	22-31 †	3200	32	2, 16	32.000
S_{18}	SK Hynix	21-28	2666	16	2, 16	32.000
S_{19}	SK Hynix	16-25	2133	16	2, 16	33.000
S_{20}	Zentel	22-31 †	2400	4	1, 16	32.000
S_{21}	*n/a*	22-15	2666	16	2, 16	32.000
U_0	Micron	22-04	4800	16	1, 32	32.000
U_1	Micron	21-41	4800	16	1, 32	32.000
U_2	SK Hynix	22-05	4800	8	1, 16	32.000
U_3	Micron	22-07	4800	16	1, 32	32.000
U_4	Samsung	21-52	4800	8	1, 16	32.000
$U_{5,6}$	Micron	21-42	4800	16	1, 32	32.000
U_7	Micron	21-49	4800	16	1, 32	32.000
U_8	Samsung	22-01	4800	16	1, 32	32.000
U_9	Samsung	21-42	4800	16	1, 32	32.000
$U_{10,11}$	Micron	22-02 †	4800	16	1, 32	32.000
$U_{12,13}$	Samsung	21-44 †	5600	16	1, 32	32.000
$U_{14,15}$	SpecTek	21-48	4800	16	1, 32	32.000

TBL. 4.7: **Specifications of the DRAM devices in our test pool.** Upper half (S_x): DDR4 SO-DIMMs, lower half (U_x): DDR5 UDIMMs. We report for each device its DRAM manufacturer (**DRAM Manuf.**) or "*n/a*" if there is no information reported by the DIMM's SPD chip; its manufacturing date (**Mf. Date**), or the date of purchase (†) in case it is not reported by the SPD chip; the frequency (**Freq.**); the device's size (**Size**); the geometry (**Geom.**) as number of ranks/banks; and its default **tRAS** value. DDR5: Same devices are same-kit modules.

4.12.6 *Performance overhead*

Fig. 4.18 and Fig. 4.17 show the performance overhead for $V = 2, 4, 8$, for the individual benchmarks of SPEC®2017 on DDR4 and DDR5. As discussed in the paper, $V = 1$ does not incur any performance overhead.

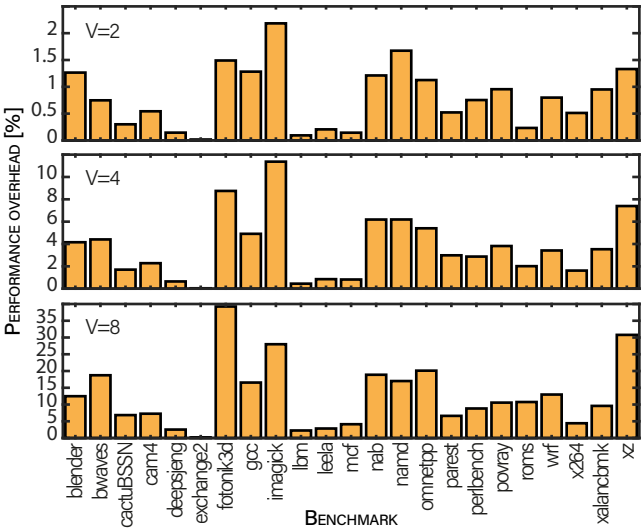

FIG. 4.17: **Individual performance overhead (DDR5).** Performance overhead for V=2,4,8 for the different benchmarks of SPEC®2017. Geometric mean = 0.8%, 3.7%, 12.7% respectively for V of 2, 4 and 8.

4.12.7 *Power overhead*

Fig. 4.20 and Fig. 4.19 show the average power overhead without considering the area overhead. The value is relative to the baseline power consumption, which depends on the device. Average power consumptions around 3 W are common [23].

4.12.8 *Mithril*

In this appendix, we point out vulnerabilities we discovered in the state-of-the-art in-DRAM mitigation Mithril [39]. Our analysis is focused on its security and standard compliance.

Missing standard compliance. Mithril is not compliant with the current DDR5 standard. It proposes using the new `RFM` command but fails to adhere to the correct parameters in the respective JEDEC specification [12]. In particular, the authors assume and evaluate an `RFM` command targeting a specific bank. However, the `RFM` can only either target all banks (`RFMab`)

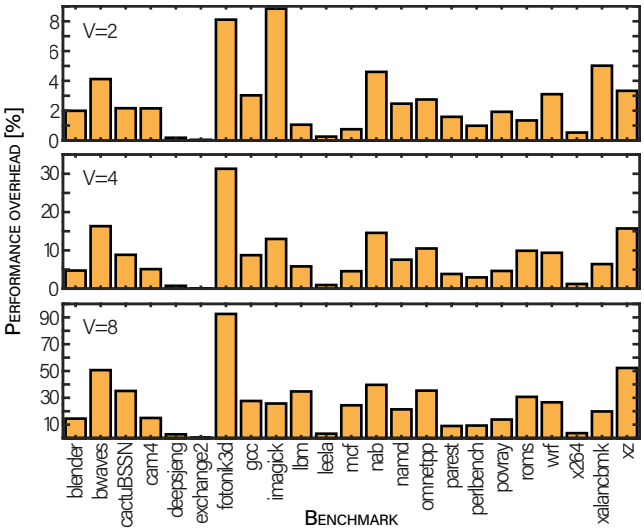

Fig. 4.18: **Individual performance overhead (DDR4).** Performance overhead for V=2,4,8 for the different benchmarks of SPEC®2017. Geometric mean = 1.6%, 5%, 16% respectively for V of 2, 4 and 8.

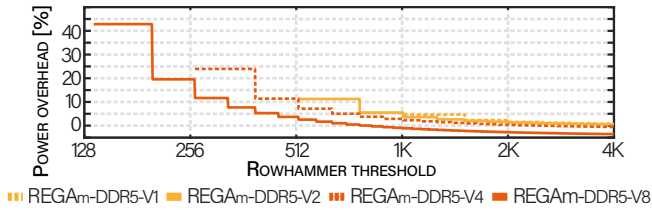

Fig. 4.19: **Power overhead (DDR5).** Average power overhead for REGA_{M}.

or banks that have the same ID in all bank groups (`RFMsb`). Moreover, the standard specifies `RFM` periods between 32 and 80 with steps of 8. Mithril, however, assumes an arbitrary `RFM` period. The standard also requires the bank activation counter to be decreased at every `REF`, which is another aspect not considered by Mithril.

Vulnerabilities. Mithril uses wrapping counters that expose the design to security issues. The counters can be manipulated to reach *MAX-1* for a row a_0, where *MAX* is the maximum number the counter can hold before wrapping around. Now, hammering a different row a_1 would lead

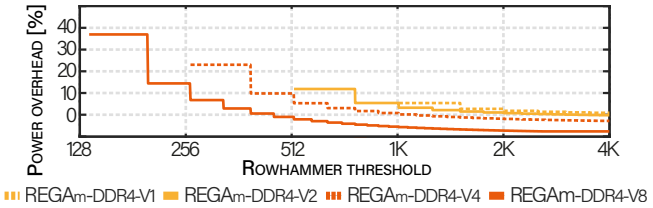

FIG. 4.20: **Power overhead (DDR4).** Average power overhead for REGA_{M}.

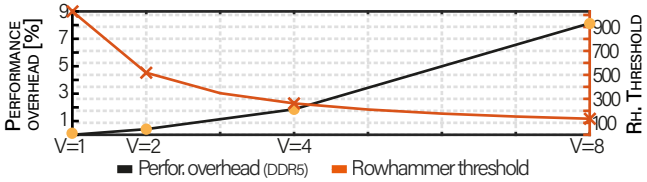

FIG. 4.21: **Performance overhead (DDR5) with L3.** Average performance overhead for $\text{REGA}_{\text{M1-8}}$ for SPEC®2017 on DDR5.

to it being replaced and subsequently refreshed. At this point, any new hammered row will replace the new minimum (0), and our victim row a_0 will never be picked for a refresh until all the others have been refreshed before.[1] Lastly, the effect of the refresh itself is not considered in the paper. It has been shown that refreshes can be used as a vector for exploitation on real-world devices [34], making the mitigation act as a confused deputy.

4.12.9 *Errata Corrige*

Overheads. We discovered a bug that affected the gem5 simulations, where the L3 cache was configured but not enabled. L3 caches are present on most desktop and server systems, while mobile devices might not have an L3 cache. The absence of L3 increased the memory load and made the original results a worst-case scenario for REGA. We have rerun all the gem5 simulations with L3 enabled, and the results are presented in Figs. 4.21–4.27.

Mithril's wrapping counters. The authors of Mithril [39] contacted us to clarify the implementation of their wrapping counters, which we now believe to be secure. However, Mithril's authors acknowledged that given

1 See §4.12.9 for an update on this vulnerability.

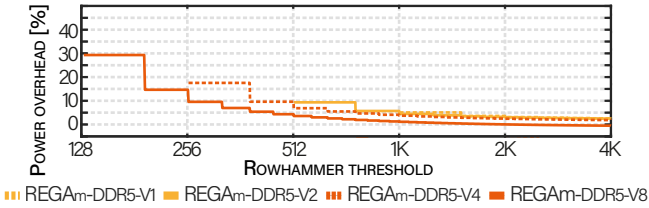

FIG. 4.22: **Power overhead per bit (DDR5) with L3.** Average power overhead per bit (for a fixed chip area) for $\text{REGA}_{\text{M1-8}}$.

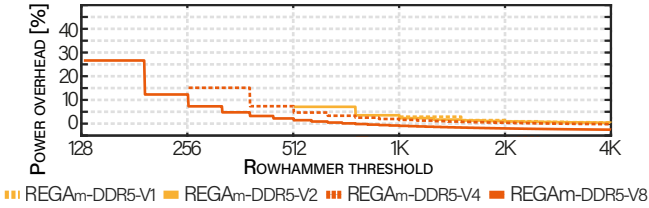

FIG. 4.23: **Power overhead (DDR5) with L3.** Average power overhead for $\text{REGA}_{\text{M1-8}}$.

the description in Mithril's paper, our attack is correct and would work. We want to thank the authors of Mithril for their clarification.

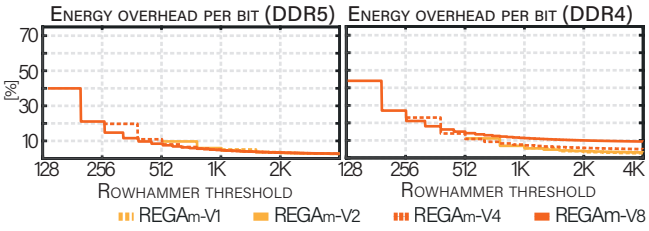

FIG. 4.24: **Energy overhead per bit (DDR5 and DDR4) of REGA$_{M1-8}$ with L3.**

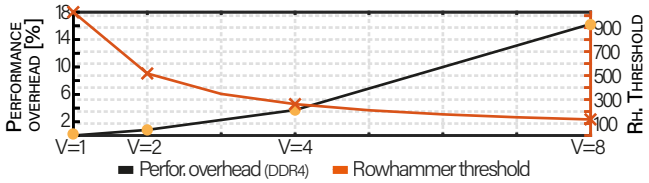

FIG. 4.25: **Performance overhead (DDR4) with L3.** Average performance overhead for REGA$_{M1-8}$ for SPEC®2017 on DDR4.

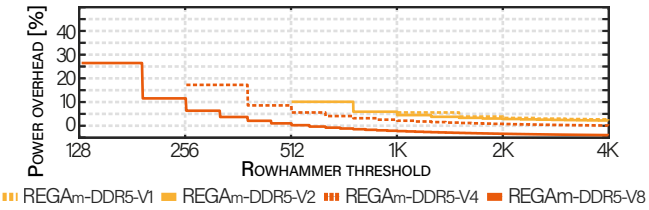

FIG. 4.26: **Power overhead per bit (DDR4) with L3.** Average power overhead per bit (for a fixed chip area) for REGA$_{M1-8}$.

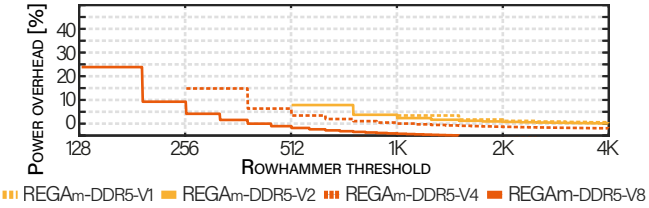

FIG. 4.27: **Power overhead (DDR4) with L3.** Average power overhead for REGA$_{M1-8}$.

5

HIFI-DRAM: ENABLING HIGH-FIDELITY DRAM RESEARCH BY UNCOVERING SENSE AMPLIFIERS WITH IC IMAGING

DRAM vendors do not disclose the architecture of the sense amplifiers deployed in their chips. Unfortunately, this hinders academic research that focuses on studying or improving DRAM. Without knowing the circuit topology, transistor dimensions, and layout of the sense amplifiers, researchers are forced to rely on best guesses, impairing the fidelity of their studies. We aim to fill this gap between academia and industry for the first time by performing Scanning Electron Microscopy (SEM) with Focused Ion Beam (FIB) on recent commodity DDR4 and DDR5 DRAM chips from the three major vendors. This required us to adequately prepare the samples, identify the sensing area, and align images from the different FIB slices. Using the acquired images, we reverse engineer the circuits, measure transistor dimensions and extract physical layouts of sense amplifiers — all previously unavailable to researchers. Our findings show that the commonly assumed classical sense amplifier topology has been replaced with the more sophisticated offset-cancellation design by two of the three major DRAM vendors. Furthermore, the transistor dimensions of sense amplifiers and their revealed physical layouts are significantly different than what is assumed in existing literature. Given commodity DRAM, our analysis shows that the public DRAM models are up to 9x inaccurate, and existing research has up to 175x error when estimating the impact of the proposed changes. To enable high-fidelity DRAM research in the future, we open source our data, including the reverse engineered circuits and layouts.

5.1 INTRODUCTION

DRAM is the target of many research efforts from academia every year with the *sense amplifier* being the fundamental section most commonly modified and simulated [22, 33, 36, 101, 124, 128–146]. Unfortunately, DRAM vendors keep the internal architecture of sense amplifiers in their chips a secret. As a result, researchers are forced to make assumptions and speculate over

crucial design factors, impacting the accuracy of their results. We aim to fill this gap by imaging, and subsequently reverse engineering, sense amplifier circuits on modern DDR4 and DDR5 devices.

Existing research commonly assumes commodity DRAM to employ the classical sense amplifier circuit, which we show not to be the case in DRAM chips from two of three major DRAM vendors. Instead, their sense amplifiers include new components and events to perform offset-compensating operations for reliable DRAM operation with smaller technology nodes. Furthermore, we obtain crucial information concerning deployed sense amplifiers, such as transistor dimensions and physical layout. Using this knowledge, we systematically analyze the feasibility and accuracy of DRAM studies spanning a decade of research. We find that the majority of these studies make inaccurate assumptions about sense amplifiers, resulting in significant errors when estimating the impact of their proposed changes. We formulate recommendations based on our findings to enable high-fidelity DRAM research in the future.

Research accuracy. DRAM provides cheap and low-latency memory based on capacitors. The transition between the analogue world of capacitors and the digital world is performed by *sense amplifiers*. Sense amplifiers enhance the extremely weak signals stored in capacitors while adhering to strict timings [12]. Their design and topology must be reliable towards process manufacturing variability and noise, yet strongly optimized to keep high die efficiency [147]. Research based on modifying sense amplifiers depends on three factors for its accuracy and validity: (i) the employed sense amplifier circuit, commonly assumed to be the classical design, (ii) the transistor dimensions, as circuits with overly large transistors will be optimistic towards their reliability, and (iii) complying with existing layouts, given that the sense amplifier region is a highly optimized area, adding new components should be done with care to achieve realistic area overheads. Unfortunately, information on these crucial factors that are necessary for high-fidelity DRAM research is not publicly available to researchers. This paper aims to fill this gap for the first time.

DRAM reverse engineering. We perform high-resolution chip imaging to reverse engineer the sense amplifier region in six commodity DDR4 and DDR5 devices from the three major DRAM vendors. For the first time, we report circuit topologies, transistor dimensions, and layouts of sense amplifiers on modern commodity DRAM devices. To this end, we combine Scanning Electron Microscopy (SEM) aided with Focused Ion

Beam (FIB) to obtain cross-section images from the samples. Performing SEM/FIB requires adequate sample preparation and the identification of the sensing areas. We then perform a highly sensitive image alignment and noise cancellation on the cross-section images, allowing us to obtain a planar view of the sense amplifier region. Using the planar view images, we reverse engineer the circuit topology by performing a multi-dimension inter- and intra-layer mapping.

HiFi-DRAM. We seek to enable high-fidelity DRAM research using our reverse engineered data. We start by comparing our findings to existing DRAM models, discovering that they employ transistors with dimensions up to 9x different than our samples counterpart. Then, we analyze existing studies that propose to modify the sense amplifier region. We find three major inaccuracies. First, these studies consider inaccurate sense amplifier designs, hence their modifications do not always apply to more modern devices. Second, the addition of new elements assumes free space in existing chips which we find not to be the case in our samples. Third, these proposals do not consider the physical layout of sense amplifiers, underestimating the impact of the proposed changes. Considering these inaccuracies, we find that existing studies can have up to 175x errors in their original estimations when considering modern commodity DRAM. Based on these findings, we formulate a set of recommendations to improve the fidelity of future DRAM research. As an example, future studies must avoid focusing on a single sense amplifier in isolation, since we find that multiple sense amplifiers are interconnected in modern DRAM chips.

Contributions. The following summarizes our contributions:

1. Using high-resolution IC imaging, we reverse engineer the sense amplifiers of modern commodity DRAM devices from the three major vendors on both DDR4 and DDR5 devices.
2. We report on important properties of sense amplifiers in our samples such as their circuitry, transistor dimensions and physical layouts.
3. We evaluate 13 papers that aim to modify sense amplifiers using our reverse engineered information to identify inaccuracies and formulate recommendations for high-fidelity DRAM research in the future.

Open sourcing. The extracted information including IC images, reversed engineered circuits, transistor dimensions and physical layouts can be reached via https://comsec.ethz.ch/hifi-dram.

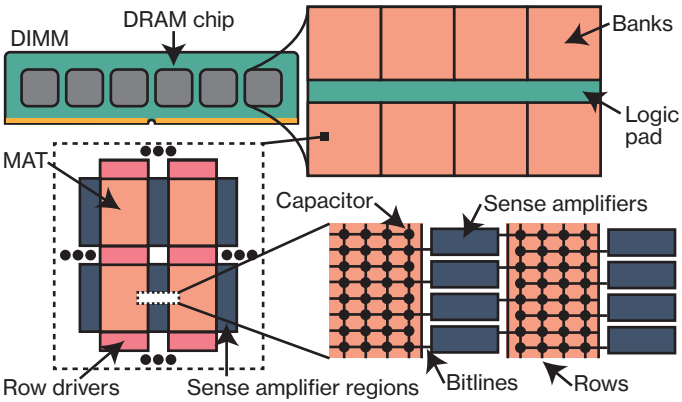

FIG. 5.1: A DIMM contains several DRAM chips, each made of multiple banks. A bank has many MATs, filled with capacitors. Capacitors are connected to SAs via bitlines after they are selected by rows. SAs are in between MATs.

5.2 BACKGROUND

We introduce the DRAM architecture and sense amplifier topologies (5.2.1) before summarizing research that aims at modifying DRAM sense amplifiers (5.2.2).

5.2.1 *DRAM and Sense Amplifier Topologies*

Commodity DRAM is available as chips complying with the DDR protocol, standardized by JEDEC [12, 20]. DRAM used in servers and desktops is usually assembled as multiple identical DRAM chips on dual-inline memory modules (DIMMs) [115, 116].

Physical organization. Internally, a DRAM chip has a hierarchical structure made of banks (Fig. 5.1), each made of multiple MATs that generally contain between half to a million capacitors [22, 148, 149]. Each capacitor in a MAT stores one bit of memory and is identified by the combination of a column, row, and bank address supplied by the memory controller. The MATs are surrounded by row drivers in one direction, and by sense amplifiers (SAs) in the other (Fig. 5.1). When the memory controller accesses memory, it first activates a specific row of a MAT. With a row *activation*, the capacitors in a

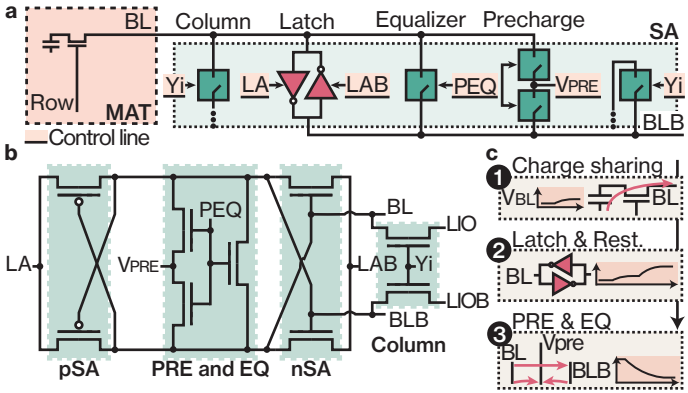

FIG. 5.2: Latch, equalizer, precharge and column are the main sense amplifier elements (**a**). In the classic circuit (**b**), PEQ activates both precharge and equalization. After a row activation, the classic circuit events (**c**) are charge sharing (**1**), and latching & restore (**2**). A precharge makes bitlines connect to Vpre (PRE) and to each others (EQ) (**3**).

MAT are connected via bitlines to the SAs of both sides. The connections are interleaved, resulting in half connections towards each side. Then, to perform reads or writes, the memory controller specifies a subset of these bits (or capacitors) with the column address. Before activating another row, the memory controller must issue a *precharge* to deactivate the current row.

Sense amplifiers. SAs are analog circuits that amplify the weak signals stored in the capacitors. Their design determines the speed of a DRAM chip and must avoid data failures. It is hence a contemporary research topic [107, 147, 150–166]. A SA operates by comparing two bitlines, one that is perturbated by the capacitor of the activated row (BL), and one that is the reference (bitline bar or BLB). The reference bitline comes from the MAT opposite to the activated MAT (Fig. 5.1). This is referred to as an open bitline scheme, currently known to be the most compact scheme and considered the standard [167]. The prime elements of SAs are *latch* circuits (Fig. 5.2a). Once activated by control lines LA and LAB, the latch circuits amplify and lock the difference between BL and BLB. Then, a *column selector* multiplexes the latched data from a particular SA (selected by control line Yi). Finally, *equalizer* and *precharge* circuits restore the SA reference voltage (Vpre with the control line PEQ), which is necessary for accessing data from a different row.

Classic sense amplifier topology and events. The classic SA is shown in Fig. 5.2b. The latch element is made of cross-coupled transistors (two pSA and two nSA). Two transistors are used for the precharge, each connecting a different bitline to Vpre. For equalization, one transistor connects BL and BLB. Lastly, the column signal multiplexes both BL and BLB. The classic SA works as follows. After the row activation, each capacitor on the row shares its charge to a specific bitline (BL) perturbating its voltage (*charge sharing*, Fig. 5.2c). Subsequently, the SA latching elements are activated. This amplification also restores the charge in the capacitor. After the voltage is latched, the memory controller can perform read/write operations. Finally, the memory controller can close the row, and internally, each BL and BLB pair is connected together (*equalized*) and set back to the reference voltage Vpre (*precharged*, Fig. 5.2c).

Contemporary research systematically assumes that the topology deployed on modern devices is the classic SA [22, 101, 124, 128, 130, 132–134, 136, 141, 168–173].

New sense amplifiers. Previous work attempts to enhance DRAM performance with a multitude of new SA topologies [107, 147, 150–166]. These proposals change the classic design by adding elements and modifying events. Meanwhile, DRAM designers aim at packing as many rows as possible per MAT, thus increasing the die efficiency. However, having many rows in a MAT reduces the signal strength latched by the SA, increasing the risk of failure (i.e., latching the opposite value) which is exacerbated by smaller technologies. This latching reliability is the result of manufacturing asymmetries in the transistors and bitlines which create an *offset* between BL and BLB. Thus, many of the new SA topologies aim at compensating for these asymmetries, and a subclass of these directly tries to reduce the offset and is known as offset-compensating (or offset-cancellation) SA [147, 157–166]. In such topologies, the SAs perform additional operations to compensate for these asymmetries.

5.2.2 *Research on DRAM*

Research focusing on commodity DRAM frequently proposes performance enhancements to the SA region. For example, optimizing the precharge event [141, 171] or speeding up the latching mechanism [174] to reduce memory latency. These improvements often require inserting new elements into the SA or MAT area, such as isolation transistors [101]. SAs, by construc-

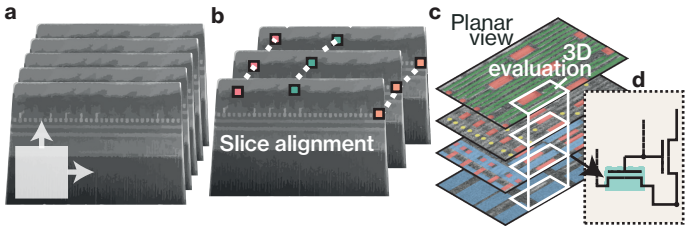

FIG. 5.3: We filter (**a**) and align (**b**) the cross section images. After we obtain a planar view (**c**), we identify connections between different layers, wires, and transistors. This allows us to reverse engineer the SA circuit (**d**).

tion, always latch all bits of a given row. In-DRAM Processing-In-Memory (PIM) exploits this parallelism by modifying SAs and MATs, typically relying on dual-contact cells (DCCs) [134, 135]. DCC is a widely-used SA addition, originally described in [124]. It aims to add an extra row in the MAT, in which each capacitor can connect to two different bitlines instead of one, as selected by two wordlines. Generally, one bitline is the standard connection (BL), while the "extra" bitline (EBL) is connected to the BLB. Lastly, recent work improves memory integrity by adding elements into the SA area [22].

Research fidelity. Multiple aspects critically undermine the accuracy of existing DRAM research that focuses on SAs. First, DRAM MATs and SA regions are highly optimized areas, and they represent the majority of a chip. Therefore, changes in these regions that were intended to be simple, could cause high overheads or require complete re-designs. Second, existing research performs analog simulations either on "best-guess" models or old technologies, and area overheads are based on old values or averages [22, 36, 101, 128]. Lastly, literature assumes that modern DRAM employs the classic SA topology. With HiFi-DRAM, we aim at providing clarity over these aspects.

5.3 OVERVIEW AND CHALLENGES

We aim to reverse engineer the SA region in multiple DRAM chips from the three major DRAM vendors. Then, we seek to use the acquired information to extend and improve the accuracy of existing and future research. To these ends, we must overcome challenges that we now describe.

Vendors do not disclose details about the SA region including its location, section dimensions or component feature sizes. Certain physical properties such as the materials as well as the thickness of the Integrated Circuit (IC) layers are further undisclosed. We want to acquire images of this area to then analyze it. Therefore, the first challenge is:

> **Challenge (C1).** Acquiring images of the SA region, in a way that the elements of interest and all the layers are visible and identifiable. Then, processing the images to obtain a planar view of the circuit.

We address this challenge in Section 5.4 by employing high-resolution imaging. In particular, we first identify the SA region using a blind approach and proceed to image it by acquiring multiple cross-section slices. Then, we post-process the slices to denoise and align them (Fig. 5.3a-b), before changing the point-of-view from cross section to top-down (i.e., planar, Fig. 5.3c).

Once we have obtained a planar view of the different layers of the circuit, we must extract meaningful information. Namely, we must reverse engineer the deployed circuitry and measure component features, such as transistor widths. This requires analyzing the images on different levels of abstraction. First, identifying different material classes that make up different electrical components and measuring their physical dimensions. Second, mapping the visible components and their connections, which might cross layers and the planar view (Fig. 5.3c). Third, understanding how these components interact (i.e., the circuit topology, Fig. 5.3d).

> **Challenge (C2).** Starting from the planar view, reverse engineering the circuits considering all the inter-connected layers, and measuring relevant features.

We describe how we address this challenge in Section 5.5. First, we find features that corresponds to gates, wires and vias. After findinding the MAT bitlines, we identify different classes of transistors. We then trace their intra- and inter-layer connections and their relation to the MAT bitlines. This way, we associate functionalities to the classes of transistors, which we link to the equivalent circuit block. Lastly, we extensively measure dimensions, including transistor sizes and region areas.

Using this newly acquired data, we aim to understand the accuracy of existing DRAM studies. To accomplish this, we review literature to identify common assumptions that are in conflict with our observations. Further,

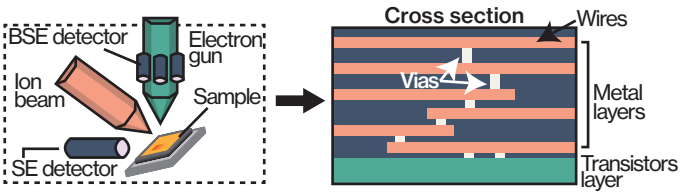

FIG. 5.4: FIB/SEM imaging requires an ion beam and an electron gun, under which the sample is positioned. The BSE detector is placed on the electron gun, while the SE detector is skewed. FIB/SEM allows to image the cross section of an IC. An IC is made of multiple metal layers, which are interconnected by vias. The transistor layer is placed at the bottom of the IC.

we must understand if the original estimations can accurately represent overheads on modern devices.

Challenge (C3). Evaluating the accuracy of existing research.

We address this challenge in Section 5.6, where we evaluate 13 different papers. We find that 8 of them result in more than 20x error in the calculation of the overheads and in the extreme case up to 175x when considering the architecture of modern commodity DRAM. We also study the existing available DRAM models, which we found to deviate substantially from the real chips. To enable high-fidelity DRAM research in the future, we formulate a number of recommendations based on our data and on the inaccuracies that we have observed of existing studies.

5.4 IMAGE ACQUISITION AND POST PROCESSING

Due to the small feature sizes of modern ICs, optical microscopy is not viable for contemporary chip imaging. On the other hand, Scanning Electron Microscopy (SEM) is an imaging technology that allows resolutions below the optical limit [175]. SEM is based on a system emitting an electron beam on the target sample (Fig. 5.4). The sample will in turn emit Secondary Electrons (SE) and BackScatter Electrons (BSE), with intensities that depend on its chemical composition.

SEM parameters. Many parameters influence the quality of a SEM image [176]. For example, the dwell time represents the time that each spot

will receive the beam [177]. A higher dwell time will produce an image with a higher signal-to-noise ratio, but it will require more time which increases the imaging cost since SEM devices are often shared across many different projects. Furthermore, the dwell time is limited by the employed technology and sample stability. The electron beam should be focused and with a high current, while the voltage that accelerates the beam affects brightness. Ultimately, optimal parameters depend on the required resolution, the chip area to image, and the sample under test.

Detectors. SEM images are based on either BSE or SE detectors, which have different contrast characteristics. Generally, BSE will enhance the difference in atomic number between the elements of a sample, while SE depends on the conductivity. Depending on the analyzed sample, the image quality might be better with either BSE or SE.

FIB. ICs are manufactured as various interconnected layers (Fig. 5.4), as such, the features of interest are buried inside the chips. Thus, imaging an IC with only SEM would result in merely viewing the upper-most external layer. Focused Ion Beam (FIB) allows milling the sample of interest. By removing material with FIB, the region of interest is exposed and can be imaged via SEM. FIB is usually implemented as GaFIB, where Gallium ion beams are used. Commercially, FIB/SEM are commonly integrated in single machines.

5.4.1 *Sample Preparation*

For each of the three major DRAM vendors, we analyze a DDR4 and a DDR5 chip, for a total of 6 chips. We extracted the chips from commodity devices sold as Dual Inline Memory Modules (DIMMs). We purchased the DIMMs from online suppliers and, for each chip, we identified the DRAM vendor using the ID reported on the packaging. The list of chips and production years can be found in Tbl. 5.1 (anonymized vendors).

Die extraction. We first aim to expose the chip die, which our imaging targets. We desolder the chip from the DIMM by applying a heatgun (400°C). We further use the heatgun to partially remove the epoxy package covering the die. Lastly, we remove the remaining epoxy with a sulfuric acid solution at 140°C (Fig. 5.5).

ROI identification. Given the die dimensions (up to 75 mm^2, Tbl. 5.1) and the expected features size (tens of nm), imaging the entire chip is not

ID	Vendor	Storage	Yr.	Size	Det.	MATs	Pixl.Res.
A4	A (DDR4)	8Gb	'17	34 mm^2	SE	V.	10.4 nm
B4	B (DDR4)	4Gb	'22	48 mm^2	BSE	N.V.	3.4 nm
C4	C (DDR4)	8Gb	'18	42 mm^2	BSE	V.	5 nm
A5	A (DDR5)	16Gb	'21	75 mm^2	SE	N.V.	5.2 nm
B5	B (DDR5)	16Gb	'22	68 mm^2	BSE	N.V.	4.2 nm
C5	C (DDR5)	16Gb	'22	66 mm^2	BSE	V.	5 nm

TBL. 5.1: **Studied chips.** We study a total of six chips, from the three major DRAM vendors (anonymized as A,B and C). We report the chip production year, its dimension, and SEM information.

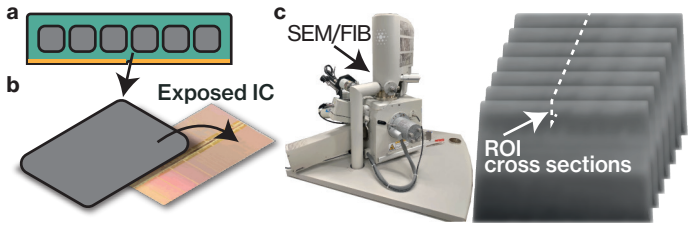

FIG. 5.5: We extract (**a**) and decap (**b**) target chips. Then, we acquire cross sections of the ROI using SEM/FIB (**c**).

realistic for time, cost and required processing. Hence, we must establish a region of interest (ROI, i.e., the SA region). On the exposed die, we first identify banks and logic pad using an optical microscope (AX10 Imager.M2, ZEISS [178]). In some cases, the die extraction exposed lower layers (Tbl. 5.1). In these chips, we identify the ROI as the largest area surrounding a MAT, as typically row drivers are smaller than SA (Section 5.2) [22].

For the remaining chips, the procedure is more challenging, as optical and electron microscopy only reveal the top layer. Its coarse features solely provide the bank-level organization, leaving the MAT locations unclear. We rely on three properties of DRAM chips to identify the ROI. First, the bank areas are dominated by MATs. Second, the feature lines are either perpendicular or parallel to MATs edges. Third, the area occupied by capacitors visually differs from the analog logic [179, 180]. On this basis, using FIB, we acquire blind cross sections in a bank perpendicularly to the feature lines (Fig. 5.6). A single image corresponds to less than one millionth of the chip area. We observe the result, and continue acquiring cross sections

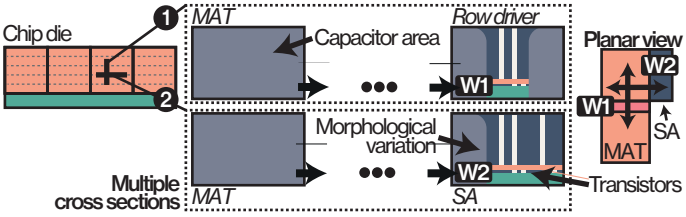

FIG. 5.6: Starting from one direction (**1**), we identify a logic region with width W_1 surrounding a MAT. The opposite direction (**2**) results in a logic region with a width W_2, bigger than W_1. We identify this second region as the SAs.

in the same direction, until we reach a morphological variation in the acquired images. In particular, until we reach an area in which we can identify transistors (Fig. 5.6). Then, based on the properties, we classify the non-logic area as a MAT. We then perform a perpendicular scan, to obtain the other edge of the MAT (Fig. 5.6). We identify the ROI as the biggest logic region surrounding the MAT. The identification procedure lasts no more than 2 hours per chip.

5.4.2 *ROI Imaging*

Once the ROI is identified, we must capture a region large enough to contain complete SAs. We configure SEM/FIB to acquire images of an area of $100\,um^2$ between two adjacent MATs. We hypothesize, based on existing DRAM models [22, 128], that this is enough to capture SAs. We perform this scan on both DDR4 and DDR5 devices (**A4-5**), confirming our theory. Each acquisition took more than 24 hours of SEM/FIB and resulted in imaging many SAs. To reduce the cost of imaging the remaining samples, we perform their acquisitions scanning areas of $30\,um^2$, enough to capture multiple SAs.

Details of SEM/FIB cross sections. We perform volumetric reconstruction using the Helios 5 UX (Thermo Fisher Scientific [181]) as follows. We use FIB to repeatedly slice the ROI, by removing perpendicular slices of 20 nm or 10 nm (30 kV Gallium ion beam with 90 pA beam current). For each slice removed, we image the cross section with SEM. The pixel resolution of the SEM images varies across experiments, down to 3.37 nm (Tbl. 5.1). We first

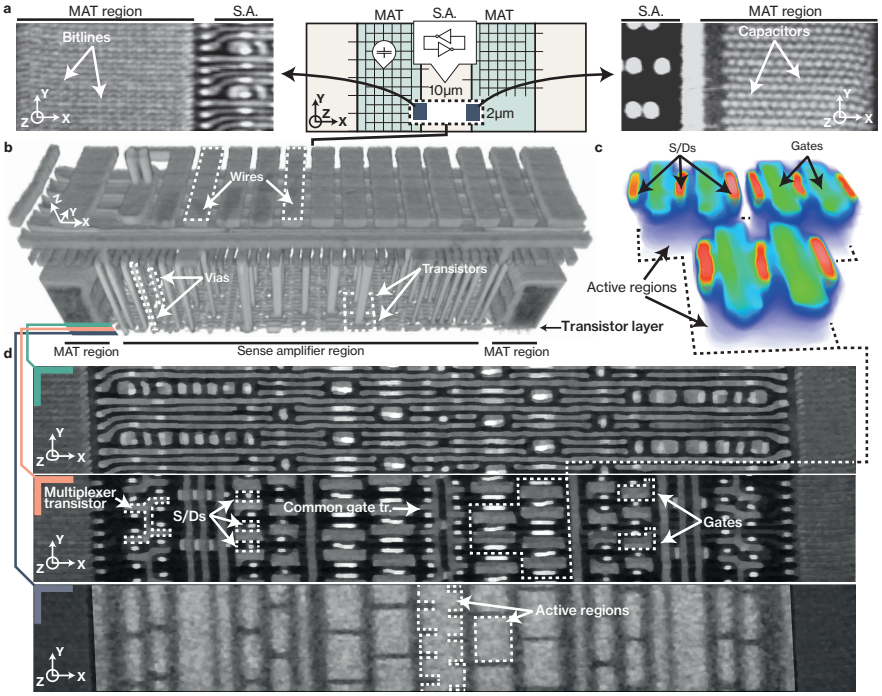

FIG. 5.7: From the imaged ROI of **C5** (**a**-center), we can identify bitlines (**a**-left) and capacitors (**a**-right). They correspond to two different layers, omitted for simplicity. In the 3D reconstruction of the SA region (**b**) wires, vias and transistors are visible. An enhanced image (**c**) shows that different transistors share the same source/drain and active region. From selected planar slices of the 3D reconstruction (**d**), we can see major elements of the SA circuitry: (**1**) connections between the different SA elements (i.e., bitlines); (**2**) gates and source/drains of the transistors; (**3**) the transistor active regions (enhanced contrast).

acquire the SEM images with SE for **A4-5**, which provides good quality. For the remaining chips, SE does not provide a good contrast, likely due to manufacturing processes, so we use BSE instead. The space parameter for SEM is large, so it may be possible for SE to produce images with higher quality for vendor B and C under different SEM parameters, such as a much longer dwell time (increasing the cost). For the acquisition, we use an accelerating voltage of 2 kV, and dwell times of 3 us (**A4-5, B4**) and 6 us (**B5, C4-5**).

5.4.3 *Image Post Processing*

For each chip, we obtain images representing slices of the SA region. To reverse engineer the circuit, however, we require a planar view. That is, we must align the slices together and change the point of view. This step requires post-processing to address high levels of noise and drift.

Noise sensitivity. We measure wire heights in the SA region to be as small as 30 nm (cross-sectional view, **B5**), while the cross section height is generally 130x this value. This makes the planar view extremely sensitive to slice alignment. We hence need to reduce the slice alignment noise and drift to less than 0.77% (1/130) of the slice. Further, this alignment error must apply across all the acquired slices for consistent planar views.

Reliable post-processing. We use the Dragonfly software [182] to perform multiple post processing steps. First, we crop the slices to include only the cross section. Then, we filter the images to reduce noise with edge preserving algorithms (split-Bregman [183] or Chambolle [184] for a total-variation denoising). Once denoised, we align the slices using the mutual-information algorithm of Dragonfly. In particular, each slide is aligned with respect to the previous one. Lastly, we change the point of view from cross section to top-down, and we further rotate the volume to correct for possible remaining misalignments. This process is semi-automatic as it requires per-scan tuning, and can be reliably performed in less than 3 hours by an analyst, including the algorithms execution time. Note that the official Thermo Fisher software (Avizo3D [185]), which would include semi-automatic tools for slice alignment and SEM processing, did not produce results matching our requirements.

5.4.4 *Imaging System Capabilities*

We focus on the SA region, yet so far, the target component feature size is unclear. As half of the evaluated chips were produced in 2022, it is uncertain if our system can provide a sufficient resolution. In Fig. 5.7, we demonstrate the reconstruction capabilities of our end-to-end imaging methodology on **C5**. After post-processing, the planar images allow for the identification of transistors, wires and vias in the SA region. For example, Fig. 5.7a reveals capacitors and bitlines. The capacitors are arranged in a honeycomb structure and are placed above the bitlines (stacked capacitor [186]). The honeycomb

structure has been proposed as a method to increase the capacitance for the same cell size [186, 187]. Fig. 5.7b shows the 3D reconstruction of the SA region. Fig. 5.7c depicts two transistors sharing a contact node. Finally, Fig. 5.7d displays three slices of the SA logic layers, including SA bitlines, transistors and active regions.

5.5 CIRCUITS REVERSE ENGINEERING

First, we describe the reverse engineering of the sense amplifier circuit and its challenges (5.5.1). We then detail the size measurements that we performed (5.5.2) and analyze the implemented sense amplifier layouts (5.5.3).

5.5.1 *Reverse Engineering the Circuit Topology*

Analog circuits are generally drawn by humans on single-layer schematics, yet this abstraction differs from the final manufactured product, which is implemented as multiple layers stacked vertically (Section 5.4). Transistors may appear close on the schematic, but they may be deployed far from each other to better utilize the available chip area. Further, some transistor characteristics (e.g., NMOS and PMOS) that impact the circuit topology might be visually indistinguishable, which is the case with our samples as opposed to previous work that targeted different technology [188]. Overall, these factors make reverse engineering analog circuits in DRAM a challenging task, even after a multi-dimensional mapping.

From images to circuits. We now describe the methodology that we devised to account for the aforementioned challenges shown in Fig. 5.7. (i) First, we determine color intensities that correspond to gates, wires and vias (Fig. 5.7b-d). (ii) Then, we identify the bitlines in the MAT, and their connections in the SA region (Fig. 5.7a). We use the bitlines as an anchor for inferring the circuit. (iii) We identify the different transistors, the corresponding wires, and the source/drains contacts. To correctly identify transistors, we include active regions in the analysis (Fig. 5.7d). (iv) We classify three different types of transistor: multiplexer transistors (Fig. 5.7d), transistors with a common gate spanning the entire region (Fig. 5.7d), and coupled transistors with a shared source (Fig. 5.7c). (v) The multiplexer transistors select a group of 4 adjacent bitlines, which are then connected to wires spanning the entire region (not shown). Each of these transistors

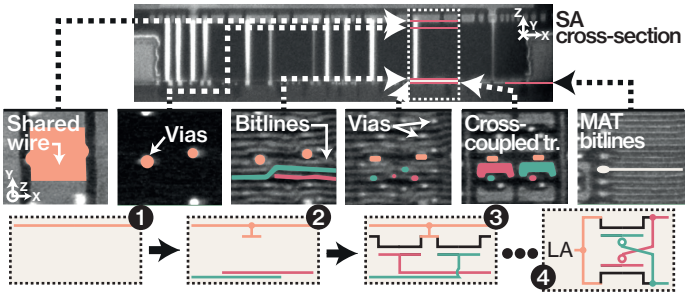

FIG. 5.8: Cross-coupled transistor reverse engineer on **B5**. From the top slice, we identify a shared line (**1-2**). Two bitlines are connected with vias to transistors gate and drains (**2-3**). Once the full circuit is mapped (**4**), the transistors are identified as the pSA latching element of the SA.

have a different gate control. Hence, we identify them as column transistors. (vi) We track the bitlines connections to the coupled transistors, which represent a latched connection, and find that the source is shared among all of these transistors. Hence, these represent the latching part of the SA. (vii) In **B4**, **C4** and **C5**, the transistors with a common gate short the bitlines together and with a global value. Therefore, we identify these as precharge/equalizer elements. (viii) Finally, in SAs, PMOS latching transistors are designed with a smaller width than NMOS [156, 164, 189, 190]. Based on this, we identify the PMOS latching transistors. We considered all other transistors to be of NMOS type. We collaborated with an independent DRAM vendor that confirmed our analysis.

Fig. 5.8 displays an example of multi-dimensional mapping. We trace inter- and intra-layer connections, and identify cross-coupling transistors. Only after the entire circuit is mapped, we can identify these elements as the pSA of the latching component. Contrary to existing models and literature about commodity DRAM, we found extra elements in **B5** and **A4-5**.

Investigating the extra elements. In chips **B5**, **A4**, and **A5**, we found that the precharge element is stand-alone, and four extra "common-gate" transistors are present (Fig. 5.10). Moreover, we could not identify the bitlines equalizer. We hypothesized that such a topology might belong to the extensive corpus of past research [107, 147, 150–166] and identified similarities with research in *offset cancellation*. Of the many different proposed offset

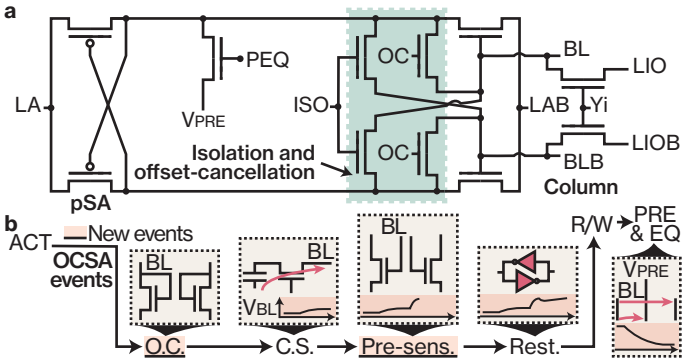

FIG. 5.9: **(a)** Offset-cancellation SA (OCSA) circuit, used on **A4**, **A5** and **B5**, and its events **(b)**, which include offset-cancellation and the pre-sensing.

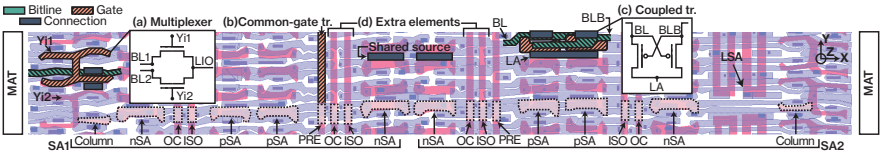

FIG. 5.10: **Reverse engineered layout of the A5 chip.** To reverse engineer the chips, we identified different transistor classes. Multiplexer (**a**), common-gate (**b**) and coupled (**c**) transistors. In chips **A4-5** and **B5** we discovered extra elements (**d**). In all the chips, two SA were stacked between two MATs (i.e., along X).

cancellation SAs (OCSAs) [147, 157–166], we could finally pin-point the reverse-engineered circuits to one design [164].

Deployed OCSAs. Half of the devices (**B4**, **C4**, **C5**) use the classic SA circuit [104]. Instead, chips **A4**, **A5**, and **B5** implement an OCSA with the circuit shown in Fig. 5.9a [164]. Our paper is the first to publicly report that OCSA topology is being used in modern commodity DRAM. The deployment of OCSA circuits is likely due to the need for reliable sensing of capacitor charge in smaller technology nodes. Hence, it is very likely that manufacturers that currently use the classic SA circuits will move to OCSA in the future as well.

The OCSA topology differs substantially from the conventional circuit and adds four transistors and two control signals (Fig. 5.9a). Two of these transistors perform isolation (ISO) and the other two offset cancellation

(OC). The OCSA adds two operations to the classic row activation of a SA (Fig. 5.9b). First, charge sharing is anteceded by an offset cancellation operation. Second, the restoring operation is preceded by a pre-sensing event. The pre-sensing operation latches the capacitor value without the bitline load and without recharging the capacitor.

Isolation and equalization in OCSAs. DRAM research often proposes adding isolation transistors to the SAs [22, 101, 141, 191]. Typically, this allows decoupling the bitlines from the latching circuit. The isolation transistors used in OCSAs differ from these proposals, as the bitlines are decoupled from the latch amplifier drains but not from the gates. Furthermore and as previously explained, precharge and equalizer circuits are necessary to operate DRAM. Normally, this is achieved by a three-transistor setup (Section 5.2). In the reverse engineered circuits that employ OCSAs, the equalizer transistor is absent. Instead, this functionality is achieved by the simultaneous activation of both the isolation and offset-cancellation elements.

5.5.2 *Measurement of the DRAM Elements*

DRAM research that performs analog simulations relies on the transistor widths and lengths. The ratios between width and length (W/L) of the various elements strongly affect the stability and speed of DRAM operations. Therefore, we measure the width and length of each transistor employed by the SA. To measure their length, we consider the gate pitch between source and drain. To measure their width, we consider the overlapping area between the gate and the active region [192]. We perform multiple distinct measurements for each dimension and for each transistor. In total, we make 835 size measurements using Dragonfly [182]. In Section 5.6, we provide more details about these measurements and their impact on existing DRAM research.

Effective sizes. Adding elements to the SA region must take IC design rules into account. As a proxy for important design rules for DRAM research, we measure the effective spacing dimensions required for each element. That is, we measure the element size including the full gate dimension and the element distance from the other components. These dimensions are higher than the width and length of transistors, as they must include safety margins. We measure the dimensions of the SA and MAT regions, and of

each die. We make all our measurements for each of the samples available online.

5.5.3 *Layout Design Analysis*

Given the absence of information, previous studies that modify SA regions have mostly ignored the physical layout of modern DRAM devices. Unfortunately, this has repercussions on the accuracy and overhead of research as we will show in Section 5.6. To bridge this gap, we re-created the physical layout of the SA regions in all of our samples, which we make available in the standard GDSII format. As an example, Fig. 5.10 shows the layout of the **A5** chip. We find that all the samples employ an open-bitline architecture, considered the standard since many years [167]. We now describe the immediate differences that we found when comparing our findings to the assumptions made on the DRAM layout in existing work.

SA arrangement. In all studied chips, SA elements are always arranged horizontally (i.e., along the X axis, Fig. 5.10). All chips have two stacked SAs (side by side) between each MAT (Fig. 5.10, "SA1" and "SA2"), with transistors positioned symmetrically. Both sets of SAs connect to bitlines from each MAT. This is different from the usual description of SAs [22, 69, 101, 128, 130, 134, 136, 139, 168], where only one SA is placed between two MATs. Consequently, the overhead of elements shared among all bitlines (e.g., isolation transistors) is lower than what previous research has assumed. The column transistors are always the first elements connected by MAT bitlines in the SA region which results in inaccuracies in previous studies as we outline in Section 5.6. Finally, the SA region further contains latching elements connected to the selected column (i.e., to LIO). They represent the next data-path step and are not part of the SA circuit (Fig. 5.10a, LSA). However, because they are part of the SA region, their presence reduces the relative overhead of new elements.

Transition from MAT to SA. The transition of a bitline from MAT to planar logic requires an overhead on average of 318 nm (DDR4) and 275 nm (DDR5) in the bitline direction. This information is useful for research that proposes to add transistors in between the MAT (i.e., creating a new logic region), and to the best of our knowledge, it has not been reported in literature so far. For example, [144] proposes to place isolation transistors in a MAT to create shorter bitlines. On top of the overhead due to a single isolation transistor,

two transitions are required, as the MAT is split in two. On average this represents 1.6% and 1.1% of a MAT in DDR4 and DDR5 respectively.

Transistor characteristics. Previous work that adds transistors to the SA region often calculates overheads as an increase in SA height (i.e., along *X*) related to the transistors width (W) [22, 171]. This is correct for the latching components, as their width is parallel to the SA height. Instead, we find that precharge, isolation and offset-cancellation transistors are designed with a common gate spanning the entire SA region (i.e., along *Y*). As result, the width of these transistors is perpendicular to the width of the other elements (Fig. 5.10). Therefore, the addition of these elements causes a SA height overhead that depends on their lengths (L). Finally, we find that the access transistors used in the MAT region have a bitline/wordline layout typical of Buried Channel Array Transistors (BCAT) across all vendors. This is consistent with literature [186].

5.6 EVALUATION OF EXISTING DRAM RESEARCH

We first analyze analog DRAM models based on our reverse engineered data (§5.6.1). Then, we evaluate previous studies that modify the sense amplifier region to identify the sources of inaccuracies (§5.6.2), and provide a more accurate measurement of area overhead in these studies (§5.6.3). Finally, we comment on the reliability of physical experiments on DRAM (§5.6.4) and conclude with a set of recommendations for accurate DRAM research in the future (§5.6.5).

5.6.1 *Inaccuracies of Existing Analog Models*

Analog simulations of DRAM SAs are widely used in research papers [22, 33, 36, 101, 124, 128–144, 193]. However, no DDR5 model exists, and only two public models exist for DDR4. In particular, DDR4 SAs are simulated with CROW (2019) [128] or with REM (2022) [22] models. Neither of these two models are based on commodity DRAM devices from the major manufacturers. CROW is employed in multiple recent work [36, 101, 128], however its transistor dimensions are based on best guesses and it does not include column transistors. REM is based on real DDR4 transistor dimensions of a smaller vendor (Zentel Japan [194]) that uses 25nm technology. This technology, however, is one generation older than current commodity DDR4

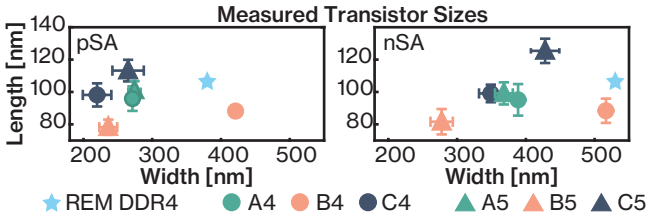

FIG. 5.11: Measured transistor sizes of the pSA and nSA for all the chips and the values used in REM. CROW values are omitted as severely out the range.

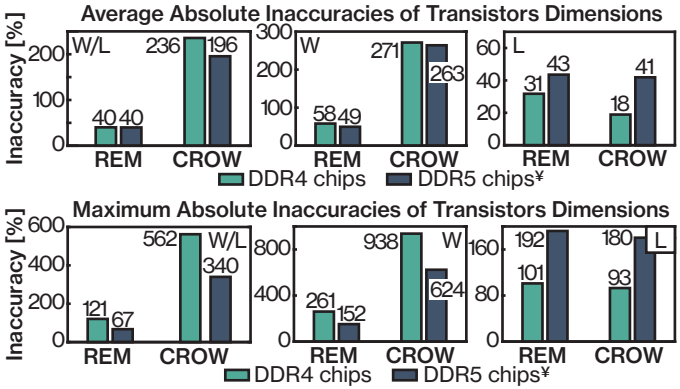

FIG. 5.12: Average and maximum absolute inaccuracies of REM and CROW compared to the measured transistors in all the chips, as W/L ratios, and separately width and length. (¥) Portability to DDR5.

device from the three major DRAM vendors [195]. Neither models include the OCSA design. In Fig. 5.11, we report the dimensions of the latching transistors (nSA and pSA) for all the chips that we reverse engineered and for REM. We omit CROW in this figure as its values are vastly out of range. To understand if existing DRAM models provide an accurate representation of commodity DRAM devices, we analyzed the width-to-length ratio (W/L) of their transistors. Generally, higher width-to-length ratios correspond to more optimistic simulations [22]. In particular, we compared each model element to each ratio obtained for that element in each chip. We included a comparison to DDR5 technology to determine whether these models can provide a reasonable approximate description of these chips as well.

Results. We report a summary of our analysis in Fig. 5.12 and discuss the inaccuracies when compared to DDR4 chips. On average, CROW has the higher inaccuracy between the two models (236%). The precharge of CROW has the highest W/L inaccuracy (562% when compared with the measured values of **C4**). We further analyzed the individual widths and lengths. CROW gives the most inaccurate widths on average (271%), with an inaccuracy of 938% when compared to the **C4**'s precharge transistors. REM has the most inaccurate lengths on average (31%), with an inaccuracy of 101% when compared to **C4**'s equalizer transistors. The models follow a similar trend when considering the DDR5 technology.

5.6.2 *Deriving Research Inaccuracies*

We now analyze research that proposes to modify the SA regions of commodity DRAM. Correctly estimating overheads and feasibility of proposed modifications to DRAM is a complex task. Even for variations that would appear minor, the absence of public information about modern devices is an obstacle that forces researchers to make blind assumptions and estimate overheads based on outdated ranges. Once a new technology is released, such as DDR5, it is further challenging to understand if existing proposals remain feasible. Unfortunately, this is a de-facto accepted status in the field. With HiFi-DRAM, we provide researchers with a realistic basis upon which they can evaluate their work.

Papers summary and analysis. We study 13 papers, crossing technologies (DDR3-DDR4) and spanning a decade (2013-2023). Among these, [101, 140, 141, 171, 173, 191] seek to improve performance by modifying DRAM. [22] modifies DRAM to improve data integrity. The remaining studies aim to implement in-DRAM PIM [124, 134, 135, 142, 196, 197]. Studying these papers, we enumerate the sources of research inaccuracy when compared to commodity devices. As research on DRAM has been based on similar repeated assumptions throughout the years, we identify common inaccuracies across most papers under study, which we describe as **I1** to **I5**.

A major inaccuracy arises from the implementation of dual-contact cells (DCCs), discussed in Section 5.2.2. Their overhead is estimated to be approximately two wordlines, i.e., negligible [124]. In all the chips that we studied, MATs do not have available space for the extra bitlines (**I1**, Fig. 5.13a). Because of this, implementing a DCC requires doubling the MAT area. As MATs represent the majority of a DRAM chip, implementing even a single

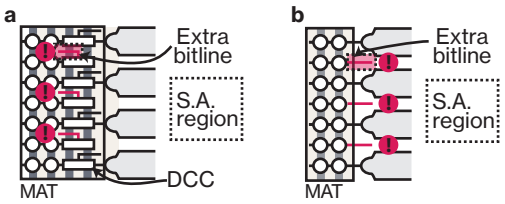

FIG. 5.13: **(a)** No free space to add new bitlines in the MAT (**I1**) and **(b)** SA region (**I2**).

DCC results in a significant overhead. Further, making the MATs larger will increase the wordline length. Due to this increase, extending the MATs will also require placing new row drivers to correctly drive the new load. Row drivers areas are comparable to the sense amplifier areas [22]. As measured from our data, all the papers affected by **I1** require on average 57% chip overhead, solely for the MAT extension.

Array size and implications related to I1. DRAM vendors are constantly trying to reduce the size of their memory arrays to increase chip yield. The current standard is the open bitline design, which has an area consumption per cell of $6F^2$ (F represents the feature size). If MAT bitlines could be made closer, this would effectively reduce the cell size to under $6F^2$. Prior work describes the same type of cell structure as a DCC, resulting in an area of $12F^2$ when reverting to a folded-bitline architecture [198], confirming the aforementioned overhead.

Inaccuracy (I1). No free space for bitlines in the MAT area.

The main DCC application is an inverter PIM operation exploiting row parallelism, so the EBL is connected to the reference bitline (BLB). There is, however, no extra space for bitlines crossing the SA region (**I2**, Fig. 5.13b). The same inaccuracy arises in [101], which aims to connect all bitlines in the MAT to the same SA region. In [22], additional wires are required for routing purposes of the new circuitry. All these papers do not consider extra overhead due to the new required wiring.

Inaccuracy (I2). No free space for bitlines in the SA area.

In standard IC processes (i.e., non-DRAM), designers could try to resolve **I1-2** by exploiting the many available IC layers. However, as confirmed by our observations and literature, this is not possible in the DRAM SA regions

and MATs, where the number of IC layers is limited [141, 199, 200]. One possibility is shrinking the bitlines. However, MAT bitlines are the main contributor of timing and signal level. Changing their dimensions would severely affect the functioning of the SA.

The feasibility of changing the SA bitlines depends on the fabrication process. These changes would affect resistance and parasitics, and must respect the design rules, such as the minimum distance between bitlines. These considerations are not addressed in the aforementioned studies. We refer the reader to Appendix 5.9.1 for further explanations.

Some studies rely on SA circuitries that differ from the ones that are currently deployed. This affects [134], in which the authors consider the precharge and equalization gates to be independent for each SA. In reality, the gates of these elements are spanning the entire SA region and are shared across all the SAs. In [173], the authors assume that isolation transistors are available in the design. As described earlier (Section 5.5), these isolation transistors differ from the one employed in OCSAs. These inaccuracies could be resolved by adding new elements to the SA region or by duplicating existing ones, introducing additional area overhead.

Inaccuracy (I3). Assuming a SA circuitry that is not deployed in practice.

Additionally, [22, 141, 191] modify SAs by adding isolation transistors and assuming that column transistors are physically placed after the SAs. In reality, column transistors are the first elements after the MATs. Therefore, these modifications require a reorganization of the SA elements.

Inaccuracy (I4). Assuming a SA physical layout that does not correspond to the ones deployed.

Finally, no paper includes the OCSA topology in their studies, in contrast to chips **A4-5, B5**. This affects the overheads of additional components and the timings of the new events as well as the reliability of analog simulations, impacting the performance, energy and power overheads of the affected operations.

Inaccuracy (I5). Not considering offset-cancellation designs as the deployed SA topologies.

In Tbl. 5.2, we provide a summary of our findings, and evaluate the overhead inaccuracies which we detail next.

Research	Inacc.	Error	Port. Cost	DDR	Yr.
CHARM [140]	**I5**	N/A	0.29x	3	'13
R.B. DEC. [141]	**I4,5**	N/A	−0.25x	3	'14
AMBIT [124]	**I1,2,5**	N/A	68x	3	'17
DrACC [196]	**I1,2,5**	35x	34x	4	'18
Graphide [135]	**I1,2,5**	54x	52x	4	'19
In-Mem.Lowcost. [197]	**I1,2,5**	70x	67x	4	'19
ELP2IM [134]	**I2,3,5**	N/A	90x	3	'20
CLR-DRAM [101]	**I2,5**	22x	21x	4	'20
SIMDRAM [142]	**I1,2,5**	70x	67x	4	'21
Nov. DRAM [191]	**I4,5**	0.49x	0.001x	4	'21
PF-DRAM [171]	**I5**	0.35x	−0.01x	4	'21
REGA [22]	**I2,4,5**	8x	7x	4	'23
CoolDRAM [173]	**I1,2,3,5**	175x	168x	4	'23

TBL. 5.2: Research inaccuracies, average overhead error and portability cost. The overhead error is evaluated on the original technology if possible. Portability cost represents the overhead variation of DDR3 to DDR4/5 and DDR4 to DDR5.

5.6.3 *Evaluation of Research Inaccuracies*

The area overhead of DRAM research is a main factor influencing its feasibility. However, previous estimations have been performed by referencing outdated values or average ranges [22, 23, 101, 134, 141, 171, 173, 196, 197, 201]. It is unclear if the reported results are realistic when considering commodity DRAM and how they would change if applied to a newer technology such as DDR5. We now study these aspects for the papers under analysis. When the paper is evaluated on its original technology, we describe the variation of overhead as overhead error. In case the original technology is older than DDR4, the analysis is not applicable (N/A). Instead, when the paper is compared to a different technology, we describe the variation in overhead as porting cost. To this end, we use the transistors effective sizes, the region areas measured in Section 5.5, and we include the effects of the inaccuracies discussed in Section 5.6.2. For our calculations, we follow the description of the original document as closely as possible, and calculate the overheads for each chip. For papers requiring isolation transistors that gave no indication about their sizing, we used dimensions from the existing isolation transistors if any isolation transistor is present in the chip, else we scaled their average dimensions to the chip values.

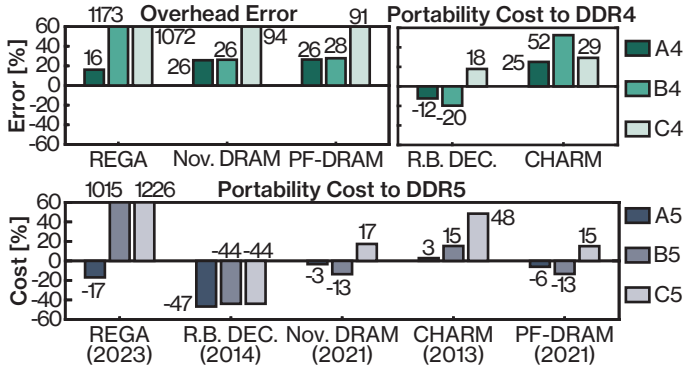

FIG. 5.14: Research portability cost and overhead error divided per DRAM vendor. Papers where the cost/error is always higher than 10x are omitted.

Effects of I1-2. Inaccuracies **I1-2** result in an extension of MAT and SA regions. For example, if for every existing bitline a new bitline must be added, this effectively means doubling the width of the region. We contacted the authors of the papers affected by **I1** or **I2**, as the quantitative effect of these inaccuracies is severe. While many authors replied ([22, 101, 124, 134, 142, 173, 197]), none provided clarifying details that would resolve the inaccuracies given the content of the original papers.

However, the authors of [134] suggested a feasible approach to implementing the NOT operation (not evaluated in the original paper). The authors of [101, 124, 142] explained that a detailed implementation was outside the scope of their paper, in line with prior work. They also suggested that if adding a new metal layer were possible, alternative implementations (not evaluated in the original paper) could reduce the overhead. The authors of [22] reported that their collaborating (smaller) DRAM vendor did not report **I2** to be an issue on their technology and are exempted from **I2** in chips **A4-5** as discussed in Appendix 5.9.1. Finally, the authors of [173] explained that their evaluations were based on the original paper on the topic ([124]) and limited by not having access to proprietary details of DRAM circuitry. We believe these communications show that HiFi-DRAM is highlighting inaccuracies in previous work, and will provide value to researchers focusing on DRAM.

Results. The average overhead errors and porting costs are reported in Table 5.2. Papers affected by **I1** or **I2** have consistently large errors and porting costs (e.g., up to 175x) across all vendors. Such large errors occur due

to the (often) very small overheads reported by the papers (e.g., 0.4% [173]). In Fig. 5.14, we report the inaccuracies and porting costs individually per DRAM vendor. We omit proposals where these are always higher than 10x. The formulas used to calculate the errors are reported in Appendix 5.9.2.

Observation 1. The overheads of papers can vary significantly across vendors. For example, [140] has a variation of 0.45x when passing from Vendor A to Vendor C on DDR5 chips.

Observation 2. Porting modifications that are originally intended for older DRAM devices results in much lower overhead in DDR5 due to smaller technology nodes. The biggest variation is for [141] (-0.47x on **A5**). This analysis shows that, in newer technologies, researchers can generally afford more complex circuits.

5.6.4 *Out-of-spec DRAM Experiments and OCSA*

Issuing commands to DRAM without complying with the DDR standard is used for reverse engineering [202], transistor speed evaluation [22], and DRAM characterization [28, 203, 204]. Furthermore, operating DRAM out of specification is used to exploit the interactions of SAs with rows to perform logic operations [170]. To these ends, researchers expect commodity DRAM to deploy classic SAs. However, chips employing OCSAs have key differences in timings and functionalities that could impact similar experiments. First, charge sharing is usually assumed to occur immediately upon a row activation with the classical SA design [170]. Instead, in chips with OCSAs, charge sharing is delayed and happens after the offset cancellation. This could impact, for example, studies that intend to perform majority-based row operations [170], where multiple rows perform charge sharing without starting the latch operation. Second, bitlines have only two states in the classic circuit, either being latched or precharged and equalized. Instead, OCSAs briefly connect bitlines to diode-connected transistors as a way to improve the sensing margin (Section 5.5). This could impact studies that skip the precharge command to avoid any effect on bitlines [170].

5.6.5 *Recommendations*

Based on our findings, we now formulate a number of recommendations for future DRAM research. First, simple changes might result in non-negligible overheads when applied to commodity devices (**I1-2**).

> **Recommendation (R1).** Overheads should be estimated including all additions to MATs or SAs, such as wires connections.

Second, research usually focuses on a single SA element, which can lead to wrong assumptions such as considering SA control lines being independent of other SAs (**I3**).

> **Recommendation (R2).** Research modifying SAs should consider the impact on all the interconnected SAs.

Third, differences between the abstract SA circuit schematic and its physical layout can create inaccuracies, for example when adding isolation elements (**I4**).

> **Recommendation (R3).** Research should consider the physical layout and organization of SAs blocks.

Finally, the deployed SA topology impacts analog simulations, overheads and timings of research proposals (**I5**). Moreover, it can impact DRAM experiments that operate the devices outside of the standard.

> **Recommendation (R4).** Research should consider OCSA in the evaluation.

On existing and future work. Our results discussed in Section 5.6.3 show that some previous work incur high overhead when considering current commodity devices. We would like to clarify that our evaluation does not reduce the value of these proposals, some of which have led to a high sprout of subsequent work. HiFi-DRAM's aim is to increase the fidelity of DRAM research, and we sincerely hope that it is not indiscriminately used to stop novel future work due to potentially higher (but more accurate) reported overheads.

5.7 RELATED WORK

5.7.1 *DRAM Reverse Engineering*

To the best of our knowledge, HiFi-DRAM is the first public research that reverse engineers the sense amplifier topologies, transistor sizes and physical layout of DRAM. In [6], the authors directly issue DRAM commands to devices with an FPGA. Their aim is to reverse engineer internal digital control mechanisms that protect against memory corruption. This results in estimating the existence of row activation counters and their sizes and is unrelated to sense amplifiers. Recent work [202] tries to obtain the number of rows in MATs by exploiting data corruption.

Techinsights [205] is a company that sells access to reverse engineered chips, including DRAM. Unfortunately, the price for accessing DRAM information equivalent to this paper is prohibitive for academic researchers (in the order of $100 ks). Moreover, the corresponding license [206] prohibits sharing the data publicly and may prevent the resulting publication. This makes it impossible to reproduce and verify research based on such data.

5.7.2 *IC Imaging*

IC imaging is commonly performed by device manufacturers for identifying manufacturing failures in the produced chips [207–210] or possible malicious modifications [188, 211, 212]. It is further used by private organizations to verify IP infringement [213–215], and to perform offensive research [216–219] such as breaking protection systems [175] or extracting private keys [220].

Previous work reports imaging of proprietary ICs on different targets, from EEPROMs [217], baseband chips [213], lockout chips [175], microcontrollers [220], processors [221], to system-on-chips [222]. A substantial orthogonal research direction is based on trying to automatize standard cell recognition and netlist extraction [209, 223–226].

5.8 CONCLUSION

We reverse engineered the sense amplifier region in DDR4 and DDR5 devices from the major DRAM vendors. We discovered that half of the chips employ an offset-cancellation sense amplifier, instead of the commonly assumed classical design. We measured transistor dimensions and further reverse engineered the physical layouts of the sense amplifiers. With this acquired knowledge, we validated the overhead and accuracy of existing research in the past decade that modify DRAM sense amplifiers. Considering commodity modern DRAM, our analysis shows that the public DRAM models are up to 9x inaccurate, and existing research has up to 175x error when estimating the impact of the proposed changes. We hope this paper and the reverse engineered information to enable high-fidelity DRAM research in the future and foster new research directions.

5.9 APPENDIX

5.9.1 *Effects of Changing Bitlines*

Among many other things, IC design rules describe the minimum width of wires and their safety distance from other elements. Bitlines are the narrowest wires placed on the lowest metal layer (M1) of the SA region, which then extend in the MAT region. We now briefly discuss on the feasibility of shrinking bitlines, which is related to **I1-2**.

Process feasibility. From a pure manufacturing point of view, shrinking wires that are already narrow can cause the interruption of their conductivity (i.e., creating disjointed wires), while reducing the distance with adjacent wires can create short circuits [227].

Electrical impact. Shrinking wires increase their electrical resistance (R), while making wires closer increases crosstalk [192]. The increase in R will reduce the speed of transmission of bitlines. In DRAM applications, this further translates to the time required to: precharge bitlines, charge sharing, latching the stored data and recharge the capacitor. Crosstalk is modeled as capacitive coupling between parallel wires. Effectively, crosstalk means that a variation in one wire will affect its adjacent wires. This is a particularly well known problem in DRAM applications and can cause read failure [228–230].

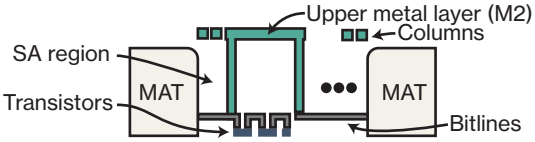

FIG. 5.15: Connection on M2 of existing bitlines (**A4-5**). Columns connect to LIO/LIOB (Fig. 5.2b).

DRAM manufacturing processes use the smallest bitlines width possible to keep the design as compact and reliable as possible. However, even in the case that shrinking bitlines would be possible and not performed in the original layout, doubling the number of bitlines would still result in large overhead. As an example, if halving **B5** SA-region bitlines were possible, the result would still add 21% chip area overhead on top of the existing overheads. As the safe distance (d) is kept, and the bitline width (B_w) is $B_w \simeq d \times 2$, the SA extension in the Y direction would be:

$$\begin{aligned} Ext &= \frac{T_B \times 2 \times (d + B_w/2)}{T_B \times (d + B_w)} - 1 \\ &= \frac{2 \times (B_w/2 + B_w/2)}{(B_w/2 + B_w)} - 1 = \frac{4}{3} - 1 \simeq 33\% \end{aligned} \tag{5.1}$$

where T_B is the initial number of bitlines in the region. Due to layout requirements, this extension is required on the MAT as well (or alternatively, it introduces empty spaces), resulting in 21% chip overhead.

Metal layer 2 in A4-5. In chips **A4-5**, bitlines that use the second set of SAs are connected via the metal layer 2 (M2). This differs from the other chips where these bitlines are directly connected on M1, and M2 is fully dedicated to other connections. These bitlines are *already* present in the SA region, and the translation is performed after the column transistors (Fig. 5.15). M2 wires are around 8x bigger than bitlines on M1, are not packed closely, and the layer presents empty spaces. We evaluated that [22] would require reducing these wires by 0.25x to accommodate new connections, and thus we consider this possible. Papers that require adding *new* bitlines to the SA region are not impacted by this because they still require the new bitlines to enter the sense amplifier region as shown in Fig. 5.13b.

5.9.2 *Overhead calculations*

We now briefly discuss the mathematical formulas we used to derive overhead errors. For each paper, we estimate its overhead (P_{chip}) for each imaged chip ($chip$). Given the paper original overhead estimation (P_{oe}), we report the average of ($P_{chip}/P_{oe} - 1$) in Tbl. 5.2. For simplicity, we define $P_{chip} = P_{extra}/Chip_{area}$ and now describe P_{extra}.

Papers that suffer **I1** or **I2** will require a severe extension of the SA or MAT region. Due to layout requirements, extending only the MAT or SA alone will require its counterpart to be extended as well (or alternatively, it introduces equivalent empty spaces). Generally, calculating the effect of **I1** and/or **I2** for a paper that doubles the bitline can be approximated as:

$$P_{extra} = MAT_{area} + SA_{area}$$

REGA [22] requires adding one new bitline every three in chips **B4-5** and **C4-5**:

$$P_{extra} = (MAT_{area} + SA_{area})/3$$

We now report the formulas of P_{extra} for the remaining papers and for REGA on chips **A4-5**. In the following, $MATs$ represents the number of MATs in a chip, SA_w the SA region width. Instead iso_{ls}, san_{ws}, sap_{ws}, col_{ws} are the horizontal (i.e., X direction) sizes of isolation, nSA, pSA and column transistors, respectively. Most of the calculations represent a horizontal extension multiplied by the width, replicated for the total number of SA regions in a chip. As all the chips implement two stacked sense amplifiers, original papers that requires adding a new SA, will actually require adding 2 SAs to connect all bitlines. Further, isolation transistors are shared among multiple rows.

REGA [22] requires new isolation transistors and SAs (**A4-5**):

$$P_{extra} = MATs \times SA_w \times (2 \times iso_{ls} + 8 \times \frac{san_{ws} + sap_{ws}}{6})$$

NR.B. DEC. [141] requires new isolation transistors:

$$P_{extra} = MATs \times SA_w \times 2 \times iso_{ls}$$

Nov. DRAM [191] adds isolation, column and SA transistors:

$$P_{extra} = MATs \times SA_w \times (2 \times iso_{ls} + 2 \times col_{ws} + 8 \times (san_{ws} + sap_{ws}))$$

CHARM [140] changes the aspect ratio of MATs, where 1% is an overhead due to layout reorganization (we use the configuration [×2,/4] from the original paper):

$$P_{extra} = MATs \times SA_w \times SA_h/4 + 0.01 \times Chip_{area}$$

PF-DRAM [171] adds independent isolation transistors and an SA imbalancer, similarly to an SA:

$$P_{extra} = MATs \times SA_w \times (4 \times iso_{ls} + 8 \times (san_{ws} + sap_{ws}))$$

We invite the reader to refer to the original content of the papers and to https://comsec.ethz.ch/hifi-dram for more details.

6

CONCLUSION AND OUTLOOK

In this dissertation, we studied DRAM security by devising novel Rowhammer mitigations and demonstrating bit flips from the new RISC-V ecosystem for the first time. By collaborating with a DRAM vendor and by reverse engineering commodity DRAM devices, we open-sourced unprecedented detailed information about DRAM internals, which we believe will improve the fidelity of future DRAM research. Overall, we demonstrated that *a principled approach to DRAM security is key to both offensive and defensive research efforts*. We now describe the main contributions of each chapter and propose research questions for future work.

RISC-H. In Chapter 2, with *RISC-H*, we demonstrated Rowhammer bit flips triggered by a RISC-V device for the first time. RISC-V is a new architecture, and only recently its first high-end CPU has become available to consumers. Compared to the mature and complex Intel and AMD counterparts, it is expected that its design might be much slower and less optimized. Porting the patterns and methods used to trigger Rowhammer bit flips by previous work results in unsuccessful attacks on this CPU and a false sense of security. Instead, we carefully analyzed internal memory bottlenecks that we could bypass with new memory patterns and devised a novel method to order memory requests based on surgical delays. Once we combined these new approaches with our reverse-engineered DRAM functions, we were able to demonstrate that the RISC-V ecosystem is also affected by Rowhammer.

Research Question for Future Work 1. What is the feasibility of Rowhammer on the RISC-V ecosystem for different DRAM DIMMs?

Our study was constrained by the extremely limited memory support provided by the RISC-V CPU, which resulted in only one DDR4 DIMM successfully booting from 85 of our lab. Therefore, a clear continuation of our work is to assess the feasibility of Rowhammer on the RISC-V ecosystem with a larger variety of DIMMs, once extended memory support is available.

Research Question for Future Work 2. What is the feasibility of Rowhammer on the RISC-V ecosystem for DDR4 and DDR5 devices with ECC?

Demonstrating successful Rowhammer bit flips on a novel system requires significant reverse engineering and extensive characterization efforts. Therefore, it is a topic of research that focuses on memory modules that do not include ECC-protected memory. Literature has shown that successful Rowhammer attacks can be performed on ECC modules; however, it is unclear how effective Rowhammer is on RISC-V devices employing ECC memory.

PROTRR. In Chapter 3, we presented our Rowhammer mitigation ProTRR. Contrary to industry results and researchers' belief, we proved that TRR can provide security to current DDR4 devices. Furthermore, we demonstrated that the new DDR5 protocol can enhance ProTRR security while reducing the overhead for protection.

Research Question for Future Work 3. Can ProTRR overhead be reduced by relying on a probabilistic approach?

ProTRR design provides deterministic guarantees against Rowhammer attacks. However, for very vulnerable devices, it can require substantial overhead. As such, future work could evaluate how to implement a probabilistic mitigation on top of ProTRR design, and what is the trade-off between security and overhead.

Research Question for Future Work 4. What is the trade-off between the addition of ECC and ProTRR?

The addition of ECC might be required on future technologies to support data integrity independently from Rowhammer attacks. The deployment of ECC will introduce an overhead, but will effectively reduce the device vulnerability towards Rowhammer. As the vulnerability is reduced, the overhead of ProTRR will decrease. Therefore, future work could analyze the trade-off of integrating ECC with ProTRR.

REGA. In Chapter 4, we studied how to secure DRAM devices in case the Rowhammer vulnerability becomes much worse in future devices. With our novel mitigation *REGA*, we proposed changes to the internal DRAM architecture that secure these devices. To achieve this, we added an extra

sense amplifier to perform refresh operations in parallel with memory requests. To evaluate the reliability of REGA, in collaboration with a DRAM vendor we developed the first accurate DRAM model for modern devices, which we further open-sourced. By not relying on states, we were able to design a mitigation that scales even with high blast radius and extremely low thresholds.

Research Question for Future Work 5. Can REGA circuitry be used to perform Processing-In-Memory operations?

With REGA, we focused on the security of DRAM devices against Rowhammer. However, adding new circuitry in the sense amplifier region can support other operations as well. In particular, a common research field in DRAM is to exploit row parallelism to perform Processing-In-Memory. Future research could study how to leverage the additional circuitry introduced by REGA to enable Processing-In-Memory. This would effectively use REGA to enhance both security and performance.

HIFI-DRAM. In Chapter 5, we imaged and reverse engineered DDR4 and DDR5 devices from the major DRAM vendors focusing on the sense amplifier region. By evaluating ten years of research, our work *HiFi-DRAM* showed that literature was based on repeatedly shared wrong assumptions. This issue is multifaceted: the commonly assumed sense amplifier circuitry has been replaced in half of the studied chips, no DRAM model accurately describes commodity devices, and many research proposals can result in overheads much larger than anticipated when layout inaccuracies are considered.

Research Question for Future Work 6. Can REGA be optimized for the circuitry of commodity devices that have no space for extra connections?

With our mitigation REGA, we proposed the addition of new sense amplifiers to perform refreshes in parallel with memory requests. To achieve this, REGA requires new connections, which we showed with HiFi-DRAM might be hard to achieve in some devices without extra overhead. Future work could study how to optimize REGA circuitry to enable seamlessly deployment on these devices.

Research Question for Future Work 7. Can this approach be used to reverse engineer deployed Rowhammer mitigations?

The deployed mitigations on DRAM devices are kept undisclosed by DRAM vendors. Their security guarantees are therefore unclear. Future work could perform our invasive reverse engineering approach to determine the circuitry deployed on modern devices to mitigate Rowhammer. While this is clearly of research interest, it will require a very complex reconstruction of the logic and significant reverse engineering effort.

Research Question for Future Work 8. Do other technologies similarly present a huge gap between Research and industry?

With HiFi-DRAM, we discovered that research on DRAM suffers from a high level of inaccuracy when compared to modern devices. Future work should assess whether other technologies that are of interest to researchers share this problem as well.

CONCLUSION. We first demonstrated that even new architectures on less mature CPUs can trigger bit flips by successfully performing Rowhammer on a RISC-V system for the first time. Contrary to industry results and previous work uncertainties [65, 231–233], we proved that securing DRAM devices is possible with TRR already on DDR4 devices, and its security guarantees are substantially enhanced with the latest DDR5 protocol. To protect future technologies that might be extremely vulnerable to Rowhammer, we designed a scalable stateless mitigation that parallelizes refreshes to activations by modifying the internal DRAM architecture. Our design has been based on a collaboration with a minor DRAM vendor; however, little information is available about commodity devices. To fill this gap, we reverse-engineered commodity DRAM devices and compared our findings to the assumptions made in existing research. We discovered critical discrepancies between industry and literature regarding the deployed circuitry, leading to significant research inaccuracies. Finally, to improve the fidelity of future studies, we open-sourced both the accurate DRAM model developed with the DRAM vendor and all the data we extracted from commodity devices.

BIBLIOGRAPHY

[1] Kuljit S. Bains, John B. Halbert, Christopher P. Mozak, Theodore Z. Schoenborn, and Zvika Greenfield. *Row hammer refresh command*. 2012.

[2] Zvika Greenfield, Kuljit S. Bains, Theodore Z. Schoenborn, Christopher P. Mozak, and John B. Halbert. *Row hammer condition monitoring*. 2012.

[3] Yoongu Kim, Ross Daly, Jeremie Kim, Chris Fallin, Ji Hye Lee, Donghyuk Lee, Chris Wilkerson, Konrad Lai, and Onur Mutlu. "Flipping Bits in Memory without Accessing Them: An Experimental Study of DRAM Disturbance Errors". In: *ISCA*. 2014.

[4] Michael Redeker, Bruce F Cockburn, and Duncan G Elliott. "An Investigation into Crosstalk Noise in DRAM Structures". In: *IEEE MTDT*. IEEE. 2002, 123.

[5] Pietro Frigo, Emanuele Vannacci, Hasan Hassan, Victor van der Veen, Onur Mutlu, Cristiano Giuffrida, Herbert Bos, and Kaveh Razavi. "TRRespass: Exploiting the Many Sides of Target Row Refresh". In: *IEEE S&P*. 2020.

[6] Hasan Hassan, Yahya Can Tugrul, Jeremie S Kim, Victor Van der Veen, Kaveh Razavi, and Onur Mutlu. "Uncovering In-DRAM RowHammer Protection Mechanisms: A New Methodology, Custom RowHammer Patterns, and Implications". In: *MICRO-54: 54th Annual IEEE/ACM International Symposium on Microarchitecture*. 2021, 1198.

[7] Patrick Jattke, Victor van der Veen, Pietro Frigo, Stijn Gunter, and Kaveh Razavi. "Blacksmith: Scalable Rowhammering in the Frequency Domain". In: *IEEE S&P*. 2022.

[8] Lucian Cojocar, Jeremie Kim, Minesh Patel, Lillian Tsai, Stefan Saroiu, Alec Wolman, and Onur Mutlu. "Are We Susceptible to Rowhammer? An End-to-End Methodology for Cloud Providers". In: *IEEE S&P*. 2020, 712.

[9] Finn de Ridder, Pietro Frigo, Emanuele Vannacci, Herbert Bos, Cristiano Giuffrida, and Kaveh Razavi. "SMASH: Synchronized Many-Sided Rowhammer Attacks from JavaScript". In: *USENIX Security*. 2021.

[10] Andrew Kwong, Daniel Genkin, Daniel Gruss, and Yuval Yarom. "RAMBleed: Reading Bits in Memory Without Accessing Them". In: *IEEE S&P*. 2020.

[11] Patrick Jattke, Max Wipfli, Flavien Solt, Michele Marazzi, Matej Bölcskei, and Kaveh Razavi. "ZenHammer: Rowhammer Attacks on AMD Zen-based Platforms". In: *33rd USENIX Security Symposium (USENIX Security 2024)*. 2024.

[12] *JESD79-5B: Double Data Rate 5 (DDR5) SDRAM*. 2022.

[13] *JESD79-5C: Double Data Rate 5 (DDR5) SDRAM*. 2024.

[14] SOPHGO. *Sophon SG2042*. 2024.

[15] Peter Pessl, Daniel Gruss, Clémentine Maurice, Michael Schwarz, and Stefan Mangard. "DRAMA: Exploiting DRAM Addressing for Cross-CPU Attacks". In: *USENIX Security*. 2016.

[16] Zelalem Birhanu Aweke, Salessawi Ferede Yitbarek, Rui Qiao, Reetuparna Das, Matthew Hicks, Yossi Oren, and Todd Austin. "ANVIL: Software-Based Protection Against Next-Generation Rowhammer Attacks". In: *ASPLOS*. 2016.

[17] Jeremie S. Kim, Minesh Patel, A. Giray Yağlıkçı, Hasan Hassan, Roknoddin Azizi, Lois Orosa, and Onur Mutlu. "Revisiting RowHammer: An Experimental Analysis of Modern DRAM Devices and Mitigation Techniques". In: *ISCA*. 2020, 638.

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

[19] Woongrae Kim, Chulmoon Jung, Seongnyuh Yoo, Duckhwa Hong, Jeongjin Hwang, Jungmin Yoon, Ohyong Jung, Joonwoo Choi, Sanga Hyun, Mankeun Kang, et al. "A 1.1 v 16gb ddr5 dram with probabilistic-aggressor tracking, refresh-management functionality, per-row hammer tracking, a multi-step precharge, and core-bias modulation for security and reliability enhancement". In: *2023 IEEE International Solid-State Circuits Conference (ISSCC)*. IEEE. 2023, 1.

[20] JEDEC Solid State Technology Association. *JESD79-4B, DDR4 Specification*. 2017.

[21] Michele Marazzi, Tristan Sachsenweger, Flavien Solt, Peng Zeng, Kubo Takashi, Maksym Yarema, and Kaveh Razavi. "HiFi-DRAM: Enabling High-fidelity DRAM Research by Uncovering Sense Amplifiers with IC Imaging". In: *51st IEEE/ACM International Symposium on Computer Architecture (ISCA)*. 2024.

[22] Michele Marazzi, Flavien Solt, Patrick Jattke, Kubo Takashi, and Kaveh Razavi. "REGA: Scalable Rowhammer Mitigation with Refresh-Generating Activations". In: *IEEE S&P*. 2023.

[23] Michele Marazzi, Patrick Jattke, Flavien Solt, and Kaveh Razavi. "PROTRR: Principled yet Optimal in-DRAM Target Row Refresh". In: *IEEE S&P*. 2022.

[24] Mark Seaborn and Thomas Dullien. "Exploiting the DRAM Rowhammer Bug to Gain Kernel Privileges". In: *Black Hat USA*. 2015.

[25] Kaveh Razavi, Ben Gras, Erik Bosman, Bart Preneel, Cristiano Giuffrida, and Herbert Bos. "Flip Feng Shui: Hammering a Needle in the Software Stack". In: *USENIX Security*. 2016.

[26] D. Gruss, M. Lipp, M. Schwarz, D. Genkin, J. Juffinger, S. O'Connell, W. Schoechl, and Y. Yarom. "Another Flip in the Wall of Rowhammer Defenses". In: *IEEE S&P*. 2018.

[27] Koksal Mus, Yarkın Doröz, M Caner Tol, Kristi Rahman, and Berk Sunar. "Jolt: Recovering tls signing keys via rowhammer faults". In: *2023 IEEE Symposium on Security and Privacy (SP)*. IEEE. 2023, 1719.

[28] Zhenrong Lang, Patrick Jattke, Michele Marazzi, and Kaveh Razavi. "BLASTER: Characterizing the Blast Radius of Rowhammer". In: *3rd Workshop on DRAM Security (DRAMSec) co-located with ISCA 2023*. ETH Zurich. 2023.

[29] Hwayong Nam, Seungmin Baek, Minbok Wi, Michael Jaemin Kim, Jaehyun Park, Chihun Song, Nam Sung Kim, and Jung Ho Ahn. "DRAMScope: Uncovering DRAM Microarchitecture and Characteristics by Issuing Memory Commands". In: *arXiv preprint arXiv:2405.02499* (2024).

[30] Wei He, Zhi Zhang, Yueqiang Cheng, Wenhao Wang, Wei Song, Yansong Gao, Qifei Zhang, Kang Li, Dongxi Liu, and Surya Nepal. "Whistleblower: A system-level empirical study on rowhammer". In: *IEEE Transactions on Computers* (2023).

[31] Ataberk Olgun, Majd Osseiran, A Giray Yağlıkçı, Yahya Can Tuğrul, Haocong Luo, Steve Rhyner, Behzad Salami, Juan Gomez Luna, and Onur Mutlu. "An experimental analysis of rowhammer in hbm2 dram chips". In: *2023 53rd Annual IEEE/IFIP International Conference on Dependable Systems and Networks-Supplemental Volume (DSN-S)*. IEEE. 2023, 151.

[32] Lois Orosa, Abdullah Giray Yaglikci, Haocong Luo, Ataberk Olgun, Jisung Park, Hasan Hassan, Minesh Patel, Jeremie S Kim, and Onur Mutlu. "A Deeper Look into RowHammer's Sensitivities: Experimental Analysis of Real DRAM Chips and Implications on Future Attacks and Defenses". In: *MICRO-54: 54th Annual IEEE/ACM International Symposium on Microarchitecture*. 2021, 1182.

[33] A Giray Yağlıkçı, Haocong Luo, Geraldo F De Oliviera, Ataberk Olgun, Minesh Patel, Jisung Park, Hasan Hassan, Jeremie S Kim, Lois Orosa, and Onur Mutlu. "Understanding RowHammer under reduced wordline voltage: An experimental study using real DRAM devices". In: *2022 52nd Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*. IEEE. 2022, 475.

[34] Andreas Kogler, Jonas Juffinger, Salman Qazi, Yoongu Kim, Moritz Lipp, Nicolas Boichat, Eric Shiu, Mattias Nissler, and Daniel Gruss. "Half-Double: Hammering from the next Row Over". In: *USENIX Security*. 2022.

[35] Tanj Bennett, Stefan Saroiu, Alec Wolman, and Lucian Cojocar. "Panopticon: A Complete In-DRAM Rowhammer Mitigation". In: *DRAMSec*. 2020.

[36] Minbok Wi, Jaehyun Park, Seoyoung Ko, Michael Jaemin Kim, Nam Sung Kim, Eojin Lee, and Jung Ho Ahn. "SHADOW: Preventing Row Hammer in DRAM with Intra-Subarray Row Shuffling". In: *HPCA*. 2023, 333.

[37] Jeonghyun Woo, Gururaj Saileshwar, and Prashant J Nair. "Scalable and secure row-swap: Efficient and safe row hammer mitigation in memory systems". In: *2023 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*. IEEE. 2023, 374.

[38] Ataberk Olgun, Yahya Can Tugrul, Nisa Bostanci, Ismail Emir Yuksel, Haocong Luo, Steve Rhyner, Abdullah Giray Yaglikci, Geraldo F Oliveira, and Onur Mutlu. "ABACuS: All-Bank Activation Counters for Scalable and Low Overhead RowHammer Mitigation". In: *arXiv preprint arXiv:2310.09977* (2024).

[39] Michael Jaemin Kim, Jaehyun Park, Yeonhong Park, Wanju Doh, Namhoon Kim, Tae Jun Ham, Jae W Lee, and Jung Ho Ahn. "Mithril: Cooperative Row Hammer Protection on Commodity DRAM Leveraging Managed Refresh". In: *IEEE HPCA*. IEEE. 2022, 1156.

[40] Moinuddin Qureshi, Aditya Rohan, Gururaj Saileshwar, and Prashant J. Nair. "Hydra: Enabling Low-Overhead Mitigation of Row-Hammer at Ultra-Low Thresholds via Hybrid Tracking". In: *ISCA*. New York New York: ACM, 2022, 699.

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

[42] A. Giray Yağlikçi, Minesh Patel, Jeremie S. Kim, Roknoddin Azizi, Ataberk Olgun, Lois Orosa, Hasan Hassan, Jisung Park, Konstantinos Kanellopoulos, Taha Shahroodi, Saugata Ghose, and Onur Mutlu. "BlockHammer: Preventing RowHammer at Low Cost by Blacklisting Rapidly-Accessed DRAM Rows". In: *HPCA*. 2021, 345.

[43] Anish Saxena, Gururaj Saileshwar, Prashant J Nair, and Moinuddin Qureshi. "Aqua: Scalable rowhammer mitigation by quarantining aggressor rows at runtime". In: *2022 55th IEEE/ACM International Symposium on Microarchitecture (MICRO)*. IEEE. 2022, 108.

[44] Gururaj Saileshwar, Bolin Wang, Moinuddin Qureshi, and Prashant J Nair. "Randomized Row-Swap: Mitigating Row Hammer by Breaking Spatial Correlation between Aggressor and Victim Rows". In: (2022), 14.

[45] Onur Mutlu and Thomas Moscibroda. "Stall-time fair memory access scheduling for chip multiprocessors". In: *40th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO 2007)*. IEEE. 2007, 146.

[46] Prathap Kumar Valsan and Heechul Yun. "MEDUSA: a predictable and high-performance DRAM controller for multicore based embedded systems". In: *2015 IEEE 3rd international conference on cyber-physical systems, networks, and applications*. IEEE. 2015, 86.

[47] William K Zuravleff and Timothy Robinson. *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. 1997.

[48] Scott Rixner, William J Dally, Ujval J Kapasi, Peter Mattson, and John D Owens. "Memory access scheduling". In: *ACM SIGARCH Computer Architecture News* 28.2 (2000), 128.

[49] Yuan Xiao, Xiaokuan Zhang, Yinqian Zhang, and Radu Teodorescu. "One Bit Flips, One Cloud Flops: Cross-VM Row Hammer Attacks and Privilege Escalation." In: *USENIX Security*. 2016.

[50] Minghua Wang, Zhi Zhang, Yueqiang Cheng, and Surya Nepal. "DRAMDig: A Knowledge-assisted Tool to Uncover DRAM Address Mapping". In: *DAC '20*. 2020.

[51] Ben Gras, Kaveh Razavi, Erik Bosman, Herbert Bos, and Cristiano Giuffrida. "ASLR on the Line: Practical Cache Attacks on the MMU." In: *NDSS*. Vol. 17. 2017, 26.

[52] Daniel Gruss, Clémentine Maurice, and Stefan Mangard. "Rowhammer.Js: A Remote Software-Induced Fault Attack in JavaScript". In: *DIMVA*. 2016.

[53] Erik Bosman, Kaveh Razavi, Herbert Bos, and Cristiano Giuffrida. "Dedup Est Machina: Memory Deduplication as an Advanced Exploitation Vector". In: *IEEE S&P*. 2016.

[54] Pietro Frigo, Cristiano Giuffrida, Herbert Bos, and Kaveh Razavi. "Grand Pwning Unit: Accelerating Microarchitectural Attacks with the GPU". In: *IEEE S&P*. 2018.

[55] Victor van der Veen, Yanick Fratantonio, Martina Lindorfer, Daniel Gruss, Clementine Maurice, Giovanni Vigna, Herbert Bos, Kaveh Razavi, and Cristiano Giuffrida. "Drammer: Deterministic Rowhammer Attacks on Mobile Platforms". In: *ACM SIGSAC*. CCS '16. New York, NY, USA: Association for Computing Machinery, 2016, 1675.

[56] Victor van der Veen, Martina Lindorfer, Yanick Fratantonio, Harikrishnan Padmanabha Pillai, Giovanni Vigna, Christopher Kruegel, Herbert Bos, and Kaveh Razavi. "Guardion: Practical Mitigation of DMA-Based Rowhammer Attacks on ARM". In: *DIMVA*. 2018.

[57] Andrei Tatar, Radhesh Krishnan Konoth, Elias Athanasopoulos, Cristiano Giuffrida, Herbert Bos, and Kaveh Razavi. "Throwhammer: Rowhammer Attacks over the Network and Defenses". In: *USENIX ATC*. 2018.

[58] Moritz Lipp, Michael Schwarz, Lukas Raab, Lukas Lamster, Misiker Tadesse Aga, Clémentine Maurice, and Daniel Gruss. "Nethammer: Inducing Rowhammer Faults Through Network Requests". In: *EuroS&PW*. 2020, 710.

[59] Ingab Kang, Eojin Lee, and Jung Ho Ahn. "CAT-TWO: Counter-Based Adaptive Tree, Time Window Optimized for DRAM Row-Hammer Prevention". In: *IEEE Access* 8 (2020), 17366.

[60] Jung Min You and Joon-Sung Yang. "MRLoc: Mitigating Row-Hammering Based on Memory Locality". In: *DAC*. IEEE. 2019, 1.

[61] Eojin Lee, Ingab Kang, Sukhan Lee, G Edward Suh, and Jung Ho Ahn. "TWiCe: Preventing Row-Hammering by Exploiting Time Window Counters". In: *ISCA*. 2019.

[62] Lucian Cojocar, Kaveh Razavi, Cristiano Giuffrida, and Herbert Bos. "Exploiting Correcting Codes: On the Effectiveness of ECC Memory Against Rowhammer Attacks". In: *IEEE S&P*. 2019.

[63] Ferdinand Brasser, Lucas Davi, David Gens, Christopher Liebchen, and Ahmad-Reza Sadeghi. "CAn't Touch This: Software-Only Mitigation against Rowhammer Attacks Targeting Kernel Memory". In: *USENIX Security*. 2017.

[64] Radhesh Krishnan Konoth, Marco Oliverio, Andrei Tatar, Dennis Andriesse, Herbert Bos, Cristiano Giuffrida, and Kaveh Razavi. "ZebRAM: Comprehensive and Compatible Software Protection Against Rowhammer Attacks". In: *USENIX OSDI*. 2018.

[65] Z. Zhang, Y. Cheng, D. Liu, S. Nepal, Z. Wang, and Y. Yarom. "PThammer: Cross-User-Kernel-Boundary Rowhammer through Implicit Accesses". In: *MICRO*. 2020, 28.

[66] Jung-Bae Lee. *Green Memory Solution*. 2014.

[67] Micron. *DDR4 SDRAM Datasheet*. Tech. rep. 2016, 380.

[68] Jayadev Misra and David Gries. "Finding Repeated Elements". In: *Science of Computer Programming* 2.2 (1982), 143.

[69] Yoongu Kim, Vivek Seshadri, Donghyuk Lee, Jamie Liu, and Onur Mutlu. "A Case for Exploiting Subarray-Level Parallelism (SALP) in DRAM". In: *ISCA*. IEEE. 2012, 368.

[70] Onur Mutlu. "The RowHammer Problem and Other Issues We May Face as Memory Becomes Denser". In: *DATE*. 2017.

[71] Google LLC. *Half-Double: Next-Row-Over Assisted Rowhammer*. Tech. rep. Google LLC, 2021, 22.

[72] Misiker Tadesse Aga, Zelalem Birhanu Aweke, and Todd Austin. "When Good Protections Go Bad: Exploiting Anti-DoS Measures to Accelerate Rowhammer Attacks". In: *HOST*. 2017.

[73] Sarani Bhattacharya and Debdeep Mukhopadhyay. "Curious Case of Rowhammer: Flipping Secret Exponent Bits Using Timing Analysis". In: *CHES*. 2016.

[74] Sarani Bhattacharya and Debdeep Mukhopadhyay. "Advanced Fault Attacks in Software: Exploiting the Rowhammer Bug". In: *Fault Tolerant Architectures for Cryptography and Hardware Security*. Ed. by Sikhar Patranabis and Debdeep Mukhopadhyay. Singapore: Springer Singapore, 2018, 111.

[75] Apostolos P Fournaris, Lidia Pocero Fraile, and Odysseas Koufopavlou. "Exploiting Hardware Vulnerabilities to Attack Embedded System Devices: A Survey of Potent Microarchitectural Attacks". In: *Electronicsweek* (2017).

[76] Damian Poddebniak, Juraj Somorovsky, Sebastian Schinzel, Manfred Lochter, and Paul Rösler. "Attacking Deterministic Signature Schemes Using Fault Attacks". In: *EuroS&P*. 2018.

[77] Rui Qiao and Mark Seaborn. "A New Approach for Rowhammer Attacks". In: *HOST*. 2016.

[78] Zhenkai Zhang, Zihao Zhan, Daniel Balasubramanian, Xenofon Koutsoukos, and Gabor Karsai. "Triggering Rowhammer Hardware Faults on ARM: A Revisit". In: *ASHES*. 2018.

[79] Xin-Chuan Wu, Timothy Sherwood, Frederic T. Chong, and Yanjing Li. "Protecting Page Tables from RowHammer Attacks Using Monotonic Pointers in DRAM True-Cells". In: *ASPLOS*. New York, NY, USA: Association for Computing Machinery, 2019, 645.

[80] Seyed Mohammad Seyedzadeh, Alex K Jones, and Rami Melhem. "Counter-Based Tree Structure for Row Hammering Mitigation in DRAM". In: *IEEE Computer Architecture Letters* 16.1 (2016), 18.

[81] Mark Kaczmarski. *Thoughts on Intel Xeon E5-2600 v2 Product Family Performance Optimisation Component Selection Guidelines*. 2014.

[82] Mungyu Son, Hyunsun Park, Junwhan Ahn, and Sungjoo Yoo. "Making DRAM Stronger Against Row Hammering". In: *DAC*. 2017, 1.

[83] Erik D Demaine, Alejandro López-Ortiz, and J Ian Munro. "Frequency Estimation of Internet Packet Streams with Limited Space". In: *ESA*. Springer. 2002, 348.

[84] Sujeet Ayyapureddi and Raghukiran Sreeramaneni. "Apparatus and Method Including Analog Accumulator for Determining Row Access Rate and Target Row Address Used for Refresh Operation". US10964378B2. 2021.

[85] Ya-Chun Lai, Po-Hsun Wu, and Jen-Shou Hsu. "Target Row Refresh Mechanism Capable of Effectively Determining Target Row Address to Effectively Mitigate Row Hammer Errors without Using Counter Circuit". US10916293B1. 2021.

[86] Koo et al. "A 1.2V 38nm 2.4Gb/s/pin 2Gb DDR4 SDRAM with bank group and ×4 half-page architecture". In: 2012, 40.

[87] Thomas Vogelsang. "Understanding the energy consumption of dynamic random access memories". In: *2010 43rd Annual IEEE/ACM International Symposium on Microarchitecture*. IEEE. 2010, 363.

[88] Niladrish Chatterjee, Naveen Muralimanohar, Rajeev Balasubramonian, Al Davis, and Norman P Jouppi. "Staged reads: Mitigating the impact of DRAM writes on DRAM reads". In: *IEEE International Symposium on High-Performance Comp Architecture*. IEEE. 2012, 1.

[89] Chun et al. "A 16Gb LPDDR4X SDRAM with an NBTI-tolerant circuit solution, an SWD PMOS GIDL reduction technique, an adaptive gear-down scheme and a metastable-free DQS aligner in a 10nm class DRAM process". In: *2018 IEEE International Solid - State Circuits Conference - (ISSCC)*. 2018, 206.

[90] Shim et al. "A 16Gb 1.2V 3.2Gb/s/pin DDR4 SDRAM with improved power distribution and repair strategy". In: *2018 IEEE International Solid - State Circuits Conference - (ISSCC)*. 2018, 212.

[91] James Bucek, Klaus-Dieter Lange, and Jóakim v. Kistowski. "SPEC CPU2017: Next-Generation Compute Benchmark". In: *ICPE*. 2018, 41.

[92] Nathan Binkert, Bradford Beckmann, Gabriel Black, Steven K. Reinhardt, Ali Saidi, Arkaprava Basu, Joel Hestness, Derek R. Hower, Tushar Krishna, Somayeh Sardashti, Rathijit Sen, Korey Sewell, Muhammad Shoaib, Nilay Vaish, Mark D. Hill, and David A. Wood. "The Gem5 Simulator". In: *SIGARCH* 39.2 (2011), 1.

[93] Shang Li, Zhiyuan Yang, Dhriaj Reddy, Ankur Srivastava, and Bruce Jacob. "DRAMsim3: A Cycle-Accurate, Thermal-Capable DRAM Simulator". In: *IEEE Computer Architecture Letters* (2020).

[94] Roland E Wunderlich, Thomas F Wenisch, Babak Falsafi, and James C Hoe. "SMARTS: Accelerating Microarchitecture Simulation via Rigorous Statistical Sampling". In: *ISCA*. 2003, 84.

[95] JEDEC Solid State Technology Association. *JEP300-1: Near-Term DRAM Level Rowhammer Mitigation*. 2021.

[96] Micron Technology Inc. *How Much Power Does Memory Use?* 2021.

[97] TechInsights Inc. *Micron MT40A4G4JC-062E_E 1z nm DDR4 Process Flow Full*. 2021.

[98] Andrei Tatar, Cristiano Giuffrida, Herbert Bos, and Kaveh Razavi. "Defeating Software Mitigations against Rowhammer: A Surgical Precision Hammer". In: *RAID*. 2018.

[99] Google LLC. *Half-Double: Next-Row-Over Assisted Rowhammer*. Tech. rep. 2021, 22.

[100] Youssef Tobah, Andrew Kwong, Ingab Kang, Daniel Genkin, and Kang G Shin. "SpecHammer: Combining Spectre and Rowhammer for New Speculative Attacks". In: *IEEE S&P*. 2022.

[101] Haocong Luo, Taha Shahroodi, Hasan Hassan, Minesh Patel, A Giray Yağlıkçı, Lois Orosa, Jisung Park, and Onur Mutlu. "CLR-DRAM: A Low-Cost DRAM Architecture Enabling Dynamic Capacity-Latency Trade-Off". In: *ACM/IEEE ISCA*. IEEE. 2020, 666.

[102] Kira Kraft, Chirag Sudarshan, Deepak M Mathew, Christian Weis, Norbert Wehn, and Matthias Jung. "Improving the Error Behavior of DRAM by Exploiting Its Z-channel Property". In: *DATE*. IEEE. 2018, 1492.

[103] Matthias Jung, Carl C Rheinländer, Christian Weis, and Norbert Wehn. "Reverse Engineering of DRAMs: Row Hammer with Crosshair". In: *ACM MEMSYS*. 2016, 471.

[104] Brent Keeth, R Jacob Baker, Brian Johnson, and Feng Lin. *DRAM Circuit Design: Fundamental and High-Speed Topics*. Vol. 13. John Wiley & Sons, 2007.

[105] Akira Kotabe. "Low-Power DRAM". In: *Green Computing with Emerging Memory*. Ed. by Takayuki Kawahara and Hiroyuki Mizuno. New York, NY: Springer New York, 2013, 87.

[106] Kibong Koo, Sunghwa Ok, Yonggu Kang, Seungbong Kim, Choungki Song, Hyeyoung Lee, Hyungsoo Kim, Yongmi Kim, Jeonghun Lee, Seunghan Oak, Yosep Lee, Jungyu Lee, Joongho Lee, Hyungyu Lee, Jaemin Jang, Jongho Jung, Byeongchan Choi, Yongju Kim, Youngdo Hur, Yunsaing Kim, Byongtae Chung, and Yongtak Kim. "A 1.2V 38nm 2.4Gb/s/Pin 2Gb DDR4 SDRAM with Bank Group and ×4 Half-Page Architecture". In: *IEEE ISSCC*. 2012, 40.

[107] Masayuki Nakamura, Tugio Takahashi, Takesada Akiba, Goro Kitsukawa, Makoto Morino, Toshihiro Sekiguchi, Isamu Asano, Katsuo Komatsuzaki, Yoshitaka Tadaki, Songsu Cho, et al. "A 29-Ns 64-Mb DRAM with Hierarchical Array Architecture". In: *IEEE Journal of Solid-State Circuits* 31.9 (1996), 1302.

[108] Tae-Young Oh, Hoeju Chung, Jun-Young Park, Ki-Won Lee, Seunghoon Oh, Su-Yeon Doo, Hyoung-Joo Kim, ChangYong Lee, Hye-Ran Kim, Jong-Ho Lee, et al. "A 3.2 Gbps/Pin 8 Gbit 1.0 V LPDDR4 SDRAM with Integrated ECC Engine for Sub-1 V DRAM Core Operation". In: *IEEE Journal of Solid-State Circuits* 50.1 (2014), 178.

[109] TechInsights Inc. *SK Hynix 21 Nm DRAM Cell Technology: Comparison of 1st and 2nd Generation*. 2017.

[110] Vivek Seshadri, Yoongu Kim, Chris Fallin, Donghyuk Lee, Rachata Ausavarungnirun, Gennady Pekhimenko, Yixin Luo, Onur Mutlu, Phillip B Gibbons, Michael A Kozuch, et al. "RowClone: Fast and Energy-Efficient in-DRAM Bulk Data Copy and Initialization". In: *IEEE/ACM MICRO*. 2013, 185.

[111] Tsugio Takahashi, Tomonori Sekiguchi, Riichiro Takemura, Seiji Narui, Hiroki Fujisawa, Shinichi Miyatake, Makoto Morino, Koji Arai, Satoru Yamada, Shoji Shukuri, et al. "A Multigigabit DRAM Technology with 6F/Sup 2/Open-Bitline Cell, Distributed Overdriven Sensing, and Stacked-Flash Fuse". In: *IEEE Journal of Solid-State Circuits* 36.11 (2001), 1721.

[112] Jyi-Tsong Lin and Cheng-Chih Hsu. "An Initial Overdriven Sense Amplifier for Low Voltage DRAMS". In: *IEEE ICCDCS*. IEEE. 2000, C31.

[113] AntMicro. *LiteX Rowhammer Tester*. 2022.

[114] *MemTest86 - Official Site of the X86 Memory Testing Tool*.

[115] JEDEC. *DDR4 SDRAM UDIMM Design Specification*. 2019.

[116] JEDEC. *DDR4 SDRAM SODIMM Design Specification*. 2019.

[117] JEDEC. *DDR3 SDRAM Unbuffered DIMM Design Specification*. 2021.

[118] JEDEC. *SPD Annex L: Serial Presence Detect (SPD) for DDR4 SDRAM Modules, Release 6*. 2022.

[119] JEDEC. *JESD4005A*. 2022.

[120] Analog Devices Inc. *LTspice Simulator*. 2022.

[121] Hasan Hassan, Gennady Pekhimenko, Nandita Vijaykumar, Vivek Seshadri, Donghyuk Lee, Oguz Ergin, and Onur Mutlu. "ChargeCache: Reducing DRAM Latency by Exploiting Row Access Locality". In: *IEEE HPCA*. IEEE. 2016, 581.

[122] Chuxiong Lin, Weifeng He, Yanan Sun, Zhigang Mao, and Mingoo Seok. "CDAR-DRAM: An in-Situ Charge Detection and Adaptive Data Restoration DRAM Architecture for Performance and Energy Efficiency Improvement". In: *ACM/IEEE DAC*. IEEE. 2021, 1093.

[123] Yaohua Wang, Lois Orosa, Xiangjun Peng, Yang Guo, Saugata Ghose, Minesh Patel, Jeremie S Kim, Juan Gómez Luna, Mohammad Sadrosadati, Nika Mansouri Ghiasi, et al. "Figaro: Improving System Performance via Fine-Grained in-Dram Data Relocation and Caching". In: *IEEE/ACM MICRO*. IEEE. 2020, 313.

[124] Vivek Seshadri, Donghyuk Lee, Thomas Mullins, Hasan Hassan, Amirali Boroumand, Jeremie Kim, Michael A Kozuch, Onur Mutlu, Phillip B Gibbons, and Todd C Mowry. "Ambit: In-memory Accelerator for Bulk Bitwise Operations Using Commodity DRAM Technology". In: *IEEE/ACM MICRO*. IEEE. 2017, 273.

[125] Nanoscale Integration and ASU Modeling (NIMO) Group. *Predictive Technology Model (PTM)*. 2022.

[126] Kevin Loughlin, Ian Neal, Jiacheng Ma, Elisa Tsai, Ofir Weisse, Satish Narayanasamy, and Baris Kasikci. "{DOLMA: Securing Speculation with the Principle of Transient {non-Observability". In: *USENIX Security*. 2021, 1397.

[127] Stefan Saroiu and Alec Wolman. "How to Configure Row-Sampling-Based Rowhammer Defenses". In: *DRAMSec*. 2022.

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

[129] Muhammad Mohsin Ghaffar, Chirag Sudarshan, Christian Weis, Matthias Jung, and Norbert Wehn. "A low power in-DRAM architecture for quantized CNNs using fast Winograd convolutions". In: *The International Symposium on Memory Systems*. 2020, 158.

[130] Fujun Bai, Song Wang, Xuerong Jia, Yixin Guo, Bing Yu, Hang Wang, Cong Lai, Qiwei Ren, and Hongbin Sun. "A Low-Cost Reduced-Latency DRAM Architecture With Dynamic Reconfiguration of Row Decoder". In: *IEEE Transactions on Very Large Scale Integration (VLSI) Systems* 31.1 (2022), 128.

[131] Gian Singh, Ankit Wagle, Sunil Khatri, and Sarma Vrudhula. "Cidan-xe: Computing in dram with artificial neurons". In: *Frontiers in Electronics* 3 (2022), 834146.

[132] Lavanya Subramanian, Kaushik Vaidyanathan, Anant Nori, Sreenivas Subramoney, Tanay Karnik, and Hong Wang. "Closed yet open dram: achieving low latency and high performance in dram memory systems". In: *Proceedings of the 55th Annual Design Automation Conference*. 2018, 1.

[133] Lois Orosa, Yaohua Wang, Mohammad Sadrosadati, Jeremie S. Kim, Minesh Patel, Ivan Puddu, Haocong Luo, Kaveh Razavi, Juan Gómez-Luna, Hasan Hassan, Nika Mansouri-Ghiasi, Saugata Ghose, and Onur Mutlu. "Codic: A low-cost substrate for enabling custom in-dram functionalities and optimizations". In: *2021 ACM/IEEE 48th Annual International Symposium on Computer Architecture (ISCA)*. IEEE. 2021, 484.

[134] Xin Xin, Youtao Zhang, and Jun Yang. "ELP2IM: Efficient and low power bitwise operation processing in DRAM". In: *2020 IEEE International Symposium on High Performance Computer Architecture (HPCA)*. IEEE. 2020, 303.

[135] Shaahin Angizi and Deliang Fan. "Graphide: A graph processing accelerator leveraging in-dram-computing". In: *Proceedings of the 2019 on Great Lakes Symposium on VLSI*. 2019, 45.

[136] Kevin K Chang, Prashant J Nair, Donghyuk Lee, Saugata Ghose, Moinuddin K Qureshi, and Onur Mutlu. "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)*. IEEE. 2016, 568.

[137] Jungwhan Choi, Wongyu Shin, Jaemin Jang, Jinwoong Suh, Yongkee Kwon, Youngsuk Moon, and Lee-Sup Kim. "Multiple clone row DRAM: A low latency and area optimized DRAM". In: *ACM SIGARCH Computer Architecture News* 43.3S (2015), 223.

[138] Sourjya Roy, Mustafa Ali, and Anand Raghunathan. "PIM-DRAM: Accelerating machine learning workloads using processing in commodity DRAM". In: *IEEE Journal on Emerging and Selected Topics in Circuits and Systems* 11.4 (2021), 701.

[139] João Dinis Ferreira, Gabriel Falcao, Juan Gómez-Luna, Mohammed Alser, Lois Orosa, Mohammad Sadrosadati, Jeremie S. Kim, Geraldo F. Oliveira, Taha Shahroodi, Anant Nori, and Onur Mutlu. "pluto: Enabling massively parallel computation in dram via lookup tables". In: *2022 55th IEEE/ACM International Symposium on Microarchitecture (MICRO)*. IEEE. 2022, 900.

[140] Young Hoon Son, O Seongil, Yuhwan Ro, Jae W Lee, and Jung Ho Ahn. "Reducing memory access latency with asymmetric DRAM bank organizations". In: *Proceedings of the 40th annual international symposium on computer architecture*. 2013, 380.

[141] O Seongil, Young Hoon Son, Nam Sung Kim, and Jung Ho Ahn. "Row-buffer decoupling: A case for low-latency DRAM microarchitecture". In: *2014 ACM/IEEE 41st International Symposium on Computer Architecture (ISCA)*. IEEE. 2014, 337.

[142] Nastaran Hajinazar, Geraldo F Oliveira, Sven Gregorio, João Dinis Ferreira, Nika Mansouri Ghiasi, Minesh Patel, Mohammed Alser, Saugata Ghose, Juan Gómez-Luna, and Onur Mutlu. "SIMDRAM: a framework for bit-serial SIMD processing using DRAM". In: *Proceedings of the 26th ACM International Conference on Architectural Support for Programming Languages and Operating Systems*. 2021, 329.

[143] Rachmad Vidya Wicaksana Putra, Muhammad Abdullah Hanif, and Muhammad Shafique. "Sparkxd: A framework for resilient and energy-efficient spiking neural network inference using approximate dram". In: *2021 58th ACM/IEEE Design Automation Conference (DAC)*. IEEE. 2021, 379.

[144] Donghyuk Lee, Yoongu Kim, Vivek Seshadri, Jamie Liu, Lavanya Subramanian, and Onur Mutlu. "Tiered-latency DRAM: A low latency and low cost DRAM architecture". In: *2013 IEEE 19th International Symposium on High Performance Computer Architecture (HPCA)*. IEEE. 2013, 615.

[145] Kevin K Chang, A Giray Yağlıkçı, Saugata Ghose, Aditya Agrawal, Niladrish Chatterjee, Abhijith Kashyap, Donghyuk Lee, Mike O'Connor, Hasan Hassan, and Onur Mutlu. "Understanding reduced-voltage operation in modern DRAM devices: Experimental characterization, analysis, and mechanisms". In: *Proceedings of the ACM on Measurement and Analysis of Computing Systems* 1.1 (2017), 1.

[146] Oghenekarho Okobiah, Saraju P Mohanty, Elias Kougianos, and Mahesh Poolakkaparambil. "Towards robust nano-CMOS sense amplifier design: a dual-threshold versus dual-oxide perspective". In: *Proceedings of the 21st edition of the great lakes symposium on Great lakes symposium on VLSI*. 2011, 145.

[147] Sanghoon Hong, Sejun Kim, Jae-Kyung Wee, and Seongsoo Lee. "Low-voltage DRAM sensing scheme with offset-cancellation sense amplifier". In: *IEEE Journal of Solid-State Circuits* 37.10 (2002), 1356.

[148] T. Kirihata, G. Mueller, M. Clinton, S. Loeffler, B. Ji, H. Terletzki, D. Hanson, Chorng-Lil Hwang, G. Lehmann, D. Storaska, G. Daniel, L. Hsu, O. Weinfurtner, T. Boehler, J. Schnell, G. Frankowsky, D. Netis, J. Ross, A. Reith, O. Kiehl, and M. Wordeman. "A 113 mm/-sup 2/600 Mb/s/pin 512 Mb DDR2 SDRAM with vertically-folded bitline architecture". In: *2001 IEEE International Solid-State Circuits Conference. Digest of Technical Papers. ISSCC (Cat. No. 01CH37177)*. IEEE. 2001, 382.

[149] Mingyu Gao, Christina Delimitrou, Dimin Niu, Krishna T Malladi, Hongzhong Zheng, Bob Brennan, and Christos Kozyrakis. "DRAF: A low-power DRAM-based reconfigurable acceleration fabric". In: *ACM SIGARCH Computer Architecture News* 44.3 (2016), 506.

[150] Akira Kotabe, Yoshimitsu Yanagawa, Satoru Akiyama, and Tomonori Sekiguchi. "0.5-V Low-VT CMOS Preamplifier for Low-Power and High-Speed Gigabit-DRAM Arrays". In: *IEEE journal of solid-state circuits* 45.11 (2010), 2348.

[151] Travis N Blalock and RC Jaeger. "A subnanosecond clamped-bit-line sense amplifier for 1T dynamic RAMs". In: *1991 International Symposium on VLSI Technology, Systems, and Applications*. IEEE Computer Society. 1991, 82.

[152] Sherif M Sharroush. "A predischarged bitline 1T-1C DRAM readout scheme". In: *Microelectronics Journal* 83 (2019), 168.

[153] Choongkeun Lee, Taegun Yim, and Hongil Yoon. "Bit-line Sense Amplifier Using PMOS Charge Transfer Pre-amplifier for Low-Voltage DRAM". In: *TENCON 2018-2018 IEEE Region 10 Conference*. IEEE. 2018, 1357.

[154] Jae-Yoon Sim. "Circuit design of DRAM for mobile generation". In: *Journal of Semiconductor Technology and Science* 7.1 (2007), 1.

[155] Choongkeun Lee and Hongil Yoon. "Highly robust and sensitive charge transfer sense amplifier for ultra-low voltage DRAMs". In: *Fifth Asia Symposium on Quality Electronic Design (ASQED 2013)*. IEEE. 2013, 227.

[156] Suk Min Kim, Byungkyu Song, and Seong-Ook Jung. "Imbalance-tolerant bit-line sense amplifier for dummy-less open bit-line scheme in dram". In: *IEEE Transactions on Circuits and Systems I: Regular Papers* 68.6 (2021), 2546.

[157] Hongil Yoon, Jae Yoon Sim, Hyun Suk Lee, Kyu Nam Lim, Jae Young Lee, Nam Jong Kim, Keum Yong Kim, Sang Man Byun, Won Suk Yang, Chang Hyun Choi, Hong Sik Jeong, Jel Hwan Yoo, Dong Il Seo, Kinam Kim, Byung Il Ryu, and Chang Gyu Hwang. "A 4 Gb DDR SDRAM with gain-controlled pre-sensing and reference bitline calibration schemes in the twisted open bitline architecture". In: *2001 IEEE International Solid-State Circuits Conference. Digest of Technical Papers. ISSCC (Cat. No. 01CH37177)*. IEEE. 2001, 378.

[158] Kyu-Nam Lim, Woong-Ju Jang, Hyung-Sik Won, Kang-Yeol Lee, Hyungsoo Kim, Dong-Whee Kim, Mi-Hyun Cho, Seung-Lo Kim, Jong-Ho Kang, Keun-Woo Park, and Byung-Tae Jeong. "A 1.2 V 23nm 6F 2 4Gb DDR3 SDRAM with local-bitline sense amplifier, hybrid LIO sense amplifier and dummy-less array architecture". In: *2012 IEEE International Solid-State Circuits Conference*. IEEE. 2012, 42.

[159] Hee-Bok Kang, Suk-Kyoung Hong, Heon-Yong Chang, Hae-Chan Park, Nam-Kyun Park, Man Young Sung, Jin-Hong Ahn, and Sung-Joo Hong. "A sense amplifier scheme with offset cancellation for giga-bit DRAM". In: *Journal of Semiconductor Technology and Science* 7.2 (2007), 67.

[160] Myoung Jin Lee. "A sensing noise compensation bit line sense amplifier for low voltage applications". In: *IEEE journal of solid-state circuits* 46.3 (2011), 690.

[161] Yohji Watanabe, Nobuo Nakamura, and Shigeyoshi Watanabe. "Offset compensating bit-line sensing scheme for high density DRAM's". In: *IEEE journal of solid-state circuits* 29.1 (1994), 9.

[162] Jinyeong Moon and Byongtae Chung. "Sense amplifier with offset mismatch calibration for sub 1-V DRAM core operation". In: *2010 IEEE International Symposium on Circuits and Systems (ISCAS)*. IEEE. 2010, 3501.

[163] Suk Min Kim, Tae Woo Oh, and Seong-Ook Jung. "Sensing voltage compensation circuit for low-power dram bit-line sense amplifier". In: *2018 International Conference on Electronics, Information, and Communication (ICEIC)*. IEEE. 2018, 1.

[164] Suk Min Kim, Byungkyu Song, and Seong-Ook Jung. "Sensing margin enhancement technique utilizing boosted reference voltage for low-voltage and high-density DRAM". In: *IEEE Transactions on Very Large Scale Integration (VLSI) Systems* 27.10 (2019), 2413.

[165] Jung-Won Sub, Kwang-Myoung Rho, Chan-Kwang Park, and Yo-Hwan Koh. "Offset-trimming bit-line sensing scheme for gigabit-scale DRAM's". In: *IEEE Journal of Solid-State Circuits* 31.7 (1996), 1025.

[166] J Park, D-H Shin, Y-H Cho, and K-W Kwon. "Inverted bit-line sense amplifier with offset-cancellation capability". In: *Electronics Letters* 52.9 (2016), 692.

[167] Alessio Spessot and Hyungrock Oh. "1T-1C dynamic random access memory status, challenges, and prospects". In: *IEEE Transactions on Electron Devices* 67.4 (2020), 1382.

[168] Jamie Liu, Ben Jaiyen, Yoongu Kim, Chris Wilkerson, and Onur Mutlu. "An experimental study of data retention behavior in modern DRAM devices: Implications for retention time profiling mechanisms". In: *ACM SIGARCH Computer Architecture News* 41.3 (2013), 60.

[169] Jeremie S Kim, Minesh Patel, Hasan Hassan, Lois Orosa, and Onur Mutlu. "D-RaNGe: Using commodity DRAM devices to generate true random numbers with low latency and high throughput". In: *2019 IEEE International Symposium on High Performance Computer Architecture (HPCA)*. IEEE. 2019, 582.

[170] Fei Gao, Georgios Tziantzioulis, and David Wentzlaff. "Computedram: In-memory compute using off-the-shelf drams". In: *Proceedings of the 52nd annual IEEE/ACM international symposium on microarchitecture*. 2019, 100.

[171] Nezam Rohbani, Sina Darabi, and Hamid Sarbazi-Azad. "PF-DRAM: a precharge-free DRAM structure". In: *2021 ACM/IEEE 48th Annual International Symposium on Computer Architecture (ISCA)*. IEEE. 2021, 126.

[172] Nezam Rohbani, Mohammad Arman Soleimani, and Hamid Sarbazi-Azad. "PIPF-DRAM: processing in precharge-free DRAM". In: *Proceedings of the 59th ACM/IEEE Design Automation Conference*. 2022, 1075.

[173] Nezam Rohbani, Mohammad Arman Soleimani, and Hamid Sarbazi-Azad. "CoolDRAM: An Energy-Efficient and Robust DRAM". In: *2023 IEEE/ACM International Symposium on Low Power Electronics and Design (ISLPED)*. IEEE. 2023, 1.

[174] Wongyu Shin, Jungwhan Choi, Jaemin Jang, Jinwoong Suh, Yongkee Kwon, Youngsuk Moon, Hongsik Kim, and Lee-Sup Kim. "Q-DRAM: Quick-access DRAM with decoupled restoring from row-activation". In: *IEEE Transactions on Computers* 65.7 (2015), 2213.

[175] Markus Kammerstetter, Markus Muellner, Daniel Burian, Christian Platzer, and Wolfgang Kastner. "Breaking integrated circuit device security through test mode silicon reverse engineering". In: *Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security*. 2014, 549.

[176] Weilie Zhou, Robert Apkarian, Zhong Lin Wang, and David Joy. "Fundamentals of scanning electron microscopy (SEM)". In: *Scanning Microscopy for Nanotechnology: Techniques and Applications* (2007), 1.

[177] Patrick Trampert, Faysal Bourghorbel, Pavel Potocek, Maurice Peemen, Christian Schlinkmann, Tim Dahmen, and Philipp Slusallek. "How should a fixed budget of dwell time be spent in scanning electron microscopy to optimize image quality?" In: *Ultramicroscopy* 191 (2018), 11.

[178] Carl Zeiss AB. *Carl Zeiss AB*. https://www.zeiss.com/. 2023.

[179] Dick James. "Recent innovations in DRAM manufacturing". In: *2010 IEEE/SEMI Advanced Semiconductor Manufacturing Conference (ASMC)*. IEEE. 2010, 264.

[180] H.J. Oh, J.Y. Kim, J.H. Kim, S.G. Park, D.H. Kim, S.E. Kim, D.S. Woo, Y.S. Lee, G.W. Ha, J.M. Park, N.J. Kang, H.J. Kim, J.S. Hwang, B.Y. Kim, D.I. Kim, Y.S. Cho, J.K. Choi, B.H. Lee, S.B. Kim, M.H. Cho, Y.I. Kim, J. Choi, D.W. Shin, M.S. Shim, W.T. Choi, G.P. Lee, Y.J. Park, W.S. Lee, and B.I. Ryu. "High-density low-power-operating DRAM device adopting 6F/sup 2/cell scheme with novel S-RCAT structure on 80nm feature size and beyond". In: *Proceedings of 35th European Solid-State Device Research Conference, 2005. ESSDERC 2005.* IEEE. 2005, 177.

[181] Thermo Fisher Scientific Inc. *Helios 5 UX DualBeam for Materials Science*. https://www.thermofisher.com/order/catalog/product/HELIOS5UX. 2023.

[182] Comet Technologies Canada Inc. *Dragonfly 2022.2*. https://www.theobjects.com/dragonfly. 2023.

[183] Tom Goldstein and Stanley Osher. "The split Bregman method for L1-regularized problems". In: *SIAM journal on imaging sciences* 2.2 (2009), 323.

[184] Antonin Chambolle. "An algorithm for total variation minimization and applications". In: *Journal of Mathematical imaging and vision* 20 (2004), 89.

[185] Thermo Fisher Scientific Inc. *Avizo Software*. https://www.thermofisher.com/ch/en/home/electron-microscopy/products/software-em-3d-vis/avizo-software.html. 2023.

[186] J. M. Park, Y. S. Hwang, S.-W. Kim, S. Y. Han, J. S. Park, J. Kim, J. W. Seo, B. S. Kim, S. H. Shin, C. H. Cho, S. W. Nam, H. S. Hong, K. P. Lee, G. Y. Jin, and E. S Jung. "20nm DRAM: A new beginning of another revolution". In: *2015 IEEE International Electron Devices Meeting (IEDM)*. IEEE. 2015, 26.

[187] Nayoung Bae, Sophie Thibaut, Toshiharu Wada, Andrew Metz, Akiteru Ko, and Peter Biolsi. "Advanced multiple patterning technologies for high density hexagonal hole arrays". In: *Advanced Etch Technology and Process Integration for Nanopatterning X*. Vol. 11615. SPIE. 2021, 24.

[188] Takeshi Sugawara, Daisuke Suzuki, Ryoichi Fujii, Shigeaki Tawa, Ryohei Hori, Mitsuru Shiozaki, and Takeshi Fujino. "Reversing stealthy dopant-level circuits". In: *Cryptographic Hardware and Embedded Systems–CHES 2014: 16th International Workshop, Busan, South Korea, September 23-26, 2014. Proceedings 16*. Springer. 2014, 112.

[189] Suk Min Kim, Byungkyu Song, Tae Woo Oh, and Seong-Ook Jung. "Analysis on sensing yield of voltage latched sense amplifier for low power DRAM". In: *2018 14th Conference on Ph. D. Research in Microelectronics and Electronics (PRIME)*. IEEE. 2018, 65.

[190] Joyce Yeung and Hamid Mahmoodi. "Robust sense amplifier design under random dopant fluctuations in nano-scale CMOS technologies". In: *2006 IEEE International SOC Conference*. IEEE. 2006, 261.

[191] Chirag Sudarshan, Lukas Steiner, Matthias Jung, Jan Lappas, Christian Weis, and Norbert Wehn. "A novel DRAM architecture for improved bandwidth utilization and latency reduction using dual-page operation". In: *IEEE Transactions on Circuits and Systems II: Express Briefs* 68.5 (2021), 1615.

[192] R Jacob Baker. *CMOS: circuit design, layout, and simulation*. John Wiley & Sons, 2019.

[193] Hoseok Seol, Wongyu Shin, Jaemin Jang, Jungwhan Choi, Jinwoong Suh, and Lee-Sup Kim. "In-dram data initialization". In: *IEEE Transactions on Very Large Scale Integration (VLSI) Systems* 25.11 (2017), 3251.

[194] Zentel Japan Corp. *Zentel Home*. https://www.zentel-japan.com/. 2023.

[195] TechInsights Inc. *Micron 1a DRAM Technology*. https://www.techinsights.com/blog/memory/micron-1a-dram-technology. 2023.

[196] Quan Deng, Lei Jiang, Youtao Zhang, Minxuan Zhang, and Jun Yang. "DrAcc: A DRAM based accelerator for accurate CNN inference". In: *Proceedings of the 55th annual design automation conference*. 2018, 1.

[197] Mustafa F Ali, Akhilesh Jaiswal, and Kaushik Roy. "In-memory low-cost bit-serial addition using commodity DRAM technology". In: *IEEE Transactions on Circuits and Systems I: Regular Papers* 67.1 (2019), 155.

[198] Louis L Hsu, Rajiv V Joshi, and Radens Carl. *Compact dual-port DRAM architecture system and method for making same*. US Patent 6,504,204. 2003.

[199] Chirag Sudarshan, Jan Lappas, Muhammad Mohsin Ghaffar, Vladimir Rybalkin, Christian Weis, Matthias Jung, and Norbert Wehn. "An in-dram neural network processing engine". In: *2019 IEEE international symposium on circuits and systems (ISCAS)*. IEEE. 2019, 1.

[200] Ytong-Bin Kim and Tom W Chen. "Assessing merged DRAM/logic technology". In: *Integration* 27.2 (1999), 179.

[201] Shih-Lien Lu, Ying-Chen Lin, and Chia-Lin Yang. "Improving DRAM latency with dynamic asymmetric subarray". In: *Proceedings of the 48th International Symposium on Microarchitecture*. 2015, 255.

[202] Hwayong Nam, Seungmin Baek, Minbok Wi, Michael Jaemin Kim, Jaehyun Park, Chihun Song, Nam Sung Kim, and Jung Ho Ahn. "X-ray: Discovering DRAM Internal Structure and Error Characteristics by Issuing Memory Commands". In: *IEEE Computer Architecture Letters* (2023).

[203] Kevin K Chang, Abhijith Kashyap, Hasan Hassan, Saugata Ghose, Kevin Hsieh, Donghyuk Lee, Tianshi Li, Gennady Pekhimenko, Samira Khan, and Onur Mutlu. "Understanding latency variation in modern DRAM chips: Experimental characterization, analysis, and optimization". In: *Proceedings of the 2016 ACM SIGMETRICS International Conference on Measurement and Modeling of Computer Science*. 2016, 323.

[204] Donghyuk Lee, Yoongu Kim, Gennady Pekhimenko, Samira Khan, Vivek Seshadri, Kevin Chang, and Onur Mutlu. "Adaptive-latency DRAM: Optimizing DRAM timing for the common-case". In: *2015 IEEE 21st International Symposium on High Performance Computer Architecture (HPCA)*. IEEE. 2015, 489.

[205] TechInsights Inc. *The Semiconductor Information Platform*. https://www.techinsights.com. 2023.

[206] TechInsights Inc. *Terms and Conditions – Content Licensing*. https://www.techinsights.com/sites/default/files/2023-09/Ts%26Cs%20-%20Content%20Licensing%202023%20-%20v1.1.pdf. 2023.

[207] Ann-Christin Bette, Patrick Brus, Gabor Balazs, Matthias Ludwig, and Alois Knoll. "Automated defect inspection in reverse engineering of integrated circuits". In: *Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision*. 2022, 1596.

[208] Ruediger Rosenkranz. "Failure localization with active and passive voltage contrast in FIB and SEM". In: *Journal of Materials Science: Materials in Electronics* 22 (2011), 1523.

[209] Adam Kimura, Jon Scholl, James Schaffranek, Matthew Sutter, Andrew Elliott, Mike Strizich, and Glen David Via. "A decomposition workflow for integrated circuit verification and validation". In: *Journal of Hardware and Systems Security* 4 (2020), 34.

[210] Bernhard Lippmann, Michael Werner, Niklas Unverricht, Aayush Singla, Peter Egger, Anja Dübotzky, Horst Gieser, Martin Rasche, Oliver Kellermann, and Helmut Graeb. "Integrated flow for reverse engineering of nanoscale technologies". In: *Proceedings of the 24th Asia and South Pacific Design Automation Conference*. 2019, 82.

[211] Bernhard Lippmann, Niklas Unverricht, Aayush Singla, Matthias Ludwig, Michael Werner, Peter Egger, Anja Duebotzky, Helmut Graeb, Horst Gieser, Martin Rasche, and Oliver Kellermann. "Verification of physical designs using an integrated reverse engineering flow for nanoscale technologies". In: *Integration* 71 (2020), 11.

[212] Matthias Ludwig, Ann-Christin Bette, and Bernhard Lippmann. "Vital: Verifying trojan-free physical layouts through hardware reverse engineering". In: *2021 IEEE Physical Assurance and Inspection of Electronics (PAINE)*. IEEE. 2021, 1.

[213] Randy Torrance and Dick James. "The state-of-the-art in IC reverse engineering". In: *International Workshop on Cryptographic Hardware and Embedded Systems*. Springer. 2009, 363.

[214] Marc Fyrbiak, Sebastian Strauß, Christian Kison, Sebastian Wallat, Malte Elson, Nikol Rummel, and Christof Paar. "Hardware reverse engineering: Overview and open challenges". In: *2017 IEEE 2nd International Verification and Security Workshop (IVSW)*. IEEE. 2017, 88.

[215] Ulbert J Botero, Ronald Wilson, Hangwei Lu, Mir Tanjidur Rahman, Mukhil A Mallaiyan, Fatemeh Ganji, Navid Asadizanjani, Mark M Tehranipoor, Damon L Woodard, and Domenic Forte. "Hardware trust and assurance through reverse engineering: A tutorial and outlook from image analysis and machine learning perspectives". In: *ACM Journal on Emerging Technologies in Computing Systems (JETC)* 17.4 (2021), 1.

[216] Franck Courbon, Jacques JA Fournier, Philippe Loubet-Moundi, and Assia Tria. "Combining image processing and laser fault injections for characterizing a hardware AES". In: *IEEE transactions on computer-aided design of integrated circuits and systems* 34.6 (2015), 928.

[217] Franck Courbon, Sergei Skorobogatov, and Christopher Woods. "Reverse engineering flash EEPROM memories using scanning electron microscopy". In: *International Conference on Smart Card Research and Advanced Applications*. Springer. 2016, 57.

[218] Arunkumar Vijayakumar, Vinay C Patil, Daniel E Holcomb, Christof Paar, and Sandip Kundu. "Physical design obfuscation of hardware: A comprehensive investigation of device and logic-level techniques". In: *IEEE Transactions on Information Forensics and Security* 12.1 (2016), 64.

[219] M Tanjidur Rahman, Qihang Shi, Shahin Tajik, Haoting Shen, Damon L Woodard, Mark Tehranipoor, and Navid Asadizanjani. "Physical inspection & attacks: New frontier in hardware security". In: *2018 IEEE 3rd International Verification and Security Workshop (IVSW)*. IEEE. 2018, 93.

[220] Christian Kison, Jürgen Frinken, and Christof Paar. "Finding the aes bits in the haystack: Reverse engineering and sca using voltage contrast". In: *Cryptographic Hardware and Embedded Systems–CHES 2015: 17th International Workshop, Saint-Malo, France, September 13-16, 2015, Proceedings 17*. Springer. 2015, 641.

[221] Mirko Holler, Manuel Guizar-Sicairos, Esther HR Tsai, Roberto Dinapoli, Elisabeth Müller, Oliver Bunk, Jörg Raabe, and Gabriel Aeppli. "High-resolution non-destructive three-dimensional imaging of integrated circuits". In: *Nature* 543.7645 (2017), 402.

[222] Dr Franck Courbon. "In-house transistors' layer reverse engineering characterization of a 45nm SoC". In: *ISTFA 2018*. ASM International. 2018, 272.

[223] Leonid Azriel, Julian Speith, Nils Albartus, Ran Ginosar, Avi Mendelson, and Christof Paar. "A survey of algorithmic methods in IC reverse engineering". In: *Journal of Cryptographic Engineering* 11.3 (2021), 299.

[224] Franck Courbon. "Practical Partial Hardware Reverse Engineering Analysis: For Local Fault Injection and Authenticity Verification". In: *Journal of Hardware and Systems Security* 4.1 (2020), 1.

[225] Sebastian Wallat, Nils Albartus, Steffen Becker, Max Hoffmann, Maik Ender, Marc Fyrbiak, Adrian Drees, Sebastian Maaßen, and Christof Paar. "Highway to HAL: open-sourcing the first extendable gate-level netlist reverse engineering framework". In: *Proceedings of the 16th ACM International Conference on Computing Frontiers*. 2019, 392.

[226] Raul Quijada, Roger Dura, Jofre Pallares, Xavier Formatje, Salvador Hidalgo, and Francisco Serra-Graells. "Large-area automated layout extraction methodology for full-ic reverse engineering". In: *Journal of Hardware and Systems Security* 2.4 (2018), 322.

[227] Hubert Kaeslin. *Top-down digital VLSI design: from architectures to gate-level circuits and FPGAs*. Morgan Kaufmann, 2014.

[228] Zemo Yang and Samiha Mourad. "Crosstalk induced fault analysis and test in DRAMs". In: *Journal of Electronic Testing* 22 (2006), 173.

[229] Yoshinobu Nakagome, M Aoki, S Ikenaga, M Horiguchi, S Kimura, Y Kawamoto, and K Itoh. "The impact of data-line interference noise on DRAM scaling". In: *IEEE Journal of Solid-state circuits* 23.5 (1988), 1120.

[230] Seyed Mohammad Seyedzadeh, Donald Kline Jr, Alex K Jones, and Rami Melhem. "Mitigating bitline crosstalk noise in dram memories". In: *Proceedings of the International Symposium on Memory Systems*. 2017, 205.

[231] Ali Fakhrzadehgan, Yale N Patt, Prashant J Nair, and Moinuddin K Qureshi. "Safeguard: Reducing the security risk from row-hammer via low-cost integrity protection". In: *2022 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*. IEEE. 2022, 373.

[232] Dayeon Kim, Hyungdong Park, Inguk Yeo, Youn Kyu Lee, Youngmin Kim, Hyung-Min Lee, and Kon-Woo Kwon. "Rowhammer Attacks in Dynamic Random-Access Memory and Defense Methods". In: *Sensors* 24.2 (2024), 592.

[233] Jonas Juffinger, Lukas Lamster, Andreas Kogler, Maria Eichlseder, Moritz Lipp, and Daniel Gruss. "CSI: Rowhammer–Cryptographic security and integrity against rowhammer". In: *2023 IEEE Symposium on Security and Privacy (SP)*. IEEE. 2023, 1702.