MCSEE: Evaluating Advanced Rowhammer Attacks and Defenses via Automated DRAM Traffic Analysis

Patrick Jattke *ETH Zurich*
Michele Marazzi *ETH Zurich*
Flavien Solt *UC Berkeley*
Max Wipfli *ETH Zurich*
Stefan Gloor *ETH Zurich*
Kaveh Razavi *ETH Zurich*

Abstract

Rowhammer attacks and defenses are constantly evolving. Recent attacks rely on hammering multiple banks or keeping rows activated for long durations. On the defense side, the DDR5 standard requires memory controllers to send Refresh Management (RFM) commands when a specific DRAM bank receives too many activations. *Are advanced Rowhammer attacks adequately exploiting their target features and do memory controllers send RFM commands adequately?*

This paper answers these questions by building an automated software platform, called *McSee*, on top of a high-frequency oscilloscope to study the behavior of DDR4 and DDR5 memory controllers under Rowhammer attacks. Leveraging a series of hardware and software optimizations, McSee is capable of reliably capturing and efficiently decoding single-cycle DDR4 and multi-cycle DDR5 traffic on the DRAM bus. We make a number of key discoveries using McSee. First, we show that hammering too many banks in parallel can actually be detrimental to the performance of Rowhammer attacks. Second, rows remain active far shorter than assumed when considering the recent Rowpress attack. Third, we show that *neither Intel nor AMD CPUs send RFM commands* even though a third of the DDR5 devices in our test pool require RFM to properly mitigate Rowhammer. Fourth, we uncover that instead of RFM, the memory controllers of Intel platforms rely on additional mitigative activations, which we characterize for the first time. We conclude by discussing the implications of our findings on the landscape of Rowhammer attacks and defenses.

1 Introduction

Rowhammer has been threatening systems security for more than a decade [1, 2], with numerous practical attacks exemplified in various scenarios [3–15]. Recent attacks rely on specific features of the underlying DDR protocol, such as parallel access to DRAM banks [15] or the possibility of keeping a DRAM row active for a long time [16]. To combat such attacks, the latest DDR5 standard requires the CPU memory controller to issue Refresh Management (RFM) commands to DRAM when certain conditions are met. Given the closed nature of memory controllers, it is unclear whether recent attacks effectively exploit the intended features or whether memory controllers correctly manage CPU-side mitigations. In this paper, we explore these aspects.

McSee. Previous work has used timing side channels to reverse engineer the bank addressing functions of memory controllers [12, 17]. Timing attacks, while conveniently launched from software, are unable to distinguish between the different commands that the memory controllers may send on the DRAM bus. As we shall soon see, this capability is crucial in understanding the effectiveness of advanced attacks and deployed mitigations. General-purpose oscilloscopes have been used in the past to reverse engineer DRAM addressing functions by probing one bit at a time [12, 17]. Studying the precise behavior of memory controllers, however, requires observing multiple command and address bits simultaneously to distinguish between specific commands and row addresses. To address this gap, we build McSee, a new software platform on top of a custom-built interposer connected to an oscilloscope, providing us with the required capabilities for automated analysis of DRAM traffic. Building McSee requires addressing several challenges related to the management, storage, and processing of the large amount of data captured on the DRAM bus. In particular, McSee features a parallel command filtering and decoding that is capable of handling both single-cycle DDR4 and multi-cycle DDR5 commands in a matter of seconds. McSee's efficient and automated analysis capabilities enabled us to investigate recent attacks and defenses.

Analyzing advanced attacks. Recent Rowhammer and Rowhammer-like attacks rely on particular features of the DDRx protocol. As an example, Sledgehammer [15] relies on the possibility of keeping multiple rows active at the same time to increase the number of banks that can be hammered in parallel. Investigating this attack with McSee, we notice that going beyond hammering six banks in parallel significantly

decreases the number of activations per bank, which is detrimental to Rowhammer attacks that require a large number of activations to decoys to bypass the in-DRAM TRR [18, 19]. Another recent Rowhammer-like attack is Rowpress, which triggers a different type of read-disturb error by keeping the DRAM rows active for long periods [16]. The system-level version of Rowpress accesses multiple columns to keep the aggressor rows activated. Using McSee, we measure — for the first time — the actual duration where the aggressor rows remain active. We find that the current implementation achieves only a fraction of what is possible according to the standard and there is significant room for better system-level Rowpress attacks in the future.

Analyzing recent mitigations. The DDR5 RFM extension requires the memory controller to keep counters for the number of activations sent to different banks. If these activations exceed a certain threshold, the memory controller is expected to send RFM commands to provide additional time for the DRAM device to perform mitigations. Our investigation of 29 DDR5 UDIMMs shows that 16 of them support RFM commands (i.e., RFM is optional) while four of them require RFM, i.e., RFM commands *must* be issued by the memory controller to ensure data integrity. Using McSee, we observe that *neither Intel nor AMD CPUs send any RFM commands* under a Rowhammer attack, even if the device requires such commands. These results are significant; fragmenting Rowhammer mitigations between CPU and DRAM creates deployment challenges that can negatively impact security. Although we do not find evidence of RFM, the commands decoded by McSee show that under a Rowhammer workload, the memory controller on Intel CPUs switches to fine-granularity refresh mode and sends additional activations that were not triggered by software. Further reverse engineering shows that these additional activations are caused by a probabilistic TRR mitigation (instead of RFM) with certain parameters that we could reverse engineer using McSee.

These findings have a number of implications for future Rowhammer attacks and defense. In particular, future reverse engineering efforts should focus on understanding the behavior of in-DRAM TRR under fine-granularity refresh instead of RFM. Future Rowhammer attacks may require tighter synchronization in their access patterns due to the increased refresh command rate, and they must consider additional CPU-triggered refreshes on top of hidden in-DRAM mitigative refreshes. Furthermore, future in-DRAM Rowhammer mitigations should be able to operate securely even when the CPUs do not always respect the standard when it comes to security features.

Contributions. We make the following contributions:

1. We present McSee, an oscilloscope-based platform for automated DRAM traffic analysis.
2. Using McSee, we show the shortcomings of recent advanced Rowhammer(-like) attacks and how they can be improved.
3. Using McSee, we find that although new DDR5 modules advertise RFM values in their SPD chips, current Intel (Alder Lake, Raptor Lake) and AMD (Zen 4) desktop CPUs do not yet send any RFM commands.
4. Using McSee, we show that the Intel DDR5 CPUs use fine-granularity refresh mode and a memory controller-based mitigation, which we reverse engineer.
5. We discuss the implications of our findings on Rowhammer attacks and defenses.

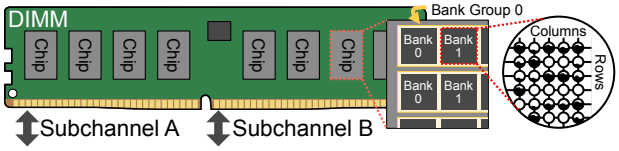

Figure 1: DDR5 DIMM architecture. DRAM chips are placed on one (single-) or both sides (dual-rank) of the DIMM. DRAM chips contain banks organized into bank groups, and each bank is comprised of rows and columns. The DIMM is split into two subchannels.

Open sourcing. The McSee platform is fully open source, including the decoder software and the PCB interposer design: https://github.com/comsec-group/mcsee.

2 Background

We explain the architecture of DRAM devices and the changes introduced with DDR5 (§2.1). We then discuss the current state of Rowhammer attacks and mitigations (§2.2).

2.1 DDR5 DRAM Devices

Today, dynamic random access memory (DRAM) is the most widely used memory in computing systems. Unbuffered dual inline memory modules (UDIMMs) are equipped with multiple DRAM *chips*. We can find these chips on either one (*single-rank*) or both sides (*dual-rank*) of the DRAM module. DIMMs are connected through *channels* to the memory controller in the CPU. Each DRAM chip consists of *banks* and *bank groups*, enabling parallel data operations for efficient access. Within each bank, a grid of *rows* and *columns* is made of capacitors to store data.

Subchannels. Compared to DDR4, the architecture of the DDR5 DIMM (Fig. 1) was redesigned to allow for a higher device density and frequency. A DDR5 DIMM is divided into two 32-bit *subchannels*. Due to the smaller bus width, certain commands now require two clock cycles instead of one (DDR4) to transmit their command and address (C/A) data, e.g., activate (`ACT`), read (`RD`), and write (`WR`).

On-die ECC. Besides rank-level ECC on most server (regis-

tered) DIMMs, the DDR5 standard mandates that all "DDR5 devices shall implement internal Single Error Correction (SEC) ECC" [20, p. 280]. This *on-die ECC* is implemented on the DRAM chip by storing parity bits along with the data and is opaque to the memory controller [21, 22].

New DRAM commands. *Refresh Management* (RFM) is a new DRAM command that gives DRAM devices more time to refresh rows during periods of high DRAM activity (e.g., Rowhammer attacks). The new same-bank `REF/RFM/PREsb` commands allow operating on a specific bank of all bank groups only, rather than targeting all banks. However, this requires that the device is in fine-granularity refresh (FGR) mode, which halves DDR5's default refresh rate (t_{REFI}) of 3.9 µs. In FGR mode, `REFsb` commands must be issued every 1.95 µs, in a round-robin fashion to all banks, followed by a `REFab` command [20, p. 172]. The memory controller can toggle the FGR mode by writing into the `MR4 OP[4]` mode register.

2.2 Rowhammer

The DRAM vulnerability *Rowhammer* [1] is a hardware fault caused by leaking capacitor charges. It allows inducing bit flips in adjacent victim DRAM rows by quickly and repeatedly activating and precharging DRAM rows. The number of activations required to trigger bit flips is known as the Rowhammer threshold, which has dropped drastically from DDR3 to the newer DDR4 devices [23]. This vulnerability can be exploited to break memory isolation and compromise system integrity, as many practical attacks have shown [3–14, 24].

In-DRAM TRR and its bypass. The original single- and double-sided [1, 3] Rowhammer patterns were rendered ineffective after in-DRAM mitigations known as *Target Row Refresh* (TRR) were introduced with DDR4 [25]. In 2020, *TRRespass* showed that increasing the number of aggressors can still trigger bit flips on roughly every third of the 40 DDR4 devices tested [25]. Later, all 40 tested DDR4 devices were shown to be vulnerable to Rowhammer by *Blacksmith* (2022) using non-uniform patterns [26].

Exploiting DRAM features for advanced attacks. Typically, Rowhammer attacks are based on interleaving fast activations of different aggressor rows. As the aggressor rows are adjacent to the victim, they reside in the same bank and their activation rate is limited by tRC (~45 ns) [27]. The tRC limit includes the minimum time that a row must be kept active (tRAS, ~32 ns) and the time to precharge the row (tRP, ~13 ns). However, in 2023, *Rowpress* showed that the Rowhammer threshold can be reduced by increasing the row open time [16]. In other words, if the aggressor rows are kept active for much longer than tRAS, fewer activations are required to induce bit flips in the victim row. Although activations that target the same bank are limited by tRC, DRAM is designed to improve performance through bank parallelism. In particular, activations targeting *different* banks only require a delay of tRRD between them (~5 ns for the same group to ~3 ns for different bank groups). *Sledgehammer* (2024) exploited this insight to issue a higher rate of activations to the DRAM device, consequently increasing the number of total bit flips [15].

DDR5 devices. *Zenhammer* [12] is the first experimental work considering DDR5 DIMMs. They showed bit flips on DDR5, albeit only one of the ten tested devices was shown to be vulnerable. The authors attribute the lack of bit flips to improved Rowhammer mitigations, on-die ECC, and a higher refresh rate.

Mitigations. A plethora of different mitigations have been proposed to stop Rowhammer attacks: software-based mitigations [5, 7, 28–30], memory controller-based mitigations [1, 31–39], and in-DRAM mitigations [18, 40–43]. Intel announced in 2014 that they would mitigate Rowhammer on servers using a memory controller-based mitigation named pseudo-TRR (pTRR) [44, 45]. In 2020, pTRR's existence was experimentally confirmed on an Intel Xeon server CPU (Sandy Bridge EP, DDR3), but it was proven absent on desktop CPUs [25]. Later, in 2021, reverse engineering efforts revealed more details about how proprietary DDR4 in-DRAM mitigations work [19, 26].

RFM. In-DRAM TRR uses the slack of the `REF` command to refresh victim rows of an attack [18, 41, 46]. As this period may be insufficient on some devices, DDR5 introduced RFM [22] to provide additional time for mitigative refreshes. For this, *rolling accumulated ACT* (RAA) counters keep track of the `ACT` count received per bank. Whenever a counter reaches the *maximum management threshold* (RAAMMT), `ACTs` to that bank are no longer allowed until an `RFM` or `REF` decreases the counter by the *initial management threshold* (RAAIMT) or 0.5–1.0 × RAAMMT, respectively. DDR5 devices must advertise the RFM values in their SPD chip, as Rowhammer mitigations may rely on it [47, 48]. Later, the *directed RFM* (DRFM) command was introduced [20] to account for the increasing *blast radius* of Rowhammer by refreshing up to four rows surrounding an aggressor row.

3 Overview

The first DDR5 bit flips in recent work [12] immediately raise the question of whether vendors use RFM — or if they configure it incorrectly, rendering it ineffective in protecting against Rowhammer. However, timing side channels alone would not allow one to distinguish whether the memory controllers are issuing RFMs, REFs, or other commands (e.g., back-to-back mitigative ACTs). Therefore, we need a platform that allows us to record and analyze what the memory controller (MC) sends to the DRAM device. Such a platform can further be leveraged to study recent attacks that utilize specific features of the DDR protocol [15, 16]. Unfortunately, it is unclear how to build such an analysis platform using

standard oscilloscopes. Existing work [12, 17, 49] that uses oscilloscopes examined only a single address bit at a time, which is insufficient to decode DDR5 commands. This raises our first **research question** (**RQ**):

RQ1. How to build a scalable analysis platform that allows us to investigate the command/address traffic from the memory controller to the DRAM device?

We present in §4 the design of *McSee*: a platform based on a general-purpose, high-speed oscilloscope that allows one to fully automate the capture and analysis of both the DDR4 and DDR5 command/address (C/A) bus traffic. McSee is backed by customized hardware and a software pipeline that we developed and optimized, including a command decoder.

Equipped with McSee, we aim to improve the understanding of recent Rowhammer(-like) attacks. In particular, these attacks rely on assumptions about the memory controller's behavior which are crucial but often not verified sufficiently or at all. Therefore, we aim to answer the question:

RQ2. Do advanced Rowhammer attacks adequately exploit the underlying DRAM features they target?

To answer this question, we validate two modern Rowhammer techniques in §5: multi-bank hammering by Sledgehammer [15] and row open time extension by Rowpress [16]. Our results show that hammering banks in parallel is only beneficial for up to six banks, after which the per-bank activation rate strongly drops. We reveal that the Rowpress system evaluation achieves row open times that are only a fraction of what the original work considers highly effective.

Next, on the defensive side, we aim to understand if RFM is advertised by DRAM devices and actually used by memory controllers of today's commodity CPUs. More precisely, we want to answer the following research questions:

RQ3. Is RFM advertised by DDR5 DRAM devices and is it supported by commodity AMD and Intel CPUs? Are there any other in-CPU mitigations in place?

Our investigations in § 6 show that the majority of our test devices report valid RFM values. However, this does not necessarily mean that the memory controllers actually respect it. To investigate this, we first reverse engineer the secret DRAM functions on recent Intel and AMD CPUs using a new systematic bit-flipping technique enabled by McSee. Using these functions, we carefully design experiments that show that *none of our test systems issues any RFM commands*.

Instead, we find additional activations to the victim rows on Intel Raptor Lake CPUs, which we show are part of a memory controller-based mitigation based on pTRR [25, 50]. This is the first report of such mitigations in consumer CPUs; previously, they were reported only on Intel server CPUs [25, 44, 45, 50]. This probabilistic mitigation makes Rowhammer attacks more difficult, as we demonstrate by reverse engineering and analyzing it. We further demonstrate that Intel CPUs, by default, employ the fine-granularity refresh mode under Rowhammer workloads, which can complicate Rowhammer attacks as refresh commands are triggered more often and are likely to alter the behavior of in-DRAM TRR mitigations. Overall, these changes seem to protect recent Intel-based systems against the current generation of Rowhammer attacks. We conclude this paper by discussing the implications of our findings on future Rowhammer attacks and defenses in §7.

Table 1: Hardware of McSee. We list all the hardware components with their exact model names.

Pos.	Component	Model Teledyne Article Name
1	Oscilloscope	SDA 806Zi-B
2	Digitizer	HDA125-18-LBUS
3	Diff. probe (8 GHz)	DH08-PB2
4	Analog probe (500 MHz)	PP021
5	18x Solder-in leads (kit)	HDA-DLS-18QL
6	DDR5 UDIMM interposer	*n/a (custom PCB)*

4 McSee

This section describes the DDR4 and DDR5 memory bus analysis platform *McSee* that we built to investigate the traffic from the memory controller to the DRAM device. To our knowledge, oscilloscopes have only been used so far in DRAM research to analyze individual signals (i.e., address bits) of the memory bus [12, 17, 49]. The McSee platform is the first open-source project that can capture DDR4/5 buses to exactly record *which DRAM commands* were issued to *which DRAM address* and *when*.

We now describe the hardware (§4.1) and then the core of our software, the DRAM decoder, which takes the traces and decodes them into DDR4 or DDR5 commands (§4.2). We then provide a more detailed description of the software components and how we optimized them (§4.3). Subsequently, we describe the validation of our platform (§4.4). We conclude by discussing the extensibility of our platform (§4.5).

4.1 Hardware Components

Before we acquired the hardware of the platform, we compared potential solutions from six different vendors (JKI, Keysight, R&S, Rigol, Tektronix, Teledyne). We did not consider the Keysight logic analyzer further due to the high price ($ 450 K) and low versatility, as we describe in §8. The Teledyne solution, which costs roughly half of the Keysight solution (i.e., $ 175 K), is the only one that provided a sufficient number of channels and a sample rate fast enough for the high speed of both DDR4 and DDR5 memory buses, which is at

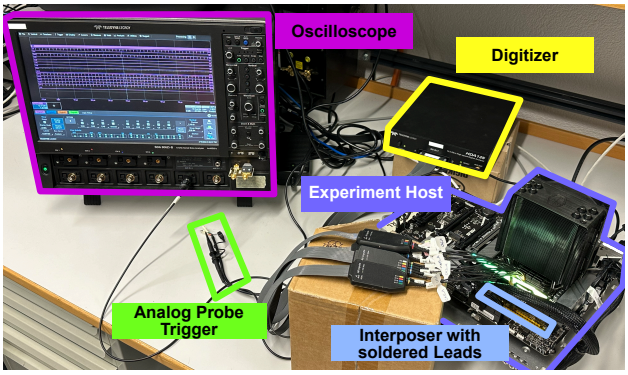

Figure 2: McSee hardware setup. The oscilloscope and digitizer are connected to the experiment host's DDR5 bus via an interposer, and an analog probe is used to trigger the recording.

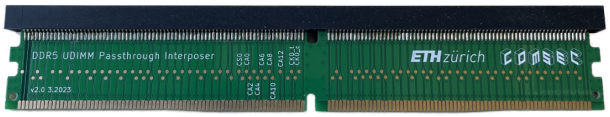

Figure 3: Our custom DDR5 passthrough interposer with soldering points in the center and labels for C/A bus signals next to them.

least 4000 MT/s. In §8, we discuss how the costs for McSee can be further reduced. The complete setup, explained in the following, is composed of the hardware components listed in Tbl. 1 and depicted in Fig. 2.

Oscilloscope & digitizer. The selected oscilloscope is a serial data analyzer (SDA), a high-speed oscilloscope that supports decoding common protocols such as USB and Ethernet. It is equipped with four analog 8 GHz channels and a serial interface (LBUS) to connect the digitizer.

The digitizer provides 18 channels with a sample rate of 12.5 GS/s each, enough to cover the command and address bits of the DDR4/5 buses. A configurable threshold voltage in the oscilloscope software defines the voltage level to be considered as logical "0" and "1". The leads connected to the digitizer are also compatible with the analog probes.

Interposer & leads. Like previous work [52–54], we use an interposer to connect the oscilloscope to the memory bus. As UDIMM interposers for oscilloscopes (e.g., DDR5-A-UDM-288 from Nexus Tech.) are expensive ($ 9'800) and off-the-shelf DIMM extenders (e.g., JET-5661AC from M-FACTORS) make soldering more challenging, we designed our own PCB (Fig. 3). Our interposer has soldering points, similar to the commercial interposer, but is much more affordable (approx. $ 100).

In Tbl. 2, we summarize the pins of the DDR5 DIMM for different DRAM commands that we capture from one of the two subchannels. For example, physical pin no. 61, which is CA bit 2 (CA2) of subchannel A, is used for row bit 0 (R0) and row bit 6 (R6) in the first and second cycle of the ACT

Table 2: Connected pins in our oscilloscope setup. We show for different DRAM commands the information provided (**Sym.**) by different pins (**Pin #**) on the command/address bus or the required voltage levels: low (▽), high (▲), or valid (𝕍). We use two columns for two-cycle commands and abbreviate bank group (BG), bank (BA), row (R), column (C), and clock (CK0).

Pin #	Sym.†	DRAM Commands							
		ACT		RD		WR		PREsb	REFsb
60	CA0	▽	R4	▲	C2	▲	𝕍	▲	▲
204	CA1	▽	R5	▽	C3	▽	C3	▲	▲
61	CA2	R0	R6	▲	C4	▲	C4	▽	▽
205	CA3	R1	R7	▲	C5	▲	C5	▲	▽
63	CA4	R2	R8	▲	C6	▽	C6	▽	▲
207	CA5	R3	R9	-	C7	-	C7	-	-
64	CA6	BA0	R10	BA0	C8	BA0	C8	BA0	BA0
208	CA7	BA1	R11	BA1	C9	BA1	C9	BA1	BA1
66	CA8	BG0	R12	BG0	C10	BG0	C10	𝕍	𝕍
210	CA9	BG1	R13	BG1	𝕍	BG1	𝕍	𝕍	▲
67	CA10	BG2	R14	BG2	▲	BG2	▲	▲	▲
211	CA11	-	R15	-	𝕍	-	-	-	-
69	CA12	-	R16	-	𝕍	-	𝕍	-	-
58	CS0	▽	▲	▽	▲	▽	▲	▽	▽
72	CK0								

† Subchannel A pins, e.g., CA0 is CA0_A in the UDIMM std. [51].

```
ts_sec    ,cmd    ,bg ,bk,row                ,col
2.209e-05,act    ,000,00,0010000000110100,
2.210e-05,ref_sb,    ,10,                    ,
2.211e-05,rd     ,000,00,                    ,001000
2.214e-05,pre_pb,000,00,                    ,
2.225e-05,ref_sb,    ,11,                    ,
```

Listing 1: Example of a decoded trace. The trace shows for each decoded DRAM command (cmd), its timestamp (ts_sec), and its address. The address depends on the command, and can include the targeted bankgroup (bg), bank (bk), row (row), or column (col).

command, respectively. Similarly, our setup on DDR4 devices allows us to capture the required command and address bits.

Probes & triggering. We use the differential probe to verify the quality of the soldered connection and to determine the digitizer's threshold voltage by measuring the signal's amplitude. To programmatically initiate data acquisition, we utilize a USB-FTDI cable connected to the experiment machine. To trigger data capture from software, we toggle the Request-to-Send (RTS) pin of the FTDI cable. Its voltage variation is detected by the oscilloscope via an analog probe and is used as a signal to start the acquisition.

4.2 DDR4 and DDR5 Command Decoder

An essential part of our software setup is the DRAM decoder, which converts CSV traces into DDR4 and DDR5 commands. As there is no open-source decoder available,

we have developed our own software from scratch, based on the DDR4/5 standards [20, 27]. Our decoder supports all 25 commands specified in the standards, including `ACT`, `RD`, `WR`, `PRE(ab|sb|pb)`, `REF(ab|sb)`, and `RFM(ab|sb)`. The decoder also extracts the full DRAM address from the traces (e.g., bank group, bank, row, column), as shown in Listing 1.

Oversampling. As sampling on the oscilloscope cannot be synchronized with the DRAM clock, we need to *oversample* the signals to capture enough information to reconstruct the DRAM commands. As a consequence, the same DRAM command may be captured in multiple (consecutive) samples. Because of this, we cannot simply match captured samples against the DDR4/5 command truth tables, as it would cause duplicate commands. Furthermore, data signals do not all change simultaneously; therefore, it is essential to determine the correct point at which the DRAM device evaluates the signals. For this, our decoder carefully analyzes the traces to determine when the signals become stable. To handle this, we divide the decoding process into three **stages (S)**: I) filtering for significant samples, II) mapping bit combinations to DRAM commands, and III) augmenting commands by address data. We explain these steps in more detail next.

S-I: Filtering for significant samples. The goal of this first step is to determine which samples are relevant to decode DRAM commands. This is necessary as the oscilloscope samples at a fixed rate (e.g., 12.5 GS/s) which is independent of the DRAM clock. Hence, this *oversampling* creates multiple identical samples if sampling is faster than the DRAM command bus, and thus leads to more data than actually needed to reconstruct the DRAM bus traffic. Using all samples in the subsequent steps would drastically increase the decoding time and lead to DRAM commands being detected multiple times. Therefore, we filter our collected data for samples s_i where the clock signal (`CK0`) changed, i.e., for which a previous sample s_{i-1} exists such that `CK0` at the previous sample s_{i-1} has a different value compared to `CK0` from the current sample s_i. As we found that the various DRAM signals do not change all at the same time, we encountered cases where the clock-changing sample s_i appeared to either (i) have *unstable* address signals that changed later, or (ii) the sample did not correspond to any DRAM command. We avoid such cases by requiring a valid DRAM command to be stable across multiple samples, i.e., $s_i = s_{i+1} = .. = s_{i+N}$. Therefore, instead of considering only the clock-changing sample, we apply a majority voting approach: we collect all samples where `CK0` is asserted and take the sample that is most common.

S-II: Mapping bit combinations to DRAM commands. In this step, we match the captured signals to the truth tables of the DDR4 [27, p. 29] and DDR5 [20, p. 103] standards, which we implemented in our decoder. As in DDR5 part of the commands require two cycles to be fully transmitted, we designed a state machine in the decoder to correctly identify DRAM commands. To distinguish between one-cycle and two-cycle commands, our decoder checks the chip select signal (`CS_n`), which is always low during the first cycle and high during the second cycle. By default, the bus uses the *2N mode* to provide larger setup and hold time margins. In this mode, the second half of a two-cycle command is transmitted two clock cycles after the first half [20, p. 276].

S-III: Augmenting DRAM commands by address data. In this final step, we augment the detected DRAM commands with their corresponding addresses, as shown in Tbl. 2. For example, from an `ACT`, we extract the bits of the bank group, bank, and row. As bits may be spread across different cycles (e.g., `R0`–`R3` in the first half and `R4`–`R16` in the second half of a `ACT` command), we need to collect and combine the bits in the correct order. Depending on the command type, the address bits are encoded in different signals. Therefore, this step occurs in parallel to Step II.

4.3 Software

Given the digitizer's sampling rate of 12.5 GS/s (i.e., 12.5×10^9 samples/second), taking multiple acquisitions quickly aggregate a very large amount of data — hundreds of megabytes per capture. As such, we need a solution that allows us to store data efficiently and in a timely manner. This represents our first **challenge (C)**:

C1. Acquired data must be stored in a space- and time-efficient way to allow long acquisition campaigns and minimize the time between consecutive acquisitions.

There are different options to process the captured traces: on the oscilloscope itself, on the experiment host, or on another (more powerful) multicore server. As neither the oscilloscope nor the experiment host is well suited to process large amounts of data, we aim to move data processing to a dedicated server. However, this requires that we can move the data quickly from the oscilloscope over the network, which represents our second challenge.

C2. Acquired data must be made available to the server quickly to enable efficient server-side data processing.

Lastly, we also envision interactive analysis sessions in addition to long-running, unattended experiment campaigns. These interactive sessions could, for example, involve tweaking the workload code before a longer-running experiment or verifying that certain changes in the experiment appear as intended in the decoded trace. Such interactive sessions become laborious if getting the decoded trace after running an experiment takes a long time. Hence, we address this requirement in our last challenge:

C3. Captured traces must be efficiently processed to minimize turnaround time for interactive analysis.

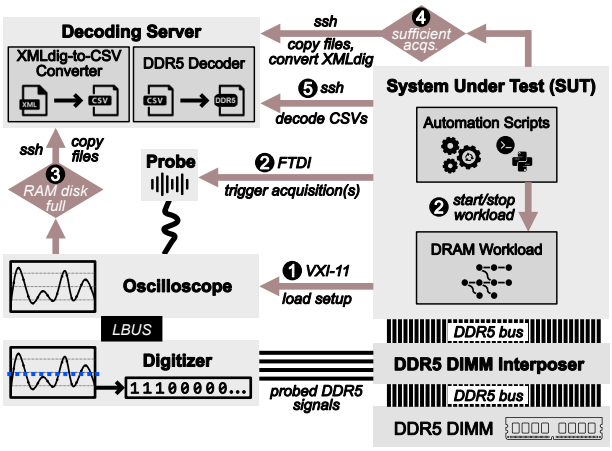

Figure 4: Data acquisition using McSee. After the oscilloscope is configured (❶), the experiment workload is executed while triggering the oscilloscope acquisition (❷). Once the RAM disk is full (❸), the data is copied to the decoding server. After acquiring sufficient traces (❹), the remaining files are copied, converted into CSV files, and decoded into DRAM commands (❺).

After giving a brief overview of how our final software setup works, we explain how we addressed the challenge **(C1–C3)** involved in building and optimizing our processing pipeline to make it around 16x faster.

Overview. To enable unattended experiments, we fully automated the execution, data transfer, and decoding of the experiments using Bash and Python scripts. The scripts run on the experiment host and perform the tasks visualized in Fig. 4:

❶ Load a configuration file in the scope containing digitizer thresholds, capture duration, and output format.

❷ Run the experiment workload while starting the capture (*triggering*) to acquire samples that are written into XMLdig [55] files on the RAM disk.

❸ Once the scope-local RAM disk is full, copy the acquired data to the decoding server through the Ethernet network.

❹ After sufficient acquisitions, copy the remaining files to the server and perform the XMLdig-to-CSV conversion.

❺ Decode the acquired data into DDR4/5 DRAM commands with their corresponding address.

We benchmarked our McSee data processing pipeline in Fig. 5 using ten traces of 2 ms each. In the following, we explain how we improved our platform to reduce the processing time from 898 seconds initially down to 55 seconds.

Efficient data storage. The oscilloscope provides an option to directly save captured traces as CSV files, which we require for our DRAM decoder. Unfortunately, writing them takes a long time (around 9 minutes) and generates large files: more than 11 GiB for ten 2 ms captures. Therefore, we use the oscilloscope's proprietary XMLdig binary format, described

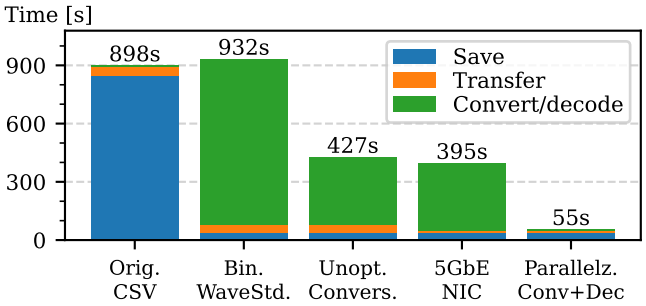

Figure 5: Performance improvements of our McSee processing pipeline. We measure the performance of our processing pipeline over ten traces of 2 ms each, repeated ten times, after each optimization that we implemented.

in its manual [55], which encodes three bytes in four ASCII characters. Writing XMLdig files is considerably faster and creates smaller files than CSVs, making it more suitable for continuous data acquisition. For example, for a 2 ms acquisition, we measured that writing XMLdig is 27x faster than directly writing CSV files while generating around 10% smaller files. However, conversion of XMLdig files with the vendor's WaveStudio[1] is slow, and in fact, made our processing pipeline less efficient (932 ms). Since conversion of XMLdig files is further not scriptable, we developed our own XMLdig converter, which reduced the processing time of the ten traces from around 932 to 432 seconds (i.e., about 54% faster).

File transfer. The oscilloscope is equipped with an integrated 1 Gbit/s Ethernet PCIe adapter, which we discovered to be the bottleneck when copying gigabytes of data to the server for processing. As the oscilloscope hardware cannot easily be upgraded without undergoing an expensive recalibration, we acquired a USB 3.1 NIC that supports up to 5 Gbit/s, which is practically around 3x faster and reduced our overall processing time to 400 seconds. As data is now only temporarily stored on the oscilloscope, we use a fast RAM disk to buffer acquisitions before transferring them to the server.

Conversion and decoding. We identified conversion and decoding as reasons for the slowdown in our pipeline. As our analysis showed that the workload is CPU-bound, we moved the conversion to a more powerful 256-core server that reduced the initial 932 seconds to 55 seconds for ten XMLdigs, corresponding to a 16x speedup. To achieve this, we process multiple trace files simultaneously and also parallelize certain steps of the pipeline, such as XMLdig-to-CSV conversion and decoding. Our highly optimized converter significantly reduces the debugging and experiment turnaround time, making McSee much more efficient and convenient to work with.

Acquisition size. We found that long acquisitions take considerably longer to process than short ones. To quantify and

[1] https://www.teledynelecroy.com/support/softwaredownload/wavestudio.aspx

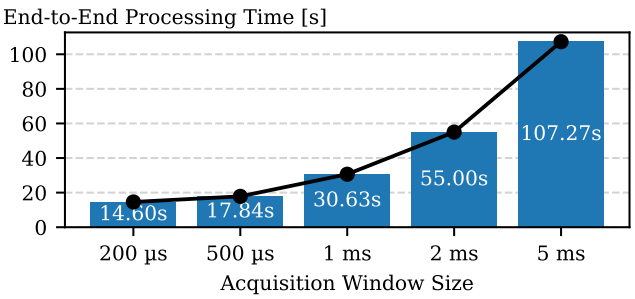

Figure 6: End-to-end processing time for different acquisition sizes over ten captured traces and averaged over ten repetitions.

compare the performance of our processing pipeline, we measured the end-to-end time for different acquisition sizes in Fig. 6. As the processing time increases non-linearly with the acquisition size, we find that 2 ms is a good trade-off between acquisition length and processing time. The results are based on DDR5 traces; however, we get similar results for DDR4 as our decoding logic is similar.

4.4 Platform and Decoder Validation

To ensure that our platform works correctly, we designed the following validity checks based on the expected device behavior derived from the DDR4/5 standards [20, 27].

Uniform address bit distribution. We read from uniformly randomly selected addresses while capturing traces using our analysis platform. After decoding, we count the frequency of all decoded DRAM address bits, e.g., how often row bit 0 (`R0`) is `0` and `1` across all traces. Due to random addresses, we expect to see both values roughly 50%; otherwise, the digitizer threshold is likely wrong, or there exists an issue with the physical (soldered) connection of the interposer.

Regular `REF` commands. We capture traces during idle system operation and expect to see `REF` commands issued regularly, on average every tREFI. If any `ACT` command has been sent to a bank before, we expect to see a `PRE(ab|sb|pb)` command before any `REF` command targeting that bank.

Complete read/write transactions. We generate memory bus traffic with MemTest86 [56]. We expect to see complete read-/write transactions consisting of `ACT`–`RD`/`WR`–`PRE` sequences. For this, we must take the address bits into account to determine which DRAM address is targeted and keep track of the state of each bank to detect missing commands. Incomplete transactions indicate that the decoder is not working correctly, in which case a manual analysis of the CSV files is required.

We will further validate our platform and decoder in §6.2, where we reverse engineer the address mappings and compare them to previous work.

4.5 Extensibility to Other DRAM Devices

Instead of a specialized analysis device, McSee is based on a general-purpose high-speed oscilloscope. This makes it possible to extend McSee to other types of DRAM (e.g., RDIMMs) or protocols (e.g., LPDDR5). As an example, we illustrate the changes that would be required to support RDIMMs.

Hardware. As the key notch alignment of RDIMMs is different from that of UDIMMs, we require a different interposer. However, there are off-the-shelf RDIMM interposers available (e.g., JET-5662AE from M-FACTORS for $ 28) that fully meet our requirements. Since RDIMMs only use seven pins for the C/A bus, we need to solder half as many pins compared to UDIMMs. However, as we discussed in §4.1, designing a custom interposer with dedicated soldering points, similar to Fig. 3, makes soldering significantly easier.

Software. We also need to adjust the decoder, as all DRAM commands are typically transmitted over more clock cycles in RDIMMs compared to UDIMMs. In particular, single-cycle commands on UDIMMs (e.g., REFab) are transmitted over two cycles on RDIMMs, and two-cycle commands (e.g., ACT) are transmitted over four cycles. This could be implemented by a logic similar to the one that is already used for the two-cycle commands on UDIMMs.

5 Analyzing Advanced Attacks

We now use McSee to investigate whether recent attacks properly exploit the underlying DRAM feature that they target. In particular, we evaluate the activation throughput in Sledgehammer's multi-bank hammering [15] (§5.1); and the row open time (t_{AggON}) in Rowpress [16] (§5.2). For the following experiments, we rely on McSee's DDR4 decoder.

5.1 Sledgehammer: Activation Throughput

Sledgehammer [15] relies on bank-level parallelism to increase the system's ACT throughput and consequently the number of Rowhammer bit flips by 7x. Their assumption is that, compared to targeting a single bank, the ACT throughput can be increased by more than 10x when activations are interleaved across different banks. We aim to validate this claim by measuring the activation throughput for both single-bank and multi-bank hammering.

Experiment. We perform our experiments on an Intel Coffee Lake (i7-8700K) system. We use the open-source implementation of Sledgehammer [57] and modify it to use the known Intel Coffee Lake DRAM addressing functions [26]. To evaluate the effectiveness of bank-level parallelism, we capture a 1 ms trace while hammering a fixed number of banks, repeating the experiment up to targeting all banks in the system. Finally, we measure the ACT throughput when targeting different numbers of banks.

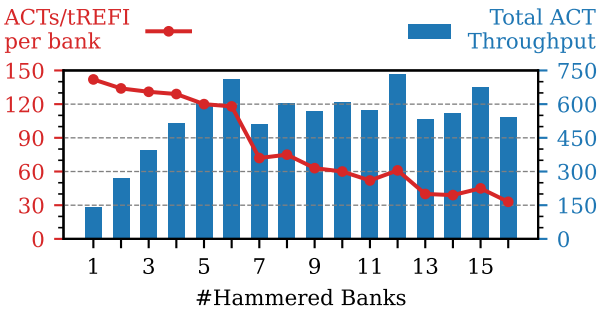

Figure 7: Sledgehammer – average activation throughput per tREFI, per bank (line) and over all banks (bar), for different numbers of hammered banks (x-axis).

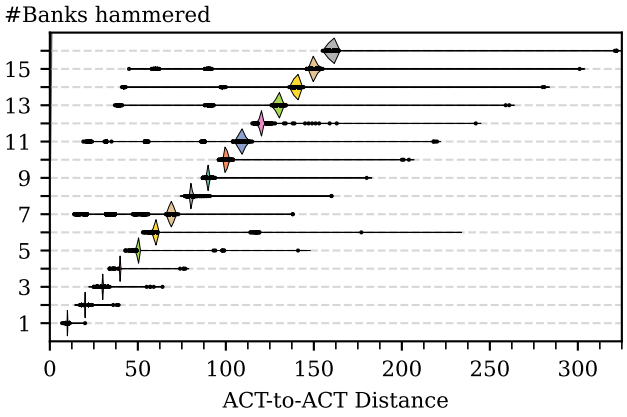

Figure 8: Sledgehammer – ACT-to-ACT distance distribution of all aggressors for different numbers of hammered banks. Some aggressors are regularly reordered, as indicated by the clusters outside the violin bodies.

Results. We present the results of our experiments in Fig. 7, where we consider ACT throughput as the number of ACTs within consecutive refresh commands (i.e., tREFI). The figure includes both the throughput per bank (line) and the total ACT throughput (bars). The results are in line with what has been reported by Sledgehammer: hammering more banks is beneficial for up to six banks. Although the total throughput steadily increases from 140 ACTs (1 bank) up to 715 ACTs (6 banks), it mostly plateaus around 500–600 ACTs when more banks are used. At the same time, the activation rate *per bank* decreases as the ACT bandwidth is divided between more banks. The ACT rate per bank drops from 140 ACTs (1 bank) to 119 ACTs (6 banks), down to 30 ACTs (16 banks).

O1. Hammering more banks is beneficial for up to six banks, after which the per-bank ACT rate strongly declines.

Reordering. The Sledgehammer authors conjecture that due to multi-bank hammering, fewer accesses get reordered. They assumed that more buffered commands reduce the opportunities for reordering. To validate this claim, we measure the ACT-to-ACT distance of the same aggressor row for different numbers of hammered banks.

Our results in Fig. 8 show that the authors' hypothesis is wrong and that reordering is happening more frequently when hammering more banks. On average, an aggressor's access is shifted 11 times when hammering one bank only, and increases to 74 times when hammering seven banks. We found that some aggressors are regularly reordered. For example, when hammering 10 aggressors on each of 7 banks (i.e., 70 aggressors in total), the ACT-to-ACT distance of some aggressors is always 20 ACTs instead of the expected 69 ACTs.

O2. Increasing the number of hammered banks from one to seven causes, on average, 6.7x more reordering of ACTs.

Implications. In conclusion, the effectiveness of Sledgehammer in inducing a higher rate of Rowhammer bit flips depends on the deployed TRR mitigation. Although attacking multiple banks in parallel leads to a higher number of total bit flips, reducing the activation rate per bank may make bypassing TRR more difficult [12]. In other words, the lower the ACT rate needed to bypass the mitigation, the more banks can be hammered in parallel, and the higher the total number of bit flips. However, if the ACTs per bank remaining for hammering are too low, the attack might fail to reach the Rowhammer threshold and not trigger any bit flips. This is probably why Sledgehammer has only been shown to be effective with up to six banks in the original paper (Figure 6) [15]. Regardless, our results confirm that multi-bank hammering is an easy and effective way to improve future Rowhammer attacks. This is particularly beneficial when patterns are near the Rowhammer threshold — where parallel row testing accelerates templating — and when the TRR mitigation is not sensitive to the order of accesses in the pattern.

5.2 Rowpress: Row Open Time

Rowpress [16] characterized a novel disturbance effect in which rows that are kept active for a long time (t_{AggON}) induce bit flips in adjacent victim rows. Rowpress allows an attacker to significantly reduce the number of activations needed to trigger a bit flip. Although the characterization is performed via an FPGA, which allows full control over the DRAM commands, the original study includes a real system evaluation of Rowpress. In that evaluation, the aggressor row is kept open by reading up to 128 cache blocks, which corresponds to the row size (128 x 64 B = 8 kB).

Using McSee, we aim to measure the aggressor t_{AggON} time while reading from the row, which could not be verified in the original study. We aim to compare the resulting row active time to what is required by Rowpress.

Experiment. We use an Intel Coffee Lake (i7-8700K) sys-

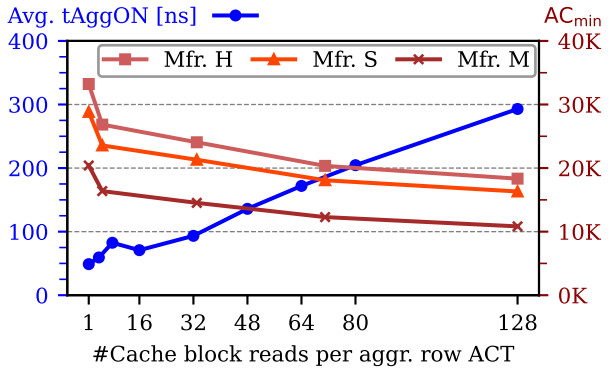

Figure 9: Rowpress – average row open time (t_{AggON}) measured over 1 ms captures for different numbers of cache block reads (y1-axis, dot), compared to AC_{min} (y2-axis), the min. number of activations to trigger the first bit flip as reported in [16] from an FPGA.

tem and the same DIMM as used in the system-level study (Samsung M378A2K43CB1-CTD) of Rowpress [58]. We verified that we can trigger bit flips on our system using the proof-of-concept Rowpress code, reproducing Figure 19 of the original work [16]. Rowpress uses 1 to 128 cache block reads to keep aggressor rows open. We measure the effective t_{AggON} resulting from each of these configurations.

Results. For the first time, we demonstrate in Fig. 9 how the average t_{AggON} of the aggressor rows varies with different numbers of cache block reads. For the resulting t_{AggON}, we report the minimum activation count (AC_{min}) to trigger a bit flip using data from the Rowpress paper [59]. As this data reports AC_{min} for specific t_{AggON} values only, we translate the t_{AggON} values using linear interpolation (of our data) to the number of cache block reads. Our results show that the average t_{AggON} time (for 128 cache block reads), for which we successfully reproduced Rowpress bit flips, is 292.95 ns. This is in the range where Rowpress shows little effect: AC_{min} only decreases by around 2x, unlike the reported 17.6x average decrease for the most effective Rowpress pattern with t_{AggON} of 7.8 µs (tREFI) in the original study.

O3. Rowpress' ability to reduce AC_{min} on commodity systems is significantly lower (2x) than for the most effective pattern (17.6x) reported in the original work.

Implications. These results show that building an effective system-level Rowpress attack is more challenging than expected. However, the results also demonstrate that there is room for improvement by increasing t_{AggON} further. This could be achieved, for example, by exploiting refresh postponement, which has recently been shown to make Rowhammer attacks more effective in bypassing TRR [60].

6 Analyzing Deployed Mitigations

The new RFM feature on DDR5 devices (§2.2) requires support from both the memory controller and the DRAM device. We investigate the RFM support of our DDR5 devices by reading out their SPD chips (§6.1). As some devices advertise RFM, we analyze their communication with the memory controller. But before that, we need to reverse engineer the secret DRAM addressing functions (§6.2), which were reported for AMD Zen 4 [12] but not for any Intel DDR5 system. Using these functions, we can precisely address the rows and check for potential RFM commands (§6.3).

6.1 RFM on DDR5 Devices

Unlike on-die ECC, RFM is not prominently advertised by any DRAM vendor, for example, in their datasheets [61–63]. Hence, it is unclear a priori whether current DDR5 devices advertise RFM support to the memory controller.

Reading SPD data. We found that existing tools such as `decode-dimms`[2] do not parse the complete SPD data from DDR5 devices. In particular, the RFM values are not supported. To address this, we obtained the raw bytes using `i2cget` and built an SPD decoder following the DDR5 SPD standard [64]. In addition to the RFM values (RAAMMT, RAAIMT, and RFM required), we also decode other fields related to the DIMM's organization (e.g., #banks, #bank groups), addressing (e.g., #row bits), and supported timings. We open source this decoder at `https://github.com/comsec-group/mcsee`.

Results. We tested 29 DDR5 devices from all three major DRAM vendors: Micron (11x), SK Hynix (10x), Samsung (7x), and an *Unknown* vendor (1x). We present the results per device in Tbl. 7, located in Appendix C.

First, we checked the device datasheets to see if on-die ECC (ODECC) is present, as mandated by JEDEC, which we confirmed for 81% of them. For 19% of the DIMMs, it is unclear as the datasheets do not mention ODECC.

O4. Almost all DDR5 devices follow JEDEC and integrate on-die ECC into the DRAM chips.

We found that 16 of 29 devices (55%) advertise valid RFM values but only four of them (14%) report that they require RFM for their device. This is our first **observation (O)**:

O5. The majority of today's off-the-shelf DDR5 devices report valid RFM values, but only a few require RFM.

Yet, the question remains: *do current DDR5-supporting Intel and AMD CPUs issue RFM commands?* To investigate this, we leverage our McSee analysis platform to analyze the DDR5 C/A bus.

[2] `https://git.kernel.org/pub/scm/utils/i2c-tools/`

Table 3: Reverse engineered address mappings and offsets of our test systems. All memory configurations are single-channel, single-DIMM, with the tuple indicating the DIMM's geometry (#ranks, #bank groups, #banks per bank group). The hex values describe the bits involved in the XOR functions. For example, 0x410000 in the Rank (RK) column means that bits 16 and 22 from the physical address are combined by XOR to calculate the rank address bit.

Sys.	Geom. #(R,G,B)	Size [GiB]	Offt. [MiB]	DRAM Addressing Functions			
				Subchan. (SC)	Rank (RK)	Bank Group (BG)	Bank Address (BA)
iAL /iRL	1, 4, 4	8	0	0x0000c3200	*n/a*	0x000081100, 0x049224000	0x092448000, 0x024910000
	1, 8, 4	16	0	0x0000c3200	*n/a*	0x000081100, 0x088844000, 0x111108000	0x222210000, 0x044420000
	2, 8, 4	32	0	0x0000c3200	0x410000	0x000081100, 0x444208000, 0x222104000	0x088820000, 0x111040000
aZ4	1, 4, 4	8	2048	0x1fffe0040	*n/a*	0x088880100, 0x111100200	0x022220400, 0x044440800
	1, 8, 4	16	2048	0x3fffc0040	*n/a*	0x084200200, 0x108401000, 0x042100100	0x210840400, 0x021080800
	2, 8, 4	32	2048	0x7fff80040	0x40000	0x084200100, 0x108400200, 0x210801000	0x421080400, 0x042100800

Table 4: Our DDR5 test systems. We report for each system the CPU microarchitecture, its release date, and model.

System	CPU (Rel. Date)	Model
iAL	**Intel Alder Lake** (11/2021)	i7-12700K
iRL	**Intel Raptor Lake** (10/2022)	i7-13700K
aZ4	**AMD Zen 4** (09/2022)	Ryzen 7 7700X

6.2 DRAM Addressing Functions

We aim to study whether the memory controllers in our three test systems (iAL, iRL, aZ4 in Tbl. 4) issue RFM commands to the DRAM devices. To verify that our McSee platform works correctly and later filter the traces more effectively, we first reverse engineer the DRAM addressing functions of these systems.

Experiment setup. Using McSee, we can precisely determine the relation between the physical address bits and the DRAM address components. Unlike earlier uses of oscilloscopes for this purpose [12, 17], which involved laborious manual probing, our setup allows us to look at all DRAM address components at the same time and is fully automated.

Systematic bit flipping. We design an experiment based on Alg. 1. In line with previous work [12,17,49,65], our approach assumes XOR-based hashing functions and the availability of 1 GiB superpages such that the least significant 30 bits between virtual and physical addresses are identical. First, we allocate as many 1 GiB superpages as possible and randomly pick an address from it. We then access the *original address* multiple times while capturing the data with the oscilloscope. After that, we systematically flip a bit and repeat the last step with the *flipped address*. For each iteration, we store the accessed physical address with the captured traces.

We repeat this process for all bits of the address and for a total of 100 different initial addresses. This takes around 5.5 hours for each system and memory configuration. By comparing the DRAM address components (e.g., bank address bits) of the original address with those of the flipped address in the decoded traces, we can clearly see if the value of any (or multiple) DRAM address component changes. If a component changes, the flipped bit is part of the addressing function of this address component. Using this approach, we can precisely reconstruct all DRAM addressing functions and also see if a physical address bit is *overlapping*, i.e., used in different DRAM functions.

Algorithm 1: Systematic bit flipping experiment. We hammer and record each possible bit flip of a randomly picked address and check for changes in the DRAM address components.

```
for addr = 0, ..., 99 do
    addr_original ← get_random_address();
    dbits_org ← hammer_rec(addr_original);
    for bit = 0, ..., 63 do
        // flip the i-th bit of the address
        addr_flipped ← addr_original ⊕ (1 ≪ bit);
        dbits_flipped ← hammer_rec(addr_flipped);
        // extract bit differences
        dbits_diff[bit] ← dbits_org ⊕ dbits_flipped;
    end
end
```

Results. The recovered DRAM addressing functions for the three platforms are presented in Tbl. 3. For the AMD Zen 4 system, we found the same functions as reported earlier [12]. Furthermore, we found that Intel Alder Lake and Raptor Lake CPUs use the same DRAM addressing functions.

O6. The DRAM addressing functions of current Intel Raptor Lake CPUs involve up to 6 bits each and bits above bit 30.

6.3 RFM on Memory Controllers

Given the DRAM addressing functions, we can precisely address specific DRAM rows, which we use to filter out noise in the captured traces more efficiently. In the next step, we investigate if memory controllers of recent DDR5 desktop CPUs, Intel Alder Lake and Raptor Lake, and also AMD Zen 4, issue RFM commands to DRAM devices.

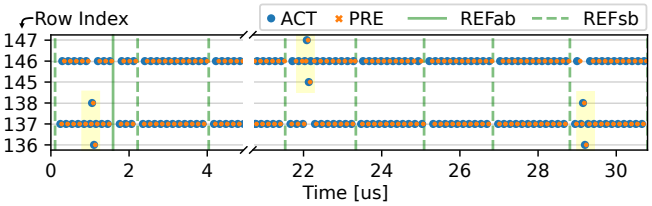

Figure 10: Activations to aggressor-adjacent rows on the Intel Alder Lake system while hammering aggressor rows 146 and 137.

Experiment setup. We use the device H_5 (SK Hynix) for this experiment, which advertises RFM support (see Tbl. 7). We then hammer two aggressor rows consecutively while capturing traces of 5 ms using McSee. We try various patterns, such as single-sided or double-sided patterns, which should all trigger RFM since it does not rely on any specific memory access pattern.

Results. We have not found any RFM commands in the captured traces of the three tested systems, i.e., `iAL`, `iRL`, and `aZ4`. We also tested another DIMM (M_3 from Micron) that sets the "RFM required" bit, but could not observe RFM commands on any of the three systems either. From this experiment, we conclude that most DDR5 DIMMs support RFM, but the memory controllers of current DDR5 CPUs do not issue RFM commands. We believe that CPU vendors might have decided against employing RFM due to requiring expensive per-bank counters in the memory controller or the performance penalty of using RFM [18].

O7. Intel Alder Lake/Raptor Lake, and AMD Zen 4 systems do not issue RFM commands to DDR5 DIMMs.

FGR Mode. Instead of RFM, we observed that both Alder Lake and Raptor Lake CPUs seem to use the fine-granularity refresh (FGR) mode by default (see §2.1). In this mode, the memory controller sends `REFsb` and `PREsb` commands to refresh and precharge one bank in all bank groups, respectively. As it halves the refresh rate (t_{REFI}) from 3.9 µs to 1.95 µs, we argue that Rowhammer attacks become harder because there is less time to hammer in between refreshes and more opportunities for TRR to refresh victims. As described in the JEDEC standard [20, p. 172], we observed that `REFab` commands are sent along with `REFsb` commands (see §2.1). We observed that while the FGR mode remains inactive on Intel CPUs during system idle states, it becomes active after a few memory accesses. On the Zen 4 system, we never observed the FGR mode.

O8. Intel Alder Lake and Raptor Lake systems use FGR mode by default. AMD Zen 4 systems do not use it.

pTRR mitigation. On the Intel Raptor Lake systems, we found that the victim rows next to our hammered aggressor rows (i.e., rows 147/146 and 138/136) were occasionally activated, as we visualize in Fig. 10. We observed this behavior whenever `ACTs` were being sent, regardless of the number of aggressor rows or their distance from each other. This behavior looks similar to the pTRR mitigation reported earlier on an Intel Xeon server CPU (Xeon E5-2620 v2, Ivy Bridge EP) with DDR3 DRAM [25]. We have not observed this behavior on the Intel Alder Lake system. We conclude from this that Intel has deployed a memory controller-based Rowhammer mitigation on their latest consumer CPUs for the first time.

O9. Intel Raptor Lake CPUs use a memory controller-based mitigation (pTRR) to protect against Rowhammer attacks.

Row remapping. We repeated the same experiment with different Micron DIMMs, which are known to employ row remapping [66]. To correctly apply the mitigation, the memory controller must be aware of the row remapping. We always observed mitigative refreshes to rows that are physically adjacent to the aggressor rows, i.e., the memory controller is aware of row remapping for particular devices from the different vendors. This is supported by our finding of "Micron row swizzling" in the pTRR-related code of the leaked Intel UEFI firmware [67]. However, we cannot exclude the possibility of further DRAM-internal row remapping.

As pTRR is a probabilistic mitigation, we aim to investigate the probability of pTRR events on the Intel Raptor Lake system in the next section.

6.4 Reverse Engineering Intel's pTRR

The memory controller-based mitigation we discovered looks similar to what has previously been reported as pseudo-TRR (pTRR) [2, 25, 44, 45]. Every time a DRAM row is accessed (i.e., activated and precharged), some adjacent row is subsequently accessed with a low probability p. As the probability p decides the mitigation's security and overhead, it is determined based on the device's vulnerability level and other protection mechanisms (e.g., on-die ECC).

Goal. To better understand the guarantees of the deployed mitigation, our objective is to experimentally reverse engineer the probability p used in the implementation. Assuming that mitigation events are stochastically independent and follow the same probability p for all events, we would expect a binomial distribution with probability p. As pTRR also affects benign workloads, we expect the probability p to be very low to minimize the overhead of mitigative refreshes.

Experiment design. In each experiment round, we hammer two aggressor rows in a loop for 8192 times while flushing (`clflush`) and fencing (`mfence`) in between hammering the aggressor rows. We choose 8192 loop rounds, as we found in preliminary experiments that it is sufficient to trigger pTRR (i.e., activations in nearby rows). We run the experiment for 512 repetitions to capture as much data as possible. After

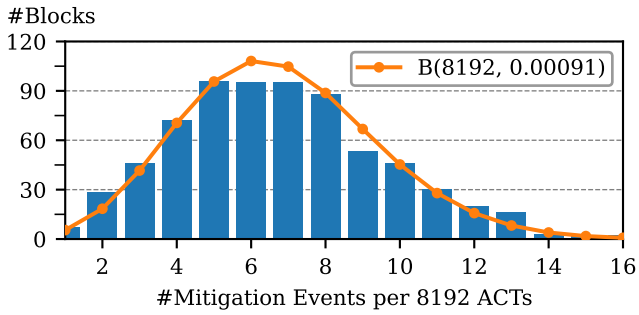

Figure 11: Distribution of Intel pTRR mitigation events on Raptor Lake while hammering the aggressor pair 8192 times, collected over four captures with each 512 repetitions. The result corresponds to a binomial distribution with a probability $p = 0.00091$ (0.091%).

each repetition, we sleep for 20 microseconds to be able to split and analyze the data more easily later.

Results. The histogram in Fig. 11 shows the frequency count (x-axis) for different numbers of pTRR mitigation events (y-axis) collected over blocks of 8192 double-sided hammer accesses. As the data fits well to a binomial distribution with $p = 0.00091$, we can conclude that the mitigation events are stochastically independent and confirm that it is indeed pTRR that we observe. We find that the probability of the distribution p is 0.00091, which means that the rows adjacent to the aggressor are refreshed with a probability of 0.091%.

We repeated the same experiment, but with hammering (consecutively) two aggressor rows of each of the four banks. Similarly to before, we found that the distribution of pTRR events fits to a binomial distribution with a probability p per bank between 0.00094 and 0.00095. This means that pTRR acts on a per-bank basis, and hammering multiple banks in parallel does make an attack time-wise more efficient, but does not reduce the probability of a victim row being mitigated.

Security Analysis. We calculate the probability that a Rowhammer attack against a pTRR-protected DIMM will succeed for different Rowhammer thresholds. We take the PARA model from previous work [68] and calculate the probability of attack success over one hour, one day, and one week. We report the success probability as a function of the Rowhammer threshold in Fig. 12. The results show that devices protected with pTRR with a Rowhammer threshold of 13 200, 16 700, and 18 800 activations can be bypassed with roughly 50% attack success probability in less than an hour, one day, and one week, respectively. From this, we conclude that pTRR alone is insufficient to protect devices in the long term due to decreasing Rowhammer thresholds [23].

Blacksmith port. We ported the state-of-the-art Blacksmith fuzzer [69] to the `iRL` system to test if it can trigger bit flips despite the presence of pTRR. We integrated the recovered DRAM addressing functions (see Tbl. 3) and adapted the refresh synchronization, which we verified using McSee. As

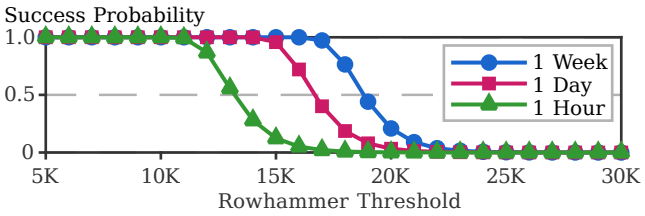

Figure 12: Rowhammer attack on pTRR. Success probability for bypassing Intel pTRR in a Rowhammer attack of one week, one day, and one hour.

Table 5: Summary of findings. Overview of detected (✓) and not detected (✗) memory controller features on our DDR5 test systems.

Feature	`iAL`	`iRL`	`aZ4`
RFM: Refresh Management	✗	✗	✗
FGR: Fine-Granularity Refresh	✓	✓	✗
pTRR: Pseudo Target Row Refresh	✗	✓	✗

we were unable to find bit flips on any of the DDR5 devices (Tbl. 7) in a 12 h fuzzer run, we conclude that pTRR is effective in mitigating state-of-the-art Rowhammer attacks. On the `aZ4` system, however, we could trigger bit flips on a DDR5 device (S_4, Micron) similar to Zenhammer [12]. As `aZ4` does not seem to employ any memory controller mitigation, we conclude that better TRR mitigations and on-die ECC inhibit us from observing bit flips on other tested DDR5 devices. We next discuss how our new findings impact the security of devices against Rowhammer.

7 Implications

We discuss the impact of our observations (§5) and findings (Tbl. 5) on the security of DDR5 systems with respect to Rowhammer. As we show, our results are crucial for the design of future Rowhammer attacks and defenses.

Attacks. Our analysis of two modern Rowhammer attacks shows that attack evaluations need to be more rigorous and consider all possible variations. For example, Sledgehammer should have reported results for all numbers of banks and activations per bank. Our results also show that such attacks have the potential to better exploit the underlying feature they target. For example, an interesting future direction is exploring system-level Rowpress attacks that rely on access patterns that keep the aggressor rows open for longer time.

RFM. While literature has proposed many RFM-based mitigations [41, 70, 71], this feature is currently not deployed on consumer CPUs. This means that there is no extra time available to mitigate Rowhammer, and TRR-based in-DRAM mitigations can only rely on REFs, thus severely impacting their security guarantees [18].

FGR. The FGR mode on the Intel systems we tested increases

Table 6: Comparison of DRAM analysis infrastructures. We compare estimated hardware costs (**HW Costs**); ease of use (**EoU**); capabilities (**Caps.**) such as viewing, manipulating, and generating memory bus traffic; and flexibility (**Flex.**). We rate aspects as ⊕ positive, ○ neutral, or ⊖ negative.

Solution	HW Costs	EoU	Caps.	Flex.
McSee (§ 4)	⊖ ($ 175 K)	⊕	view	⊕
Logic Analyzer [72, 73]	⊖ ($ 450 K)	⊕	view	○
Oscilloscope [12, 17, 49]	⊕ ($ 1 K)	○	view	○
FPGA Platform	○ ($ 5 K)[1]	○	generate	⊕
Fault Inj. Platform [54]	⊕ ($ 500)	⊖	manipulate	⊖

the refresh *rate* while reducing the duration of each REF command [22]. This can impact TRR, for example, by allowing a higher or more granular invocation frequency. However, there exists no study of TRR under FGR [19] and whether this modality is more favorable to the mitigation or the attacker. Given that previous work [26] relies on TRR synchronization to induce Rowhammer bit flips, it is fundamental for future research to consider the effect of FGR.

pTRR. Raptor Lake deploys pTRR, making it harder to trigger bit flips reliably. This impedes end-to-end Rowhammer attacks, whose success now depends on the probabilistic nature of pTRR. Hence, future work on Intel must consider *both* in-DRAM and in-CPU TRR to trigger bit flips. As pTRR cannot account for the varying Rowhammer thresholds across devices (HC_{min}), future DDR5 characterization results should be compared with our pTRR reverse engineering results.

PRAC. The latest DDR5 standard [48] contains the Rowhammer mitigation feature *Per Row Activation Counting* (PRAC) involving the memory controller. Although it promises stronger protection, our findings are disconcerting: even if there would be a principled mitigation, *there is no guarantee* that CPU vendors will support it. Once the DRAM vendors deploy PRAC, McSee is an ideal platform to study when ALERT and RFM commands are sent for an adequate security evaluation of PRAC.

8 Related Work

We compare different, previously used DRAM analysis infrastructures with McSee (§4) w.r.t. the estimated hardware costs (**HW Costs**), the ease of use (**EoU**) in terms of required expertise, the capabilities (**Caps.**) (i.e., if memory bus traffic can be viewed, manipulated, or DRAM workloads can be generated), and the platform's flexibility (**Flex.**) regarding different use cases. In Tbl. 6, we summarize our results. We now discuss these aspects in detail.

McSee. Our platform provides us with a comprehensive view of the entire command/address bus. As an oscilloscope is a general tool, it is flexible regarding captured signals, supported DRAM types, and even capturing other buses and protocols. Thanks to our custom-built software stack, we can easily capture and efficiently process DDR5 traces.

As an alternative to buying the oscilloscope used by McSee, academic groups can rent our exact setup for around $ 9 K/month. Alternatively, it is also possible to obtain a second-hand device at a cheaper price. Given that the software part of McSee is platform-agnostic, it can be used with the output from any oscilloscope or logic analyzer.

Specialized logic analyzer. A logic analyzer for DRAM analysis (e.g., Keysight U4164A) provides a detailed view of the DDRx memory bus, including the C/A and the data bus. Having an analysis SW makes it the easiest-to-use solution, requiring little expertise to set up and use. However, it is by far the most expensive solution and its flexibility is limited.

Oscilloscope. An oscilloscope can capture very few signals from the DDRx memory bus at a time, for example, for verification [12, 17]. However, this is insufficient to fully reconstruct the DRAM commands. In addition, it requires short repetitive workloads, making it impossible to detect a TRR-triggered ACT for a victim amid hundreds of regular ACTs to aggressors. Therefore, the low costs are diminished by low usability and a limited view of the memory bus. The low sampling frequency of devices in this price range (e.g., 200 MHz) restricts their use for research on high-speed DRAM.

FPGA platform. An FPGA-based memory controller allows running custom workloads with fine-grained control over DRAM commands and device behavior (e.g., refreshes) [19, 25]. It is the most versatile platform for studying DRAM chips in isolation, as it avoids all "noise" (e.g., optimizations, mitigations) caused by memory controllers in COTS CPUs. The main drawbacks are the high platform development costs and the expertise required to run and debug it. Besides that, we cannot use them to study in-CPU Rowhammer mitigations.

Fault injection platform. The mFIT [54] DDR4 platform manipulates DRAM commands *on-the-fly* by controllably faulting bits to turn one command into another. Because only a custom PCB and a microcontroller are needed, the costs are low. However, the possible command transformations are limited (since it can only pull-up/-down voltage levels), and precise synchronization with commands is missing, thus strongly reducing possible use cases. Similar limitations apply to the REFault DDR5 fault injection platform [74]. Due to the two-cycle commands in DDR5, the possible command transformations are even more limited than on DDR4.

9 Conclusion

We presented McSee, a new platform for reliable and efficient analysis of DDR4/5 traffic on the DRAM bus. Using McSee, we found that advanced Rowhammer(-like) attacks do not always exploit the underlying DRAM features that they tar-

get as intended. We also uncovered the DRAM addressing functions on Intel and AMD CPUs that support DDR5 and showed that — although new DDR5 modules advertise RFM values in their SPD chips — current CPUs do not send any RFM commands. We further discovered that Intel CPUs use fine-granularity refresh mode and Raptor Lake systems additionally employ a memory controller-based probabilistic mitigation under Rowhammer attacks, which we also reverse engineered using McSee. Finally, we explored the implications of our discoveries for practical Rowhammer attacks and defenses in the future.

Acknowledgments

We thank our anonymous reviewers and shepherd for their valuable feedback. This research was supported by the Swiss State Secretariat for Education, Research and Innovation under contract number MB22.00057 (ERC-StG PROMISE).

Appendices

A Research Ethics

This work presents a novel platform for validating DRAM-based attacks (such as Rowhammer and Rowpress), and more generally, for studying the memory controller of CPUs. We study the memory controller to analyze the presence of in-CPU Rowhammer mitigations.

The insights of our work are useful to better understand the multi-layered security of DDR5 DRAM memory in today's computing systems. However, we believe that the potential for misuse is limited due to the advanced nature of these attacks. Moreover, Rowhammer is an industry-wide known problem, and as we do not present any new attack vectors, we do not think that this work raises any ethical concerns.

B Open Science

We open source McSee's software, the oscilloscope-agnostic data processing pipeline, and our decoder on GitHub. Additionally, we provide the PCB interposer design for enabling straightforward replication. The artifacts can be found at https://github.com/comsec-group/mcsee.

C DIMM Details

In Tbl. 7, we provide details of the DDR5 UDIMMs used in our experiments, including their reported RFM values and the presence of on-die ECC (ODECC).

Table 7: Our DDR5 UDIMM testpool. We report for each device, its manufacturing date (Mf. Date) as year-month; size; frequency (Freq.); device width (Wd.); DRAM geometry as number of ranks (RK), bank groups (BG), banks per bank group (BA), and row bits (R); their RFM values (RFM) where "R" denotes reserved for future use (RFU), and if on-die ECC (ODECC) is present (✓) or it is unclear (?). Unavailable values are denoted by *n/a*.

ID	Mf. Date	Size [GiB]	Wd. [b]	Geometry #(RK,BG,BA,R)	RFM†	OD-ECC
M_1	22-05	16	x8	1, 8, 4, 16	0, 80, 6x	✓
M_2	22-08	16	x8	1, 8, 4, 16	0, R, R	?
M_3	22-01	16	x8	1, 8, 4, 16	1, 80, 4x	✓
M_4	21-10	16	x8	1, 8, 4, 16	1, 80, 4x	✓
M_5	21-10	16	x8	1, 8, 4, 16	1, 80, 4x	✓
M_6	21-10	16	x8	1, 8, 4, 16	1, 80, 4x	✓
M_7	22-02	16	x8	1, 8, 4, 16	0, R, R	?
M_8	21-12	16	x8	1, 8, 4, 16	0, R, R	✓
M_9	*n/a*	16	x8	1, 8, 4, 16	0, R, R	✓
M_{10}	21-11	16	x8	1, 8, 4, 16	0, R, R	✓
M_{11}	22-10	32	x8	2, 8, 4, 16	0, 80, 6x	✓
H_1	22-01	8	x16	1, 4, 4, 16	0, R, R	✓
H_2	22-12	8	x16	1, 4, 4, 16	0, 80, 6x	?
H_3	22-07	16	x8	1, 8, 4, 16	0, 80, 6x	✓
H_4	22-08	16	x8	1, 8, 4, 16	0, 80, 6x	✓
H_5	22-07	16	x8	1, 8, 4, 16	0, 80, 6x	✓
H_6	23-01	32	x8	2, 8, 4, 16	0, 80, 6x	✓
H_7	22-12	32	x8	2, 8, 4, 16	0, 80, 6x	?
H_8	23-01	32	x8	2, 8, 4, 16	0, 80, 6x	✓
H_9	22-08	32	x8	2, 8, 4, 16	0, 80, 6x	?
H_{10}	23-01	32	x8	2, 8, 4, 16	0, 80, 6x	?
S_1	22-02	8	x16	1, 4, 4, 16	0, 80, 6x	?
S_2	21-12	8	x16	1, 4, 4, 16	0, R, R	✓
S_3	22-01	16	x8	1, 8, 4, 16	0, R, R	✓
S_4	21-10	16	x8	1, 8, 4, 16	0, R, R	✓
S_5	*n/a*	16	x8	1, 8, 4, 16	0, R, R	✓
S_6	22-09	16	x8	1, 8, 4, 16	0, R, R	✓
S_7	23-05	16	x8	1, 8, 4, 16	0, R, R	✓
U_1	22-05	8	x16	1, 4, 4, 16	0, 80, 6x	✓

† RFM values: RFM is required, RAAIMT, RAAMMT.

References

[1] Y. Kim, R. Daly, J. Kim, C. Fallin, J. H. Lee, D. Lee, C. Wilkerson, K. Lai, and O. Mutlu, "Flipping Bits In Memory Without Accessing Them: An Experimental Study of DRAM Disturbance Errors," in *ISCA '14*. Minneapolis, MN, USA: IEEE, Jun. 2014, pp. 361–372. [Online]. Available: http://ieeexplore.ieee.org/document/6853210/

[2] K. S. Bains, J. B. Halbert, C. P. Mozak, T. Z. Schoenborn, and Z. Greenfield, "Row Hammer Refresh Command," Patent, Jun., 2012. [Online]. Available: https://patents.google.com/patent/US20140006703A1/en

[3] M. Seaborn and T. Dullien, "Project Zero: Exploiting the DRAM Rowhammer Bug to Gain Kernel Privileges," Mar. 2015. [Online]. Available: https://googleprojectzero.blogspot.com/2015/03/exploiting-dram-rowhammer-bug-to-gain.html

[4] K. Razavi, B. Gras, C. Giuffrida, E. Bosman, B. Preneel, and H. Bos, "Flip Feng Shui: Hammering a Needle in the Software Stack," in *USENIX Security '16*, 2016. [Online]. Available: https://www.usenix.org/conference/usenixsecurity16/technical-sessions/presentation/razavi

[5] V. van der Veen, Y. Fratantonio, M. Lindorfer, D. Gruss, C. Maurice, G. Vigna, H. Bos, K. Razavi, and C. Giuffrida, "Drammer: Deterministic Rowhammer Attacks on Mobile Platforms," in *CCS '16*. Vienna Austria: ACM, Oct. 2016, pp. 1675–1689. [Online]. Available: https://dl.acm.org/doi/10.1145/2976749.2978406

[6] Y. Xiao, X. Zhang, Y. Zhang, and R. Teodorescu, "One Bit Flips, One Cloud Flops: Cross-VM Row Hammer Attacks and Privilege Escalation," in *USENIX Security '16*, Austin, TX, Aug. 2016, pp. 19–35. [Online]. Available: https://www.usenix.org/conference/usenixsecurity16/technical-sessions/presentation/xiao

[7] A. Tatar, R. K. Konoth, C. Giuffrida, H. Bos, E. Athanasopoulos, and K. Razavi, "Throwhammer: Rowhammer Attacks over the Network and Defenses," in *USENIX ATC '18*, 2018, p. 14. [Online]. Available: https://www.usenix.org/conference/atc18/presentation/tatar

[8] F. de Ridder, P. Frigo, E. Vannacci, H. Bos, C. Giuffrida, and K. Razavi, "SMASH: Synchronized Many-sided Rowhammer Attacks From JavaScript," in *USENIX Security '21*, Aug. 2021. [Online]. Available: https://www.usenix.org/conference/usenixsecurity21/presentation/ridder

[9] Z. Zhang, Y. Cheng, D. Liu, S. Nepal, Z. Wang, and Y. Yarom, "PThammer: Cross-User-Kernel-Boundary Rowhammer through Implicit Accesses," in *MICRO '20*, Oct. 2020, pp. 28–41. [Online]. Available: https://ieeexplore.ieee.org/stamp/stamp.jsp?arnumber=9251982

[10] 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 '18*, May 2018, pp. 245–261. [Online]. Available: https://ieeexplore.ieee.org/stamp/stamp.jsp?arnumber=8418607

[11] M. Fahr Jr, H. Kippen, A. Kwong, T. Dang, J. Lichtinger, D. Dachman-Soled, D. Genkin, A. Nelson, R. Perlner, A. Yerukhimovich *et al.*, "When Frodo Flips: End-to-End Key Recovery on FrodoKEM via Rowhammer," in *ACM CCS '22*, 2022, pp. 979–993. [Online]. Available: https://dl.acm.org/doi/pdf/10.1145/3548606.3560673

[12] P. Jattke, M. Wipfli, F. Solt, M. Marazzi, M. Bölcskei, and K. Razavi, "ZenHammer: Rowhammer Attacks on AMD Zen-based Platforms," in *USENIX Security '24*, Aug. 2024. [Online]. Available: https://www.usenix.org/system/files/sec24fall-prepub-1050-jattke.pdf

[13] K. Mus, Y. Doröz, M. C. Tol, K. Rahman, and B. Sunar, "Jolt: Recovering TLS Signing Keys via Rowhammer Faults," in *2023 IEEE Symposium on Security and Privacy (SP)*. San Francisco, CA, USA: IEEE, May 2023, pp. 1719–1736. [Online]. Available: https://ieeexplore.ieee.org/document/10179450/

[14] A. Adiletta, C. Tol, and B. Sunar, "Leapfrog: The Rowhammer Instruction Skip Attack," *arXiv preprint arXiv:2404.07878*, 2024. [Online]. Available: https://arxiv.org/pdf/2404.07878

[15] I. Kang, W. Wang, J. Kim, S. van Schaik, Y. Tobah, D. Genkin, A. Kwong, and Y. Yarom, "SledgeHammer: Amplifying Rowhammer via Bank-level Parallelism."

[16] H. Luo, A. Olgun, A. G. Yağlıkçı, Y. C. Tuğrul, S. Rhyner, M. B. Cavlak, J. Lindegger, M. Sadrosadati, and O. Mutlu, "RowPress: Amplifying Read Disturbance in Modern DRAM Chips," in *Proceedings of the 50th Annual International Symposium on Computer Architecture*. Orlando FL USA: ACM, Jun. 2023, pp. 1–18. [Online]. Available: https://dl.acm.org/doi/10.1145/3579371.3589063

[17] P. Pessl, D. Gruss, C. Maurice, M. Schwarz, and S. Mangard, "DRAMA: Exploiting DRAM Addressing for Cross-CPU Attacks," in *USENIX Security '16*, p. 18. [Online]. Available: https://www.usenix.org/conference/usenixsecurity16/technical-sessions/presentation/pessl

[18] M. Marazzi, P. Jattke, F. Solt, and K. Razavi, "ProTRR: Principled yet Optimal In-DRAM Target Row Refresh," in *2022 IEEE Symposium on Security and Privacy (SP)*. San Francisco, CA, USA: IEEE, May 2022, pp. 735–753. [Online]. Available: https://ieeexplore.ieee.org/document/9833664/

[19] H. Hassan, Y. C. Tugrul, J. S. Kim, V. van der Veen, K. Razavi, and O. Mutlu, "Uncovering In-DRAM RowHammer Protection Mechanisms:A New Methodology, Custom RowHammer Patterns, and Implications," in *MICRO '21*. Virtual Event Greece: ACM, Oct. 2021, pp. 1198–1213. [Online]. Available: https://dl.acm.org/doi/10.1145/3466752.3480110

[20] "JESD79-5B: Double Data Rate 5 (DDR5) SDRAM," Sep. 2022. [Online]. Available: https://www.jedec.org/standards-documents/docs/jesd79-5

[21] M. Patel, J. S. Kim, T. Shahroodi, H. Hassan, and O. Mutlu, "Bit-Exact ECC Recovery (BEER): Determining DRAM On-Die ECC Functions by Exploiting DRAM Data Retention Characteristics," in *2020 53rd Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*. Athens, Greece: IEEE, Oct. 2020, pp. 282–297. [Online]. Available: https://ieeexplore.ieee.org/document/9251987/

[22] "JESD79-5A: Double Data Rate 5 (DDR5) SDRAM," Jul. 2020. [Online]. Available: https://www.jedec.org/standards-documents/docs/jesd79-5

[23] J. S. Kim, M. Patel, A. G. Yaglikci, H. Hassan, R. Azizi, L. Orosa, and O. Mutlu, "Revisiting RowHammer: An Experimental Analysis of Modern DRAM Devices and Mitigation Techniques," in *ISCA '20*. Valencia, Spain: IEEE, May 2020, pp. 638–651. [Online]. Available: https://ieeexplore.ieee.org/document/9138944/

[24] M. Marazzi and K. Razavi, "Risc-h: Rowhammer attacks on risc-v," in *4th Workshop on DRAM Security (DRAMSec) co-located with ISCA 2024*, 2024.

[25] P. Frigo, E. Vannacc, H. Hassan, V. v. der Veen, O. Mutlu, C. Giuffrida, H. Bos, and K. Razavi, "TRRespass: Exploiting the Many Sides of Target Row Refresh," in *IEEE S&P '20*, 2020, pp. 747–762. [Online]. Available: https://download.vusec.net/papers/trrespass_sp20.pdf

[26] P. Jattke, V. van der Veen, P. Frigo, S. Gunter, and K. Razavi, "BLACKSMITH: Scalable Rowhammering in the Frequency Domain," in *IEEE S&P '22*, May 2022. [Online]. Available: https://comsec.ethz.ch/wp-content/files/blacksmith_sp22.pdf

[27] "JESD79-4D: Double Data Rate 4 (DDR4) SDRAM," Jul. 2021. [Online]. Available: https://www.jedec.org/standards-documents/docs/jesd79-4a

[28] R. K. Konoth, M. Oliverio, A. Tatar, D. Andriesse, H. Bos, C. Giuffrida, and K. Razavi, "ZebRAM: Comprehensive and Compatible Software Protection Against Rowhammer Attacks," in *USENIX Security '18*, 2018, p. 15. [Online]. Available: https://www.usenix.org/conference/osdi18/presentation/konoth

[29] F. Brasser, L. Davi, D. Gens, C. Liebchen, and A.-R. Sadeghi, "CAn't touch this: Software-only mitigation against rowhammer attacks targeting kernel memory," in *USENIX Security '17*. Vancouver, BC: USENIX Association, Aug. 2017, pp. 117–130. [Online]. Available: https://www.usenix.org/conference/usenixsecurity17/technical-sessions/presentation/brasser

[30] Z. B. Aweke, S. F. Yitbarek, R. Qiao, R. Das, M. Hicks, Y. Oren, and T. Austin, "ANVIL: Software-Based Protection Against Next-Generation Rowhammer Attacks," in *ASPLOS '16*. Atlanta, Georgia, USA: ACM Press, 2016, pp. 743–755. [Online]. Available: http://dl.acm.org/citation.cfm?doid=2872362.2872390

[31] S. Sylvester, J. Sandersan, and S. R. Hasan, "Mitigation of Rowhammer Attack on DDR4 Memory: A Novel Multi-Table Frequent Element Algorithm Based Approach," in *2023 IEEE 66th International Midwest Symposium on Circuits and Systems (MWSCAS)*. Tempe, AZ, USA: IEEE, Aug. 2023, pp. 1098–1102. [Online]. Available: https://ieeexplore.ieee.org/document/10405885/

[32] M. Qureshi, A. Rohan, G. Saileshwar, and P. J. Nair, "Hydra: Enabling low-overhead mitigation of row-hammer at ultra-low thresholds via hybrid tracking," in *Proceedings of the 49th Annual International Symposium on Computer Architecture*. New York New York: ACM, Jun. 2022, pp. 699–710. [Online]. Available: https://dl.acm.org/doi/10.1145/3470496.3527421

[33] G. Saileshwar, B. Wang, M. Qureshi, and P. J. Nair, "Randomized Row-Swap: Mitigating Row Hammer by Breaking Spatial Correlation between Aggressor and Victim Rows," p. 14, 2022. [Online]. Available: https://dl.acm.org/doi/pdf/10.1145/3503222.3507716

[34] A. G. Yağlikçi, M. Patel, J. S. Kim, R. Azizi, A. Olgun, L. Orosa, H. Hassan, J. Park, K. Kanellopoulos, T. Shahroodi, S. Ghose, and O. Mutlu, "BlockHammer: Preventing RowHammer at Low Cost by Blacklisting Rapidly-Accessed DRAM Rows," in *HPCA '21*, Feb. 2021, pp. 345–358. [Online]. Available: https://ieeexplore.ieee.org/document/9407238

[35] I. Kang, E. Lee, and J. H. Ahn, "CAT-TWO: Counter-Based Adaptive Tree, Time Window Optimized for DRAM Row-Hammer Prevention," *IEEE Access*, vol. 8, pp. 17 366–17 377, 2020. [Online]. Available: https://ieeexplore.ieee.org/abstract/document/8962085

[36] Y. Park, S. N. University, W. Kwon, S. N. University, E. Lee, S. N. University, T. J. Ham, S. N. University, J. H. Ahn, S. N. University, J. W. Lee, and S. N. University, "Graphene: Strong yet Lightweight Row Hammer Protection," in *MICRO '20*, 2020, p. 13. [Online]. Available: https://ieeexplore.ieee.org/abstract/document/9251863

[37] E. Lee, I. Kang, S. Lee, G. E. Suh, and J. H. Ahn, "TWiCe: Preventing row-hammering by exploiting time window counters," in *ISCA '19*. Phoenix Arizona: ACM, Jun. 2019, pp. 385–396. [Online]. Available: https://dl.acm.org/doi/10.1145/3307650.3322232

[38] J. M. You and J. Yang, "MRLoc: Mitigating Row-hammering based on memory Locality," in *DAC '19*, Jun. 2019, pp. 1–6. [Online]. Available: https://ieeexplore.ieee.org/abstract/document/8806778

[39] S. M. Seyedzadeh, A. K. Jones, and R. Melhem, "Counter-Based Tree Structure for Row Hammering Mitigation in DRAM," *IEEE Computer Architecture Letters*, vol. 16, no. 1, pp. 18–21, Jan. 2017. [Online]. Available: https://ieeexplore.ieee.org/abstract/document/7579600

[40] M. Marazzi, F. Solt, P. Jattke, K. Takashi, and K. Razavi, "REGA: Scalable Rowhammer Mitigation with Refresh-Generating Activations," in *2023 IEEE Symposium on Security and Privacy (SP)*. San Francisco, CA, USA: IEEE, May 2023, pp. 1684–1701. [Online]. Available: https://ieeexplore.ieee.org/document/10179327/

[41] M. J. Kim, J. Park, Y. Park, W. Doh, N. Kim, T. J. Ham, J. W. Lee, and J. H. Ahn, "Mithril: Cooperative Row Hammer Protection on Commodity DRAM Leveraging Managed Refresh," *arXiv:2108.06703 [cs]*, Aug. 2021. [Online]. Available: http://arxiv.org/abs/2108.06703

[42] T. Bennett, S. Saroiu, A. Wolman, L. Cojocar, and A.-G. Llc, "Panopticon: A Complete In-DRAM Rowhammer Mitigation," p. 7, 2021. [Online]. Available: https://dramsec.ethz.ch/papers/panopticon.pdf

[43] M. Son, H. Park, J. Ahn, and S. Yoo, "Making DRAM Stronger Against Row Hammering," in *DAC '17*. Austin TX USA: ACM, Jun. 2017, pp. 1–6. [Online]. Available: https://dl.acm.org/doi/10.1145/3061639.3062281

[44] J.-B. Lee, "Green Memory Solution," 2014. [Online]. Available: http://aod.teletogether.com/sec/20140519/SAMSUNG_Investors_Forum_2014_session_1.pdf

[45] M. Kaczmarski, "Thoughts on Intel® Xeon® E5-2600 v2 Product Family Performance Optimisation – component selection guidelines," Aug. 2014.

[46] Z. Zhang, J. Qi, Y. Cheng, S. Jiang, Y. Lin, Y. Gao, S. Nepal, and Y. Zou, "A Retrospective and Futurespective of Rowhammer Attacks and Defenses on DRAM," *arXiv:2201.02986 [cs]*, Jan. 2022. [Online]. Available: http://arxiv.org/abs/2201.02986

[47] W. Kim, C. Jung, S. Yoo, D. Hong, J. Hwang, J. Yoon, O. Jung, J. Choi, S. Hyun, M. Kang, S. Lee, D. Kim, S. Ku, D. Choi, N. Joo, S. Yoon, J. Noh, B. Go, C. Kim, S. Hwang, M. Hwang, S.-M. Yi, H. Kim, S. Heo, Y. Jang, K. Jang, S. Chu, Y. Oh, K. Kim, J. Kim, S. Kim, J. Hwang, S. Park, J. Lee, I. Jeong, J. Cho, and J. Kim, "A 1.1V 16Gb DDR5 DRAM with Probabilistic-Aggressor Tracking, Refresh-Management Functionality, Per-Row Hammer Tracking, a Multi-Step Precharge, and Core-Bias Modulation for Security and Reliability Enhancement," in *2023 IEEE International Solid-State Circuits Conference (ISSCC)*, Feb. 2023, pp. 1–3. [Online]. Available: https://ieeexplore.ieee.org/abstract/document/10067805

[48] "JESD79-5C: Double Data Rate 5 (DDR5) SDRAM," Apr. 2024. [Online]. Available: https://www.jedec.org/standards-documents/docs/jesd79-5

[49] M. Schwarz, "DRAMA: Exploiting DRAM Buffers for Fun and Profit," Ph.D. dissertation, 2016. [Online]. Available: https://www.blackhat.com/docs/eu-16/materials/eu-16-Schwarz-How-Your-DRAM-Becomes-A-Security-Problem-wp.pdf

[50] M. Lipp, M. Schwarz, L. Raab, L. Lamster, M. T. Aga, C. Maurice, and D. Gruss, "Nethammer: Inducing Rowhammer Faults through Network Requests," in *EuroS&PW*, Sep. 2020, pp. 710–719. [Online]. Available: https://ieeexplore.ieee.org/document/9229701

[51] "JESD308A: DDR5 Unbuffered Dual Inline Memory Module (UDIMM) Common Standard," Jan. 2024. [Online]. Available: https://www.jedec.org/standards-documents/docs/jesd308a

[52] L. Cojocar, J. Kim, M. Patel, L. Tsai, S. Saroiu, A. Wolman, and O. Mutlu, "Are We Susceptible to Rowhammer? An End-to-End Methodology for Cloud Providers," in *2020 IEEE Symposium on Security and Privacy (SP)*. San Francisco, CA, USA: IEEE, May 2020, pp. 712–728. [Online]. Available: https://ieeexplore.ieee.org/document/9152654/

[53] W. He, Z. Zhang, Y. Cheng, W. Wang, W. Song, Y. Gao, Q. Zhang, K. Li, D. Liu, and S. Nepal, "WhistleBlower: A System-level Empirical Study on RowHammer," *IEEE Transactions on Computers*, pp. 1–15, 2023. [Online]. Available: https://ieeexplore.ieee.org/document/10014649/

[54] L. Cojocar, K. Loughlin, S. Saroiu, B. Kasikci, and A. Wolman, "mFIT: A Bump-in-the-Wire Tool for Plug-and-Play Analysis of Rowhammer Susceptibility Factors," p. 37.

[55] Teledyne Technologies Inc., "MAUI Remote Control and Automation Manual," 2023. [Online]. Available: https://cdn.teledynelecroy.com/files/manuals/maui-remote-control-and-automation-manual.pdf

[56] "MemTest86 - Official Site of the x86 Memory Testing Tool." [Online]. Available: https://www.memtest86.com

[57] I. Kang, "Mojomojo52/multibank_hammer." [Online]. Available: https://github.com/mojomojo52/multibank_hammer

[58] SAFARI, ETH Zurich, "RowPress Source Code: CMU-SAFARI/RowPress." [Online]. Available: https://github.com/CMU-SAFARI/RowPress/blob/1af25db2ebd430a79b6876d40f88c59182a207be/characterization/DRAM-Bender/sources/apps/RowPress/src/Tester.cpp

[59] H. Luo, A. Olgun, G. Yaglikci, Y. C. Tuğrul, S. Rhyner, M. B. Cavlak, J. Lindegger, M. Sadrosadati, and O. Mutlu, "Artifact of "RowPress: Amplifying Read-Disturbance in Modern DRAM Chips"," Zenodo, Mar. 2023. [Online]. Available: https://doi.org/10.5281/zenodo.7768005

[60] F. Ridder, P. Jattke, and K. Razavi, "Posthammer: Pervasive Browser-based Rowhammer Attacks with Postponed Refresh Commands," in *USENIX Security '25*. Seattle, WA, USA: USENIX Association, Aug. 2025. [Online]. Available: https://comsec.ethz.ch/research/dram/posthammer/

[61] Kingston Technology Corp., "KF556C40BB-32 - Memory Module Specifications." [Online]. Available: https://www.kingston.com/datasheets/KF552C40BB-32.pdf

[62] "Crucial 16GB DDR5-4800 UDIMM | CT16G48C40U5 | Crucial.com." [Online]. Available: https://www.crucial.com/memory/ddr5/ct16g48c40u5

[63] "M323R1GB4BB0-CQK(DDR5) | DRAM." [Online]. Available: https://semiconductor.samsung.com/dram/module/udimm/m323r1gb4bb0-cqk

[64] "JESD400-5B: DDR5 Serial Presence Detect (SPD)," Oct. 2023. [Online]. Available: https://www.jedec.org/standards-documents/docs/jesd400-5b

[65] M. Wang, Z. Zhang, Y. Cheng, and S. Nepal, "DRAMDig: A Knowledge-assisted Tool to Uncover DRAM Address Mapping," *arXiv:2004.02354 [cs]*, Jul. 2020. [Online]. Available: http://arxiv.org/abs/2004.02354

[66] L. Orosa, U. Rührmair, A. G. Yaglikci, H. Luo, A. Olgun, P. Jattke, M. Patel, J. Kim, K. Razavi, and O. Mutlu, "SpyHammer: Using RowHammer to Remotely Spy on Temperature," Oct. 2022. [Online]. Available: http://arxiv.org/abs/2210.04084

[67] P. Alcorn, "Intel's Alder Lake BIOS Source Code Reportedly Leaked Online," Oct. 2022. [Online]. Available: https://www.tomshardware.com/news/intels-alder-lake-bios-source-code-reportedly-leaked-online

[68] A. G. Yağlikçi, A. Olgun, M. Patel, H. Luo, H. Hassan, L. Orosa, O. Ergin, and O. Mutlu, "HiRA: Hidden row activation for reducing refresh latency of off-the-shelf DRAM chips," in *MICRO '22*, 2022, pp. 815–834. [Online]. Available: https://ieeexplore.ieee.org/stamp/stamp.jsp?arnumber=9923850

[69] P. Jattke, V. van der Veen, P. Frigo, S. Gunter, and K. Razavi, "GitHub: Comsec-group/blacksmith," Computer Security Group @ ETH Zurich, Apr. 2024. [Online]. Available: https://github.com/comsec-group/blacksmith

[70] A. Jaleel, S. W. Keckler, and G. Saileshwar, "Probabilistic Tracker Management Policies for Low-Cost and Scalable Rowhammer Mitigation," Apr. 2024. [Online]. Available: http://arxiv.org/abs/2404.16256

[71] M. Wi, J. Park, S. Ko, M. J. Kim, N. Sung Kim, E. Lee, and J. H. Ahn, "SHADOW: Preventing Row Hammer in DRAM with Intra-Subarray Row Shuffling," in *HPCA '23*, Feb. 2023, pp. 333–346. [Online]. Available: https://ieeexplore.ieee.org/abstract/document/10070966

[72] M. Patel, J. S. Kim, and O. Mutlu, "The Reach Profiler (REAPER): Enabling the Mitigation of DRAM Retention Failures via Profiling at Aggressive Conditions," in *Proceedings of the 44th Annual International Symposium on Computer Architecture*. Toronto ON Canada: ACM, Jun. 2017, pp. 255–268. [Online]. Available: https://dl.acm.org/doi/10.1145/3079856.3080242

[73] J. Kim, "Improving DRAM Performance, Security, and Reliability by Understanding and Exploiting DRAM Timing Parameter Margins," Thesis, Carnegie Mellon University, Dec. 2020. [Online]. Available: https://arxiv.org/pdf/2109.14520

[74] S. Gloor, P. Jattke, and K. Razavi, "Refault: A fault injection platform for rowhammer research on ddr5 memory," in *Proceedings of the Microarchitecture Security Conference*, 2025.