Microsoft Research

# mFIT: A Bump-in-the-Wire Tool for Plug-and-Play Analysis of Rowhammer Susceptibility Factors

**Lucian Cojocar, Kevin Loughlin[†], Stefan Saroiu, Baris Kasikci[†], Alec Wolman**

Microsoft, [†]University of Michigan

MSR-TR-2021-25

October 25, 2021

**Abstract**

Understanding susceptibility to Rowhammer bit flips in DRAM (i.e., main memory) is vital to ensure reliability and security on today's systems, particularly on multitenant cloud servers. However, existing tools for analyzing susceptibility either suffer from a lack of precision or are not broadly-compatible with cloud platforms.

In this work, we introduce mFIT, a low-cost tool for characterizing Rowhammer susceptibility factors. mFIT operates as a "bump-in-the-wire" between the host's memory controller and a DDR4 DRAM module, offering the ability to analyze both the host platform's and DRAM's influences on Rowhammer susceptibility. We show that mFIT offers "plug-and-play" support for analyzing the latest three generations of DDR4 server-grade DRAM modules from all three major DRAM manufacturers, using two different Intel server platforms. In addition to providing the first public evidence of worsening Rowhammer susceptibility in these modules, we show that mFIT can experimentally determine various factors' roles in Rowhammer susceptibility. Using this knowledge, we demonstrate per-module "Rowhammer-optimal" data patterns that average 60% more bit flips than prior work, offering researchers insights on how to construct stronger Rowhammer attacks and defenses.

We open-sourced mFIT's design[1] to help other researchers use our techniques to further the community's understanding of Rowhammer and DRAM.

# 1 Introduction

Rowhammer attacks [25, 18, 15, 30, 29, 21, 22, 26, 31, 34, 36, 38, 39, 41, 42, 45, 46, 47, 16] demonstrate that certain DRAM (i.e., main memory) access patterns can cause bit flips in DRAM, corrupting data and potentially causing machine crashes or even subversion. Accordingly, understanding Rowhammer susceptibility is vital to ensure reliability and security on today's systems, particularly on multitenant cloud servers (i.e., where one tenant's DRAM access patterns can affect the reliability and security of other tenants). Unfortunately, cloud providers lack a thorough understanding of the various factors affecting Rowhammer susceptibility on modern DDR4 DIMMs (i.e., DRAM modules). This lack of understanding is exacerbated by limited available tooling to precisely characterize these factors.

Existing tools suffer from one of two limitations. First, many are not broadly-compatible with commodity servers and/or server-grade DIMMs.

[1] https://github.com/microsoft/mfit

Industrial memory testers are built for standalone DRAM module testing, use their own custom-made memory controllers and busses, and are quite expensive with prices starting at hundreds of thousands of dollars [9, 8]. As an alternative, researchers have used FPGA-based memory controllers [23, 18, 29] that offer precise control of traffic issued to *client*-grade DIMMs (and thus, precise analysis capabilities), but they also remove the host platform's memory controller from the analysis and require significant programmer effort to support arbitrary DIMMs (e.g., server-grade). To the best of our knowledge, no custom memory controller offers "plug-and-play" support for an arbitrary DDR4 DIMM, nor the ability to characterize a memory subsystem's Rowhammer susceptibility in the presence of both the DIMM and the memory controller.

Second, while the remaining set of primarily software-only tools [18, 42, 43, 15, 14] enable analyses that include the host platform's memory controller, they lack the ability to precisely characterize Rowhammer susceptibility. These tools lack sufficient control of DRAM traffic, whether via lack of programmability or inability to influence specific DRAM commands. For instance, prior work [14, 30, 29] shows that a key component in characterizing Rowhammer susceptibility is preventing the memory controller from sending *refresh* (REF) commands to the DIMM, since suppressing REFs allows relevant aspects of Rowhammer susceptibility to be reliably reverse engineered. However, outside of custom memory controllers and industrial memory testers, existing tools cannot control when a REF command is issued, meaning they lack the ability to precisely characterize DRAM.

To achieve both the precision of a custom hardware approach *and* broad host compatibility across DDR4 DIMMs and platforms, we instead propose *mFIT* – a memory fault injection tool. mFIT's key innovation is operating as a "bump-in-the-wire" design between the host platform's memory controller and an arbitrary DDR4 DIMM, and can be built for just US$ 61. At a high level, mFIT suppresses REF commands for a programmable period of time by masking a signal that otherwise indicates the memory controller has issued a REF command, thereby "tricking" the attached DIMM into thinking no REFs have been issued. In doing so, mFIT allows researchers to more easily observe the properties of internal DRAM structures, while simultaneously benefiting from a commodity host platform for workload/experiment generation and analysis.

Using this REF suppression technique, we demonstrate mFIT's simple-but-effective ability to evaluate susceptibility to Rowhammer. We highlight mFIT's "plug-and-play" operation via a broad analysis across the latest three generations of server-grade DIMMs from all three major DRAM man-

ufacturers, using two different generations of Intel servers/host memory controllers. Notably, we offer the first public evidence of increasing Rowhammer susceptibility for DDR4 server-grade DIMMs (previously observed in client-grade DIMMs [29]).

We then use mFIT to advance the state-of-the-art in Rowhammer characterization, producing the first extensive experimental analysis of various Rowhammer susceptibility factors for these DIMMs—including internal DRAM row addressing/scrambling [17], subarray design [35], data dependency [30], and host data scrambling [13, 15]. We show that, using the internal system properties recovered by mFIT, one can construct malicious data patterns that average 60% more bit flips than state-of-the-art data patterns used in existing Rowhammer literature [29], offering future researchers valuable information when designing Rowhammer attacks and defenses.

To summarize, we make the following contributions:

- We design *mFIT*, a low-cost tool for Rowhammer susceptibility analysis, capable of characterizing both DDR4 DIMMs and a host memory controller's role in Rowhammer.
- We demonstrate mFIT's broad applicability and ease-of-use via an analysis of Rowhammer susceptibility in the latest three generations of server-grade DIMMs from all three major DRAM manufacturers, using two different generation of Intel servers/memory controllers.
- We provide the first public evidence of worsening Rowhammer susceptibility in DDR4 server-grade DIMMs, validating trends observed for client-grade DIMMs [29].
- We advance the state-of-the-art in Rowhammer characterization, producing the first extensive experimental analysis of various Rowhammer susceptibility factors for these DIMMs.
- Using insights obtained with mFIT, we demonstrate per-DIMM "Rowhammer-optimal" data patterns that average 60% more bit flips than state-of-the-art data patterns [29].

## 2 Background

This section provides background on memory subsystems, DRAM, and Rowhammer as necessary to understand the design and analyses of mFIT.

### 2.1 Memory Subsystems

A typical main memory subsystem contains one or more memory controllers that manage load and store requests to physical addresses (e.g., from pro-

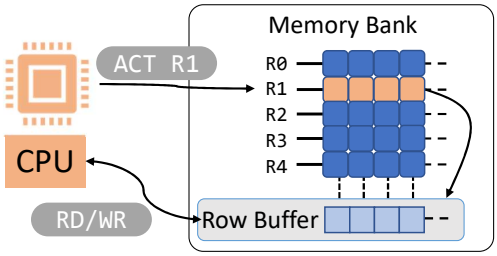


Figure 1: To access data in DRAM, the CPU's integrated memory controller needs to first activate (ACT) the row containing the data, before reading (RD) or writing (WR) at cache-line offsets within the row buffer.

cessor cores). After translating the requests into *DDR4 commands* (i.e., a communication standard between memory controllers and DRAM [1]), each memory controller is responsible for issuing the appropriate commands to one or more attached DRAM modules (e.g., client- or server-grade *DIMMs*: dual in-line memory modules). Unlike client-grade DIMMs, server-grade DIMMs support error correction codes via a wider data bus and additional circuitry; nonetheless, we note that both classes process largely the same set of commands.

### 2.2 DRAM Organization

From the memory controller's perspective (i.e., the logical DDR4 space), each DIMM consists of a set of *banks*, and each bank is a row-column array of cells. In reality, within each logical bank are a set of row-column *subarrays* of cells. Within a subarray, rows of cells are accessed via the same *word line*, while columns of cells are accessed via the same *bit line*. We note that a typical row contains thousands of DRAM cells, and each DRAM cell in the row stores 1 bit of information as charged/discharged states.

Memory controllers read/write cells by first issuing an *activate* (ACT) command to a particular row within a bank, as shown in Fig. 1. This connects the row's word line to the bank's *row buffer*, where subsequent read (RD) and write (WR) commands are performed on cells from the row at cache-line granularity offsets. When the memory controller wishes to access another row in the same bank, it must first *precharge* (PRE) the bank, closing the open row and preparing the row buffer for new data from a different row. Accordingly, only one row per bank can be open at a time.

Because DRAM cells discharge over time, the memory controller periodically issues *refresh* (REF) commands, such that the charges in each row's cells are replenished and information is retained. According to the DDR4 standard, each row should be refreshed within 64ms of its last refresh; the memory controller issues a REF command every 7.8$\mu$s on average, such that the DIMM can refresh different rows throughout the 64ms refresh window.

### 2.3 Rowhammer

Rowhammer attacks [25, 18, 15, 30, 29, 21, 22, 26, 31, 34, 36, 38, 39, 41, 42, 45, 46, 47, 16] show that frequent ACTs of the same row(s)—produced by certain memory access patterns—can corrupt data in nearby rows. This data corruption is both a reliability and security threat, as Rowhammer bit flips can result in incorrect results, machine crashes, and even system subversion.

At a high level, alternating RDs or WRs to a set of *aggressor* rows within a single bank necessitate alternating ACTs of these aggressors due to row buffer contention. However, because of electrical interference among nearby rows, these frequent ACTs may sufficiently disturb (i.e., alter) the charges in nearby *victim* rows so as to produce logical bit flips.

More precisely, each row can safely withstand a per-DIMM *maximum activation count* (MAC) of ACTs within a refresh window. However, if one or more aggressors surpass their MACs before a (potential) victim row is refreshed, the victim's data may be disturbed. We note that victim rows are those found up to $b$ rows away from an aggressor, where $b$ defines an aggressor's *blast radius* (which varies across DIMMs).

Notably, more than one aggressor row may be used in a Rowhammer attack, where an attack with $k$ aggressor rows is referred to as an $k$-sided attack. In such scenarios, the *hammer count* (HC) refers to the number of times *each* aggressor is activated.

Modern DIMMs internally mitigate a subset of Rowhammer attacks via Target Row Refresh (TRR) [18]. TRR effectively refreshes potential victim rows ahead of schedule, using the time allocated to regularly-scheduled REF commands to perform these additional, targeted refreshes. However, TRR fails to mitigate Rowhammer attacks with sufficient numbers of aggressor rows (e.g., $> 20$). As such, the underlying DRAM technology remains susceptible to Rowhammer [12, 16].

To prevent TRR from obfuscating susceptibility to Rowhammer during DIMM characterization, state-of-the-art tools [23, 14] suppress/do not issue REF commands during experimentation. REF suppression ensures that

TRR is not engaged, such that the underlying properties of the DRAM technology (e.g., data retention time) can be more easily obtained.

### 2.4 DRAM Internals

While the logical view of DRAM is sufficient to understand Rowhammer at a high level, certain aspects of mFIT's analyses require a deeper understanding of various DRAM internals. For simplicity, we provide relevant details here, and return to them later in the paper.

In reality, each logical bank of DRAM is actually split among many chips within the DIMM. Each chip stores a portion of the data used in an operation. For example, an "x4" (e.g., the format of the server-grade DIMMs analyzed in this study) DIMM means that each chip processes 4 data bits, referred to as a "nibble." The DIMM is free to "swizzle" [6] (effectively, intertwine/remap) logically-adjacent bit lines among chips.

While data transfers between the system and DIMM appear to operate at a cache-line sized granularity, they are actually performed in "bursts" according to the data bus width. For instance, the DIMM responds to a RD/WR command with 8-sequential 64-bit transfers, totaling the 512 bits of data in a 64-byte cache line.

## 3 Characterizing Rowhammer: Challenges and Goals

Given the reliability and security ramifications of Rowhammer bit flips, the research community has a vested interest in understanding Rowhammer susceptibility from both an offensive and defensive perspective. However, characterizing Rowhammer is difficult. In this section, we describe the challenges for such characterization, providing motivating goals for mFIT's design.

### 3.1 Full System Compatibility

Many existing tools operate as custom or platform-specific memory controllers [23, 10, 9]; while this offers fine-grained control over the testing environment (e.g., the ability to control the DDR4 commands sent to the DIMM), it can inhibit compatibility across platforms. In particular, the host memory controller's effects on the system are often removed (replaced by those of the custom memory controller). Furthermore, supporting arbitrary DIMMs (e.g., both client- and server-grade) requires significant engineering effort. To the best of our knowledge, no custom memory controller offers "plug-and-play" support for an arbitrary DDR4 DIMM, nor the ability to

characterize the host platform's role in Rowhammer susceptibility (i.e., in addition to the DIMM's role). As such, an ideal tool would achieve broad compatibility across memory controllers and DIMMs.

### 3.2 Indirection

Rowhammer characterization is complicated by the many layers of indirection/"blackbox" hardware in between a system-level user and a DRAM-internal bit flip. For instance, a system-level user interacts with DRAM via loads/stores to virtual addresses that go through several layers of translation—and may be broken into multiple DDR4 commands (§2.2)—before resolving to DRAM internal addresses. Notably, commodity memory controllers do not offer fine-grained control over physical-to-DDR address translation logic, and DRAM modules do not directly expose internal addressing information. Thus, an ideal characterization platform must be able to account for these layers of indirection to characterize Rowhammer.

### 3.3 Precision

Many prior tools and approaches [42, 15, 14] lack the ability to extract fine-grained information about Rowhammer due to their lack of control over the DDR commands sent to DRAM. For example, prior work [29] demonstrates that the ability to suppress REF commands is key element in characterizing Rowhammer, as REF commands are otherwise used to obfuscate internal DRAM properties contributing to Rowhammer susceptibility [18]. Furthermore, because the refresh window—the maximum length of time between refreshes of a single row—is a key component in Rowhammer susceptibility (§2.3), fine-grained and programmatic control over the duration of REF suppression is paramount. Ultimately, an ideal Rowhammer characterization tool should grant sufficiently-precise control over REF suppression so as to effectively characterize Rowhammer.

### 3.4 Interference

Because DRAM loses data upon reboot, it is important to keep the test system alive during and after experimentation to collect reliable data from the DIMM-under-test. Furthermore, an ideal tool should not alter the relevant aspects of system operation necessary to characterize Rowhammer behavior. For example, the tool should not interfere with the ACT commands at the root of a Rowhammer attack (§2.3). Effectively, the tool should be as transparent as possible in terms of normal system operation.

### 3.5 Cost

Finally, to be widely-adoptable among the research community, a Rowhammer characterization tool must be affordable. While the thousands of dollars in costs for a custom memory controller or industrial memory tester may be acceptable to some, lower costs increase the approachability of Rowhammer characterization, and therefore the potential quantity of analyses. As described in the next section, mFIT is built using materials costing a total of just US$ 61.

## 4 Design of mFIT

This section describes the design and implementation of mFIT. We design mFIT as a low-cost hardware and software solution to enable experimental analysis of Rowhammer effects. Our primary goal is *compatibility* – we want to ensure that mFIT works with the host systems, memory controllers, and DIMM modules that users of mFIT deploy in their production environments. To ensure this wide compatibility, we design mFIT as a bump-in-the-wire device between the memory controller and a DRAM DIMM – an interposer that plugs into a standard DDR4 DIMM slot on the host motherboard. The primary mechanism that mFIT offers to enable a wide variety of Rowhammer experimentation is *refresh suppression*. mFIT offers fine-grained programmatic control over when REF commands are suppressed, and prior work [29] demonstrates that this ability is key to characterizing Rowhammer.

Fig. 2 shows mFIT installed in a server, with one DIMM attached to the mFIT interposer. Fig. 3 shows a block diagram of the main components of mFIT. Beyond the interposer, mFIT provides a control platform consisting of a logic controller board connected to the interposer and a software client (not shown in the photo). The software client runs on a separate PC and connects via USB to the mFIT controller board and connects via a remote console to the experimental platform (i.e., the server with the interposer installed) to initiate running workloads that will be measured during mFIT experiments.

The rest of this section describes how mFIT implements refresh suppression with a low-cost design, some of the system-level effects of mFIT's operation, how mFIT offers fine-grained control and synchronization, and finally a summary of the platform software we use to run workloads on the experimental platform.

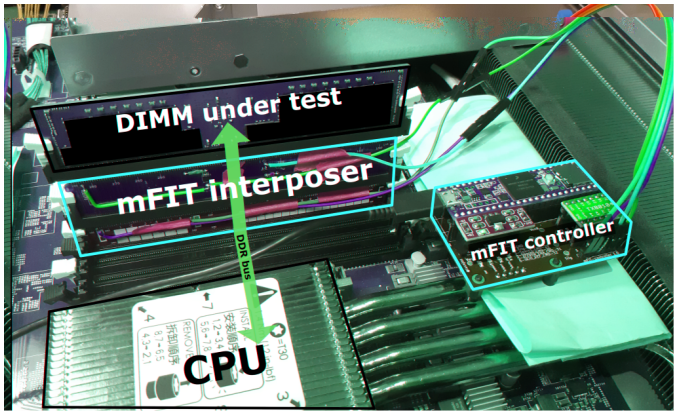


Figure 2: Deployment of mFIT in a server. Only one DIMM is attached so that the interposer is visible.

### 4.1 Refresh Suppression

To suppress REF commands, mFIT uses a physical interposer on the DDR bus placed between the CPU and one DIMM slot. The interposer can be configured at runtime to either allow or suppress refresh commands. Fig. 2 shows mFIT deployed on a server.

Fig. 3 shows a block diagram of the mFIT architecture, consisting of a control platform (on the left) and an experimental platform (on the right). In this figure, mFIT's components are shown with red boxes. To use mFIT, a software client running on the control platform loads a workload on the experimental platform. The workload includes *trigger points* to indicate when REF suppression begins and ends. These triggers are captured by the interposer and relayed to the logic controller board (with the `HWTRIG_CTL` signal). In turn, the controller board toggles the REF suppression by configuring two signals (`A14_CTL` and `ALERTn_CTL`) on the interposer.

The two signals used to suppress REF commands are inspired by a technique introduced in our earlier work [14]. A straightforward way to transparently suppress the REF command is to "fault" the `A14` signal on the DDR memory bus by driving it from high to low. The DDR4 encoding of the REF command sets the `A14` signal to high. When the interposer lowers the `A14` signal, this causes the DIMM to detect a parity error. The DIMM will then discard all corrupted commands and raise an error to the memory controller through the `ALERTn` signal. However, mFIT also suppresses the `ALERTn` signal to stop the parity error from reaching the memory controller. This error suppression is done via the `ALERTn_CTL` control signal shown in

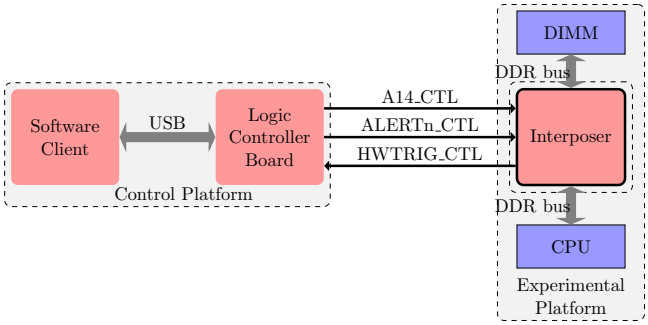


Figure 3: mFIT consists of a control platform and an interposer (mFIT's components are shown in red). The interposer is attached to an experimental platform. mFIT manipulates three control signals (ending in '_CTL') to transparently suppress REF commands.

Fig. 3.

The combination of these two signals enables mFIT to accomplish its goal of suppressing REF commands in a transparent manner. The REF command is faulty and ends up unprocessed by the DIMM. Also, the experimental platform never receives the parity error alert and continues to execute its workload unperturbed.

During the period of time when refresh suppression is enabled, mFIT forces the `A14` signal of the DDR4 command and address (CA) bus to low, regardless of which commands are sent on the bus. While this achieves REF suppression, it also comes with two limitations. First, the ACT command is limited to only activate rows whose address has bit 14 encoded as zero. Second, RD commands are converted to WR commands because RD and WR commands are encoded with identical signals except for `A14`. A RD command is encoded with a high `A14` signal, whereas a WR has `A14` set to low. Note that PRE commands are encoded with low `A14` and remain unaltered. As we will see in §6, these limitations still allow mFIT to perform a broad range of experiments.

The mFIT hardware components are simple, leading to a low-cost design. The logic controller board is an off-the-shelf design with an ARM Cortex-M4 micro-controller. This board uses GPIO pins to control the signals to and from the interposer board. The total cost of our logic controller board plus components is US$ 37. The total cost of our DIMM interposer board plus components is US$ 24. This leads to a total cost of US$ 61.

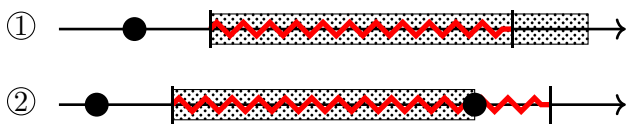


Figure 4: Two typical scenarios supported by mFIT for synchronization between the workload and REF suppression. ● is when the trigger event fires. ∿ is the time interval when mFIT suppresses REFs, and ▒ is the time interval when the workload runs.

### 4.2 Fine-grained Control and Synchronization

To meet our goal of precision for mFIT, we provide fine-grained and programmatic control over when REF suppression occurs. To accomplish this, we face two major challenges. The first is how to design a lightweight triggering mechanism to enable and disable REF suppression. The second challenge is how to *synchronize* the workload running on the target system with the REF suppression implemented by the interposer on the DDR bus.

Accordingly, we first summarize how a user controls mFIT to start and stop experiments. Next, we describe the design and implementation of our trigger mechanism. Finally, we describe a methodology that enables us to characterize the trigger lag and effectively synchronize REF suppression with the running workload.

**Controlling mFIT.** mFIT uses two mechanisms for fine-grained control over REF suppression. The first mechanism is a trigger: a sequence of code that runs on the CPU of the experimental platform (i.e., the target platform) that toggles the state of REF suppression (i.e., enables or disables REF suppression). The second mechanism, used only to disable REF suppression, is a software configured fixed duration for REF suppression. mFIT also supports manual control of REF suppression via the software client.

Fig. 4 shows two typical usage models of mFIT using the above mechanisms. For example, Scenario 1 shows when the user embeds a trigger before the start of the workload to achieve REF suppression for a fixed duration after the trigger event. Scenario 2 shows when the user embeds triggers at both the start and end of the workload they want to measure. As depicted in the figure, there is a short lag between when the trigger code executes on the CPU and when REF suppression starts or stops on the DDR bus. Below, we address how to compensate for this trigger lag.

**Trigger Design.** Traditional DRAM testing equipment uses sophisticated trigger designs based on a combination of DDR commands and

data [9, 19]. One common example of a trigger is to instruct the CPU to access memory at a specific DRAM address, which the experimentation tool (e.g., mFIT) interprets as a trigger event. Unfortunately, such a design would require the interposer to interpret DDR address signals *at bus clock rates* that range from 800 MHz to 1.6 GHz. Due to these high clock rates, this would require highly-optimized, complex, and expensive hardware on the interposer or logic controller board to satisfy timing constraints.

To meet our goal of designing a lightweight, inexpensive, and transparent trigger, we chose to monitor the (slower) System Management Bus (SMBus) that is adjacent to the DDR CA bus and used by the CPU to read metadata information from a DIMM. Because the SMBus maximum clock frequency is only 1 MHz, opting to trigger along this bus enables a comparatively simple and inexpensive hardware design. In this design HWTRIG_CTL is simply an wire connected to the signal that caries the data on the SMBus (SDA).

mFIT's trigger consists of programmatically issuing a metadata read command from the CPU of the experimental platform where the DIMM and interposer are installed. The metadata read only occurs on the SMBus, and is not affected by the interposer manipulating signals on the DDR CA bus. This approach is transparent because it does not interfere with the memory workload, and its overhead is negligible. In this way, the CPU only needs to execute a single load instruction to signal the REF suppression to start or stop.

**Characterizing Trigger Lag.** As shown in Fig. 4, we define the trigger lag as the period of time from when the CPU executes the instruction that enables REF suppression to when REF suppression actually begins on the DDR bus. Trigger lag has two inherent causes: first, there is a propagation delay from when the CPU issues the load command that reads the DIMM metadata to when the logic controller board detects the trigger signal and then reconfigures the `A14_CTL` and `ALERTn_CTL` signals sent to the interposer. Second, there is CPU instruction reordering and also memory operation reordering at the memory controller.

Our goal is to demonstrate that this lag is short relative to the rate of REF commands, and that it is consistent across experimental workloads. Once we determine the maximum trigger lag, we insert a fixed delay matching that lag after we enable REF suppression yet before we start to execute the experimental workloads – thereby preventing trigger lag from affecting our experimental data.

Because we have no direct visibility on the DDR bus, we focus on measuring the trigger lag as the period of time from when the CPU executes the instruction that enables REF suppression to when the first memory accesses

issued by the CPU are affected by the suppression of REF commands. We develop a calibration methodology to measure the trigger lag, and use it to characterize the lag on our platform.

To characterize the trigger lag, we develop a technique we call *memory casting*. With memory casting, the experimental platform runs a workload that writes a monotonically increasing counter value to DRAM continuously at a uniform rate. These writes are carefully constructed to use a memory location whose DRAM row address is encoded with a high `A14` signal. This address choice is important because when mFIT suppresses REFs, it sets the `A14` signal to low. Therefore, as soon as mFIT's trigger suppresses `A14`, the writes will be performed at a different address (where the DRAM row address is encoded with a low `A14` signal). This shift in addressing allows us to inspect memory to determine which was the last counter value written before REF suppression started. We also prevent reordering from affecting our casting workload by inserting the appropriate memory fence instructions between each iteration of the casting workload.

The above method enables us to precisely measure the trigger lag. Using this approach, we experimentally determine that the median trigger lag is 1.4$\mu$s and remains below 2$\mu$s even with other workloads. Even though this technique does not require a bus analyzer, we use a DDR bus analyzer to further validate that our memory casting technique works as expected. After we determine the maximum trigger lag (2$\mu$s), we prevent this lag from affecting our experiments by inserting a fixed delay matching the lag value after we enable REF suppression and before we start to execute the experimental workload.

### 4.3 Software Experiments

We implement our software that runs on the target system as an UEFI [3] application to ensure a precise control over the memory accesses. We choose the UEFI environment for two reasons. First, we want to avoid spurious memory accesses caused by other processes that are typically present when running a general purpose operating system. The UEFI environment allows us to implement tasks that run on a single CPU. Second, we want to precisely target a specific DIMM, bank, row, and column. The memory mapping in UEFI allows a direct mapping between the virtual and the physical address space (i.e., a virtual pointer can be used as a physical pointer). Further, using the open-source Error Detection And Correction (EDAC) [2] driver as an inspiration, we implement the physical-to-logical mapping that allows us to decompose CPU's physical address to a DIMM, row, bank, and

column address. For example, we use this ability to map a physical address to a specific DIMM and row address in implementing the memory casting technique described above.

# 5 Experiment Platform and Proof-of-Concept

This section describes the experimental setup for mFIT's analyses of Rowhammer susceptibility. We provide an initial proof-of-concept experiment demonstrating increasing Rowhammer susceptibility, before analyzing specific susceptibility factors in §6.

## 5.1 Platform Setup

At a high level, we place mFIT between a host memory controller and a DIMM-under-test. mFIT suppresses REFs issued to the DIMM-under-test as specified in various experiments. We reserve a second attached DIMM for regular system functionality (i.e., the UEFI application described in the previous section). Running the application on an isolated DIMM avoids system crashes due to mFIT's REF suppression and experimentally-induced Rowhammer bit flips, important for preventing interference (§3.4).

Given the lack of server platform data for Rowhammer, we focus our experiments on commodity server platforms used by a major cloud provider. However, we note that mFIT is capable of operating on both server and client platforms.

**DIMMs.** We analyze the latest three generations of DDR4 server-grade DIMMs from three major DRAM vendors, for a total of 9 DIMMs. More specifically, the three generations of DIMMs are differentiated by their process nodes (older to newer: 1x, 1y, and 1z—with this information provided by industrial partners). The DRAM vendors are referred to anonymously as `VendorA`, `VendorB`, and `VendorC`.

Previous work [29] uses the manufacturing date code of DRAM modules to estimate the process node. However, we find that the date code does not always correlate with the process node.

**CPUs/Memory Controllers.** We test across two generations of Intel server blades/integrated memory controllers: Skylake Scalable Processor (Skylake-SP) and Cascade Lake-SP. We examine each DIMM on one of the two blades, depending on host compatibility (i.e., some of the newer DIMMs are only compatible with the newer Cascade Lake blades).

Because mFIT is limited to a maximum operating frequency of 1200MT/s, we lower the memory bus speed to this frequency in the BIOS. Crucially,

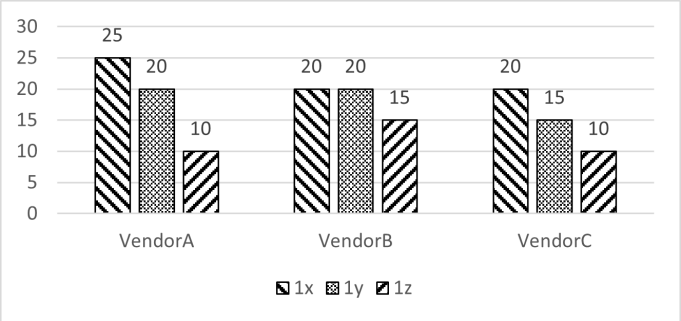


Figure 5: The Y-axis represents the "Hammer Count" (in thousands) at which we observe the first bit flip. We use 2048 rows and a double-sided attack. We test four different data patterns on each row (Section 6.3 will describe these patterns in detail). To select a single representative number per vendor, we take the minimum of the mean values of hammer counts for each of the four patterns.

this does not significantly impact the testing environment, as (a) REF commands–which mFIT suppresses–continue to be issued at the same rate, regardless of bus frequency, and (b) though the number of ACT commands issued within a refresh window may be slightly decreased, the system can still issue an orders-of-magnitude-sufficient number of ACT commands within a refresh window to cause Rowhammer bit flips (§2.3)—thereby retaining characterization capabilities.

Additionally, to more easily observe bit flips and control the testing environment, we disable the host's error correction code (ECC) support and data scrambling; effectively, these are BIOS options that alter the data stored in DRAM, and therefore complicate experiments unless disabled. We note that disabling these options does *not* change the underlying properties of the DRAM. Furthermore, prior work has demonstrated that ECC is not an effective defense against Rowhammer [15]; we characterize the impact of data scrambling in §6.4.

### 5.2 Proof-of-Concept: Measuring Rowhammer Susceptibility via Hammer Count

In line with prior work [29], we establish increasing susceptibility to Rowhammer in newer DIMMs by measuring the *hammer count* (§2.3) at which the first bit flip is observed, with REFs suppressed by mFIT to mitigate obfuscation effects. Following this prior methodology, we encode data in a logically-

checkered pattern throughout the DIMM (i.e., from the memory controller's perspective, 0-1-0-1 on odd rows, and 1-0-1-0 on even rows), and repeatedly perform a 2-sided Rowhammer attack (i.e., 2 aggressor rows) across a set of rows within bank of memory.

As shown in Fig. 5, across all vendors, we observe that lower HCs are needed to induce bit flips in newer DIMMs. This data—gathered for server-grade DIMMs—is consistent with the worsening susceptibility observed for client-grade DIMMs in previous work [29], motivating an in-depth experimental analysis of Rowhammer susceptibility factors.

# 6 Evaluating Rowhammer Susceptibility Factors

Having confirmed increasing susceptibility to Rowhammer, we use mFIT to provide the first experimental quantification of various susceptibility factors in DDR4 server-grade DIMMs. Using the platform setup described in §5.1, we analyze three key DRAM-internal susceptibility factors identified in prior work: row address scrambling [17, 15] (§6.1), subarray boundaries [35] (§6.2), and data dependency among cells [30, 32, 29] (§6.3). We additionally exploit mFIT's ability to characterize the host memory controller (i.e., in addition to DRAM itself), analyzing the role of data scrambling in Rowhammer susceptibility (§6.4). Finally, we use our analyses to construct an "optimal" Rowhammer data pattern (§6.5)—capable of flipping up to 2X more bits than the state-of-the-art pattern [29]—validating our findings and offering researchers insights on how to construct stronger Rowhammer attacks and defenses.

## 6.1 Factor #1: Row Address Scrambling

DRAM vendors often scramble row addresses by remapping the MC's *logical row addresses* to different *physical row addresses* inside a DRAM device. Such forms of scrambling improve the geometry of the die, optimize internal address decoding, or make better re-use of internal circuitry [44]. Characterizing row address scrambling lets us determine row adjacency, a vital Rowhammer susceptibility factor [30]. Note that server-grade DIMMs are subject to two additional forms of scrambling: (1) output inversion done by the Registered Clock Driver (RCD) [11], and (2) bit swizzling done by the memory controller. Both RCD output inversion and bit swizzling are well-documented [6, 4], and we can therefore account for them in our experiments.

| DIMM-ID | row address scrambling? |
|---|---|
| VendorA-1X | ✗ |
| VendorB-1X | ✗ |
| VendorC-1X | ✓ |
| VendorA-1Y | ✗ |
| VendorB-1Y | ✗ |
| VendorC-1Y | ✓ |
| VendorA-1Z | ✓ |
| VendorB-1Z | ✗ |
| VendorC-1Z | ✓ |

Table 1: Presence/absence of row address scrambling as recovered with mFIT.

mFIT's refresh suppression capability lets us borrow a technique from industrial DRAM testing equipment [19, 9] to determine physical row adjacency: we activate the row repeatedly (similar to mounting a single-sided Rowhammer attack) and measure the bit flip density of victim rows. Adjacent victim rows show significantly higher number of bits flips than non-adjacent victim rows. We note that previous work [40, 37, 42] also used this technique on general-purpose platforms without suppressing refresh commands; however, their results were limited to testing only previous generation DDR3 DRAM devices and client-grade DIMMs.

We found a bimodal distribution among the DIMMs: some have no form of row address scrambling (i.e., the map from logical to physical row addresses is the identity function) whereas all others share the following mapping function:

$$PhysicalRowAddress = LogicalRowAddress \oplus (b_3 \ll 2) \oplus (b_3 \ll 1) \quad (1)$$

where $b_3$ is the bit at index 3 in the logical row address ($b_0$ is the least significant bit).

Our results (summarized in Table 1) show that `VendorC` uses row scrambling in all its DIMMs, whereas `VendorA` uses it in its newest DIMMs only.


Figure 6: An example of a many-sided attack pattern with 4 aggressor rows and 5 victim rows.

The use of the same row scrambling function suggests that different vendors could be using *similar* optimization techniques for their devices or could be *sharing* their DRAM designs amongst themselves. Unfortunately, we are unaware of any publicly available design documents of row scrambling functions. In their absence, it is difficult to determine the exact origin of similarities across vendors. Our mapping function matches the results of a recent study that tests older DDR3 DIMMs (manufactured in 2007 and 2008) [17]. Like us, they found two different vendors to share this pattern across their DIMMs.

**The impact of row address scrambling on Rowhammer** Recent work has shown that mounting Rowhammer on newer DRAM requires crafting a careful pattern of aggressor rows. One such common pattern is *many-sided* formed by a tuple of $k$ aggressor rows [18]. Fig. 6 shows an example of a many-sided attack pattern.

Unfortunately, row address scrambling can distort a many-sided attack pattern in logical space by re-mapping both aggressor and victim rows. To understand the impact of row address scrambling, we examined all possible *many-sided* patterns with $k$ aggressor rows. We varied $k$ from 2 to 19 to re-construct the attack patterns used by recent work [18, 16]. We examined a full DDR4 bank ($2^{18}$ rows) [1].

We classified each many-sided pattern in logical space in four different categories depending of its layout in physical space: (1) neither the aggressors nor the victims remain many-sided, (2) only the victim rows remain many-sided, (3) only the aggressor rows remain many-sided, and (4) both aggressor and victim rows remain many-sided. Fig. 7 shows our classification results for each $k$ value.

Based on our results we make the following observations. First, a many-sided attack in logical space has a 50% or less chance to remain many-sided

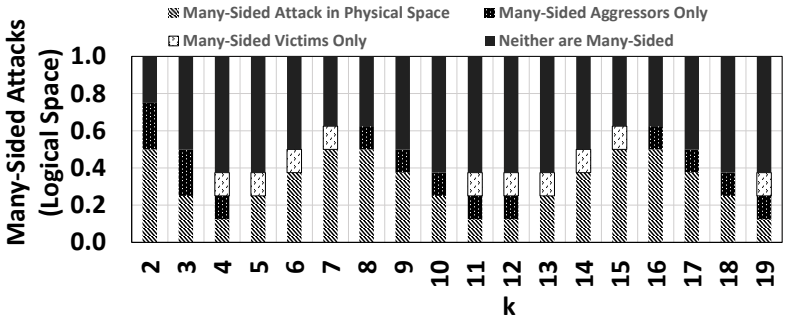


Figure 7: Impact of row address scrambling to many-sided Rowhammer patterns in logical space. Once mapped in physical space, the pattern can (1) remain many-sided, (2) only the victim rows remain many-sided, (3) only the aggressor rows remain many-sided, or (4) neither aggressor nor victim rows remain many-sided.

in the physical space due to address scrambling. For k=4, 11, 12, and 19, only 12.5% of many-sided attacks in logical space remain many-sided in physical space. The best case scenario is for k=2, 7, 8, 15, and 16 when half of many-sided attacks in logical space remain many-sided in physical space.

Second, address scrambling often affects both the aggressor rows and the victim rows of a k-sided attack. At least 25% (and as much as 62.5%) of many-sided attacks have both aggressors and victims mapped differently.

These results indicate that most previous evaluations of many-sided patterns are impacted by address scrambling for all values of $k$. When address scrambling affects victim but not aggressor rows, previous papers likely under-reported the number of bit flips because they did not check for bit flips on the correct set of victim rows. When address scrambling affects aggressor but not victim rows, previous papers failed to mount the correct many-sided attack pattern.

### 6.2 Factor #2: Subarray Boundaries

Recall from §2.2 that banks of DRAM—where aggressor and victim rows must co-reside to yield Rowhammer bit flips—are actually broken down into a set of row-column subarrays of DRAM cells. As suggested in prior work [35], the electromagnetic isolation of each subarray should mean that Rowhammer attacks require not only rows co-located in the same *bank*, but within the same *subarray* within that bank. Accordingly, we use mFIT to experimentally test this theory.

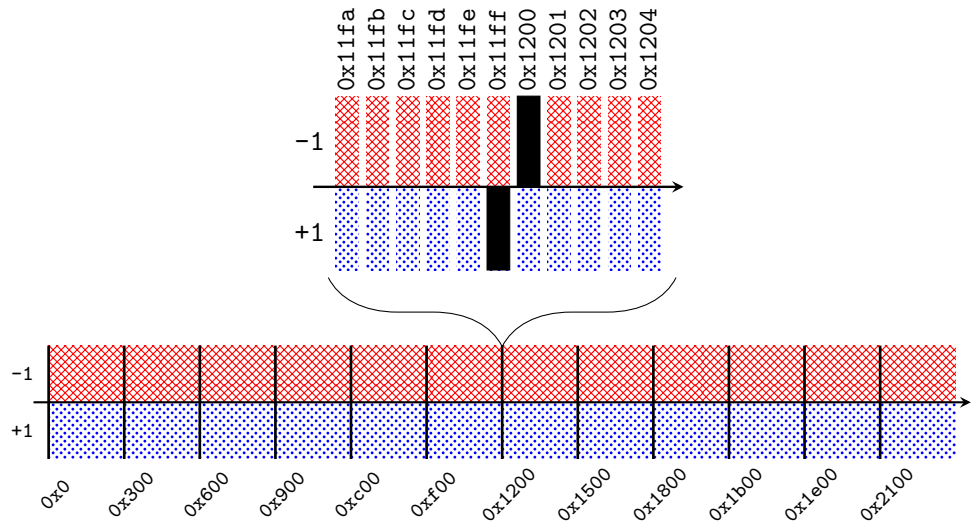


Figure 8: Subarray configuration revealed by a single-sided Rowhammer experiment on `VendorB-1Y`. The X-axis corresponds to the unscrambled row address of the aggressor row ($r$). A vertical blue bar ( ) below the axis represents that victim row $r+1$ experienced bit flip(s), and a red bar ( ) that victim row $r-1$ experienced bit flip(s). A black vertical bar means that one of the victims (either $r+1$ or $r-1$) did not experience any bit flips. We observe that every `0x300` rows, there are no bit flips on $r+1$, immediately followed by no bit flips on $r-1$.

At a high level, we test whether there is a pattern for which aggressor-victim row pairs fail to produce bit flips during a single-sided Rowhammer attack. For instance, given aggressor row $r$ and adjacent row $r+1$, a lack of bit flips in the potential victim row $(r+1)$ for $r \mod 100 = 0$ (but an otherwise successful attack) would provide strong evidence that (a) the subarray size is 100 rows, (b) $r$ and $r+1$ are on different subarrays, and (c) subarray boundaries prevent Rowhammer attacks.

However, we must account for two potentially-confounding factors in this test. First, a DIMM can internally "repair" defective rows by remapping them to spare rows within a chip, obfuscating the pattern of successful/unsuccessful attacks. In practice, we find that the row repair phenomenon is rare for all DIMMs tested (i.e., we only found evidence of around 0.15% of rows being repaired), meaning that the pattern can still be inferred with a sufficiently-large sample size.

Second, even without defective row repairs, *logically*-adjacent rows are not necessarily *physically*-adjacent in DRAM. Thankfully, as discussed in §6.1, we find that 5 samples map logically-adjacent rows to physically-adjacent rows (aside from occasional defective row repairs). For the remaining 4 samples, we can account for row address scrambling via our reverse-engineered scrambling equation, meaning we can produce appropriate logical row addresses for the aggressor and victim to ensure we test physically-adjacent rows.

Thus, accounting for these factors, we use the following methodology to test for a relationship between subarray boundaries and Rowhammer efficacy. According to the DIMM-specific mappings recovered in §6.1, we sweep through physically-adjacent rows in a bank one at a time. For each row $r$, we prime $r$ and its potential victims $r-1$ and $r+1$ with a DIMM-specific effective aggressor-victim data pattern—e.g., checkered (§5.2).

We then perform a single-sided Rowhammer attack with $r$ as the aggressor (i.e., we repeatedly activate $r$), with REFs suppressed by mFIT to simplify flipping bits. Note that in order to repeatedly-activate (i.e., open) $r$, we must force $r$ to close between accesses, which we achieve by alternating writes between row $r$ and some other dummy row $d$ in the same bank. To avoid Rowhammer interference between $r$, $d$, and potential victim rows, we choose $d$ to be fixed, near the end of the bank, much farther than any known *blast radius* (§2.3).

After completing an attack for a given $r$, we check for bit flips in $r-1$ and $r+1$. Note that we always observe bit flips in at least 1 of the victim rows due to the effectiveness of the aggressor-victim data pattern. If we observe bit flips in *both* rows, we cannot conclude that $r-1$, $r$, and $r+1$ are on different subarrays.

However, if we only observe bit flips in *one* of the potential victims (e.g., $r-1$), we conclude that the other victim row ($r+1$) *might* be on a different subarray than $r$. To confirm this suspicion, we repeat the experiment with various different aggressor-victim patterns, to eliminate different data encoding in row $r+1$ as the reason for the unsuccessful attack. If we fail to observe bit flips in $r+1$ for all data patterns, we conclude that there is a high likelihood that $r$ and $r+1$ are on different subarrays.

After completing the experiment for a sufficient number of rows, we check for a pattern in unsuccessful attacks. If the attack fails at regular intervals (e.g., every $n$ rows), we infer that the subarrays consist of $n$ rows, and that Rowhammer attacks fail across subarray boundaries.

After accounting for row address scrambling, we observed a DIMM-specific, steady pattern of failed Rowhammer attacks for all three vendors

| DIMM-ID | subarray size (in #rows) |
|---|---|
| VendorA-1X | 0x02b0 |
| VendorB-1X | 0x0300 |
| VendorC-1X | 0x0339 |
| VendorA-1Y | 0x0338 |
| VendorB-1Y | 0x0300 |
| VendorC-1Y | 0x0402 |
| VendorA-1Z | 0x03fa |
| VendorB-1Z | 0x0400 |
| VendorC-1Z | 0x04f9 |

Table 2: Subarray sizes recovered with mFIT.

across all three generations; we show the pattern observed for an example DIMM (VendorB-1Y) in Fig. 8. We reiterate that rare exceptions to the pattern are believed to be defective row remaps, and do not affect our ability to infer the pattern. For simplicity, we present the inferred subarray sizes in Table 2.

Thus, we offer strong experimental evidence in support of the theory that Rowhammer attacks require aggressor and victim rows to be co-located on the same subarray within a bank [35]. Accordingly, an optimally-effective Rowhammer attack will ensure that all aggressor and victim rows are in the same subarray, while effective defenses could aim to ensure that rows from two different trust domains (e.g., processes) are mapped to different subarrays.

### 6.3 Factor #3: Data Dependency Among Cells

Prior work [30] has established that the efficacy of a Rowhammer attack is influenced by the data stored in the rows of interest in two ways. First, victim cells tend to be disturbed from the positively "charged" to "discharged" state (i.e., by an injection of [negative] electrons), meaning that whether a logical 1 or logical 0 is encoded as the charged state affects Rowhammer

| ID | Even rows (0,2,4,...) | Odd rows (1,3,5,...) |
|---|---|---|
| RS0 | 0x0000000000000000 | 0xffffffffffffffff |
| RS1 | 0xffffffffffffffff | 0x0000000000000000 |
| CH0 | 0x5555555555555555 | 0xaaaaaaaaaaaaaaaa |
| CH1 | 0xaaaaaaaaaaaaaaaa | 0x5555555555555555 |

Table 3: 64 bit baseline data patterns used in previous work [30, 29].

susceptibility. Second, the electrons that discharge (i.e., disturb) these victim cells tend to leak from nearby aggressor cells, where the aggressor cells themselves are in the discharged state.

Based on these foundational observations, prior work [31, 15] attempts to achieve susceptibility to Rowhammer via data patterns that—at first glance—ensure aggressor cells are discharged and nearby victim cells are initially charged. These patterns are listed in Table 3. RS indicates "Row Stripe", where RS0 indicates that even rows (starting at row 0) are all logical 0s, while odd rows are all logical 1s. RS1 is the inverse of this pattern. CH indicates "Checkered" as described in §5.2, where CH0 and CH1 are distinguished by whether the first bit in even rows is a logical 0 or 1, respectively.

To our knowledge, no work has experimentally determined whether a given "hammerable pattern" remains effective throughout an entire DIMM, nor experimentally-confirmed the maximally-hammerable pattern. Accordingly, we first calculate the state-of-the-art optimal baseline from among these four patterns; this baseline is determined by the pattern yielding the maximum number of bit flips when using a 20-sided Rowhammer attack—after having accounted for row address scrambling (§6.1) to ensure true physical adjacency.

We perform a 20-sided Rowhammer attack (as opposed to simpler single- or double-sided) for two reasons. First, a 20-sided attack has been shown to be effective on modern systems, even with state-of-the-art mitigations enabled (§2.3). Second, for reasons yet unknown, we found the data dependence pattern was easier to infer when using 20 aggressor rows versus a smaller number. Notably, the baseline pattern varies by DIMM (i.e., row stripe for some, checkered for others), likely due to variations in cell and bitline encodings among different DIMMs.

From these per-DIMM baselines, we then set out to modify the pattern according to knowledge of various internal DIMM structures and behaviors

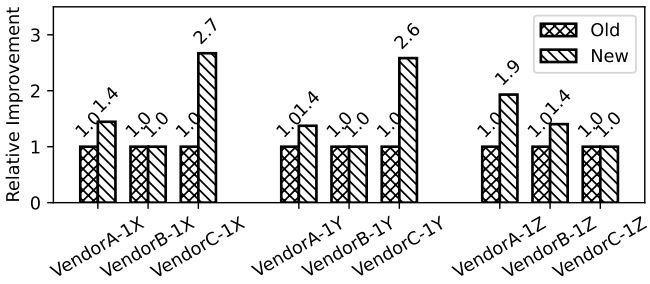


Figure 9: Relative improvement in the number of bits flipped with our new pattern for each DIMM.

across which data encodings could potentially change (e.g., a logical 1 might be stored as the discharged state instead of the charged state). First, using the previously-inferred subarray boundaries (§6.2) we invert the pattern at each subarray, to account for potentially-alternating data encoding between subarrays. Next, because a cache line is returned from a DIMM in successive "bursts" (§2.4), we invert the pattern for the columns of data returned by each burst; we note that the mapping between cache line word and burst number are public. Finally, because each "nibble" within a burst comes from a different chip on the DIMM (§2.4), we invert the pattern at the nibble boundary (every 4 bits on the DIMMs-under-test), accounting for swizzling where applicable.

We tested all possible combinations of these three independent considerations for the maximum number of bit flips. We list our newfound optimum pattern descriptions in Table 4, plotting their improvements compared to prior baselines in Fig. 9. We found that, in the majority of cases, the previously-used patterns could be significantly improved upon (60% on average and even 167% in the case of `VendorC-1X`) by accounting for subarray, burst, and chip boundaries within the DIMM. Interestingly, for the HammerCount that we use for each the imbalance between the best pattern and its inverted counterpart is relatively large (for example 23:7 for `VendorA-1Y`, 27:1.3 for `VendorA-1Z` and 44:0.3 for `VendorB-1Z`) suggesting that a Rowhammer-highly-immune data pattern is possible.

Ultimately, our findings indicate that greater Rowhammer susceptibility can be achieved by (1) adjusting the pattern according to available DIMM-specific metadata, and (2) data that is reverse engineered (e.g., the subarray size and row-addressing). Conversely, certain Rowhammer effects can be entirely negated on select DIMMs when using the inverse pattern, an

| DIMM-ID | Base-line | Im-prove-ment | 64-bit victim (w/DQ sw.) on Row 0 Col 0 | Invert w/ SA | Pattern description |
|---|---|---|---|---|---|
| VendorA-1X | CH | 45% | 0x5555 5a5a 5a5a 5555 | ✓ | Invert every new SA, except between $3^{rd}$ and $4^{th}$ |
| VendorB-1X | RS | 0% | 0x0000 0000 0000 0000 | ✗ | - |
| VendorC-1X | RS | 167% | 0x0000 0000 0000 0000† | ✗ | Inverts in upper-bursts (cols 8K+4, 8K+5, 8K+6, 8K+7) |
| VendorA-1Y | CH | 37% | 0x5555 5a5a 5a5a 5555‡ | ✗ | Upper cols. differ |
| VendorB-1Y | RS | 0% | 0x0000 0000 0000 0000 | ✗ | - |
| VendorC-1Y | RS | 158% | 0x0000 0000 0000 0000† | ✗ | Inverts in upper-bursts (cols 8K+4, 8K+5, 8K+6, 8K+7) |
| VendorA-1Z | CH | 93% | 0x3c3c 3c3c 3c3c 3c3c‡ | ✓ | Invert every new SA, except between $8^{th}$ and $9^{th}$ SA. Upper cols. differ |
| VendorB-1Z | RS | 40% | 0x0000 0000 0000 0000† | ✗ | Inverts in every-other burst (cols 2K+1) |
| VendorC-1Z | RS | 0% | 0xffff ffff ffff ffff | ✗ | - |
| Average | | 60% | | | |

Table 4: Optimal-hammering patterns (i.e., most bit flips) recovered with mFIT for a 20-sided attack. The baseline is the original-optimal pattern from those in Table 3, followed by the relative improvement we were able to achieve via modifications. We list the updated data pattern per DIMM, as well as whether we inverted the pattern by subarray (A)—i.e., CH0 on even subarrays, CH1 on odd subarrays—and a brief pattern description. †: pattern is inverted on cache line granularity as some bursts must have the inverted value. ‡: data is reported on the lower columns (columns from 0 to 511 including). The columns from 512 to 1023 have a different susceptibility pattern.

interesting consideration for future mitigations that may seek to encode data according to minimally-hammerable pattern where possible.

### 6.4 Factor #4: Host Data Scrambling

Given that the data pattern stored in a row affects the susceptibility to a Rowhammer attack [30], it is important not only to understand how *DRAM* itself encodes data (§6.3), but also how the *host memory controller* encodes data. Notably, so long as the host memory controller can recover the original data sent to it from the upper layers of the memory subsystem (e.g., cores), it is free to encode the data in memory as it sees fit (e.g., in a similar manner to data encryption/decryption).

In this vein, for electrical reliability reasons, the host memory controller of modern servers "scrambles" (i.e., masks) data before storing it in DRAM [13]. More specifically, upon a DRAM write, the memory controller XORs the original data with mask(s) that are likely to produce an even distribution of physical 1's and 0's in each row; when later reading the physical data back, the memory controller XORs the physical data with the same mask(s) to recover its original (logical) value ($\texttt{data} \oplus \texttt{mask} \oplus \texttt{mask} = \texttt{data}$). Accordingly, it is important for both DRAM analyses and Rowhammer attackers to account for data scrambling, such that the desired data pattern is encoded into DRAM.

From a DRAM analysis standpoint (e.g., determining cell layout), we can trivially account for data scrambling on our test platforms by disabling it in the BIOS. However, because this is not an option for all platforms—and because accounting for data scrambling remains crucial to producing an optimized Rowhammer attack/defense on a production system—we use mFIT to reverse engineer the data scrambling implementation in the host memory controller. Notably, we find that the memory controller's implementation of data scrambling is independent of the attached DIMM. Therefore, it suffices to reverse-engineer the mask(s) for any single DIMM.

At a high level, we combine mFIT's ability to suppress REFs with knowledge of a specific DIMM's cells' orientations to recover the scrambling mask(s). We use a DIMM composed of "true" cells—i.e., those that encode a logical 1 as the charged state, and therefore discharge from $1 \rightarrow 0$ in the absence of REFs—because this allows us to easily determine the XOR mask(s).

More specifically, we activate mFIT such that all data on the DIMM will eventually move to the discharged state of 0 (due to lack of REFs). On the host system, we then read the data back from the DIMM. When the

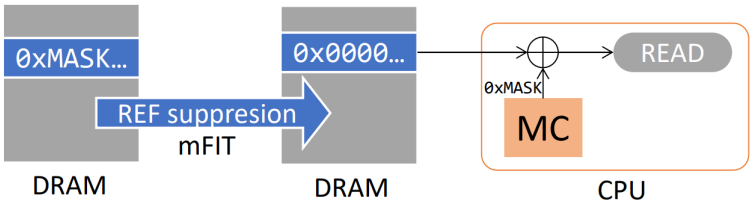


Figure 10: Recovering the MC's data scrambling pattern by REF suppression after the mask lands in DRAM.

memory controller receives the physical 0 values from the DIMM, it will apply the XOR mask(s) to unscramble the data. Because $0 \oplus \texttt{mask} = \texttt{mask}$, the data ultimately returned to the processor core is in fact the scrambling mask, and has therefore been recovered.

Repeating this experiment at different locations throughout the DIMM, we notice two interesting properties. First, there is a single mask applied to all data for scrambling. Second, the mask does not change across system reboots; in fact, we find that the mask is calculated from a 64-bit seed value, configurable in the system BIOS. We note that these observations for Intel DDR4 memory controllers (i.e., those used on the Skylake-SP and Cascade Lake-SP micro-architectures) are consistent with those previously-observed for DDR3 memory controllers [13, 15].

The ramifications of these findings for Rowhammer susceptibility are significant. Because the scrambling mask does not change between rows, once an attacker recovers the scrambling mask, there are able to prime an arbitrary row with a "maximally-hammerable" data pattern. Effectively, commodity implementations of data scrambling do not pose significant obstacles for Rowhammer attackers, and therefore should not be considered effective mitigations.

### 6.5 Putting It All Together

To summarize, mFIT offers the ability to characterize various Rowhammer susceptibility factors, include those within the DRAM itself as well as the host memory controller. By accounting for row address scrambling, subarray boundaries, data dependencies among cells, and data scrambling in the host memory controller, one can produce up to 60% on average (see Table 4) greater bit flips on a DIMM than state-of-the-art baseline patterns provide. Conversely, we additionally find that select factors *entirely prevent* Rowhammer bit flips, including placing aggressor and potential victim rows

on separate subarrays, as well as select data patterns for certain DIMMs. These findings enable researchers to better reason about efficacy of current and future Rowhammer attacks and defenses.

# 7 Related Work

**Industrial memory testers.** There is a small market of industrial memory testers aimed at hardware vendors and system integrators [9, 8]. These testers are built for standalone DRAM module testing and often come with pre-defined testing sequences and benchmarks. However, some testers offer more flexibility by including a custom-made memory controller that lets operators develop their own sequence of testing commands. Such testers are expensive with prices starting at hundreds of thousands of dollars. In contrast, mFIT is less expensive and can reveal insights into the behavior of an entire memory system including the DRAM modules, the DIMMs, the busses, and the memory controller.

**FPGA-based memory controllers.** Researchers have developed FPGA-based hardware platforms to give them increased control over the commands sent to a DRAM module [23, 18, 29]. These platforms are flexible allowing for fine-grained control over the sequence of DRAM commands and their timings. This high degree of flexibility comes with a high development cost due to the complexity of developing of a full-featured memory controller. In our experience, an additional barrier of FPGA-based platforms is their lack of supporting server-grade DIMMs. Instead, most FPGA platforms either have different DIMM formats (such as client-grade SO-DIMMs) or they only allow for a couple of specific DDR4 DIMM parts. In contrast, mFIT is more flexible albeit with less control of DRAM commands. Its main goal is refresh suppression.

**DDR bus analyzers.** DDR bus analyzers capture traces of DDR commands sent by a memory controller to one or more DIMMs [24, 19, 7, 33, 14]. Previous work used industrial bus analyzers to characterize Rowhammer sequences on real hardware [14], mount an attack against secure enclaves [33], summarize memory accesses to analyze the efficiency of a newly designed system [28], characterize memory workloads [27], and recover the memory controller address mappings [37, 14]. An earlier project showed how to design and implement a platform for tracing DDR commands and saving them for off-line analysis [24]. Bus analyzers are also expensive with costs approaching US$ 100K. Bus analyzers' functionality is orthogonal to that of mFIT; their traces of DDR commands can complement mFIT's refresh

suppression capability and provide additional insights into DRAM characterization.

**DDR command fault injectors.** DDR command fault injection techniques were previously used [14, 15, 20] to characterize DIMMs running in a host system. These devices are more rudimentary than mFIT and require manual human interaction, such as pressing a button [14] or inserting a shunt [15]. Fault injection on the memory bus data lines was used to understand system level error correcting codes [15] and to study the error reporting mechanism overhead [20].

Intel offers a commercial tool that can inject errors in the data lines of a DIMM [5]. This tool's main purpose is testing the error correction functionality of a memory controller by simulating data failures.

# 8 Conclusion

As DRAM becomes increasingly susceptible to Rowhammer bit flips, understanding the factors influencing the likelihood of a bit flip is important for maintaining system reliability and security. mFIT is the first analysis tool to offer precise and low-cost "plug-and-play" Rowhammer analysis capabilities for both arbitrary DDR4 DIMMs and associated host platforms (e.g., memory controllers). Having used mFIT to confirm increasing susceptibility to Rowhammer in DDR4 server-grade DIMMs, we have extensively analyzed the role of various Rowhammer susceptibility factors discussed in prior work. With the system properties unearthed by mFIT, we have constructed a Rowhammer data pattern averaging 60% greater bit flips than the existing state-of-the-art patterns. Using the open-source design of mFIT and our results thus far, we hope to enable future researchers to increase the efficacy of Rowhammer attacks, defenses, and understanding.

## References


[1] "DDR4 SDRAM STANDARD," JESD79-4.

[2] "The Linux Kernel – Error Detection And Correction (EDAC) Devices," https://github.com/torvalds/linux/tree/master/drivers/edac.

[3] "Unified Extensible Firmware Interface (UEFI)," https://uefi.org/specifications.

[4] "DDR4 SDRAM Registered DIMM Design Specification," Sep. 2014.

[5] "Intel: Memory Error Injection MEI Test Card and Utility," https://designintools.intel.com/MEI_Test_Card_and_Utility_p/stlgrn61.htm, 2017.

[6] "Annex L: Serial Presence Detect (SPD) for DDR4 SDRAM Modules," 2018.

[7] "Teledyne LeCroy - DDR Debug Toolkit," 2018.

[8] "Eureka 2933 DDR4 Memory Tester," 2019.

[9] "The TurboCATS III-2667ST Memory Test System," http://www.turbocats.com.hk/, 2019.

[10] "LiteX Row Hammer Tester," https://github.com/antmicro/litex-rowhammer-tester, 2021.

[11] "DDR4 Registering Clock Driver Definition (DDR4RCD02)," Augus 2016.

[12] AMD, "TRRespass (CVE-2020-10255) – 3/10/20," https://www.amd.com/en/corporate/product-security, 2020.

[13] J. Bauer, M. Gruhn, and F. C. Freiling, "Lest we forget: Cold-boot attacks on scrambled DDR3 memory," *Digital Investigation*, vol. 16, pp. S65–S74, 2016.

[14] L. Cojocar, J. S. 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 2020, San Francisco, CA, USA, May 18-21, 2020*. IEEE, 2020, pp. 712–728. [Online]. Available: https://doi.org/10.1109/SP40000.2020.00085

[15] L. Cojocar, K. Razavi, C. Giuffrida, and H. Bos, "Exploiting correcting codes: On the effectiveness of ECC memory against rowhammer attacks," in *2019 IEEE Symposium on Security and Privacy, SP 2019, San Francisco, CA, USA, May 19-23, 2019*. IEEE, 2019, pp. 55–71. [Online]. Available: https://doi.org/10.1109/SP.2019.00089

[16] F. de Ridder, P. Frigo, E. Vannacci, H. Bos, C. Giuffrida, and K. Razavi, "SMASH: synchronized many-sided rowhammer attacks from javascript," in *30th USENIX Security Symposium, USENIX Security 2021, August 11-13, 2021*, M. Bailey and R. Greenstadt, Eds. USENIX Association, 2021, pp. 1001–1018. [Online]. Available: https://www.usenix.org/conference/usenixsecurity21/presentation/ridder

[17] M. Farmani, M. M. Tehranipoor, and F. Rahman, "RHAT: efficient rowhammer-aware test for modern DRAM modules," in *26th IEEE European Test Symposium, ETS 2021, Bruges, Belgium, May 24-28, 2021*. IEEE, 2021, pp. 1–6. [Online]. Available: https://doi.org/10.1109/ETS50041.2021.9465436

[18] P. Frigo, E. Vannacci, H. Hassan, V. van der Veen, O. Mutlu, C. Giuffrida, H. Bos, and K. Razavi, "Trrespass: Exploiting the many sides of target row refresh," in *2020 IEEE Symposium on Security and Privacy, SP 2020, San Francisco, CA, USA, May 18-21, 2020*. IEEE, 2020, pp. 747–762. [Online]. Available: https://doi.org/10.1109/SP40000.2020.00090

[19] FuturePlus Systems, "The DDR Detective," http://futureplus.com/datasheets/FS2800A%20DDR%20Detective%20Data%20Sheet.pdf, 2018.

[20] M. Gottscho, M. Shoaib, S. Govindan, B. Sharma, D. Wang, and P. Gupta, "Measuring the impact of memory errors on application performance," *IEEE Comput. Archit. Lett.*, vol. 16, no. 1, pp. 51–55, 2017. [Online]. Available: https://doi.org/10.1109/LCA.2016.2599513

[21] D. Gruss, M. Lipp, M. Schwarz, D. Genkin, J. Juffinger, S. O'Connell, W. Schoechl, and Y. Yarom, "Another flip in the wall of rowhammer defenses," in *2018 IEEE Symposium on Security and Privacy, SP 2018, Proceedings, 21-23 May 2018, San Francisco, California, USA*. IEEE Computer Society, 2018, pp. 245–261. [Online]. Available: https://doi.org/10.1109/SP.2018.00031

[22] D. Gruss, C. Maurice, and S. Mangard, "Rowhammer.js: A remote software-induced fault attack in javascript," in *Detection of Intrusions and Malware, and Vulnerability Assessment - 13th International Conference, DIMVA 2016, San Sebastián, Spain, July 7-8, 2016, Proceedings*, ser. Lecture Notes in Computer Science, J. Caballero, U. Zurutuza, and R. J. Rodríguez, Eds., vol. 9721. Springer, 2016, pp. 300–321. [Online]. Available: https://doi.org/10.1007/978-3-319-40667-1_15

[23] H. Hassan, N. Vijaykumar, S. M. Khan, S. Ghose, K. K. Chang, G. Pekhimenko, D. Lee, O. Ergin, and O. Mutlu, "Softmc: A flexible and practical open-source infrastructure for enabling experimental DRAM studies," in *2017 IEEE International Symposium on High Performance Computer Architecture, HPCA 2017, Austin, TX, USA, February 4-8, 2017*. IEEE Computer Society, 2017, pp. 241–252. [Online]. Available: https://doi.org/10.1109/HPCA.2017.62

[24] Y. Huang, L. Chen, Z. Cui, Y. Ruan, Y. Bao, M. Chen, and N. Sun, "HMTT: A hybrid hardware/software tracing system for bridging the DRAM access trace's semantic gap," *ACM Trans. Archit. Code Optim.*, vol. 11, no. 1, pp. 7:1–7:25, 2014. [Online]. Available: https://doi.org/10.1145/2579668

[25] Y. Jang, J. Lee, S. Lee, and T. Kim, "Sgx-bomb: Locking down the processor via rowhammer attack," in *Proceedings of the 2nd Workshop on System Software for Trusted Execution, SysTEX@SOSP 2017, Shanghai, China, October 28, 2017*. ACM, 2017, pp. 5:1–5:6. [Online]. Available: https://doi.org/10.1145/3152701.3152709

[26] S. Ji, Y. Ko, S. Oh, and J. Kim, "Pinpoint rowhammer: Suppressing unwanted bit flips on rowhammer attacks," in *Proceedings of the 2019 ACM Asia Conference on Computer and Communications Security, AsiaCCS 2019, Auckland, New Zealand, July 09-12, 2019*, S. D. Galbraith, G. Russello, W. Susilo, D. Gollmann, E. Kirda, and Z. Liang, Eds. ACM, 2019, pp. 549–560. [Online]. Available: https://doi.org/10.1145/3321705.3329811

[27] T. Jiang, Q. Zhang, R. Hou, L. Chai, S. A. McKee, Z. Jia, and N. Sun, "Understanding the behavior of in-memory computing workloads," in *2014 IEEE International Symposium on Workload Characterization, IISWC 2014, Raleigh, NC, USA, October 26-28,*

*2014*. IEEE Computer Society, 2014, pp. 22–30. [Online]. Available: https://doi.org/10.1109/IISWC.2014.6983036

[28] S. M. Khan, C. Wilkerson, Z. Wang, A. R. Alameldeen, D. Lee, and O. Mutlu, "Detecting and mitigating data-dependent DRAM failures by exploiting current memory content," in *Proceedings of the 50th Annual IEEE/ACM International Symposium on Microarchitecture, MICRO 2017, Cambridge, MA, USA, October 14-18, 2017*, H. C. Hunter, J. Moreno, J. S. Emer, and D. Sánchez, Eds. ACM, 2017, pp. 27–40. [Online]. Available: https://doi.org/10.1145/3123939.3123945

[29] J. S. Kim, M. Patel, A. G. Yaglikçi, H. Hassan, R. Azizi, L. Orosa, and O. Mutlu, "Revisiting rowhammer: An experimental analysis of modern DRAM devices and mitigation techniques," in *47th ACM/IEEE Annual International Symposium on Computer Architecture, ISCA 2020, Valencia, Spain, May 30 - June 3, 2020*. IEEE, 2020, pp. 638–651. [Online]. Available: https://doi.org/10.1109/ISCA45697.2020.00059

[30] Y. Kim, R. Daly, J. S. Kim, C. Fallin, J. 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 *ACM/IEEE 41st International Symposium on Computer Architecture, ISCA 2014, Minneapolis, MN, USA, June 14-18, 2014*. IEEE Computer Society, 2014, pp. 361–372. [Online]. Available: https://doi.org/10.1109/ISCA.2014.6853210

[31] A. Kwong, D. Genkin, D. Gruss, and Y. Yarom, "Rambleed: Reading bits in memory without accessing them," in *2020 IEEE Symposium on Security and Privacy, SP 2020, San Francisco, CA, USA, May 18-21, 2020*. IEEE, 2020, pp. 695–711. [Online]. Available: https://doi.org/10.1109/SP40000.2020.00020

[32] M. Lanteigne, "How Rowhammer Could be Used to Exploit Weaknesses in Computer Hardware," http://www.thirdio.com/rowhammer.pdf, 2016.

[33] D. Lee, D. Jung, I. T. Fang, C. Tsai, and R. A. Popa, "An off-chip attack on hardware enclaves via the memory bus," in *29th USENIX Security Symposium, USENIX Security 2020, August 12-14, 2020*, S. Capkun and F. Roesner, Eds. USENIX Association, 2020, pp. 487–504. [Online]. Available: https://www.usenix.org/conference/usenixsecurity20/presentation/lee-dayeol

[34] M. Lipp, M. Schwarz, L. Raab, L. Lamster, M. T. Aga, C. Maurice, and D. Gruss, "Nethammer: Inducing rowhammer faults through network requests," in *IEEE European Symposium on Security and Privacy Workshops, EuroS&P Workshops 2020, Genoa, Italy, September 7-11, 2020*. IEEE, 2020, pp. 710–719. [Online]. Available: https://doi.org/10.1109/EuroSPW51379.2020.00102

[35] K. Loughlin, S. Saroiu, A. Wolman, and B. Kasikci, "Stop! hammer time: rethinking our approach to rowhammer mitigations," in *HotOS '21: Workshop on Hot Topics in Operating Systems, Ann Arbor, Michigan, USA, June, 1-3, 2021*, S. Angel, B. Kasikci, and E. Kohler, Eds. ACM, 2021, pp. 88–95. [Online]. Available: https://doi.org/10.1145/3458336.3465295

[36] O. Mutlu, "The rowhammer problem and other issues we may face as memory becomes denser," in *Design, Automation & Test in Europe Conference & Exhibition, DATE 2017, Lausanne, Switzerland, March 27-31, 2017*, D. Atienza and G. D. Natale, Eds. IEEE, 2017, pp. 1116–1121. [Online]. Available: https://doi.org/10.23919/DATE.2017.7927156

[37] P. Pessl, D. Gruss, C. Maurice, M. Schwarz, and S. Mangard, "DRAMA: exploiting DRAM addressing for cross-cpu attacks," in *25th USENIX Security Symposium, USENIX Security 16, Austin, TX, USA, August 10-12, 2016*, T. Holz and S. Savage, Eds. USENIX Association, 2016, pp. 565–581. [Online]. Available: https://www.usenix.org/conference/usenixsecurity16/technical-sessions/presentation/pessl

[38] R. Qiao and M. Seaborn, "A new approach for rowhammer attacks," in *2016 IEEE International Symposium on Hardware Oriented Security and Trust, HOST 2016, McLean, VA, USA, May 3-5, 2016*, W. H. Robinson, S. Bhunia, and R. Kastner, Eds. IEEE Computer Society, 2016, pp. 161–166. [Online]. Available: https://doi.org/10.1109/HST.2016.7495576

[39] K. Razavi, B. Gras, E. Bosman, B. Preneel, C. Giuffrida, and H. Bos, "Flip feng shui: Hammering a needle in the software stack," in *25th USENIX Security Symposium, USENIX Security 16, Austin, TX, USA, August 10-12, 2016*, T. Holz and S. Savage, Eds. USENIX Association, 2016, pp. 1–18. [Online]. Available: https://www.usenix.org/conference/usenixsecurity16/technical-sessions/presentation/razavi

[40] M. Schwarz, "DRAMA: Exploiting DRAM Buffers for Fun and Profit," *Graz University Technology, Master's Thesis*, 2016.

[41] M. Seaborn and T. Dullien, "Exploiting the DRAM rowhammer bug to gain kernel privileges," *Black Hat*, pp. 7–9, 2015.

[42] A. Tatar, C. Giuffrida, H. Bos, and K. Razavi, "Defeating software mitigations against rowhammer: A surgical precision hammer," in *Research in Attacks, Intrusions, and Defenses - 21st International Symposium, RAID 2018, Heraklion, Crete, Greece, September 10-12, 2018, Proceedings*, ser. Lecture Notes in Computer Science, M. Bailey, T. Holz, M. Stamatogiannakis, and S. Ioannidis, Eds., vol. 11050. Springer, 2018, pp. 47–66. [Online]. Available: https://doi.org/10.1007/978-3-030-00470-5_3

[43] A. Tatar, R. K. Konoth, E. Athanasopoulos, C. Giuffrida, H. Bos, and K. Razavi, "Throwhammer: Rowhammer attacks over the network and defenses," in *2018 USENIX Annual Technical Conference, USENIX ATC 2018, Boston, MA, USA, July 11-13, 2018*, H. S. Gunawi and B. Reed, Eds. USENIX Association, 2018, pp. 213–226. [Online]. Available: https://www.usenix.org/conference/atc18/presentation/tatar

[44] A. J. van de Goor and I. Schanstra, "Address and data scrambling: Causes and impact on memory tests," in *1st IEEE International Workshop on Electronic Design, Test and Applications (DELTA 2002), 29-31 January 2002, Christchurch, New Zealand*. IEEE Computer Society, 2002, pp. 128–136. [Online]. Available: https://doi.org/10.1109/DELTA.2002.994601

[45] 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 *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security, Vienna, Austria, October 24-28, 2016*, E. R. Weippl, S. Katzenbeisser, C. Kruegel, A. C. Myers, and S. Halevi, Eds. ACM, 2016, pp. 1675–1689. [Online]. Available: https://doi.org/10.1145/2976749.2978406

[46] V. van der Veen, M. Lindorfer, Y. Fratantonio, H. P. Pillai, G. Vigna, C. Kruegel, H. Bos, and K. Razavi, "Guardion: Practical mitigation of dma-based rowhammer attacks on ARM," in *Detection of Intrusions*

*and Malware, and Vulnerability Assessment - 15th International Conference, DIMVA 2018, Saclay, France, June 28-29, 2018, Proceedings*, ser. Lecture Notes in Computer Science, C. Giuffrida, S. Bardin, and G. Blanc, Eds., vol. 10885. Springer, 2018, pp. 92–113. [Online]. Available: https://doi.org/10.1007/978-3-319-93411-2_5

[47] Y. Xiao, X. Zhang, Y. Zhang, and R. Teodorescu, "One bit flips, one cloud flops: Cross-vm row hammer attacks and privilege escalation," in *25th USENIX Security Symposium, USENIX Security 16, Austin, TX, USA, August 10-12, 2016*, T. Holz and S. Savage, Eds. USENIX Association, 2016, pp. 19–35. [Online]. Available: https://www.usenix.org/conference/usenixsecurity16/technical-sessions/presentation/xiao