HammerScope: Observing DRAM Power Consumption Using Rowhammer

Yaakov Cohen*
Ben-Gurion University of the Negev and Intel
Beer-Sheva, Israel
yaakoc@post.bgu.ac.il

Kevin Sam Tharayil*
Georgia Institute of Technology
Atlanta, Georgia, USA
kevinsam@gatech.edu

Arie Haenel*
Jerusalem College of Technology and Intel
Jerusalem, Israel
arie.haenel@jct.ac.il

Daniel Genkin
Georgia Institute of Technology
Atlanta, Georgia, USA
genkin@gatech.edu

Angelos D. Keromytis
Georgia Institute of Technology
Atlanta, Georgia, USA
angelos@gatech.edu

Yossi Oren
Ben-Gurion University of the Negev and Intel
Beer-Sheva, Israel
yos@bgu.ac.il

Yuval Yarom
University of Adelaide
Adelaide, Australia
yval@cs.adelaide.edu.au

Abstract

The constant reduction in memory cell sizes has increased memory density and reduced power consumption, but has also affected its reliability. The Rowhammer attack exploits this reduced reliability to induce bit flips in memory, without directly accessing these bits. Most Rowhammer attacks target software integrity, but some recent attacks demonstrated its use for compromising confidentiality.

Continuing this trend, in this paper we observe that the Rowhammer attack strongly correlates with the memory instantaneous power consumption. We exploit this observation to design HammerScope, a Rowhammer-based attack technique for measuring the power consumption of the memory unit. Because the power consumption correlates with the level of activity of the memory, HammerScope allows an attacker to infer memory activity.

To demonstrate the offensive capabilities of HammerScope, we use it to mount three information leakage attacks. We first show that HammerScope can be used to break kernel address-space layout randomization (KASLR). Our second attack uses memory activity as a covert channel for a Spectre attack, allowing us to leak information from the operating system kernel. Finally, we demonstrate the use of HammerScope for performing website fingerprinting, compromising user privacy. Our work demonstrates the importance of finding systematic solutions for Rowhammer attacks.

CCS Concepts

• **Hardware → Dynamic memory**; • **Security and privacy → Side-channel analysis and countermeasures**.

*All student authors contributed equally to this paper

Keywords

Side-channel attack, Rowhammer

ACM Reference Format:
Yaakov Cohen, Kevin Sam Tharayil, Arie Haenel, Daniel Genkin, Angelos D. Keromytis, Yossi Oren, and Yuval Yarom. 2022. HammerScope: Observing DRAM Power Consumption Using Rowhammer. In *Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security (CCS '22), November 7–11, 2022, Los Angeles, CA, USA.* ACM, New York, NY, USA, 15 pages. https://doi.org/10.1145/3548606.3560688

1 Introduction

Rowhammer is a hardware vulnerability affecting nearly all modern systems equipped with dynamic random access memory (DRAM). At a high level, by performing certain carefully-orchestrated memory access patterns, an attacker can flip bits in memory, without directly writing to these bits. From a security perspective, Rowhammer is typically viewed as a (restricted) write primitive, where an attacker can modify the contents of memory they are prohibited from accessing.

Ever since its first public disclosure in 2014 [30], and consequent introduction to the security research community in 2015 [50], the Rowhammer fault attack is a source of considerable academic interest. In particular, Rowhammer's ability to reliably flip bits across security boundaries has been exploited for sandbox escapes [24, 50], privilege escalation attacks on operating systems and hypervisors [22, 24, 47, 50, 56, 61, 65], denial-of-service attacks [22, 26], and even for fault injection in cryptographic protocols [4]. More recently, Rowhammer attacks have been demonstrated against hardware-based Rowhammer countermeasures, such as targeted row refresh [17] and error correcting codes (ECC-RAM) [10].

While each of these works may be harmful from a security perspective, a common trend remains. In nearly all Rowhammer-based exploits the attacker is actively trying to break the system's data integrity by flipping bits they cannot otherwise access, usually due to software isolation mechanisms. Much less is known about other security implications of Rowhammer, such as data confidentiality and authenticity.

A first indication that the effects of Rowhammer span beyond just integrity was given by RAMBleed [34], which showed that data-dependent bit flips can be used to read the contents of memory cells, as opposed to flipping bits in them. More recently, SpecHammer [54] demonstrated that Rowhammer can be used to enhance Spectre, again leaking data across security boundaries. With data confidentiality being a new front for Rowhammer research, in this paper we ask the following question:

Can Rowhammer be used for additional confidentiality applications? What will it take for an attacker to mount such attacks and what information can be leaked with them?

1.1 Our Contribution

We present HammerScope, a new technique for using Rowhammer to measure the power consumption of memory modules.

A Changing Threshold. To flip a specific victim bit using Rowhammer, an attacker must activate nearby aggressor rows a certain number of times in quick succession. We study the activation threshold required for a successful Rowhammer-induced bit flip, and discover that it correlates with the power consumption of the hammered memory module. In particular, the higher the power consumption, the more row activations are required to flip a bit.

Constructing HammerScope. Armed with this observation we construct HammerScope, a side-channel attack which measures the power consumption of direct in-line memory modules (DIMMs) using only unprivileged software. At a high level, HammerScope continuously Rowhammers a targeted bit, noting how many activations are required to successfully flip it. As the activation threshold correlates with the DIMM's power consumption, HammerScope can trace the power consumption as it changes over time. Thus, HammerScope effectively turns Rowhammer into a tool which allows attackers to monitor the DIMM's power consumption via unprivileged software, without requiring any additional equipment.

Benchmarking and Analyzing HammerScope. Benchmarking HammerScope across six systems, we show that it can track the DIMM's power consumption on both DDR3 and DDR4 memory. Ironically, we show that doubling the row refresh period on modern DDR4 modules also doubles HammerScope's sampling rate, allowing for more accurate power measurements. Finally, we hypothesize about the mechanism behind HammerScope and propose several explanations for the phenomenon under the parasitic capacitance model.

Mounting Rowhammer-based Power Analysis Attacks. As a final contribution, we show that Rowhammer has privacy implications beyond fault attacks. We show how recording the DIMM's power consumption can be used for accurate instruction timing, derandomizing KASLR, mounting power-analysis-based Spectre attacks, and even for website fingerprinting. We demonstrate that all of these use cases are possible via HammerScope, and show that HammerScope provides a vector for mounting power analysis attacks using software-only means.

Summary of Contributions. In summary, in this paper we make the following contributions:

- We formalize and characterize the behavior of Rowhammer fault attacks under varying power conditions, and show that there is a distinct relationship between DRAM susceptibility to Rowhammer bit flips and its power consumption (Sections 4.1 to 4.4).
- We introduce a new attack technique, called HammerScope, and show that it can be applied to multiple desktop and laptop computers with DDR3 and DDR4 memories from multiple vendors (Section 4.5).
- We apply HammerScope to carry out a series of end-to-end attacks, which can break KASLR, leak memory from the kernel, and perform website fingerprinting (Section 5).

1.2 Artifact Availability and Responsible Disclosure

We will release all the artifacts related to this project as open source, including code for synchronized Rowhammer, implementations of our feedback-based and threshold-based HammerScope attack methods, and the code for our end-to-end attack on KASLR. In addition, we will release datasets containing to our website fingerprinting traces, collected both using HammerScope and using RAPL, together with the machine learning pipeline used for classification. A subset of the artifacts is already available in the repository found at https://github.com/hammerscope/artifacts.

We provided an early version of this paper to contacts at Intel, AMD, Google and HP. We also shared our paper with JEDEC JC-42 committee, which contains representatives from various DRAM manufacturers and other hardware vendors.

2 Background

In this section we explain some key concepts and previous related works associated with HammerScope.

2.1 DRAM

Dynamic random-access memory (DRAM) is used as the main volatile memory for most desktops, servers and mobile devices. The fundamental unit of storage in DRAM is a DRAM cell, which stores one bit of data using a capacitor.

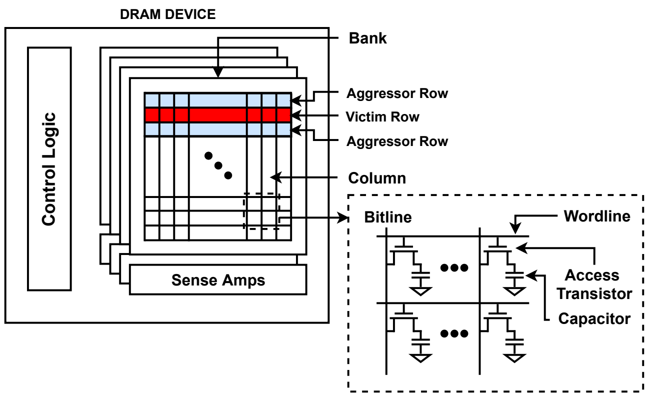


Figure 1: Structure of a DRAM chip. Taken from [3]

As Figure 1 shows, these cells are typically arranged into rows consisting of multiple cells in parallel. Each cell is connected to a vertical *bit line* through an access transistor [28]. Each of these bit lines are shared by multiple rows of DRAM cells. The access transistors themselves are controlled through *word lines*, which are arranged horizontally and shared by multiple columns.

Accessing data from DRAM. The CPU cannot access data stored in the capacitors directly. Instead, it interfaces with the cells using an on-DRAM element called the *row buffer*, using a three-step protocol: To read from a DRAM cell, the DRAM controller of the CPU first issues an Activate command. This command toggles the row's word line, activating all access transistors in the row and connecting the cells' storage capacitors to the bit line. This transfers the contents of the capacitors in the row containing the cell into the row buffer, where they are converted to discrete digital values using a sense amplifier circuit. Next, the DRAM controller issues a Read command, moving data from the sense amplifiers to the DRAM bus. Finally, the DRAM controller issues a Precharge command, which restores the values of the row buffer back into the DRAM cells.

Refresh command. To prevent decay due to capacitors losing their charge over time, DRAM chips must be periodically refreshed. During the refresh operation, the DRAM reads a single row into the row buffer and immediately writes it back into the cells. The CPU, and in some cases the DRAM controller itself, performs a refresh operation every few microseconds, iterating over all the rows in a set pattern. The nominal refresh period of desktop-class DRAM chips is 64 milliseconds, and a typical DRAM chip has 8192 rows, which means a row refresh operation is scheduled approximately every 8 microseconds.

Charge leakage. Writing a value to a DRAM cell sets the charge stored in the cell's capacitor to a fixed value, either to the DRAM supply voltage V_{DD} or to 0. For clarity, the rest of the discussion assumes the case of writing V_{DD} to the cell. This charge decays over time, due to the cell capacitor's intrinsic leakage. If it decays to within a certain distance from $V_{DD}/2$, the sense amplifier will not be able to correctly recover the value originally stored in DRAM, causing a potential bit flip. While the intrinsic leakage is simply caused by the passage of time, there is another type of leakage, called coupling leakage, which occurs when the target cell's neighboring rows are activated. This coupling leakage is the foundation of the Rowhammer effect.

2.2 Rowhammer

Rowhammer is a fault attack on DRAM cells, first introduced by Kim et al. [30] and further investigated by Jiang et al. [28]. The Rowhammer attack builds on the observation that activating a row of bits in DRAM depletes the charge of nearby rows due to coupling effects. As a result, an adversary who maliciously accesses DRAM in a controlled manner, causing a row to activate and deactivate multiple times in a single refresh cycle, can cause bits in an adjacent row to flip. We refer to bits vulnerable to row-hammering as *flippy* and to the rows which are activated to flip a bit as *aggressor rows.*

Rowhammer attack methodology. A typical Rowhammer attack consists of several steps. The adversary first reverse engineers the mapping of virtual addresses in its address space to physical memory addresses and from those to DRAM locations in terms of rows, banks, and ranks. The adversary then profiles the memory to discover flippy bits. Next, the adversary *massages* memory to bring a vulnerable victim memory location to map to a flippy bit. Finally, the adversary repeatedly activates the aggressor rows to flip the target bit. This final step is challenging because the adversary needs to bypass the cache hierarchy to ensure that memory accesses are served from DRAM.

Security impact of Rowhammer Past works proposed multiple ways of leveraging this attack to produce a security impact. For example, Razavi et al. [47] use Rowhammer to break OpenSSH public-key authentication and forge GPG signatures, Xiao et al. [61] show how it can break the memory isolation in Xen and access arbitrary physical memory on a shared machine, and Kwong et al. [34] show how to extract an RSA key from an OpenSSH daemon. Rowhammer attacks have been demonstrated in many settings: from JavaScript [12, 24], over the network [39, 53], through heterogeneous FPGA-based systems [60], and even using GPUs [16].

Rowhammer defense. Multiple works proposed detecting and preventing Rowhammer attacks using a combination of hardware and software mechanisms [1, 5, 29, 35, 51]. DRAM vendors also implemented countermeasures for Rowhammer in their own modules. The usual approach taken by most vendors is *targeted row refresh* (TRR). In TRR, the DRAM module monitors which rows are activated the most and preemptively issues extra row refresh commands to neighboring rows. Yet, many implementations of TRR are vulnerable to dedicated activation patterns that bypass it [12, 17, 25, 27, 32].

Minimum activation count. One performance metric of Rowhammer which is important to this work is the *minimum activation count* AC_{min} – how many times must the attacker activate the aggressor rows before a particular bit flips. While the first published attacks used millions of activations, newer attacks require as few as 6,000 activations for some memory variants [25]. As we show in our work, the minimum activation count differs between individual cells in the same DRAM module, and it acts as a sharp threshold – when properly configured, Rowhammer attacks which perform less than AC_{min} row activations will always fail, whereas performing slightly more than AC_{min} activations will flip the bit with a very high probability.

2.3 Power Analysis

Power analysis is one of the most well-established forms of side-channel attack [15, 31]. It exploits the dependency correlation between the instantaneous power consumption of CMOS devices and the internal state changes of these devices over time. These state changes depend on the instructions and data that are being processed [41]. Consequently, an adversary who can monitor the power consumption of a device can learn about its internal state. To set up a power analysis attack, the attacker traditionally places measurement probes between the device under test (DUT) and its power supply, and collects the instantaneous current traveling across the probes using a deep-memory oscilloscope. Due to the required physical access, such attacks are considered a practical threat only when the attacker has physical control of the DUT.

Reducing proximity requirement. Replacing the power probes with near-field electromagnetic probes [18] can relax the requirement for a physical contact, allowing attacks from within a few millimeters from DUT. A further relaxation can be achieved by measuring power consumption modulations over far-field electromagnetic waves, extending the range to a few meters [6, 45].

Genkin et al. [20] exploited power-related fluctuations of a laptop's ground line, accessible by touching exposed metal on the computer's chassis or the remote end of Ethernet, VGA or USB cables. Despite the increase of range, these setups are still fundamentally similar to contact-based attacks, since they require the attacker to be physically adjacent to the DUT.

Removing Proximity Requirement. A more dramatic increase in attack distance is achieved by causing a DUT to measure its own power consumption and report it to the attacker. For example, an on-board analog to digital converter (ADC) circuit can be used to measure power consumption, allowing untrusted code to extract secret keys from a TrustZone secure element [43]. In the case of FPGA-based devices, Zick et al. [66] showed how reconfigurable logic can make an FPGA measure its own supply voltage, and Schellenberg et al. [48] showed how this primitive can be built into a Trojan that enables a remote power analysis attack. Genkin et al. [19] showed how a PC's internal microphone can pick up power-related emanations, allowing an adversary with remote access to the microphone (for example, during a teleconference) to recover secret information. In the domain of personal computers, the Platypus attack showed how an adversary with access to the running average power level (RAPL) register on Intel [38] and AMD [37] processors can perform power analysis attacks. Since Platypus' original discovery, access to the RAPL interface has been limited to privileged software, thereby mitigating this attack vector.

3 Attack Model and Experimental Setup

We assume a user-level adversary, which is capable of performing repeated Rowhammer attacks on a target machine, as well as observing whether the attacks are successful. We explicitly note that we assume no additional CPU or OS support for reporting power consumption, battery levels, RF emanations, or any other power consumption artifacts. We also do not require root-level access, performing our HammerScope measurements using only user-level permissions. Finally, unlike many Rowhammer attacks, we do not make any assumptions regarding the victim's memory location and do not depend on "memory massaging" techniques.

Experimental setup. We evaluated our attack on a variety of machines, as shown in Table 1, spanning multiple CPU generations and both DDR3 and DDR4 modules. In the DDR3 machines, we used code from the HammerTime software suite [52] to recover the mapping between physical memory and DRAM layout, then applied a standard double-sided Rowhammer attack to flip vulnerable bits. For the DDR4 machines, we used code from the TRRespass artifact repository [17] to recover the mapping, then used 9-sided Rowhammer to flip bits. To convert the code of [17] to work in user mode, we applied the methods of Kwong et al. [34] and Tobah et al. [54] to force the Linux allocator to allocate a contiguous 2 MB block, and then applied the time measurement method of Pessl et al. [46] to find consecutive DRAM rows in this block.

4 HammerScope

As outlined above, the HammerScope attack is based on observing a correlation between DRAM power consumption and number of Rowhammer attempts required for obtaining a bit flip. More specifically, we observe that as the DIMM's power consumption increases, bits in the DIMM become harder to flip, requiring more

CPU Model	DRAM Type	DRAM Vendor	Kernel Version	OS Version
Core i7-4770	DDR3-1	Samsung	5.13.0	Ubuntu 20.04.1
Core i7-4790 (1)	DDR3-2	Kingston	5.9.0	Debian 5.9.1
Core i7-4790 (2)	DDR3-3	Kingston	5.9.0	Debian 5.9.1
Core i7-6600U	DDR4-1	Samsung	5.4.1	Ubuntu 18.04.6
Core i7-7700	DDR4-2	Samsung	5.4.1	Ubuntu 18.04.6
Core i7-10700K	DDR4-3	Samsung	5.13.0	Ubuntu 20.04.3
Core i7-8700K	DDR4-4	Samsung	5.15.0	Ubuntu 20.04.2

Table 1: Test Machine Specifications

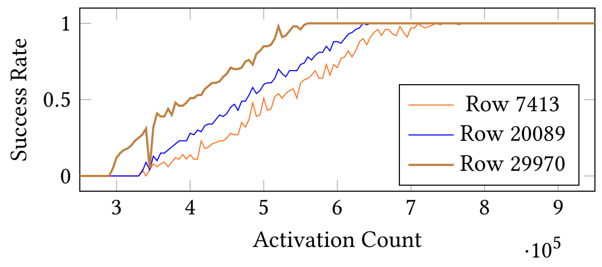


Figure 2: Activation count for three different rows (DDR3)

Rowhammer attempts. Finally, as the DIMM's power consumption correlates with system activity, we can use Rowhammer as a power-based side channel to monitor various system events.

4.1 Establishing the Activation Threshold

As HammerScope relies on using Rowhammer to observe DRAM power consumption, our first task is to establish a baseline of the number of row activations required to flip a bit. To that aim, we first carried out a Rowhammer profiling step on a DDR3 memory module (Table 1 DDR3-2), identifying several rows that contain flippy bits. For each of these rows, we measured the probability that it would experience a bit flip as a function of the number of row activations. We measured 150 different activation counts between 250,000 and 995,000, repeatedly attempting to flip a bit 100 times for each activation count.

Figure 2 shows the results of the experiment. Specifically, when the number of activations is below some minimal threshold AC_{min} there are no Rowhammer bit flips. Moreover, the threshold depends on the selected vulnerable row. Finally, we observe that the probability of Rowhammer bit flips when the activation count is close to the activation threshold is low, which adds noise to our measurement.

4.2 Observing System Activity

Having established the value of AC_{min} across different rows for an idle system, we now demonstrate how the number of row activation required to flip a bit can be used to monitor the system's memory activity. To that aim, we run a program that repeatedly writes to DRAM for 30 seconds and then leaves the memory idle for 10 seconds. In parallel to running our program, we repeatedly compute the number of row activation required for flipping a pre-selected Rowhammer vulnerable bit. Figure 3 presents the number of row activations required as a function of time. As can be seen, the number of row activations required to flip the targeted bit varies, and is low during idle periods (white background), and high during times of memory activity (gray background).

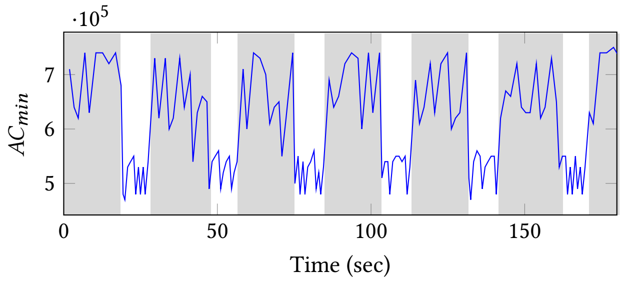


Figure 3: Variation in AC_{min} while memory is idle and while it is accessed.

4.3 Reducing Noise

While the signal depicted in Figure 3 clearly allows us to distinguish between regions of high and low memory activity, the signal observed is still noisy. This is due to the low success probability of Rowhammer with only AC_{min} activations (Figure 2), which prevents HammerScope from accurately measuring the DIMM's power consumption. In this section we show how to denoise the signal observed via HammerScope, by identifying the best time to start a Rowhammer attempt for a given vulnerable bit.

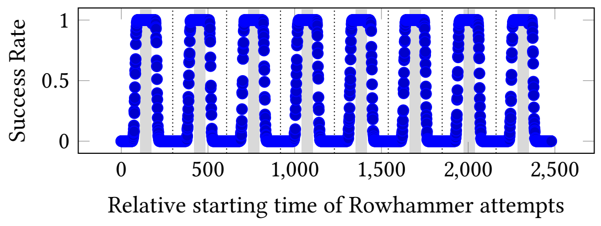
The Mechanics of Row Activations. We first recall that DRAM cells are capacitors, which leak charge over time, resulting in bit errors. To avoid loss of data, the memory controller periodically issues Refresh commands, which replenish the charge in all the capacitors of a row. We use the term refresh period to refer to the time between two successive Refresh commands to the same row.

Increasing Rowhammer Success Probability. As Rowhammer activations increase the rate of charge depletion from neighboring rows, we hypothesize that a sequence of Rowhammer attacks which fits within a row's refresh period, and is not interrupted by a Refresh command, will have a higher probability of flipping a vulnerable bit in that row. In other words, each row has a *vulnerable window* such that Rowhammer attacks started within the window have a higher likelihood of being successful compared to attacks that start outside the window, presumably because the attack can flip the bit before the subsequent Refresh command.

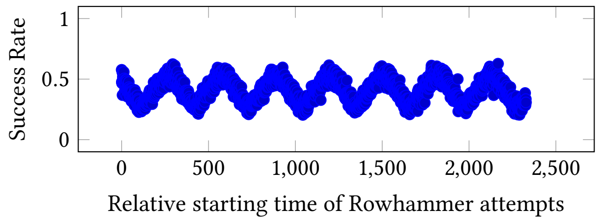
To find this vulnerable window, we first carry out multiple Rowhammer attempts on a chosen row, where each attempt consists of a fixed number of row activations. For each attempt i, we record the starting time, T_i, and whether the attempt is successful (i.e., results in a bit flip). We then use the collected information to find the refresh period, and from it the row's vulnerable window.

Identifying Refresh Period. To identify the refresh period, we search a space of candidates spanning all possible refresh periods between 20 ms and 65 ms with a resolution of 1 nanosecond, resulting in a search space of 45 million possibilities. For each candidate T_R, we test if whether the start times T_i of hammering successes fall within a fixed window relative to the candidate refresh period, as we describe below.

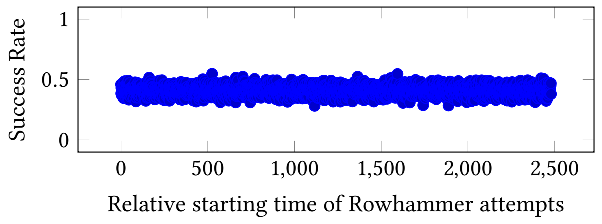
For each Rowhammer attempt i with a starting time T_i, we compute the starting time relative to a candidate T_R period, by $T_i' = T_i \bmod T_R$. If we picked the correct value of T_R, the T_i's of different refresh windows will align, and we will see a well-defined region of high success rate.



(a) Correct refresh period. Shaded areas indicate vulnerable windows, and dotted lines indicate (conjectured) locations of refresh commands.



(b) Closer to the correct refresh period



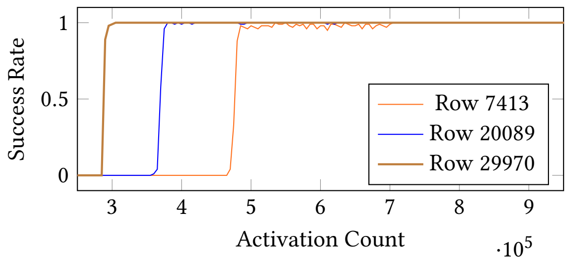
(c) Wrong refresh period

Figure 4: Finding the Rowhammer vulnerable window on DDR3

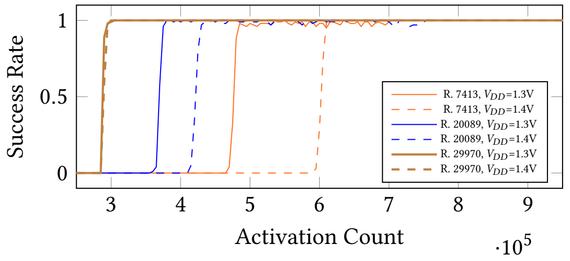
Observing Refresh Periods. Figure 4 shows this effect, repeated over 8 consecutive refresh periods for the purpose of illustration. For the correct value of T_R (Figure 4a), Rowhammer attempts that begin within a well-defined window between consecutive Refresh commands are typically successful, while attempts outside the window usually fail. For incorrect values of the refresh period (Figures 4b and 4c), this effect does not happen, as the starting times of successful Rowhammer attempts are no longer at the same location relative to the (incorrectly guessed) refresh period. The value of T_R which produces the narrowest regions of high Rowhammer success rates is thus the correct refresh period.

Identifying the Rowhammer Vulnerable Window. In addition to computing the refresh rate, Figure 4a also identifies the vulnerable windows, which correspond to time regions with a high Rowhammer success rate despite only using AC_{min} activations. More specifically, inspecting the gray-shaded areas in Figure 4a, we observe that there is an interval (a, b) relative to the refresh command (dotted line), such that Rowhammer attacks starting within such an interval have a high likelihood of success. In other words, the row's vulnerable window spans all times x such that $jT_R + a < x < jT_R + b$ for some $j \in \{0, 1, 2, \ldots\}$, making Rowhammer attacks performed during the window highly likely to succeed.

A Synchronized Rowhammer. With the locations of Rowhammer vulnerable windows in hand, we can now re-examine the success probability of different values of AC_{min} across different flippable bits. Indeed, Figure 5a presents the results of repeating the experiment of Figure 2, but this time having the Rowhammer attempts be synchronized to the Rowhammer vulnerable windows of the targeted rows. As can be seen, synchronization with the vulnerable window results in much sharper transitions between failing and successful Rowhammer attempts, which will significantly denoise the power signals measured by HammerScope.



(a) Synchronized with refresh period, Fixed Voltage



(b) Synchronized with refresh period, Variable Voltage

Figure 5: Hammering Success vs. Activation Count (DDR3)

Handling DDR4 Systems. Our technique for identifying the DIMM's refresh period and the rows' vulnerable windows applies not only to DDR3, but also to newer DDR4 systems. Figure 6 is the DDR4-3 counterpart of Figure 4a. As can be seen, correct values of T_R still produce distinct areas of high Rowhammer success rates, albeit with lower (about 50%) overall success rate compared to DDR3 systems. We hypothesize that this is because the 9-sided hammering method we used to attack this system was only partially successful in overcoming the module's TRR mitigations.

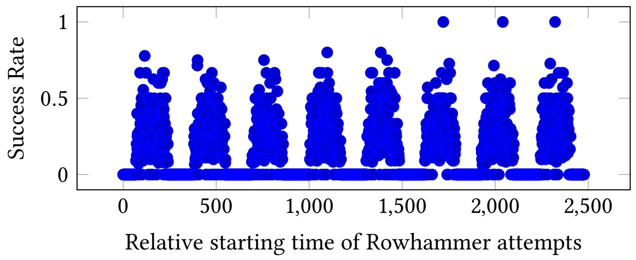


Figure 6: Finding the Rowhammer vulnerable window on DDR4

Observing Refresh Periods. Applying our methodology for finding the refresh period and vulnerable windows to our DDR3 and DDR4 experimental setups, we summarize our findings in Table 2. As the table shows, the measured refresh period was never equal to 64 ms, and varied between experiments on the same machine, depending on the DRAM's clock rate. Finally, like Hassan et al. [25], we find that the refresh periods of all of our DDR4 modules are below 32 ms, presumably to protect against Rowhammer.

DRAM Module	DRAM Frequency	Refresh Period (ms)	HammerScope Sampling Rate (Hz)
DDR3-1	1066 MHz	63.897672	15.65
DDR3-1	1333 MHz	42.598448	23.47
DDR3-2	1333 MHz	64.045484	15.61
DDR3-3	1400 MHz	42.614029	23.47
DDR3-3	1600 MHz	42.598448	23.47
DDR4-1	2133 MHz	30.529955	32.75
DDR4-2	2400 MHz	30.628924	32.65
DDR4-3	2400 MHz	30.611017	32.67
DDR4-4	2400 MHz	30.558066	32.72

Table 2: Measured refresh periods and corresponding sampling rate across different DRAM configurations.

4.4 Effect of Voltage on AC_{min}

Having obtained a methodology of accurately evaluating the value of AC_{min} for a targeted bit flip, we now proceed in observing how the value of AC_{min} changes as the function of the DIMM's supply voltage. To that aim, we repeat the experiment of Figure 5a and intentionally modify the DRAM supply voltage of the system from 1.4V to 1.3V[1]. For each voltage level, we subsequently measure the AC_{min} value across 3 rows, plotting our findings in Figure 5b.

As can be seen in Figure 5b, the value of AC_{min} for a given row is directly proportional to the DIMM's supply voltage, meaning that less row activations are required for lower voltages in order to obtain a successful bit flip. Besides indicating that DIMMs are more flippy when undervolted, the connection between AC_{min} and voltage suggests that we can use Rowhammer in order to obtain information about the DIMM's supply voltage. For example, looking at Figure 5b, attacking row 7413 with 500,000 activations has a 100% success rate in case V_{DD}=1.3V, and a 0% success rate if V_{DD}=1.4V. Thus, by measuring the amount of activations during a Rowhammer attack on row 7413, the attacker is able to distinguish between V_{DD}=1.4V and V_{DD}=1.3V.

4.5 Constructing HammerScope

Having observed a connection between DIMM supply voltage and the activation threshold, AC_{min}, required for a successful Rowhammer bit flip, we now describe how we use Rowhammer to measure the DIMM's power consumption.

A Naive Attempt. In theory, the most basic method for mounting HammerScope would be to wait for the next vulnerable window and start hammering the targeted bit, recording the number of activation counts until the bit flips. While this method sounds

[1]We note here that both 1.3V and 1.4V are technically below the recommended V_{DD} rating for DDR3, which is 1.425V to 1.575V. However, this undervolted condition was used only in this section for visual clarity, and all subsequent measurements and attacks used the recommended voltage range.

appealing, the act of reading the bit's current value also refreshes its capacitors, thus voiding the efficacy of further activation attempts.

Guess and Record. Instead, the attacker can first choose an activation count that is slightly higher than AC_{min} in the idle state. The attacker then performs a Rowhammer attack synchronized to the vulnerable window, checking if the targeted bit has flipped after reaching the chosen activation count. The attacker then records the outcome, and proceeds with another attack iteration.

The downside of this approach is that it produces binary values. A successful bit flip indicates that the system is in the idle state, while a failed bit flip suggests that the DRAM was active.

Closed-Loop Feedback. To further improve HammerScope's measurement resolution, we borrow a technique from control theory called Closed-Loop Feedback [14]. Here, the main idea is to still perform Guess and Record, but adaptively choose the value of AC_{min} for the next attack iteration based on the success (or failure) of the current attack iteration at flipping the targeted bit. As we essentially track the value of AC_{min} as it changes through time, this approach gives us a higher-sensitivity measurement.

More specifically, Figure 7 presents a pseudocode of our approach. In each iteration of the loop in Line 2, we first set the targeted victim row to a known state for Rowhammer (Line 3) and flush it from the CPU cache. We then pause, waiting for the next vulnerable window (Line 4). Once the vulnerable window time arrives, we perform a Rowhammer attack with a controlled number of activation_counts (Line 5) and record activation_counts in the trace. Finally, we check if the Rowhammer attempt was successful, updating activation_counts accordingly for the next attack iteration (Lines 7–10).

Using this method, we essentially track the value of AC_{min} as it changes over time, and use it as a proxy of the DIMM power consumption level. In particular, the output of this method is not binary, as in the previous approach, and allows a significantly finer resolution compared to the Guess and Record approach.

```
1  let activation_count = AC_START
2  while true:
3    reset_row()
4    wait_for_vulnerable_window() //As explained in Sec. 4.3
5    perform_rowhammer(activation_counts)
6    trace.append(activation_counts)
7    if bit_flipped():
8      activation_counts -= DELTA
9    else:
10     activation_counts += DELTA
```

Figure 7: The HammerScope control loop Algorithm.

4.6 Characterizing HammerScope

We now compare HammerScope with power measurements obtained via RAPL and Oscilloscope and discuss HammerScope's sampling rate, bandwidth and SNR.

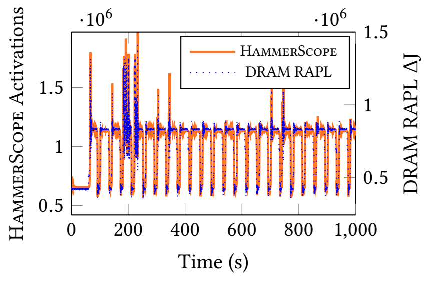
Comparing HammerScope with RAPL. To show the effectiveness of our method in measuring DRAM power consumption, we compare HammerScope results with the DIMM's power consumption as reported by the RAPL interface. RAPL is a mechanism for measuring and controlling the power consumption of individual components of Intel- and ARM-based machines. It provides, among other things, a real-time direct measurement of the power consumption of the DRAM subsystem, based on high resolution sampling of the integrated voltage regulator supplying power to DRAM [11, 13]. In this experiment, we run a program that repeatedly writes to the DRAM for 30 seconds, then remains idle for 10 seconds. We sample the DRAM power consumption using both the Intel RAPL interface and the control loop implementation of HammerScope on our DDR3-1 setup, at a sampling rate 16 Hz. See Figure 8a.

As the figure shows, both RAPL and the HammerScope measurements can accurately capture the DRAM activity. The RAPL and HammerScope measurements are highly correlated ($\rho = .97$, $p < .001$). Similar results were obtained on our DDR3-3 and DDR4-2 setups. See Figure 8b and Figure 8c respectively.

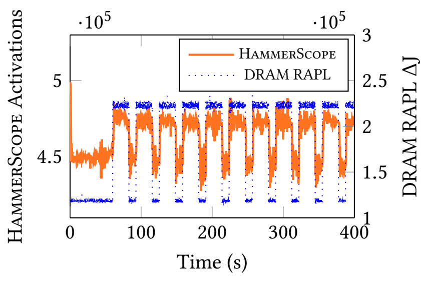
Comparing HammerScope with Oscilloscope Measurements. In addition to comparing HammerScope with measurements taken using the RAPL interface, we have also compared HammerScope with a direct physical measurement of the DRAM line using an oscilloscope on our DDR4-4 set up. To this aim, we connected a passive probe to the DRAM's V_{PP} supply line, and sampled the voltage level using a Picotech PicoScope 3425 deep memory oscilloscope at a sampling rate of 100 kHz while simultaneously taking both HammerScope and RAPL measurements. The computer executed a script which repetitively accessed memory for two seconds, and then remained idle for two seconds. Since our probe connection was in parallel to the voltage supply, and not in series, this physical measurement actually measured the transient voltage droops caused by changes in power consumption, and not the power consumption directly. To recover the power consumption trace, we applied a digital band pass filter centered at 33 kHz, followed by a rectifier and a low-pass filter. The results of this evaluation are shown in Figure 9. The oscilloscope reading is displayed in relative scale. As the figure indicates, Rowhammer measurements very accurately track both RAPL and oscilloscope measurements, albeit at a low sampling rate.

Calculating HammerScope's Sampling Rate. In order to execute HammerScope, the attacker performs exactly one measurement per DIMM refresh period, at the previously-identified vulnerable window. Thus, HammerScope' sampling rate is determined by the machines specific memory modules and their configuration, see Table 2. Among the different memory configurations we tested, we observed the highest sampling rate of 32.67 Hz (corresponding to a 32ms refresh period on DDR4) with the lowest rate being 15.65 (64 ms refresh interval on DDR3). In particular, the doubling of the DIMM's refresh rate on DDR4 machines appears to provide for a higher HammerScope sampling rate, improving the measurement resolution on modern machines. Finally, we note that while 32.67 Hz is quite a low sampling rate compared to other power analysis primitives, it is still sufficient for mounting attacks on realistic targets (see Section 5).

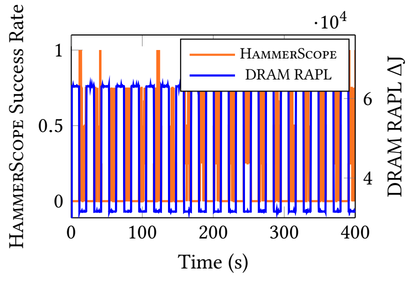
Evaluating HammerScope's Bandwidth. To characterize HammerScope's bandwidth (i.e., horizontal accuracy) we executed a program that repeatedly accesses DRAM and then sleeps for a predetermined time, thus generating a square wave signal in the DRAM's power consumption. Next, we measured whether the DRAM's RAPL interface and HammerScope were capable of tracking this signal. Our results indicate that both methods can track periodic signals at frequencies of up to 4 Hz. At a higher frequency, DRAM does not



(a) HammerScope vs. RAPL on DDR3-1



(b) HammerScope vs RAPL on DDR3-3



(c) HammerScope vs RAPL on DDR4-2

Figure 8: Comparing HammerScope and RAPL

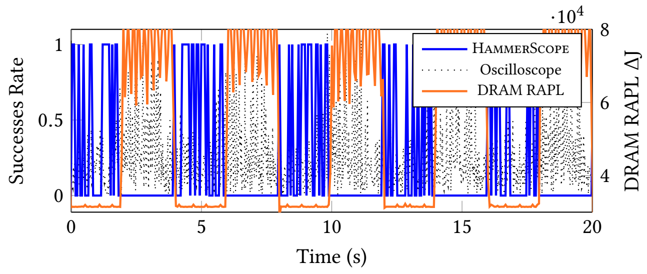


Figure 9: HammerScope vs RAPL vs Oscilloscope on DDR4-4

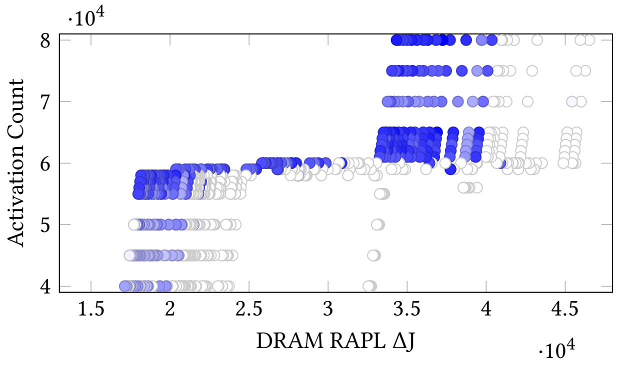


Figure 10: DRAM RAPL vs Rowhammer success rates on DDR4 (dark circles indicate success)

have sufficient time to recover from its high-power state, making both RAPL and HammerScope measurements unreliable.

Characterizing HammerScope's SNR. To identify HammerScope's Signal-to-Noise Ratio (SNR) (i.e., vertical accuracy), we executed a different test program that alternates between memory accesses and a controllable number of NOP instructions in a tight loop. This effectively creates a controlled power consumption using pulse width modulation (PWM), where the duty cycle of the modulation is controlled by the ratio between memory accesses and NOP instructions. Next, we varied the number of activation counts, while measuring both the Rowhammer success rate and the actual power level of the DRAM (using RAPL). The results of this experiment are shown in Figure 10.

In the figure, each circle represents a single experiment out of 1200 in total, and the color of the circle indicates the hammering success rate, with darker circles representing higher success rates. As the figure shows, for each given DRAM RAPL value there is a very sharp threshold value for the activation count, with high hammering success rates (dark circles) if the activation count is above this threshold, and nearly no successes (white circles) if it is below the threshold. In an actual HammerScope attack, an attacker has control over the activation count (equivalent to being able to move vertically in the figure), and needs to discover the current power consumption level of the DRAM (equivalent to discovering his horizontal position in the figure). In this setting, a measurement error event will occur only if multiple power consumption levels share the same hammering threshold. We observed no such cases in our measurements.

4.7 Explaining the Mechanism Behind HammerScope

Our experimental results indicate that increased DRAM power consumption decreases the effectiveness of Rowhammer attacks, allowing HammerScope to indirectly measure DRAM power consumption via Rowhammer. In this section we propose some possible explanations for our observations, based on the capacitive crosstalk model, as originally proposed by Kim et al. [30].

Memory Organization. We begin by recalling that DRAM cells are arranged in a matrix structure, with each DRAM cell containing one storage capacitor C_S connected to a bit line (arranged vertically) through an access transistor. The access transistors are controlled through word lines (arranged horizontally), which regulate which transistors open at which times based on the cell being accessed, as shown in Figure 1. When a word line is activated, the access transistor is turned on, connecting the storage capacitor to the bit line. The stored value is detected by a sense amplifier by comparing the charge stored on the capacitor with a reference voltage.

A Physical Model for Rowhammer. Several theoretical explanations were proposed for Rowhammer in the reliability community, including electron injection and capture, capacitive crosstalk, or some combination of the two [59]. According to the capacitive crosstalk model, the matrix structure of DRAM cell arrays creates parasitic coupling capacitance between adjacent word lines, see Figure 11. When the word line of an aggressor row is activated, it experiences a rapid voltage level transition, from low to high. Due to capacitive crosstalk, this transition results in a temporary voltage spike in the word lines of adjacent rows. This spike, in turn, briefly activates the access transistor of the victim row, unintentionally causing the cells in this row to leak charge from their storage capacitors. Continuous and repetitive activations of the aggressor row amplify the effect of this unintentional charge leakage from the victim row. In particular, after AC_{min} activations, the voltage on one or more storage capacitors will come close enough to $V_{DD}/2$, causing a bit flip. Examining our results through this lens, we can propose several models to explain the mechanism behind HammerScope.

The Slew Rate Explanation. As Figure 12 shows, the length of the parasitic activation on the victim's word line is dependent on the rate of change, or *slew rate*, of the aggressor's word line. More specifically, a rapidly-changing aggressor signal will result in a narrow peak on the victim word line, with relatively high amplitude, while a slowly-changing aggressor signal will result in a wider peak with a relatively lower amplitude [58]. If increased DRAM power consumption causes a faster slew rate on the word lines, this could explain our experimental observations. Some evidence in this direction is presented in Chang et al. [9, Figure 5], where the authors show via simulation that the control lines of a DRAM module have a slower slew rate when the module is undervolted. Under this explanation, we conjecture that changes in the DIMM's power consumption result in changes to its instantaneous voltage levels, affecting the *slew rate* of its control lines and in turn the effectiveness of Rowhammer. This in turn results in the power-correlated signal observed by HammerScope.

The Charge Pump Depletion Explanation. Assuming the slew rate is not affected by DRAM activity, another possible explanation for the phenomenon we observe considers the instantaneous voltage level of the DRAM word line. In DDR3 modules this voltage rail is generated by an internal charge pump circuitry, while in DDR4 modules it is provided externally via a dedicated input, V_{PP}. If increased DRAM power consumption causes a temporary voltage droop in the charge pump's output, it would result in a lower voltage level on the aggressor's word line. If the slew rate remains constant, this in turn would result in a smaller peak on the victim word line, making the victim cells less susceptible to Rowhammer. Possible support for this explanation, comes from a concurrent independent work by Yaglikçi et al. [62]. There, [62] measured multiple DDR4 DRAM chips using an FPGA test board, observing the Rowhammer success rate while maintaining a constant activation count of 300K and varying the word-line boost voltage rail V_{PP}. They observed that on most of their evaluated chips, decreasing V_{PP} from its nominal value made the chips more resistant to Rowhammer attacks. Under this explanation, we conjecture that changes in the DIMM's power consumption result in changes to the level of its internal word-line boost signal, affecting the *amplitude* of the DIMM's control lines and in turn the effectiveness of Rowhammer.

The DRAM Contention Explanation. A final explanation assumes that both the slew rate and the voltage level are not affected by the DRAM's power consumption. Instead, it is possible that Rowhammer accesses made to an otherwise-idle DRAM system have a different temporal layout than accesses made to DRAM busy

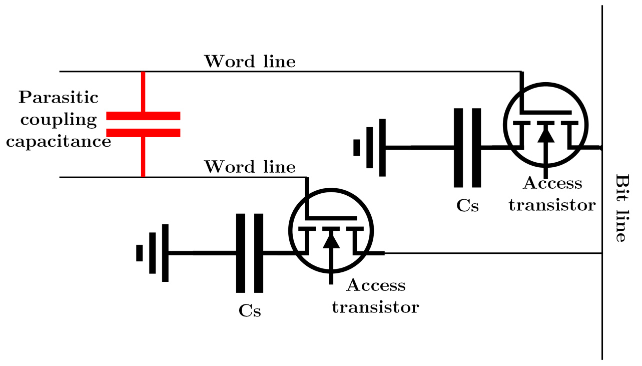


Figure 11: Parasitic coupling capacitance between two adjacent DRAM cells

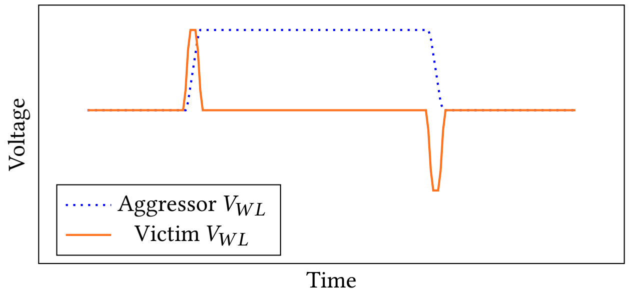


Figure 12: Parasitic coupling and row activation

with simultaneously serving both Rowhammer and other workloads. According to this explanation, the way in which Rowhammer-inducing accesses to the aggressor rows become interleaved with non-Rowhammer related memory accesses makes them decrease in potency, making Rowhammer attacks harder. Under this explanation, we conjecture that areas of high DRAM activity, which indirectly correlate with the DIMM's power consumption, result in contention on the DRAM's control bus which in turn results in less effective Rowhammer attacks.

With all the above explanations being equally plausible, we leave the task of systematically exploring the physical reasons behind HammerScope and Rowhammer to future works.

4.8 Differentiating Instructions Using HammerScope

In this section we show how to use HammerScope to measure the time it takes to execute different instructions, thereby distinguishing between them.

Measurement Technique. We begin by observing that if we interleave commands which activate the DRAM with commands which do not activate the DRAM, the overall average DRAM power consumption will depend on the proportion of the time during which the DRAM is active, namely the DRAM duty cycle. We use this property to observe the runtime of instructions executed by the core, via Rowhammer-based memory measurements. More specifically, we execute a program which repeatedly activates the DRAM by accessing uncached addresses, and then subsequently

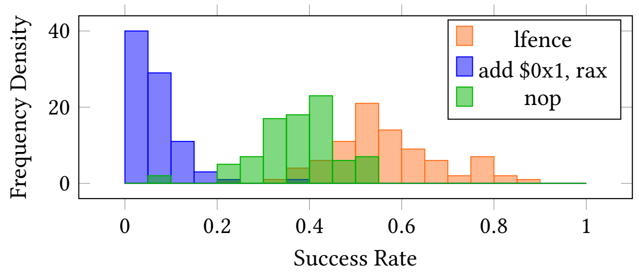


Figure 13: Differentiating instructions using HammerScope

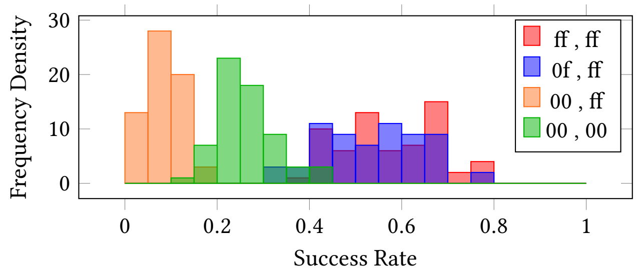


Figure 14: Observing `DIV` operands using HammerScope

executes the target instruction. In parallel, on another physical core, we continuously measure the DIMM's power consumption using HammerScope. Instructions that take longer to execute will result in a lower average duty cycle for the DRAM. This in turn results in decreased DRAM power consumption. Conversely, an instruction with a faster execution time results in a higher duty cycle, thereby increasing DRAM power consumption. Thus, by observing the DRAM power consumption when various instructions are executed we are able to distinguish between them.

Differentiating Instruction Types. To demonstrate HammerScope's capability of distinguishing between different instruction types, we used our DDR4-3 setup and repeatedly executed 8 q-word write operations followed by 16 instances of the instruction being tested. In parallel, using another physical core, we continuously took 128 samples of the DIMM's power consumption using HammerScope. Figure 13 summarizes our findings. As can be seen, we are able to distinguish between the three different instructions due to their difference in execution time.

Observing `DIV` operands. Going beyond distinguishing different instructions, we now proceed to demonstrate HammerScope's capability to differentiate between different operands of variable time instructions. To that aim, we focus on the division operation, that is known to be variable time [55] with known security implications [33]. Demonstrating the ability to distinguish between division operands, we again used our DDR4-3 setup and again executed 8 q-word write operations followed by 16 instances of the division instruction using different operands. More specifically, we used four different 128-bit dividends and a fixed divisor `0xffffffffffffffff`, see Figure 14. As the figure shows, we are able to distinguish between the division operations on different operands.

The Prefetch Command. Prefetch commands [49] allow programs to indicate to the CPU that a location in memory is likely to be accessed soon, allowing the CPU to optionally load the referenced address to cache. As observed by prior works [21, 23, 37] Prefetch commands are slower if they reference invalid or unmapped addresses. This is because translations for invalid addresses are not stored in the Translation Look-aside Buffer (TLB), requiring prefetches to such addresses to walk the page table. Conversely, translations to valid addresses are stored in the TLB even if the page is not accessible. Consequently, prefetches of valid addresses are faster than prefetches of invalid addresses, allowing us to observe this via HammerScope. We now show how HammerScope can be used as an alternative way of measuring the duration of Prefetch commands, allowing us to infer the validity of addresses.

Timing Prefetch Commands via HammerScope. As described above, Prefetch commands execute slower on invalid or unmapped addresses, compared to normal addresses. To exploit this, we execute a program which repeatedly activates DRAM, by reading from several consecutive uncached addresses, and subsequently executes a Prefetch command to a targeted address. In parallel, we measure the DIMM's power consumption using HammerScope running on another physical core similar to the previous experiments. Since the Prefetch command only accesses the memory in the first iteration (to actually fetch the target) and not afterwards, longer execution time for the Prefetch command results in a lower average duty cycle for the DRAM. This in turn results in decreased DRAM power consumption. Conversely, a faster execution time for the Prefetch command results in a higher duty cycle, thereby increasing DRAM power consumption. As a result, an attacker that measures the DRAM's power consumption can deduce the executing time of the Prefetch instruction, thereby deducing the validity of the prefetched address.

Measuring Prefetch Commands via HammerScope. Empirically demonstrating this effect, we used our DDR4-2 setup to capture DRAM RAPL and HammerScope measurements while the system executes a loop consisting of 8 DRAM memory accesses and a variable number of repeated executions of Prefetch commands. For the Prefetch address, we targeted both mapped and unmapped memory addresses. Our results are shown in Figure 15. Y-axis indicates the average hammering success probability of a 9-sided synchronized Rowhammer attack with a fixed activation count of 60,000, measured over 32 attempts and the X-axis shows the Prefetch instruction counts. As the figure shows, there is a significant difference in HammerScope behavior between mapped and unmapped addresses.

5 Attacks Using HammerScope

Having established the feasibility of using HammerScope to distinguish instruction types and operands, in this section we demonstrate how HammerScope can be used to mount end-to-end attacks.

5.1 Distinguishing Mapped and Unmapped Addresses

We begin by recalling HammerScope's ability to measure the executing time of the Prefetch command, and that the execution time of Prefetch command depends on whether the address is mapped or unmapped. In this section we show how we can use this to distinguish between a mapped and unmapped address. As Figure 15 shows, when a small amount of Prefetch commands is executed in every loop iteration, the system spends most of the time executing the DRAM access command, and as a result the power consumption measurements for the mapped and unmapped memory cases are similar. However, when the Prefetch command is executed enough times per iteration, the difference in runtime between the quickly-terminating Prefetch commands issued to mapped addresses and the slower Prefetch commands to unmapped addresses manifests itself through DRAM power consumption.

More specifically, as a result of the high average power levels when prefetching mapped addresses, Rowhammer attempts performed while prefetching mapped memory tend to fail. On the other hand, Rowhammer attempts while prefetching unmapped memory have a non-zero success rate, as long as enough Prefetch instructions are issued in a row. As can be seen in Figure 15, Rowhammer attacks issued when 8 DRAM access commands are interleaved with 64 Prefetch commands will only succeed if the address being prefetched is an invalid, unmapped address. This property can be exploited for side-channel attacks, allowing attackers to bypass SMAP and ASLR, assuming access to a high resolution timer.

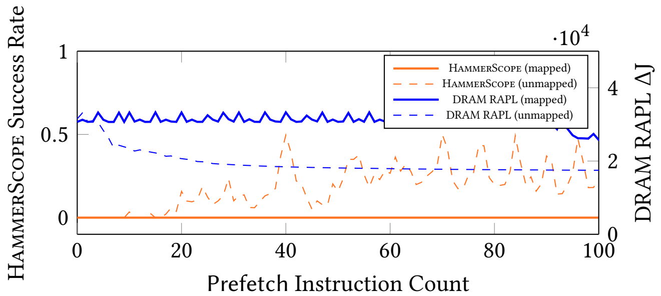


Figure 15: Timing Instructions with HammerScope

5.2 Attacking KASLR

Address Space Layout Randomization (ASLR) is a mechanism that protects against code-reuse attacks by randomizing the memory locations of sensitive code and data [42]. KASLR is the Linux implementation of ASLR aimed at protecting the kernel by randomizing the base address of the text segment, which is the segment holding the code corresponding to the kernel and associated kernel modules. The text segment of the kernel is mapped to an address in the range `0xffffffff80800000` to `0xffffffffc0800000`, with a maximum size of 1 GB [7]. As observed by [8], the kernel text segment is aligned to a 2 MB boundary, which allows 512 possible base addresses for the kernel code.

Breaking KASLR using HammerScope. Using HammerScope ability to distinguish mapped addresses from unmapped addressed using the Prefetch command, we can attack KASLR from an unprivileged process, without access to the (now privileged) RAPL interface or a timing measurement.

For the attack, we use the PREFETCHNTA instruction to repeatedly prefetch the first byte of each of the possible base addresses for the kernel text segment. We run our measurement 512 times, prefetching data from addresses in 2 MB intervals within the range `0xffffffff80800000` to `0xffffffffc0800000`. Finally, to distinguish Prefetch commands to mapped addresses from Prefetch commands to unmapped addresses, we use the methodology of Section 5.1, interleaving the Prefetch commands with memory reads, and observing the activation threshold for successful bit flips.

Attacking DDR3 Systems. Figure 16 presents the results of our experiment using our DDR3-3 setup. The X-axis represents the candidate address for the text segment. To mount HammerScope, we used eight consecutive Prefetch commands to candidate address, interleaving these with eight consecutive DRAM accesses. Concurrently with the accesses, we performed double-sided Rowhammer, and recorded the values of AC_{min} using the control loop method presented in Section 4.5. As Figure 16 shows, when accessing non-mapped memory the Prefetch command is slower and, as a result, the threshold activation count is low. However, when prefetching an address mapped to the kernel, the Prefetch command completes faster and resulting in a higher activation count.

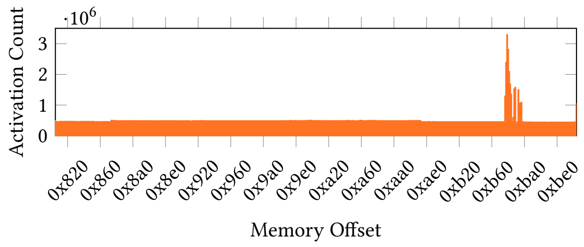


Figure 16: Attacking KASLR using HammerScope on DDR3

Attacking DDR4 Systems. Moving away from DDR3 machines to DDR4, Figure 17 shows the results of our attack on our DDR4-3. Here, we used 64 consecutive Prefetch commands to the candidate address, interleaving these with 8 consecutive DRAM accesses. Finally, we use 9-sided Rowhammer [17] to overcome the targeted row refresh countermeasure, experimentally setting the activation count to threshold to 65,000. We use the HammerScope attack with the Guess and Record method of Section 4.5, plotting the average success probability over 50 HammerScope attempts on the Y-axis. As can be seen in Figure 17, prefetching unmapped addresses results in high Rowhammer success rate whereas prefetching mapped addresses result in a very low success rate. Overall, just as in the DDR3 case, we can clearly identify the value of the kernel base address, thereby derandomizing KASLR.

Attack Demo. A video demonstrating an end-to-end unprivileged attack on an Intel i7-7700 system with DDR4 memory can be found in the paper's artifact repository. The overall attack time is

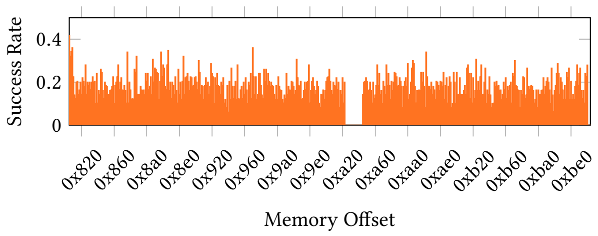


Figure 17: Attacking KASLR using HammerScope on DDR4

12:30 minutes, out of which 15 seconds are used for memory allocation and templating, 3:45 minutes are used for locating vulnerable rows and calculating the refresh period and the phase, and 8:30 are dedicated to the attack itself.

5.3 Combining HammerScope and Spectre

Spectre can allow attackers to read out kernel memory in the presence of a speculative execution gadget in the kernel by using a cache-based covert channel. To do so, the adversary first locates a Spectre v1 gadget in the kernel. This gadget performs a bounds-checked array access (Line 2 of Figure 18), obtaining the value `secret`. Finally, the gadget then performs another access (Line 3 of Figure 18) to an address dependent on the value of `secret`, thus leaking `secret` over a cache-based covert channel.

```
1   if (offset < array1_size)
2     secret = array1[offset]
3     temp &= array2[secret * 512];
```

Figure 18: Spectre v1 Gadget

Using HammerScope for Spectre. We implement a custom kernel module containing a Spectre v1 gadget similar to the one in Figure 18, loading it into the Linux kernel, and allowing user-mode application to control it using `ioctl` calls. We first call the gadget with small `offset` values, aiming to mistrain the branch predictor. Next, we call the Spectre gadget with an out-of-bounds value for `offset`, causing the CPU to speculate past the check in Line 1 of Figure 18. This speculative execution loads into `secret` a value from an attacker-controlled address (Line 2 of Figure 18). Next, when the speculation reaches Line 3 of Figure 18, a `secret`-dependent offset of `array2` is loaded into the CPU's cache. Finally, the CPU discovers that incorrect speculative execution has occurred, attempts to revert the outcome, and returns execution using the correct control flow.

A Rowhammer based Covert Channel. Rather than recovering the value of `secret` by measuring the access latency of elements of `array2`, we use HammerScope to monitor the power consumed by the gadget's transient memory accesses. We flush the elements of `array2` one at a time, and then measure the DIMM's power during the subsequent transient access to `array2`'s elements using HammerScope. When we guess the array index correctly, speculatively accessing the cached `array2` element corresponding to the value of the secret causes a significantly different power consumption pattern, compared to the rest of `array2`'s elements, which are not cached.

Experimental Results. We evaluated this attack on our DDR4-2 setup, using 9-sided Rowhammer attacks with a fixed activation count of 65,000, using the Guess and Record variant of HammerScope outlined in Section 4.5. The result of this experiment is shown in Figure 19. As the figure shows, the Rowhammer success rates are measurably lower when accessing the (cached) element of `array2` corresponding to the value of `secret` (`s` in this example), compared to accessing other (non-cached) elements of `array2`. Finally, we acknowledge that using HammerScope as a Spectre covert channel results in a significantly slower read rate compared to cache-based channels, taking 8 second per `array2` element and about 30 minutes per byte.

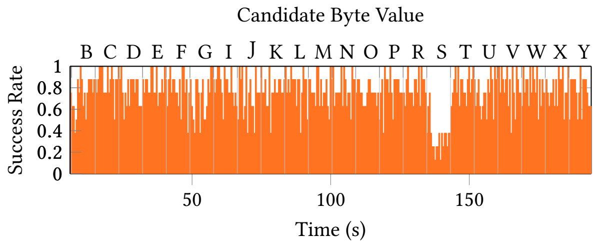


Figure 19: Reading out kernel memory using Spectre and HammerScope

5.4 Website Fingerprinting

As a final use case of HammerScope, we show how the ability to record the DIMM's power consumption can be use in order to mount a website fingerprinting attack, where the adversary attempts to identify the webpage being browsed by the user. Existing works already showed that power consumption can be used for website fingerprinting, but these works either assume a vendor-supplied API, such as the JavaScript Battery API, or the existence of some external hardware such as a malicious battery or charger [36, 44, 63].

Experimental Setup. We carried out our experiments using the DDR4-3 setup, equipped with an i7-10700K CPU and running Firefox version 96.0 for Linux. We collected data in the closed-world model, which assumes the user visits one of a predetermined set of 25 websites. The base rate accuracy of a trivial classifier in this case is 4%. In total, we collected 100 traces for each of the 25 pages, each trace consisting of 40 seconds of consecutive HammerScope measurements captured at a sampling rate of 18 Hz while Firefox was loading the page. Each of the resulting side-channel traces was a binary vector of length 721. For reference, we also collected DRAM RAPL traces at the same sampling rate, to serve as an upper bound on our classifier's performance. Our data processing pipeline was based on SKLearn for Python 3.9. We fed the side-channel traces without any preprocessing into a deep learning network consisting of 3 convolution and pooling layers, followed by two dense layers and a final Softmax output layer. We used a ReLU activation function for all layers, and optimized the network using the Adam optimizer with sparse categorical cross-entropy loss. The data was then split into training and testing sets using stratified 5-fold cross validation.

Classification Results. The accuracy of our classifier was 40%, well above the base rate of 4%. A classifier trained on the reference RAPL traces obtained an accuracy of 59%, implying that HammerScope's signal quality is not significantly worse than RAPL's.

6 Limitations and Countermeasures

6.1 Limitations

Low Sampling Rate. One of HammerScope's main limitations is its sampling rate of about 15–32 Hz (see Table 2), which is considerably lower than the sampling rates of even low-end oscilloscopes. HammerScope's sampling rate does present a significant challenge for mounting more advanced attacks such as differential power analysis [31] against side-channel hardened implementations. We thus leave the task of developing low sampling rate attacks against hardened implementations to future work.

Low Vertical Accuracy. In addition to its low sampling rate, HammerScope measurements also suffer from being significantly less accurate compared to those taken using dedicated analog-to-digital converters. This means the separation between energy levels must be big enough in order to allow us to distinguish them using HammerScope. This requires effort from the attacker, as instructions being observed must be somehow interleaved between instructions which cause a high degree of DRAM activity, see Section 4.8.

Inability to Attack Constant Time Code. While technically a power-measurement attack, HammerScope cannot, in its current form, extract information from constant time instructions, such as `aes-ni`, or integer multiplication operations. We acknowledge this limitation, attributing it to HammerScope's limitations in vertical accuracy and sampling rate, and leave the task of extracting information from constant-time operations to future work.

The Need for Code Execution. In its core HammerScope is a Rowhammer attack, and thus inherits Rowhammer's threat model of having (unprivileged) code execution on the target machine. While this scenario is natural in some cases (e.g., a remote server), it does present challenges for some devices, such as smart cards, Hardware Security Modules (HSMs), and proprietary microcontrollers. This is in contrast to more traditional power analysis attacks, which merely require passive observation of the target using external measurement equipment.

6.2 Countermeasures

Since HammerScope measures a physical phenomenon, and does not rely directly on a manufacturer-provided machine status register, it is not trivial to mitigate. To protect against HammerScope attacks, we must observe the chain of events that make them possible: First, the machine's secret behavior is modulated onto its power consumption; Next, this power consumption affects the susceptibility of DRAM rows to Rowhammer attacks; Finally, an adversary triggers the Rowhammer attack. To mitigate HammerScope, this chain must be broken through non-trivial changes to the software and the underlying hardware.

Constant-Power Code. To help prevent secret behavior from leaking into a system's power consumption, writers of security-sensitive code could consider writing *constant-power code*, whose power consumption does not depend on the instructions or data being processed. While such code is already applied programming devices considered at risk of power analysis, such as payment cards [41], we note that HammerScope's current limitations make attacks on data challenging. In particular, further improvements seem to be required in order to obtain HammerScope-based data extraction.

Adding Noise. Another well-known, but slightly less effective, method for preventing side-channel leakage is the random noise countermeasure, which aims to reduce the signal-to-noise ratio of the side-channel trace available to the attacker, increasing the attack time to the point of impracticality. To be applied in this case, sensitive code can consider intentionally accessing DRAM during sensitive operations.

Rowhammer Mitigation. The next point of intervention could be helping to prevent or detect Rowhammer attacks. There is a large body of work that proposes both software-based and hardware-based mitigation mechanisms for Rowhammer, as surveyed by Kim et al. in [29]. Loughlin et al. [40] present a taxonomy which divides mitigations into three groups: isolation-centric, frequency-centric and refresh-centric. Kim et al. in [30] suggested increasing the refresh rate until it is impossible to perform enough activations within a refresh window to flip a bit. This is becoming increasingly difficult as miniaturization and power optimization make DRAM hardware ever more susceptible to Rowhammer attacks. Kim et al. also proposed PARA, which refreshes one or more adjacent rows with a low probability every time a row is refreshed. Kim et al. in [29] implement an ideal refresh mechanism which tracks row activation and issues a targeted refresh command to a row right before it can flip a bit. ANVIL [1] detects Rowhammer attacks by tracking the latency of DRAM access using hardware counters to identify frequently accessed rows. Brasser et al. implemented CATT [5], a mechanism which prevents attackers from leveraging Rowhammer to corrupt kernel memory from user mode, by using the OS physical memory allocator to physically isolate between the kernel and user memory. ProHIT [51] mitigate Rowhammer by using low-overhead probabilistic tables to maintain DRAM activation history. Similarly, TWiCe [35] detects Rowhammer attacks using counters with low performance impact. Panopticon [2] proposes a complete in-DRAM Rowhammer mitigation by maintaining a counter table with an activation threshold. GuardION [57] mitigates Rowhammer exploitation on ARM by isolating DMA buffers with guard rows. You et al. propose to refresh victim rows using a probability that is dynamically adjusted based on each row's access history [64].

7 Conclusions

In this work we introduced HammerScope, a method for software-based power analysis using Rowhammer attacks, and showed how it can be used to compromise secret information in some scenarios. The Rowhammer attack is unique among fault attacks, since the actions required to trigger it are purely in the digital domain, while its impact manifests in the analog domain. This work shows that the reverse relationship also holds – the analog conditions of the system can be sensed using Rowhammer and brought back into the digital domain.

While we demonstrate some security implications of HammerScope, we acknowledge that all the attacks presented in this paper could also be mounted via other channels. We leave the task of investigation additional HammerScope applications, as well as systematically analyzing the root cause of HammerScope to future work.

Acknowledgments

The authors wish to thank Anatoly Shusterman, Liad Oz, Eilam Gal, Lucian Cojocar, Ron Anderson, and David Blankenbeckler for their assistance and advice.

This work was supported by an ARC Discovery Early Career Researcher Award DE200101577; an ARC Discovery Project number DP210102670; the Blavatnik ICRC at Tel-Aviv University; the Air Force Office of Scientific Research (AFOSR) under award number FA9550-20-1-0425; the National Science Foundation under grant CNS-1954712; and gifts from AMD, Intel, and Qualcomm.

References

[1] Zelalem Birhanu Aweke, Salessawi Ferede Yitbarek, Rui Qiao, Reetuparna Das, Matthew Hicks, Yossi Oren, and Todd M. Austin. 2016. ANVIL: Software-Based

Protection Against Next-Generation Rowhammer Attacks. In *ASPLOS*. 743–755. https://doi.org/10.1145/2872362.2872390

[2] Tanj Bennett, Stefan Saroiu, Alec Wolman, and Lucian Cojocar. 2021. Panopticon: A Complete In-DRAM Rowhammer Mitigation. In *DRAMSec.* https://dramsec.ethz.ch/papers/panopticon.pdf

[3] Ishwar Bhati, Mu-Tien Chang, Zeshan Chishti, Shih-Lien Lu, and Bruce L. Jacob. 2016. DRAM Refresh Mechanisms, Penalties, and Trade-Offs. *IEEE Trans. Computers* 65, 1 (2016), 108–121. https://doi.org/10.1109/TC.2015.2417540

[4] Sarani Bhattacharya and Debdeep Mukhopadhyay. 2016. Curious Case of Rowhammer: Flipping Secret Exponent Bits Using Timing Analysis. In *CHES*. 602–624. https://doi.org/10.1007/978-3-662-53140-2_29

[5] Ferdinand Brasser, Lucas Davi, David Gens, Christopher Liebchen, and Ahmad-Reza Sadeghi. 2017. CAn't Touch This: Software-only Mitigation against Rowhammer Attacks targeting Kernel Memory. In *USENIX Security*. 117–130. https://www.usenix.org/conference/usenixsecurity17/technical-sessions/presentation/brasser

[6] Giovanni Camurati, Sebastian Poeplau, Marius Muench, Tom Hayes, and Aurélien Francillon. 2018. Screaming Channels: When Electromagnetic Side Channels Meet Radio Transceivers. In *CCS*. 163–177. https://doi.org/10.1145/3243734.3243802

[7] Claudio Canella, Daniel Genkin, Lukas Giner, Daniel Gruss, Moritz Lipp, Marina Minkin, Daniel Moghimi, Frank Piessens, Michael Schwarz, Berk Sunar, Jo Van Bulck, and Yuval Yarom. 2019. Fallout: Leaking Data on Meltdown-resistant CPUs. In *CCS*. 769–784. https://doi.org/10.1145/3319535.3363219

[8] Claudio Canella, Michael Schwarz, Martin Haubenwallner, Martin Schwarzl, and Daniel Gruss. 2020. KASLR: Break It, Fix It, Repeat. In *AsiaCCS*. 481–493. https://doi.org/10.1145/3320269.3384747

[9] Kevin K. Chang, Abdullah Giray Yaglikçi, Saugata Ghose, Aditya Agrawal, Niladrish Chatterjee, Abhijith Kashyap, Donghyuk Lee, Mike O'Connor, Hasan Hassan, and Onur Mutlu. 2017. Understanding Reduced-Voltage Operation in Modern DRAM Devices: Experimental Characterization, Analysis, and Mechanisms. *Proc. ACM Meas. Anal. Comput. Syst.* 1, 1 (2017), 10:1–10:42. https://doi.org/10.1145/3084447

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

[11] Howard David, Eugene Gorbatov, Ulf R. Hanebutte, Rahul Khanna, and Christian Le. 2010. RAPL: memory power estimation and capping. In *ISLPED*. 189–194. https://doi.org/10.1145/1840845.1840883

[12] Finn de Ridder, Pietro Frigo, Emanuele Vannacci, Herbert Bos, Cristiano Giuffrida, and Kaveh Razavi. 2021. SMASH: Synchronized Many-sided Rowhammer Attacks from JavaScript. In *USENIX Security*. 1001–1018. https://www.usenix.org/conference/usenixsecurity21/presentation/ridder

[13] Spencer Desrochers, Chad Paradis, and Vincent M. Weaver. 2016. A Validation of DRAM RAPL Power Measurements. In *MEMSYS*. 455–470. https://doi.org/10.1145/2989081.2989088

[14] Steven A. Frank. 2018. *Control Theory Tutorial*. Springer International Publishing.

[15] Jeffrey Friedman. 1972. Tempest: A signal problem. *NSA Cryptologic Spectrum* 2, 3 (1972). https://www.nsa.gov/portals/75/documents/news-features/declassified-documents/cryptologic-spectrum/tempest.pdf

[16] Pietro Frigo, Cristiano Giuffrida, Herbert Bos, and Kaveh Razavi. 2018. Grand Pwning Unit: Accelerating Microarchitectural Attacks with the GPU. In *IEEE SP*. 195–210. https://doi.org/10.1109/SP.2018.00022

[17] Pietro Frigo, Emanuele Vannacci, Hasan Hassan, Victor van der Veen, Onur Mutlu, Cristiano Giuffrida, Herbert Bos, and Kaveh Razavi. 2020. TRRespass: Exploiting the Many Sides of Target Row Refresh. In *IEEE SP*. 747–762. https://doi.org/10.1109/SP46214.2022.9833664

[18] Karine Gandolfi, Christophe Mourtel, and Francis Olivier. 2001. Electromagnetic Analysis: Concrete Results. In *CHES*. 251–261. https://doi.org/10.1007/3-540-44709-1_21

[19] Daniel Genkin, Noam Nissan, Roei Schuster, and Eran Tromer. 2022. Lend Me Your Ear: Passive Remote Physical Side Channels on PCs. In *USENIX Security*. 4437–4454. https://outflux.net/slides/2013/lss/kaslr.pdf

[20] Daniel Genkin, Itamar Pipman, and Eran Tromer. 2015. Get your hands off my laptop: physical side-channel key-extraction attacks on PCs - Extended version. *J. Cryptogr. Eng.* 5, 2 (2015), 95–112. https://doi.org/10.1007/s13389-015-0100-7

[21] Daniel Gruss, Moritz Lipp, Michael Schwarz, Richard Fellner, Clémentine Maurice, and Stefan Mangard. 2017. KASLR is Dead: Long Live KASLR. In *ESSoS*. 161–176. https://doi.org/10.1007/978-3-319-62105-0_11

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

[23] Daniel Gruss, Clémentine Maurice, Anders Fogh, Moritz Lipp, and Stefan Mangard. 2016. Prefetch Side-Channel Attacks: Bypassing SMAP and Kernel ASLR. In *CCS*. 368–379. https://doi.org/10.1145/2976749.2978356

[24] Daniel Gruss, Clémentine Maurice, and Stefan Mangard. 2016. Rowhammer.js: A Remote Software-Induced Fault Attack in JavaScript. In *DIMVA*. 300–321. https://doi.org/10.1007/978-3-319-40667-1_15

[25] Hasan Hassan, Yahya Can Tugrul, Jeremie S. Kim, Victor van der Veen, Kaveh Razavi, and Onur Mutlu. 2021. Uncovering In-DRAM RowHammer Protection Mechanisms: A New Methodology, Custom RowHammer Patterns, and Implications. In *MICRO*. 1198–1213. https://doi.org/10.1145/3466752.3480110

[26] Yeongjin Jang, Jaehyuk Lee, Sangho Lee, and Taesoo Kim. 2017. SGX-Bomb: Locking Down the Processor via Rowhammer Attack. In *SysTEX*. 5:1–5:6. https://doi.org/10.1145/3152701.3152709

[27] Patrick Jattke, Victor van der Veen, Pietro Frigo, Stijn Gunter, and Kaveh Razavi. 2022. BLACKSMITH: Scalable Rowhammering in the Frequency Domain. In *IEEE SP*. 716–734. https://doi.org/10.1109/SP46214.2022.9833772

[28] Yichen Jiang, Huifeng Zhu, Dean Sullivan, Xiaolong Guo, Xuan Zhang, and Yier Jin. 2021. Quantifying Rowhammer Vulnerability for DRAM Security. In *DAC*. 73–78. https://doi.org/10.1109/DAC18074.2021.9586119

[29] Jeremie S. Kim, Minesh Patel, Abdullah Giray Yaglikçi, Hasan Hassan, Roknoddin Azizi, Lois Orosa, and Onur Mutlu. 2020. Revisiting RowHammer: An Experimental Analysis of Modern DRAM Devices and Mitigation Techniques. In *ISCA*. 638–651. https://doi.org/10.1109/ISCA45697.2020.00059

[30] Yoongu Kim, Ross Daly, Jeremie S. Kim, Chris Fallin, Ji-Hye Lee, Donghyuk Lee, Chris Wilkerson, Konrad Lai, and Onur Mutlu. 2014. Flipping bits in memory without accessing them: An experimental study of DRAM disturbance errors. In *ISCA*. 361–372. https://doi.org/10.1109/ISCA.2014.6853210

[31] Paul C. Kocher, Joshua Jaffe, and Benjamin Jun. 1999. Differential Power Analysis. In *CRYPTO*. 388–397. https://doi.org/10.1007/3-540-48405-1_25

[32] Andreas Kogler, Jonas Juffinger, Salman Qazi, Yoongu Kim, Moritz Lipp, Nicolas Boichat, Eric Shiu, Mattias Nissler, and Daniel Gruss. 2022. Half-Double: Hammering from the Next Row Over. In *USENIX Security*. 3807–3824. https://www.usenix.org/conference/usenixsecurity22/presentation/kogler-half-double

[33] Robert Kotcher, Yutong Pei, Pranjal Jumde, and Collin Jackson. 2013. Cross-Origin Pixel Stealing: Timing Attacks Using CSS Filters. In *CCS*. 1055–1062. https://doi.org/10.1145/2508859.2516712

[34] Andrew Kwong, Daniel Genkin, Daniel Gruss, and Yuval Yarom. 2020. RAMBleed: Reading Bits in Memory Without Accessing Them. In *IEEE SP*. 695–711. https://doi.org/10.1109/SP40000.2020.00020

[35] Eojin Lee, Ingab Kang, Sukhan Lee, G. Edward Suh, and Jung Ho Ahn. 2019. TWiCe: Preventing Row-hammering by Exploiting Time Window Counters. In *ISCA*. 385–396. https://doi.org/10.1145/3307650.3322232

[36] Pavel Lifshits, Roni Forte, Yedid Hoshen, Matt Halpern, Manuel Philipose, Mohit Tiwari, and Mark Silberstein. 2018. Power to peep-all: Inference Attacks by Malicious Batteries on Mobile Devices. *PoPETs* 2018, 4 (2018), 141–158. https://doi.org/10.1515/popets-2018-0036

[37] Moritz Lipp, Daniel Gruss, and Michael Schwarz. 2022. AMD Prefetch Attacks through Power and Time. In *USENIX Security*. 643–660. https://www.usenix.org/conference/usenixsecurity22/presentation/lipp

[38] Moritz Lipp, Andreas Kogler, David F. Oswald, Michael Schwarz, Catherine Easdon, Claudio Canella, and Daniel Gruss. 2021. PLATYPUS: Software-based Power Side-Channel Attacks on x86. In *IEEE SP*. 355–371. https://doi.org/10.1109/SP40001.2021.00063

[39] Moritz Lipp, Michael Schwarz, Lukas Raab, Lukas Lamster, Misiker Tadesse Aga, Clémentine Maurice, and Daniel Gruss. 2020. Nethammer: Inducing Rowhammer Faults through Network Requests. In *EuroS&P Workshops*. 710–719. https://doi.org/10.1109/EuroSPW51379.2020.00102

[40] Kevin Loughlin, Stefan Saroiu, Alec Wolman, and Baris Kasikci. 2021. Stop! Hammer Time: Rethinking our Approach to Rowhammer Mitigations. In *HotOS*. 88–95. https://doi.org/10.1145/3458336.3465295

[41] Stefan Mangard, Elisabeth Oswald, and Thomas Popp. 2007. *Power Analysis Attacks: Revealing the Secrets of Smart Cards*. Springer.

[42] Hector Marco-Gisbert and Ismael Ripoll Ripoll. 2019. Address space layout randomization next generation. *Applied Sciences* 9, 14 (2019), 2928. https://doi.org/10.3390/app9142928

[43] Colin O'Flynn and Alex Dewar. 2019. On-Device Power Analysis Across Hardware Security Domains. Stop Hitting Yourself. *IACR Trans. Cryptogr. Hardw. Embed. Syst.* 2019, 4 (2019), 126–153. https://doi.org/10.13154/tches.v2019.i4.126-153

[44] Lukasz Olejnik, Gunes Acar, Claude Castelluccia, and Claudia Díaz. 2015. The Leaking Battery - A Privacy Analysis of the HTML5 Battery Status API. In *DPM/QASA at ESORICS*. 254–263. https://doi.org/10.1007/978-3-319-29883-2_18

[45] Yossef Oren and Adi Shamir. 2007. Remote Password Extraction from RFID Tags. *IEEE Trans. Computers* 56, 9 (2007), 1292–1296. https://doi.org/10.1109/TC.2007.1050

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

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

[48] Falk Schellenberg, Dennis R. E. Gnad, Amir Moradi, and Mehdi Baradaran Tahoori. 2018. An inside job: Remote power analysis attacks on FPGAs. In *DATE*. 1111–1116. https://doi.org/10.23919/DATE.2018.8342177

[49] Martin Schwarzl, Thomas Schuster, Michael Schwarz, and Daniel Gruss. 2021. Speculative dereferencing of registers: Reviving Foreshadow. In *FC*. 311–330. https://doi.org/10.1007/978-3-662-64322-8_15

[50] Mark Seaborn and Thomas Dullien. 2015. Exploiting the DRAM rowhammer bug to gain kernel privileges. https://googleprojectzero.blogspot.com/2015/03/exploiting-dram-rowhammer-bug-to-gain.html. (2015).

[51] Mungyu Son, Hyunsun Park, Junwhan Ahn, and Sungjoo Yoo. 2017. Making DRAM Stronger Against Row Hammering. In *DAC*. 55:1–55:6. https://doi.org/10.1145/3061639.3062281

[52] Andrei Tatar, Cristiano Giuffrida, Herbert Bos, and Kaveh Razavi. 2018. Defeating Software Mitigations Against Rowhammer: A Surgical Precision Hammer. In *RAID*. 47–66. https://doi.org/10.1007/978-3-030-00470-5_3

[53] Andrei Tatar, Radhesh Krishnan Konoth, Elias Athanasopoulos, Cristiano Giuffrida, Herbert Bos, and Kaveh Razavi. 2018. Throwhammer: Rowhammer Attacks over the Network and Defenses. In *USENIX ATC*. 213–226. https://www.usenix.org/conference/atc18/presentation/tatar

[54] Youssef Tobah, Andrew Kwong, Ingab Kang, Daniel Genkin, and Kang G. Shin. 2022. SpecHammer: Combining Spectre and Rowhammer for New Speculative Attacks. In *IEEE SP*. 681–698. https://doi.org/10.1109/SP46214.2022.9833802

[55] Jo Van Bulck, Frank Piessens, and Raoul Strackx. 2018. Nemesis: Studying Microarchitectural Timing Leaks in Rudimentary CPU Interrupt Logic. In *CCS*. 178–195. https://doi.org/10.1145/3243734.3243822

[56] Victor Van Der Veen, Yanick Fratantonio, Martina Lindorfer, Daniel Gruss, Clémentine Maurice, Giovanni Vigna, Herbert Bos, Kaveh Razavi, and Cristiano Giuffrida. 2016. Drammer: Deterministic Rowhammer Attacks on Mobile Platforms. In *CCS*. 1675–1689. https://doi.org/10.1145/2976749.2978406

[57] Victor Van der Veen, Martina Lindorfer, Yanick Fratantonio, Harikrishnan Padmanabha Pillai, Giovanni Vigna, Christopher Kruegel, Herbert Bos, and Kaveh Razavi. 2018. GuardION: Practical Mitigation of DMA-Based Rowhammer Attacks on ARM. In *DIMVA*. 92–113. https://doi.org/10.1007/978-3-319-93411-2_5

[58] VLSI System Design Corporation. 2017. Effect of Coupling Capacitance. (2017). https://www.vlsisystemdesign.com/effect-of-coupling-capacitance/

[59] Andrew J. Walker, Sungkwon Lee, and Dafna Beery. 2021. On DRAM Rowhammer and the Physics of Insecurity. *IEEE Transactions on Electron Devices* 68, 4 (2021), 1400–1410. https://doi.org/10.1109/TED.2021.3060362

[60] Zane Weissman, Thore Tiemann, Daniel Moghimi, Evan Custodio, Thomas Eisenbarth, and Berk Sunar. 2020. JackHammer: Efficient Rowhammer on Heterogeneous FPGA-CPU Platforms. *IACR Trans. Cryptogr. Hardw. Embed. Syst.* 2020, 3 (2020), 169–195. https://doi.org/10.13154/tches.v2020.i3.169-195

[61] Yuan Xiao, Xiaokuan Zhang, Yinqian Zhang, and Radu Teodorescu. 2016. One Bit Flips, One Cloud Flops: Cross-VM Row Hammer Attacks and Privilege Escalation. In *USENIX Security*. 19–35. https://www.usenix.org/conference/usenixsecurity16/technical-sessions/presentation/xiao

[62] Abdullah Giray Yaglikçi, Haocong Luo, Geraldo F. de Oliviera, Ataberk Olgun, Minesh Patel, Jisung Park, Hasan Hassan, Jeremie S. Kim, Lois Orosa, and Onur Mutlu. 2022. Understanding RowHammer Under Reduced Wordline Voltage: An Experimental Study Using Real DRAM Devices. In *DSN*. 475–487. https://doi.org/10.1109/DSN53405.2022.00054

[63] Qing Yang, Paolo Gasti, Gang Zhou, Aydin Farajidavar, and Kiran S. Balagani. 2017. On Inferring Browsing Activity on Smartphones via USB Power Analysis Side-Channel. *IEEE Trans. Inf. Forensics Secur.* 12, 5 (2017), 1056–1066. https://doi.org/10.1109/TIFS.2016.2639446

[64] Jung Min You and Joon-Sung Yang. 2019. MRLoc: Mitigating Row-hammering Based on Memory Locality. In *DAC*. 19. https://doi.org/10.1145/3316781.3317866

[65] Zhi Zhang, Yueqiang Cheng, Dongxi Liu, Surya Nepal, Zhi Wang, and Yuval Yarom. 2020. PThammer: Cross-User-Kernel-Boundary Rowhammer through Implicit Accesses. In *MICRO*. 28–41. https://doi.org/10.1109/MICRO50266.2020.00016

[66] Kenneth M. Zick, Meeta Srivastav, Wei Zhang, and Matthew French. 2013. Sensing Nanosecond-Scale Voltage Attacks and Natural Transients in FPGAs. In *FPGA*. 101–104. https://doi.org/10.1145/2435264.2435283