FP-Rowhammer: DRAM-Based Device Fingerprinting

Hari Venugopalan
hvenugopalan@ucdavis.edu
UC Davis

Kaustav Goswami
kggoswami@ucdavis.edu
UC Davis

Zainul Abi Din
zdin@ucdavis.edu
UC Davis

Jason Lowe-Power
jlowepower@ucdavis.edu
UC Davis

Samuel T. King
kingst@ucdavis.edu
UC Davis

Zubair Shafiq
zubair@ucdavis.edu
UC Davis

Abstract

Device fingerprinting leverages attributes that capture heterogeneity in hardware and software configurations to extract unique and stable fingerprints. Fingerprinting countermeasures attempt to either present a uniform fingerprint across different devices through normalization or present different fingerprints for the same device each time through obfuscation. We present FP-Rowhammer, a Rowhammer-based device fingerprinting approach that can build unique and stable fingerprints even across devices with normalized or obfuscated hardware and software configurations. To this end, FP-Rowhammer leverages the DRAM manufacturing process variation that gives rise to unique distributions of Rowhammer-induced bit flips across different DRAM modules. Our evaluation on a test bed of 98 DRAM modules shows that FP-Rowhammer achieves 99.91% fingerprinting accuracy. FP-Rowhammer's fingerprints are also stable, with no degradation in fingerprinting accuracy over a period of ten days. We also demonstrate that FP-Rowhammer is efficient, taking less than five seconds to extract a fingerprint. FP-Rowhammer is the first Rowhammer fingerprinting approach to extract unique and stable fingerprints efficiently and at scale.

1 Introduction

Growing restrictions against stateful identifiers [3, 25, 56] has led to the emergence of device fingerprinting for tracking [6, 30, 43]. Fingerprinters capture distinguishing hardware and software attributes of devices to construct a *fingerprint* that identifies clients without needing to store any client-side state [43]. For a fingerprint to be useful, it needs to be *unique* and *stable*. First, a fingerprint should have sufficiently high entropy to uniquely identify a device within a population of devices [17]. Second, it should remain sufficiently stable over time so it can be linked to previous fingerprints from the same device for re-identification [76].

Fingerprinting techniques typically aggregate a variety of software and hardware attributes, such as screen resolution, the number of processors, and the type and version of the operating system, to construct device fingerprints [19–21]. To attain high entropy, such fingerprints are dependent on devices having diverse configurations that are sufficiently distinguishable. A common countermeasure to reduce entropy is to normalize the attributes that capture device configurations to present the same values across different devices [9, 55]. Fingerprinters also have to contend with how the attributes change over time with the goal of producing a stable fingerprint. These changes can either arise from natural evolution of device configurations (e.g., software updates) or from fingerprinting countermeasures that intentionally obfuscate attributes [10, 57]. To attain high stability, fingerprinters attempt to predict fingerprint changes [76] or employ stemming to improve stability [63].

In this work, we investigate the threat model where a fingerprinter aims to extract unique and stable fingerprints for devices with identical hardware and software configurations over an extended time period. To this end, we aim to capture fundamental differences in the physical properties of the device's hardware. The key insight is that a fingerprinter can capture inherent differences that arise as a result of *process variation* in the hardware manufacturing process. As users rarely modify their device hardware, these fingerprints would remain stable. While prior research has exploited the manufacturing process variation in CPUs [11], GPUs [42], and clocks [69], we successfully employ memory (specifically DRAM) for device fingerprinting.

We leverage Rowhammer [40] to extract fingerprints by capturing the side-effects of process variation in memory modules. At a high level, "hammering" a memory row (i.e., repeated read or write operations in a short time interval) results in bit flips in adjacent memory rows. In this paper, we investigate whether the pattern of bit flips due to Rowhammer can be leveraged for fingerprinting. To build intuition, Figure 1 visualizes the distribution of bit flips produced by executing Rowhammer at the same locations on two identical DRAM modules (also called Dual Inline Memory Modules or DIMMs) at two different points in time. The figure shows that the distribution of bit flips is *reasonably similar* on the same DRAM modules at different points in time while being *noticeably different* across different DRAM modules. While recent research has extended Rowhammer-based PUFs [70] for fingerprinting [44], as we discuss next, there are several non-trivial, technical concerns that make it challenging to use Rowhammer for fingerprinting [23].

We present FP-Rowhammer, a Rowhammer-based fingerprinting approach that overcomes these challenges and exploits bit flip distributions to extract *unique* and *stable* fingerprints even among homogeneous devices with identical software and hardware configurations over an extended period of time. We address the following technical challenges with FP-Rowhammer:

- First, we find that the bit flips triggered by Rowhammer are non-deterministic (i.e., hammering the same memory location does not always flip the same set of bits). In FP-Rowhammer, we use statistical sampling to account for **non-determinism** by hammering the same memory locations multiple times to extract and compare probability distributions of bit flips. This leads to accurate and robust fingerprinting, even in presence of external factors (§ 6.7).
- Second, the operating system's **memory abstractions limit access** to contiguous physical memory. Since bit flips

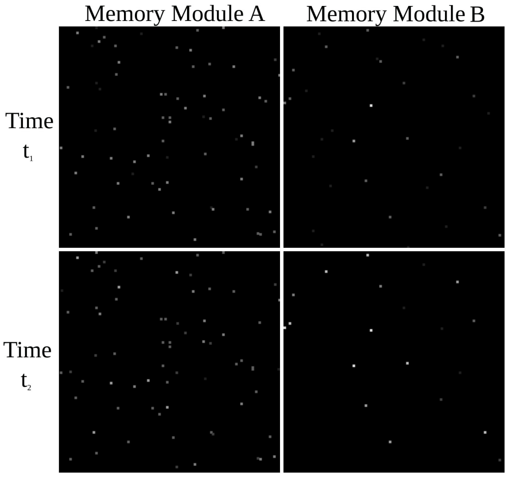


Figure 1: Visualization of the Rowhammer bit flip distribution with a brighter spot representing a higher bit flip probability. The top row shows the distributions on two different but identical DRAM modules. The bottom row shows the distribution on the same DRAM modules at a later point in time. It shows that the distribution of bit flips is *reasonably similar* on the same DRAM modules at different points in time while being *noticeably different* across different DRAM modules.

are not uniformly distributed across rows [23], fingerprinters will struggle to match fingerprints if these abstractions present them with different chunks of memory at different points in time. In FP-Rowhammer, we sample chunks of memory for near-guaranteed access to the same chunk.

- Third, memory modules implement **Rowhammer mitigations**, such as Target Row Refresh (TRR) [48]. While many-sided and non-uniform hammering can bypass mitigations [22, 32], we find that certain patterns are not suitable for fingerprinting (§6.7). In FP-Rowhammer, we systematically identify hammering patterns to bypass these mitigations for at-scale Rowhammer-based device fingerprinting.

We evaluate FP-Rowhammer on a test bed of nearly 100 DIMMs across 6 sets of identical DRAM modules for 2 major DRAM manufacturers with the largest DRAM marketshare. FP-Rowhammer achieves a high fingerprint accuracy of 99.91%, corresponding to 100% precision and 97.06% recall. FP-Rowhammer also achieves high stability, with no degradation in fingerprint accuracy over a period of ten days. FP-Rowhammer is able to efficiently extract fingerprints in less than five seconds. To the best of our knowledge, our evaluation of DRAM modules is the largest scale to date.

2 Background

2.1 DRAM basics

All DRAM technologies follow the same basic architecture [29, 35, 49–54]. We focus on DIMM-based DRAM packages in this paper, but our findings also apply to other packaging techniques.

Each physical DIMM is installed on a DRAM channel on the motherboard. Channels enable issuing concurrent requests to multiple DIMMs. A DIMM can have one or more ranks. Each rank contains multiple logical structures called banks. A bank is a two-dimensional array of cells organized into rows and columns. Each cell contains a capacitor and an access transistor, with the capacitor's charged state representing a single bit. The number of cells in a column is given by the width (x8, x16 etc) of the DIMM.

The memory controller issues commands to the DRAM to perform memory operations at the granualrity of a *row*. The `ACT` command activates a row by loading it into the row-buffer before reading or writing to it. The `PRE` command deactivates a row and prepares the row-buffer to load another row by restoring previously held values. The memory controller periodically issues `REFI` commands to refresh the charge held by capacitors since the charge naturally drains over time.

2.2 Rowhammer

DIMMs are susceptible to memory corruption as a result of electrical interference. Rowhammer [5, 40] corrupts the data stored in some capacitors leading to bit flips in memory. Specifically, Rowhammer triggers bit flips at a particular address by repeatedly accessing neighboring addresses. This leads to the repeated activation and deactivation of rows containing the accessed addresses. The resulting electro-magnetic interference between the accessed rows (referred to as aggressors or aggressor rows) and their neighboring rows (referred to as victim rows) accelerates the charge dissipation of the capacitors in the victim rows, resulting in memory corruption.

Two prerequisites must be satisfied to reliably execute Rowhammer: access to physically contiguous memory and fast, uncached memory access [7, 13, 37, 75]. Physically contiguous memory enables two-sided Rowhammer which has been shown to be more effective at triggering bit flips [4]. Fast, uncached memory access is required to activate aggressors in the same row at a sufficiently high rate to trigger bit flips. Modern DDR4 DIMMs implement Target Row Refresh or TRR to mitigate Rowhammer [48]. While several undocumented, proprietary implementations of TRR exist, all of them essentially track memory accesses to identify aggressor rows and issue additional refreshes to the associated victim rows [22, 28].

2.3 Rowhammer for fingerprinting

The rate at which a capacitor loses its charge depends on its physical properties [31, 61]. These properties are not uniform on all chips due to process variation induced during manufacturing [73]. This process variation also determines the susceptibility of each bit to flipping under Rowhammer. Thus, when running Rowhammer with identical parameters (under comparable environmental conditions) on different chips, differences in the bit flip behavior can be attributed to differences in the physical properties of the DIMMs.

Rowhammer PUF [70] studies the distribution of bit flips on a PandaBoard [18], but they do not compare the uniqueness of bit flips across devices. They also present results on DDR2 memory which did not incorporate any mitigations against Rowhammer. Fingerprinters on a PandaBoard also do not have to contend with the restrictions imposed by the OS to access memory. Recently, researchers have extended the approach proposed by Rowhammer PUF for fingerprinting [44] on desktops with DDR4 DIMMs. However, they do not discuss ways to overcome key challenges that complicate the extraction of fingerprints on commodity devices:

First, fingerprinters have to ensure that they compare bit flip distributions from the same memory chunk for fingerprinting. However, fingerprinters have to rely on the abstractions provided by the OS to access memory, which make it difficult to guarantee access to the same chunk. Fingerprinters cannot identify devices by matching bit flip distributions across different chunks of memory from the same module since we find that each chunk has a unique distribution of bit flips (Takeaway 1 in §4.2).

Second, fingerprinters have to overcome the non-deterministic behavior of bit flips [23] when identifying devices. We see that bit flips are non-deterministic since some capacitors are more susceptible to Rowhammer (higher probability of flipping) when compared to others (Takeaway 2 in §4.2). This results in cases where the approach proposed by prior research [44] is unreliable in identifying devices even when building up references across multiple hammering attempts (Figure 3 and Figure 4). Fingerprinters also have to contend with the presence of external factors such as background applications and changes to the CPU's frequency (that they cannot control) which further impact the non-determinism in bit flips (§6.7).

Third, the choice of the hammering pattern employed to overcome TRR also has an impact on which bits flip [23]. As a result, certain hammering patterns are more suitable for fingerprinting. However, prior research [44] does not discuss ways to identify patterns that help with fingerprinting. Lastly, prior research presents results on a limited set of DRAM modules and does not discuss if their results generalize across DRAM modules of different configurations and manufacturers.

With FP-Rowhammer, we first make the observation that the bit flips in each contiguous 2 MB chunk of memory (one among several ways to allocate contiguous memory using Transparent Huge Pages or THP [38]) are highly unique and persistent (§4.2). Leveraging this observation, we propose a novel sampling strategy as part of FP-Rowhammer's design to overcome the memory restrictions imposed by the operating system (§5.3). When overcoming TRR to trigger bit flips for fingerprinting, we operationalize hammering patterns at scale by prioritizing those patterns that can trigger a large number of bit flips (§5.1). Such patterns help improve the robustness of the fingerprint in presence of external factors (§6.7). To account for the inherent non-determinism in bit flips, we reset and hammer the memory chunk multiple times to extract a probability distribution of bit flips (§5.2). We then compare the divergence of these probability distributions to reliably fingerprint DIMMs. Crucially, when compared to prior work, we always execute Rowhammer multiple times to ensure that our fingerprints are not affected by those capacitors that have a low probability of showing bit flips. FP-Rowhammer is the first technique to demonstrate the extraction of unique and stable fingerprints on the largest scale using Rowhammer while overcoming practical limitations enforced by the OS and by Rowhammer mitigations such as TRR.

3 FP-Rowhammer Overview

3.1 Threat model

In this paper, we take the role of the fingerprinter whose goal is to extract unique and stable fingerprints from devices even among those that have identical hardware and software configurations. In our threat model, the fingerprinter can run code on a user's device. Specifically, we consider host-based fingerprinting [11, 69, 70] where users run a native application on their device that was developed by the fingerprinter. We assume that the fingerprinter can run unprivileged (without root privileges) native code on the user's device, which attempts to extract a device fingerprint [20, 21].

As a malicious use case, fingerprinters controlling two or more applications can use such fingerprints for cross-app tracking [65, 77]. Malicious fingerprinters can also use such fingerprints to execute targeted attacks against specific victims [69]. Fingerprinters cannot accomplish this using native identifiers such as IDFA [3] or ADID [25] since they require user consent or can be reset by users. Fingerprinters could also use such fingerprints for security and authentication [11]. For example, game developers can use them to detect aim bots in multiplayer games [26] or detect and ban devices for cheating [16].

In this paper, we do not consider web-based fingerprinting (where a user visits a website controlled by the fingerprinter) since it is more challenging to trigger Rowhammer from the browser [37]. We discuss these challenges and discuss ways to extend our approach to operate within the browser in §7.1.

As mentioned in §2, we focus on DIMM-based DRAM packages in this paper, but our findings are also applicable to other packaging techniques. We assume that the fingerprinter possess a wide array of devices and DRAM modules (DIMMs) with different configurations. Fingerprinters use these devices to discover ways to overcome mitigations such as TRR.

3.2 FP-Rowhammer architecture

At a high level, FP-Rowhammer triggers bit flips on multiple contiguous chunks of memory on a user's device and uses the distribution of the triggered bit flips as a fingerprint. FP-Rowhammer then uses a similarity metric to compare fingerprints extracted from different sessions at different points in time to recognize if these sessions were executed on the same device.

FP-Rowhammer's operation consists of three phases, namely, a *templating phase*, a *hammering phase*, and a *matching phase.*

- In the templating phase, fingerprinters conduct experiments on their own devices to discover ways to overcome Rowhammer mitigations and trigger bit flips.
- In the hammering phase, fingerprinters execute code on users' devices. The fingerprinter's code uses the knowledge gained from the templating phase to trigger bit flips on their devices. They then create a probability distribution out of the triggered bit flips which serves as a fingerprint for the user's device.
- In the matching phase, fingerprinters compare the fingerprint extracted from a user's device against other reference fingerprints to identify the user. They also use the extracted fingerprints to create new or update existing references.

4 Bit Flip Measurement and Analysis

In this section, we first calculate a theoretical upper bound on the entropy that can be obtained from the bits that flip across multiple contiguous chunks of 2 MB of memory. While calculating this theoretical upper bound, we assume that the process variation during the manufacturing of DIMMs is such that the bits that flip within each row (on the same DIMM and across DIMMs) are independent. In this analysis, we also assume that bit flips are deterministic, i.e., hammering the same memory regions always results in the same set of bit flips. Then, we relax these assumptions and validate our analysis by measuring the actual entropy on a set of 3,611 such chunks across 36 DIMMs. We focus on contiguous 2 MB chunks of memory since we can obtain such chunks from the OS without requiring root privileges. Transparent Huge Pages is one of the methods employed by past Rowhammer research to allocate 2 MB chunks of contiguous memory. We highlight that while we present FP-Rowhammer in context of 2 MB huge pages, FP-Rowhammer can operate with any method of allocating contiguous memory [37, 75]. From our measurement, we observe that bit flips are unique across chunks and despite exhibiting non-deterministic behavior, are persistent within each chunk. We use takeaways from our measurement study to design our fingerprinting technique.

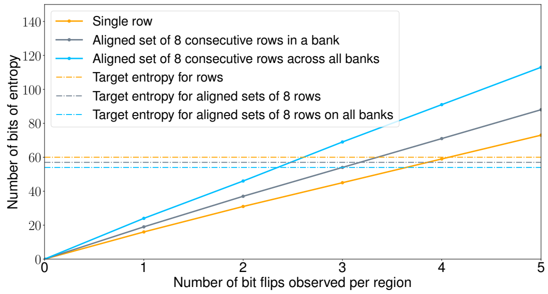


Figure 2: Plots showing the variation in the number of bits of entropy that can be obtained to represent different regions of memory across a trillion DIMMs with varying number of bit flips.

4.1 Theoretical entropy analysis

If we consider that there are approximately 1 trillion DIMMs on the planet, with each DIMM having approximately 10 banks and each bank having approximately 100,000 rows, we will have a total of 10^{18} possible rows. We will need approximately $\log_2(10^{18}) = 59.79 \approx 60$ bits of entropy to represent all these rows. Since rows within DRAM DIMMs are a finer granularity than the number of DIMMs (and correspondingly the number of devices), obtaining an entropy of 60 bits would be sufficient to represent all devices on the planet.

Every row of a DRAM DIMM contains 65,536 capacitors. If process variation results in Rowhammer triggering exactly one bit flip per row (i.e., only one capacitor losing its charge), we can use the index of the flipped bit within each row to at most identify 65,536 rows (equivalent of 16 bits of entropy). Thus, if exactly one bit flips per row, using the index of the flipped bit within the row does not have enough entropy to represent all possible rows across all DIMMs. The solid orange line in Figure 2 shows the amount of entropy available to represent all rows with varying number of bit flips observed per row and the dashed orange line showing the required entropy. We see that if 5 bits flip per row, we can represent all possible rows since we get 73 bits of entropy ($\log_2 \binom{65,536}{5}$).

The analysis presented so far limits us to only observe bit flips within a single row. However, with a contiguous chunk of 2 MB of memory, we can access multiple rows from each bank. For example, in case of dual rank DIMMs having a width of 8 bits (going forward, we refer to this configuration as 2Rx8), a contiguous 2 MB chunk of memory corresponds to an aligned set of 8 consecutive rows (or 524,288 capacitors) within each bank. In this case, we will need an entropy of 57 bits to represent all 10^{17} chunks across all DIMMs. We see that if 4 bits flip among these rows, we get 71 bits of entropy ($\log_2 \binom{524,288}{4}$) to represent them. As contiguous 2 MB chunks are interleaved across banks, we can access these consecutive rows across all banks. If we consider all 16,777,216 capacitors spread across all banks in a 2 MB chunk, we see that 3 bit flips in each chunk is sufficient to obtain an entropy of 54 bits ($\log_2 \binom{16,777,216}{3}$). This entropy is sufficient to represent all 10^{16} such chunks across a trillion DIMMs. The grey and blue solid lines in Figure 2 show the variation in entropy with varying number of bit flips produced per aligned set of 8 consecutive rows and per aligned set of consecutive of 8 consecutive rows across all banks respectively. Dashed lines of the same colors show the required entropy in both cases.

In summary, our analysis indicates that the distribution of bit flips triggered by Rowhammer in individual 2 MB contiguous chunks of memory is potentially unique even if they can produce at least 5 bit flips. We reiterate that the theoretical analysis assumed that all chunks produce bit flips, the distribution of bit flips is independent and that bit flips do not exhibit any non-deterministic behavior. We now perform experiments to measure the actual entropy across such chunks across multiple DIMMs.

4.2 Empirical entropy analysis

Existing Rowhammer research has primarily focused on developing techniques to trigger bit flips [13, 22, 27, 32, 40, 75] in memory. To the best of our knowledge, prior work lacks any analysis of the distribution of bit flips, particularly in terms of their entropy. In this section, we present the first such study on DDR4 DIMMs. Concretely, we first validate our theoretical analysis by measuring the entropy of the distribution of bit flips within a given bank across multiple 2 MB chunks of memory across DIMMs. As mentioned in §1, we find that bit flips are not deterministic and, as a result, merely measuring the entropy of the distribution of bit flips is insufficient to extract a reliable fingerprint. Thus, we also measure the persistence of the distribution of bit flips across repeated measurements to the same chunks across DIMMs.

4.2.1 Test Bed. Our test bed for this measurement consists of 36 identical 2Rx8 DDR4 DIMMs. These DIMMs do not use ECC but they use TRR to mitigate Rowhammer. We fuzz one of these DIMMs to discover a non-uniform hammering pattern that can overcome TRR and trigger bit flips. We observe that the discovered pattern triggers bit flips on all 36 DIMMs.

4.2.2 Methodology. We conduct our experiments in a controlled setting with root privileges that allow us to allocate 1 GB of contiguous memory. This setting gives us control over where we perform our hammering since we observe that the allocation of huge pages in physical memory rarely changes in Linux (verified using pagemap [39]). We confine our hammering to randomly chosen contiguous 2 MB chunks within the allocated huge page that lie within an arbitrarily chosen bank. While hammering, we modify the hammering pattern determined by the fuzzer to only trigger bit flips within the randomly chosen 2 MB chunks.

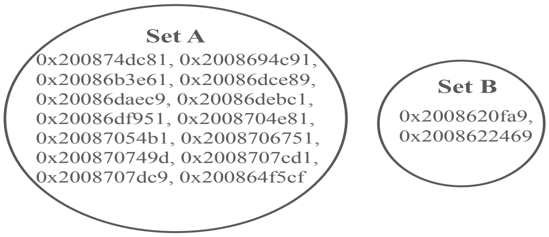

Set A
0x200874dc81, 0x2008694c91,
0x20086b3e61, 0x20086dce89,
0x20086daec9, 0x20086debc1,
0x20086df951, 0x2008704e81,
0x20087054b1, 0x2008706751,
0x200870749d, 0x2008707cd1,
0x2008707dc9, 0x200864f5cf
Set B
0x2008620fa9,
0x2008622469

Figure 3: Set A shows the addresses of bits that flipped when hammering a particular chunk of a DIMM. Set B shows the addresses of bits that flipped when restoring data to the chunk and hammering it again. The two sets are disjoint demonstrating the non-deterministic behavior of bit flips.

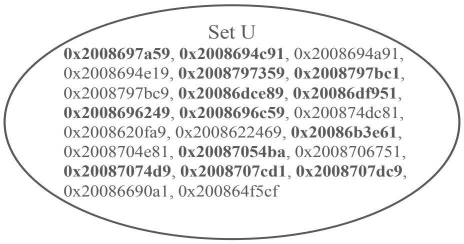

Set U
0x2008697a59, **0x2008694c91**, 0x2008694a91,
0x2008694e19, **0x2008797359**, **0x2008797bc1**,
0x2008797bc9, **0x20086dce89**, **0x20086df951**,
0x2008696249, **0x2008696c59**, 0x200874dc81,
0x2008620fa9, 0x2008622469, **0x20086b3e61**,
0x2008704e81, **0x20087054ba**, 0x2008706751,
0x20087074d9, **0x2008707cd1**, **0x2008707dc9**,
0x20086690a1, 0x200864f5cf

Figure 4: Set U shows the set of all addresses that flipped across 8 attempts to restore the data and hammering the same chunk from Figure 3. The boldfaced addresses are addresses that flipped more than once across the 8 attempts.

4.2.3 Results. Across all 36 DIMMs, we hammered a total of 3,611 chunks. 99.77% of these chunks (3,603 chunks) produced at least one bit flip. Among these chunks, the number of bit flips ranged from 1 bit flip to 1,799 bit flips at an average of 711 bit flips per chunk. Across the chunks that produced bit flips, we record an entropy of 12 bits for the triggered bit flips. We calculated entropy in terms of the number of chunks that had the same set of bit flips as a given chunk. Crucially, we highlight that in our test bed, 12 bits of entropy corresponds to the highest possible normalized entropy of 1.0 [2], which demonstrates that each chunk has a unique set of bit flips (i.e., all 3,611 chunks can be uniquely identified with 12 bits of entropy). We summarize these findings in Table 1. We use the fact that the bit flips in every chunk in our experiment is unique to estimate the expected entropy on all possible chunks. Extrapolating our results, based on the average of 711 bit flips per chunk yields over 7,700 bits of entropy, which is significantly higher than the 60 bits needed to represent such chunks on a trillion DIMMs.

Measurement	Value
Percentage of chunks showing bit flips	99.77%
Minimum number of bit flips per chunk	1 flip
Maximum number of bit flips per chunk	1,799 flips
Average number of bit flips per chunk	711 flips
Measured entropy across 3,603 chunks	12 bits
Normalized entropy	1.0

Table 1: Summary of the measured distribution of bit flips across 36 identical DIMMs.

Takeaway 1: Every contiguous 2 MB chunk of memory has a *unique* set of bit flips when subjected to Rowhammer.

To use the bit flips produced by Rowhammer as a fingerprint, ensuring that they have high entropy on different chunks is not sufficient unless they are also persistent within the same chunk. In our study, we notice that reinitializing regions with the same data and hammering them again does not guarantee that the same bit will flip. In other words, we observe that the bits that flip within a given chunk are not deterministic. For example, Figure 3 shows the sets of bits that flipped when hammering the same chunk on a particular DIMM twice (while restoring the data written to the chunk before hammering again). Set A shows the addresses that flipped during the first attempt which is completely disjoint to set B shows the set of addresses that flipped during the second attempt.

Takeaway 2: Bit flips exhibit *non-deterministic behavior*, i.e., hammering the same set of aggressor rows multiple times does not result in the same set of bit flips.

When we restored the data and attempted to hammer the same chunk 6 more times, we observed 2 attempts with no bit flips and 4 attempts where some addresses in set A flipped again. We visually represent the list of bits that flipped across all 8 attempts in Figure 4. This figure shows that we cannot re-identify a particular chunk by merely employing a set similarity metric like Jaccard index (as proposed by prior research [44, 70]). More importantly, the figure also indicates that some bits (such as those in set A) have a higher probability of flipping and other bits (such as those in set B) have a lower probability of flipping. Leveraging this observation, we compute a probability distribution for the bit flips in each chunk and match similarity of distributions to measure persistence. To extract a probability distribution from a given chunk, we hammer it

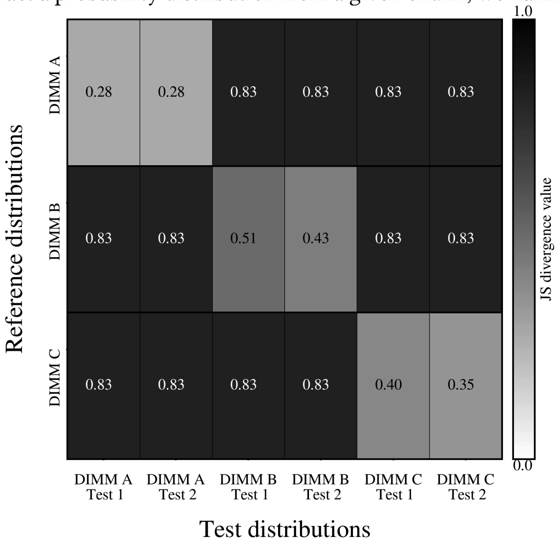


Figure 5: Visualization of the relative persistence of bit flips within given 2 MB chunks of memory across multiple DIMMs.

multiple times and use the count of flips at different indices across all hammering attempts. Then, we use Jensen-Shannon (JS) divergence [58] to compute the similarity of distributions. In Figure 5, we compare the distributions for 3 randomly chosen chunks from 3 different DIMMs across 3 different hammering attempts. We see similarities in the probability distributions computed from the same chunk as compared to distributions across chunks.

Takeaway 3: Different bits have different probabilities of flipping. The distribution of these probabilities is *unique* across contiguous 2 MB chunks of memory and *persistent* within each 2 MB chunk.

5 FP-Rowhammer

5.1 Templating phase

In the templating phase, we (the fingerprinters) seek to discover hammering patterns that can overcome Rowhammer mitigations to trigger bit flips on our own devices, so that we can employ the patterns to trigger bit flips on users' devices in the hammering phase. Concretely, in our experiments on DDR4 DIMMs (that employ in-DRAM TRR [22]), we run Blacksmith's [33] fuzzer to discover non-uniform hammering patterns that can trigger bit flips. We observe that hammering patterns tend to be successful in evading TRR on multiple DIMMs from the same manufacturer. To account for all TRR implementations in the wild, our goal is to discover sufficiently many patterns to overcome all of them.

Once the fuzzer discovers patterns that can trigger bit flips, we evaluate them on our own DIMMs to decide which patterns to employ on users' devices in the hammering phase. We prioritize those patterns that trigger more bit flips as well as those that generalize to trigger bit flips on more DIMMs.

Patterns that trigger more bit flips help account for differences in bit flip behavior when extracting fingerprints. These differences either arise as a result of the inherent non-deterministic behavior of bit flips (§4.2) or as a result of external factors that fingerprinters cannot control. For example, we observe fewer bit flips (with the same pattern on the same DIMM) when running at lower CPU frequency (§6.7). A pattern that produces very few bit flips in the controlled setting of our own devices may not produce any bit flips on a user's device in presence of factors outside our control. In contrast, a pattern that produces a large number of bit flips in our controlled setting may still produce enough bit flips to be able to fingerprint a user's device.

We observe that patterns that need fewer aggressors to overcome TRR (we refer to these as secondary aggressors) tend to trigger more bit flips. Intuitively, by virtue of being shorter patterns, these patterns would activate aggressors that trigger bit flips (we refer to these as primary aggressors) at a higher frequency, resulting in more bit flips. We prioritize patterns that generalize to trigger bit flips on more DIMMs in our possession since they are also likely to generalize to more DIMMs in the wild.

5.2 Hammering phase

In this phase, our goal is to trigger bit flips in a user's device and extract the distribution of bit flips as a fingerprint. Since we do not have knowledge of the type of DIMMs (or their corresponding TRR implementations) on the user's device, we rely on the patterns discovered in the templating phase to trigger bit flips. The hammering patterns discovered by the fuzzer are non-uniform hammering patterns that are defined by a set of aggressors, a phase, an amplitude and a frequency [32]. The phase, amplitude and frequency are such that some aggressors engage with TRR (secondary aggressors) and the others trigger bit flips (primary aggressors). Within the execution of each pattern, we observe that the aggressors accessed at certain points in time always served as primary aggressors regardless of the choice of which addresses were used as primary and secondary aggressors. We also observe that the position of the primary aggressors within the pattern is fixed across all DIMMs where the pattern is able to trigger bit flips.

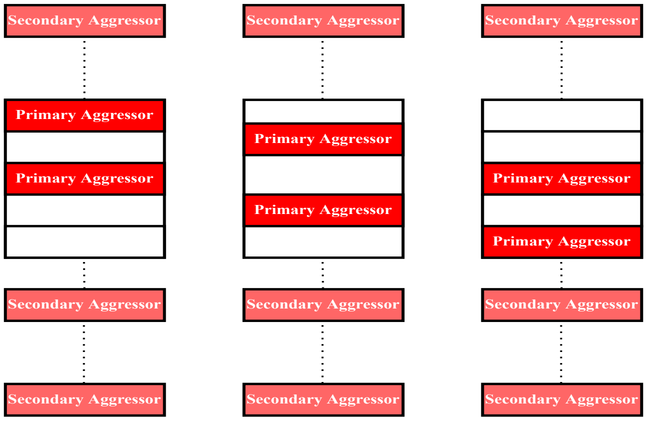


Figure 6: Visualization of a hammering sweep performed within one bank of a contiguous 2 MB chunk. The central rectangular blocks in the visualization represent rows within the chunk. We map primary aggressors in the discovered patterns to rows within the chunk and secondary aggressors to random rows within the same bank. The hammering sweep involves sequentially hammering all pairs of double-sided aggressors as primary aggressors within the bank of the chunk and scanning the other rows for bit flips.

Concretely, for a pattern $a_1, a_2, \ldots a_i, a_{i+1}, \ldots a_n$, addresses placed at indices i and $i + 1$ served as primary aggressors on all DIMMs where the pattern triggered bit flips. To reliably produce bit flips in the hammering phase, we pick addresses such that the primary aggressors form a double-sided aggressor pair, and the secondary aggressors are other addresses within the same bank as the primary aggressors to form our hammering pattern. Fingerprinters can employ timing side channels (described in Appendix H) or other APIs (decode-dimms [72], lspci [46], etc) to infer DRAM configuration/-manufacturer and pick appropriate hammering patterns.

In this section, we discuss the hammering phase in context of a single DIMM present on the user's device. We discuss ways to extend the hammering phase to devices having multiple DIMMs across multiple channels in Appendix I. To execute the discovered patterns, we allocate transparent huge pages on a user's device to obtain contiguous chunks of 2 MB of memory. Transparent huge pages can be obtained using the *madvise* [47] system call that does not require root privileges. We can access all addresses within the huge page by modifying the lower 21 bits of the starting address of the chunk. This allows us to pick double-sided aggressor pairs since such chunks typically provide access to contiguous rows across multiple banks of a DIMM. For example, in case of the user's device having one 1Rx8 DIMM, the chunk gives us access to 16 contiguous rows within each of the 16 banks on the DIMM. To trigger bit flips, we first choose a particular bank within the chunk. We can do this since most bits in the address that determine the bank are contained with the lower 21 bits in most CPU architectures [13, 62]. Then, we map the primary aggressors in the discovered patterns to double-sided aggressors within the chosen bank of the chunk and secondary aggressors to random addresses within the same bank (possibly outside the chunk). While we cannot determine the exact row of a given address from the lower 21 bits, we do know the relative position of rows with respect to the row corresponding to the start address of the chunk. This allows us to choose different pairs of rows as primary aggressors within the chunk.[1]

We sequentially consider all pairs of double-sided aggressors within the bank of the 2 MB chunk as primary aggressors and execute the discovered pattern. Upon executing the pattern with each pair of primary aggressors, we record the addresses that had bit flips within the allocated chunk and the corresponding change in the data written at those addresses. We then restore the original data written to the chunk, shift our primary aggressors to the next set of double-sided aggressors within the chunk and repeat the same procedure. We refer to this operation of hammering all possible double-sided aggressors as primary aggressors within the allocated chunk as a hammering sweep. Figure 6 visualizes the *hammering sweep*. To account for non-determinism in bit flips, we repeat the hammering sweep multiple times on the chunk.

For uncached memory access, we use the clflush [74] instruction which is unprivileged on Intel and AMD (x86-based processors).

5.3 Matching phase

With the observation from §4.2 that the distribution of bit flips within a bank of contiguous 2 MB chunks is highly unique and stable, we compare the similarity of these distributions to fingerprint them. From the information recorded in the hammering phase, we identify the relative positions and counts of the capacitors that flipped within a contiguous 2 MB chunk (indexed from 0 to 1,048,576 in case of 1Rx8 DIMMs). We use these counts to create an empirical probability distribution for each capacitor to flip within the chunk. We then compare the similarity of this distribution against previously extracted distributions using JS divergence to identify the chunk. In case the newly extracted distribution is significantly different from all previously extracted distributions, we consider the distribution to have been extracted from a new DIMM and use the distribution as a reference for that DIMM.

However, we cannot guarantee access to the same 2 MB chunks on a user's device, since memory allocation is handled by the OS. Thus, if we obtain two different 2 MB chunks of memory on a user's device during two different sessions and compared their distributions, we would incorrectly conclude that different devices were used during these sessions. For example, the OS may allocate a particular huge page in one session and a different huge page from the same DIMM in a subsequent session. The bit flip probability distribution extracted from these huge pages would be different since the distribution is unique for different 2 MB chunks.

We overcome this challenge by taking inspiration from the birthday paradox [1, 7], which shows that there is a high probability for at least one entity to be present in multiple groups even for modest-sized groups. Concretely, we hammer multiple 2 MB chunks during each session to ensure that we hammer at least one previously hammered chunk. For example, suppose the user's device has 1 GB of memory which corresponds to 512 different 2 MB chunks of memory. From the birthday paradox, if we were to hammer 64 chunks each in two different sessions with the user's device, then the probability that at least one chunk would overlap between them is over 99.9%. We derive this from first principles in Appendix B.

One drawback to this approach is that hammering a large number of chunks would prolong the duration of the hammering phase, thereby making it less efficient. However, we can overcome this by building up references across sessions, which would result in hammering fewer chunks in the long term. For example, suppose we have reference distributions to 64 different chunks from one session. In a subsequent session with the same device, we hammer 64 chunks such that only one chunk happens to overlap with the reference. We can now combine the distributions of the 63 non-matching chunks to our reference to have an updated reference from 127 different chunks from that device. When running a subsequent session on the same device, we have a higher probability that an allocated chunk would match our reference, since the reference size has increased. Upon reaching the limiting case where we have references for all possible chunks, merely hammering one chunk in the hammering phase would be sufficient, thereby resulting in higher efficiency. We show the progressive decrease in number of chunks to sample with growing reference sizes in Appendix B.

6 Evaluation

We evaluate FP-Rowhammer in terms of the uniqueness of its fingerprints, stability of its fingerprints, time taken to extract fingerprints and robustness of its fingerprints in presence of external factors.

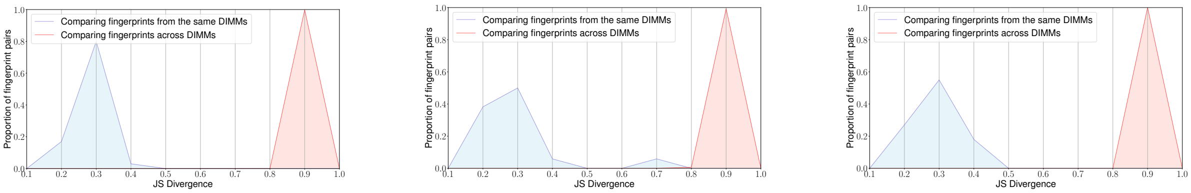
6.1 Test bed

Since the likelihood of an unintended bit flip that results in a crash is non-trivial, we do not evaluate FP-Rowhammer in the wild. Instead, we evaluate FP-Rowhammer on a test bed in our lab.

6.1.1 DIMMs. Our test bed consists of 35 single rank DIMMs having a width of 8 bits (1Rx8), 11 single rank DIMMs having a width of 16 bits (1Rx8) and 36 dual rank DIMMs having a width of 8 bits (2Rx8) from one major DRAM manufacturer. Our test bed also contains 10 1Rx8 DIMMs, 2 1Rx16 DIMMs and 4 2Rx8 DIMMs from another major DRAM manufacturer. All the DIMMs in our test bed are DDR4 DIMMs that do not employ ECC but employ undocumented TRR implementations to combat Rowhammer. Overall, our test bed contains 98 DIMMs that include 3 different geometries or topologies each across 2 major DRAM manufacturers.

6.1.2 Processors. We installed these DIMMs among 12 Intel Core i7 desktops, which includes 2 Skylake desktops, 9 Kaby Lake desktops and one Coffee Lake desktop. While we focus on Intel desktops

[1] Altering the lower 21 bits also enables a timing side-channel (also used by Blacksmith [33]) to verify the allocation of contiguous memory.



(a) Distribution of JS divergence values across all pairs of fingerprints from 36 identical 2Rx8 DIMMs

(b) Distribution of JS divergence values across all pairs of fingerprints from 35 identical 1Rx8 DIMMs

(c) Distribution of JS divergence values across all pairs of fingerprints from 11 identical 1Rx16 DIMMs

Figure 7: Plots showing the distribution of JS divergence values when comparing bit flip distributions obtained from the same pair of DIMMs and across different DIMMs.

running Linux in our experiments, FP-Rowhammer is not specifically tied to a particular processor or operating system[2]. Bit flip data from our test bed is available at this link for future research.

6.2 Experimental methodology

Eleven hammering patterns were sufficient to trigger bit flips on all 98 DIMMs in our test bed, showing that they were effective at triggering bit flips across DIMMs. We were able to reuse the same hammering patterns across DIMMs with the same geometry from the same manufacturer [32]. As a concrete example, one hammering pattern was successful in triggering bit flips on all 35 1Rx8 DIMMs from a particular manufacturer.

For each DIMM in our test set, we perform a hammering sweep (visualized in Figure 6) on a particular bank of multiple 2 MB chunks of memory with the appropriate pattern and record the resulting distribution of bit flips in that chunk. To compute the resulting distribution, we repeat the hammering sweep operation multiple times on each chunk. We consider the set of bit flip probability distributions of all hammered chunks on a DIMM as its fingerprint.

Given two fingerprints, we compute the JS divergence on all pairs of bit flip probability distributions between them. We consider the two fingerprints to match (correspond to the same DIMM) if the minimum JS divergence value across all pairs is below an empirically determined threshold. In our experiments, we extract multiple fingerprints from each DIMM and refer to the first fingerprint extracted from a particular DIMM as a reference fingerprint for that DIMM. We refer to subsequent fingerprints extracted from each DIMM as test fingerprints. When computing accuracy, precision and recall, we compare all test fingerprints against each reference.

6.3 How unique are the fingerprints extracted by FP-Rowhammer?

For this evaluation, we extract 2 fingerprints from each DIMM in our test bed using the aforementioned methodology. In these experiments, we extracted test fingerprints within the space of a few hours of extracting the reference fingerprint. We did not reseat the DIMMs in the interim period. We repeated the hammering sweep operation 8 times and activated aggressors 10,000,000 times.

Figure 7(a) shows the recorded minimum JS divergence when matching fingerprints from each of the 36 2Rx8 DIMMs against reference fingerprints from each of them. From the figure, we clearly see that the minimum JS divergence computed among distributions taken from the same DIMMs is significantly lower than the minimum JS divergence computed among distributions taken across different DIMMs. Figure 7(b) and Figure 7(c) shows similar plots across 35 1Rx8 DIMMs and 11 1Rx16 DIMMs respectively. We report similar plots for the other manufacturer in Appendix D.

The clear separation in JS divergence computed on pairs of fingerprints (bit flip distributions) taken from the same DIMM against those taken from different DIMMs allows us to pick multiple thresholds for JS divergence to uniquely identify DIMMs. By picking appropriate thresholds[3], we attain an overall fingerprint accuracy of 99.91%, corresponding to a precision of 100% and recall of 97.06%. We highlight that even though we incrementally grew the size of our test bed by purchasing new DIMMs, we did not have to tune thresholds to attain the reported accuracy. Overall, these results show that FP-Rowhammer has very high discriminative power regardless of DRAM manufacturer, geometry and processor.

6.4 How stable are the fingerprints extracted by FP-Rowhammer?

We evaluate the stability of FP-Rowhammer's fingerprints over time. Since we have more DIMMs than desktops, we first evaluate stability on a set of 10 random DIMMs across both manufacturers (6 and 4 DIMMs respectively) by extracting fingerprints from them once a day for 10 days. We highlight that we do not re-seat DIMMs at anytime during this evaluation which is what we would expect from users in wild. We repeated the hammering sweep operation 8 times and activated the aggressors 200,000 times.

Figure 8 shows the variation in FP-Rowhammer's accuracy, precision and recall on these DIMMs over time. From the plot we see that all metrics roughly remain constant with some minor fluctuations. Importantly, the plot does not show any trend of decline which indicates that the fingerprints extracted by FP-Rowhammer are stable. These metrics were computed using the same threshold for each day which indicates that the JS divergence values (and the corresponding bit flip probabilities) remain unchanged. We highlight that the stability is not a result of our specific choice for the

[2] Researchers have already demonstrated Rowhammer on other processors [34, 75] and operating systems [8, 75]

[3] We picked thresholds by running the same experiment on a smaller subset of DIMMs.

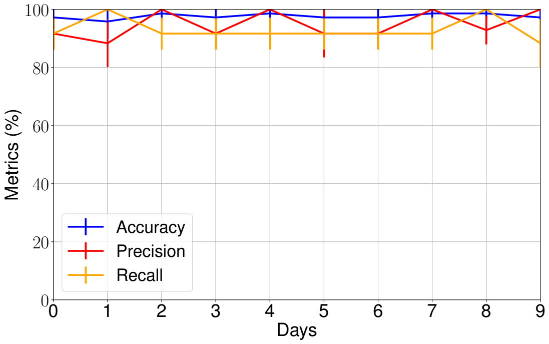


Figure 8: Variation in the accuracy, precision and recall of the fingerprints extracted by FP-Rowhammer on a set of 10 DIMMs over 10 days. The metrics roughly remain constant with minor fluctuations which provides strong evidence that the fingerprints extracted by FP-Rowhammer are stable.

threshold since we record similar accuracy, precision and recall even with slightly altered thresholds.

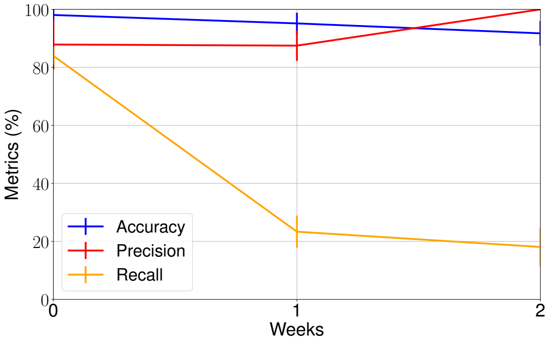


Figure 9: Variation in the accuracy, precision and recall of FP-Rowhammer's fingerprints extracted on our entire test bed (98 DIMMs) over 2 weeks and again over 4 weeks. Unlike our previous observation, there is a change in metrics with the recall declining significantly. We suspect that this decline emerges from having to re-seat DIMMs in our experiments.

Motivated by these results, we increased the scale of our evaluation by evaluating on all DIMMs, first over a period of 2 weeks and then over a period of 4 weeks. For these experiments, we had to re-seat DIMMs in the interim period to cover all DIMMs due to the limited number of desktops at our disposal. Figure 9 shows how FP-Rowhammer's accuracy, precision and recall change over two weeks and over four weeks on our entire set of DIMMs. At a common threshold of 0.8 for JS divergence, we only see minor fluctuations in precision and accuracy, but we see a significant decline in recall. When we tried to change the threshold to improve the recall, we observed that it came at the cost of precision. We were unable to pick a threshold that gave us high precision and recall.

We suspect that this instability in FP-Rowhammer's fingerprints is a result of our experimental setup where we re-seat DIMMs. If this is indeed the case, FP-Rowhammer's fingerprints would be stable in the wild since users rarely re-seat their DIMMs. The ideal way to confirm our suspicion would be to evaluate the stability of FP-Rowhammer's fingerprints on our entire test bed over an extended period of time (4 weeks) without having to re-seat DIMMs. Since this is not feasible due to the limited number of desktops at our disposal, we run experiments that provide strong evidence that re-seating induces the instability in FP-Rowhammer's fingerprints.

Experiment	Accuracy	Precision	Recall
No rebooting/re-seating	100%	100%	100%
Only Rebooting	100%	100%	100%
Re-seating (and rebooting)	91.67%	95.83%	50%

Table 2: Even when comparing fingerprints extracted over a span of a few minutes, we see that there is a significant drop in recall only when we re-seat DIMMs.

Concretely, we run three different experiments on six randomly chosen identical DIMMs from our test bed. In the first experiment, we hammer and extract two fingerprints within the space of a few minutes. We also extract two fingerprints within the space of a few minutes in the second experiment, but we re-seat the DIMM (i.e., take out and insert back) after extracting the first fingerprint. We do the same thing in the last experiment but reboot the desktop after extracting the first fingerprint. Since we cannot re-seat a DIMM without rebooting the device, we run the third experiment to see the impact of rebooting on the stability of our fingerprint.

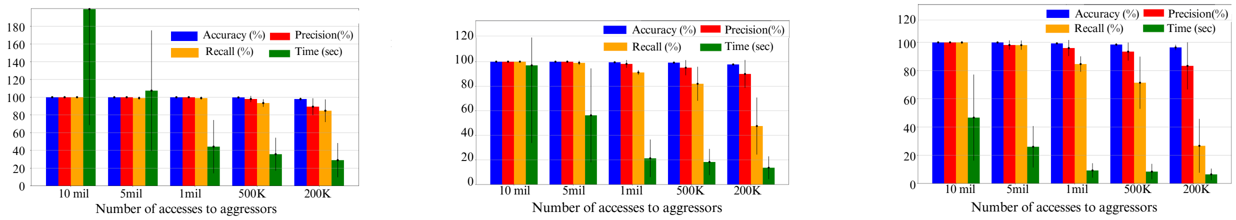
We present the highest accuracy attained, with the corresponding precision and recall for all three experiments in Table 2. The table shows a drastically lower value of 50% recall for the experiment where we re-seated the DIMM. The other two experiments show perfect precision and recall. These observations support our hypothesis that the instability in FP-Rowhammer's fingerprints is a result of re-seating the DIMMs.

We suspect that minute differences in alignment when re-seating DIMMs change the electrical characteristics (resistance, capacitance, inductance, etc) of the wires in the DRAM chips, which ultimately changes the set of bits that flip. Consistent with prior research [23], these results show that the set of bits that flip (and correspondingly the fingerprints extracted by FP-Rowhammer) are a property of the entire system being subjected to Rowhammer and not just the DRAM devices. Thus, we can expect FP-Rowhammer's fingerprints to be stable in the wild since users rarely re-seat their DIMMs.

Processor independence Results in the previous section (§6.3) show that FP-Rowhammer can uniquely fingerprint DIMMs across different processors (our testbed consists of Skylake, Kaby Lake and Coffee Lake CPUs). However, fingerprints extracted from the same DIMM across processors would be affected since placing the DIMM on different processors necessarily requires re-seating the DIMM.

6.5 How long does FP-Rowhammer take to extract fingerprints?

Efficiency quantifies the time taken to extract fingerprints [69]. When accessing aggressors 10,000,000 times to execute Rowhammer, the hammering sweep operation takes an average of 20 seconds on a single 2 MB chunk, which further increases when repeating the operation to account for non-determinism. Even in the limiting case where we have enough references to cover all 2 MB chunks



(a) Plots when repeating the hammering sweep operation 8 times

(b) Plots when repeating the hammering sweep operation 4 times

(c) Plots when repeating the hammering sweep operation 2 times

Figure 10: Plots showing the variation in accuracy, precision, recall and time elapsed to extract fingerprints when varying the number of accesses to trigger bit flips and varying the number of times we repeat the hammering sweep operation.

in a given DIMM, FP-Rowhammer takes almost 3 minutes to extract a fingerprint when accessing aggressors 10,000,000 times and repeating the hammering sweep 8 times. Such a long duration for fingerprinting would only be acceptable in certain use cases where a user keeps an application open for a long time, such as a gaming application or a video streaming application. We can improve FP-Rowhammer's efficiency by reducing the number of times we access aggressors to trigger bit flips or reducing the number of times we repeat the hammering sweep operation. Employing either approach is subject to making sure that they do not significantly degrade FP-Rowhammer's fingerprinting accuracy.

In this section, we present a comprehensive analysis of the tradeoff between fingerprint accuracy and efficiency when running FP-Rowhammer. Concretely, we present the fingerprint accuracy and the average time taken to extract a fingerprint from one 2 MB chunk across 15 different configurations on all the DIMMs in our test bed. These 15 configurations differ in terms of the number of times we access the aggressors when executing Rowhammer and the number of times we repeat the hammering sweep operation. We consider 5 different values for the number of accesses to each aggressor ranging from 10 million accesses to 200,000 accesses and 3 different values for the number of times we repeat the hammering sweep operation (8 times, 4 times and 2 times).

Figure 10 shows the accuracy, precision, recall and time elapsed in all 15 configurations. These results show that we can drastically reduce the time taken to extract fingerprints while maintaining high accuracy. For example, accessing aggressors 1,000,000 times and repeating the hammering sweep operation twice only takes 9.92 seconds, while still maintaining 95.96% precision and 84.61% recall (99.27% accuracy). In this configuration, FP-Rowhammer is suitable as a second factor of authentication [14, 78].

To further decrease the time taken to extract fingerprints, we ran an experiment where we hammered half the rows in each 2 MB chunk instead of all rows in the hammering sweep operation. With this optimization, it took us 4.54 seconds to extract fingerprints corresponding to 95.33% precision and 83.84% recall (99.09% accuracy). Fingerprinters can further constrain the hammering sweep to fewer rows or employ multi-back hammering [37] to further decrease the time taken to extract fingerprints.

6.6 What is the impact of temperature on FP-Rowhammer?

Prior research has shown that temperature affects Rowhammer bit flips [60, 70]. They study the differences in bit flips at different temperatures by clamping the DRAM module with heater pads to raise its operating temperature [60]. Since most users are not likely to use such pads to heat up their DIMMs, it would be unrealistic to use them to evaluate FP-Rowhammer's fingerprints. However, since changes to ambient temperature are more realistic, in this section, we compare bit flip distributions at different ambient temperatures and evaluate their impact on FP-Rowhammer.

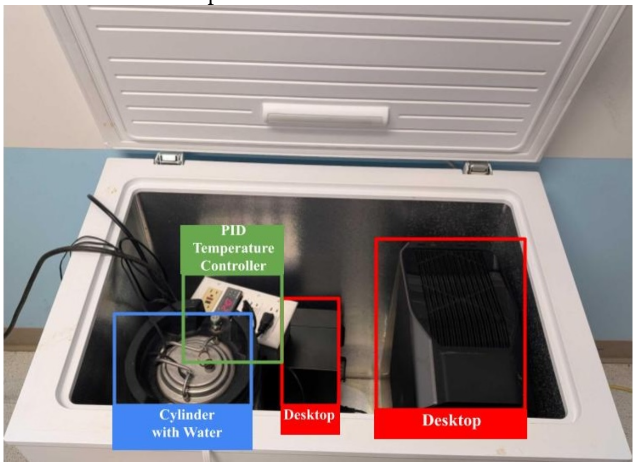
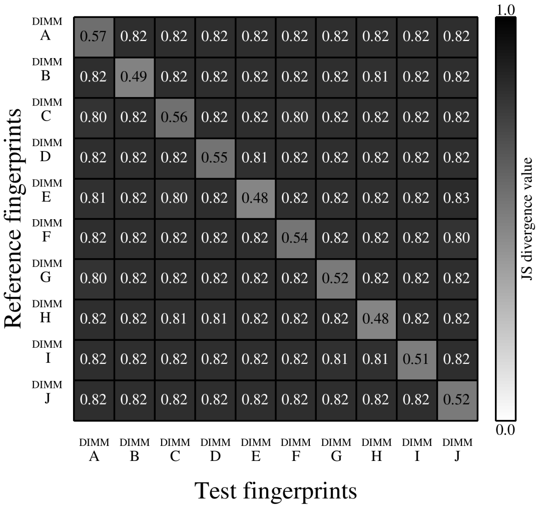


Figure 11: We placed desktops in a temperature controlled environment and altered its temperature using a PID temperature controller. We place a cylinder of water within this environment to minimize temperature fluctuations.

Concretely, we place desktops inside a temperature controlled environment (Insignia chest freezer) and alter its temperature using a PID temperature controller [24]. We place a cylinder of water within this environment to minimize temperature fluctuations. The temperature controller also includes a thermometer which records the current temperature. We use the temperature controller to set distinct temperatures and triggered bit flips once the temperature stabilized. We set the temperatures to 15°C, 20°C, 25°C, 30°C, 35°C and 40°C. Then, we arbitrarily chose 10 DIMMs with identical



	DIMM A	DIMM B	DIMM C	DIMM D	DIMM E	DIMM F	DIMM G	DIMM H	DIMM I	DIMM J
DIMM A	0.57	0.82	0.82	0.82	0.82	0.82	0.82	0.82	0.82	0.82
DIMM B	0.82	0.49	0.82	0.82	0.82	0.82	0.82	0.81	0.82	0.82
DIMM C	0.80	0.82	0.56	0.82	0.82	0.80	0.82	0.82	0.82	0.82
DIMM D	0.82	0.82	0.82	0.55	0.81	0.82	0.82	0.82	0.82	0.82
DIMM E	0.81	0.82	0.80	0.82	0.48	0.82	0.82	0.82	0.82	0.83
DIMM F	0.82	0.82	0.82	0.82	0.82	0.54	0.82	0.82	0.82	0.80
DIMM G	0.80	0.82	0.82	0.82	0.82	0.82	0.52	0.82	0.82	0.82
DIMM H	0.82	0.82	0.81	0.81	0.82	0.82	0.82	0.48	0.82	0.82
DIMM I	0.82	0.82	0.82	0.82	0.82	0.82	0.81	0.81	0.51	0.82
DIMM J	0.82	0.82	0.82	0.82	0.82	0.82	0.82	0.82	0.82	0.52

Figure 12: A heatmap showing JS divergence values across DIMMs. When computing values for the same DIMM, we compared bit flip distributions obtained at $15°C$ against those obtained at $40°C$. When computing values across DIMMs, we compared bit flip distributions obtained at $15°C$. The clear separation in JS divergence values shows that changes to ambient temperature do not affect FP-Rowhammer.

configurations, placed them within these desktops to trigger bit flips at different temperatures and compared their distributions. We picked these values since they span the realistic range of ambient temperatures. Figure 11 shows our experimental setup.

In this section, we present results across the two extreme temperatures, i.e., when extracting reference fingerprints (bit flip distributions) at $15°C$ and comparing them against fingerprints extracted at $40°C$. We did not observe a clear trend in the number of bit flips when increasing the temperature, with 6 DIMMs showing fewer bit flips at higher temperatures and 4 DIMMs showing more bit flips at higher temperatures. We report JS divergence values ranging from 0.48 to 0.57 when comparing bit flip distributions across different temperatures for the same DIMM and JS divergence values over 0.80 when comparing bit flip distributions across DIMMs at the same temperature. The clear separation in JS divergence implies 100% accuracy on these DIMMs even when comparing fingerprints across temperatures. These results show that FP-Rowhammer's fingerprints are stable even with changes to ambient temperature. Figure 12 shows a heatmap of JS divergence values across the 10 arbitrarily chosen DIMMs. Appendix G lists the number of bit flips recorded at these temperatures on all 10 DIMMs. We observed similar results at the other temperatures.

6.7 What is the impact of other external factors on FP-Rowhammer?

We evaluate the robustness of FP-Rowhammer's fingerprints to other external factors (outside the control of the fingerprinter) that can influence bit flips. Concretely, we evaluate FP-Rowhammer's robustness in context of CPU frequency since fewer bits flip at lower frequencies. CPU frequencies are subject to change since some CPU governors (e.g., ondemand [15]) dynamically scale the CPU frequency based on the CPU load, which depend on other applications running on a user's device. Thus, matching fingerprints extracted at different frequencies corresponds to matching fingerprints when running other applications with different loads.

We first extract fingerprints (bit flip distributions) on all DIMMs in our test bed when running the CPU at 3600 MHz (its highest frequency). Then, we extract fingerprints from the same DIMMs when running the CPU at 2800 MHz. Matching fingerprints across these frequencies simulates an extreme scenario where we trigger bit flips in presence of different applications that exert different loads on the CPU. On average, we report 2 orders of magnitude difference in the number of bit flips that trigger at the two frequencies. When matching fingerprints across these frequencies, FP-Rowhammer attains an accuracy of 99.09%, corresponding to a precision of 94.2% and recall of 81.56%. Thus, FP-Rowhammer only suffers a modest drop in fingerprint accuracy when matching fingerprints extracted at different frequencies.

Importantly, these results also demonstrate the impact of employing hammering patterns that produce the most number of bit flips. We repeated the same experiment by choosing a hammering pattern that triggered fewer bit flips in the templating phase. We notice that even when running at the highest frequency of 3600 MHz, this pattern triggers bit flips on only 10 DIMMs. On these DIMMs, this pattern did not trigger any bit flips at the lower frequency of 2800 MHz, resulting in 9% recall when comparing fingerprints.

We also compared fingerprints while running certain common applications in the background. Concretely, on a subset of 10 random identical DIMMs, we extracted fingerprints when running 1) no background applications, 2) running LibreOffice Writer, and 3) running YouTube on Mozilla Firefox in the background. We report 100% accuracy when matching these fingerprints against each other.

7 Discussion

7.1 Extension to web and mobile

Executing Rowhammer, which is a prerequisite to use FP-Rowhammer requires access to contiguous memory and fast, uncached memory access (§2.2). In this section, we discuss ways to satisfy these requirements on web and mobile systems to execute Rowhammer. Upon executing Rowhammer, fingerprinters can match bit flip distributions (§5) on the resultant bit flips for fingerprinting.

Both Rowhammer.js [27] and SMASH [13] describe ways to allocate contiguous memory from JavaScript using Transparent Huge Pages. SMASH also describes self-evicting patterns that extend the many-sided hammering patterns proposed by TRRespass [22] to overcome TRR and trigger Rowhammer from the browser. However, TRResspass was unable to trigger bit flips on the DIMMs in our test bed, even after running their fuzzer for two weeks. We can also use Blacksmith's fuzzer [33] to discover such patterns, although porting Blacksmith to JavaScript is non-trivial because we cannot guarantee precisely accessing aggressors at particular points of time within a refresh interval without explicitly flushing the cache.

Most recently, Sledgehammer [37] proposed a novel method to allocate contiguous memory without relying on Transparent Huge Pages to trigger bit flips from the browser. Drammer [75] describes ways to allocate contiguous memory on mobile platforms such as

Android. Techniques proposed by SMASH and SledgeHammer to flush the cache can potentially be applied on mobile too to ensure fast, uncached memory access to execute Rowhammer.

The main contribution of our work, FP-Rowhammer, is exploiting the distribution of bit flips for fingerprinting. In this paper, we trigger Rowhammer bit flips using native code on desktop systems, but with considerable engineering effort, our contributions can be extended to web and mobile by leveraging techniques proposed by Drammer, SMASH, Rowhammer.js and SledgeHammer.

7.2 Applications of FP-Rowhammer

The unique and stable fingerprints extracted by FP-Rowhammer can be used for authentication. Even FP-Rowhammer's least efficient configuration that takes almost 3 minutes to extract fingerprints can be used for cheat detection in multiplayer gaming since users tend to keep gaming applications open for extended periods of time. Shorter variants of FP-Rowhammer that take 4.5 seconds to extract fingerprints are comparable to secondary forms of authentication such as CAPTCHAs or SMS-based authentication. FP-Rowhammer provides better user experience over these factors since FP-Rowhammer can run seamlessly in the background without requiring user interaction. Even shorter variants of FP-Rowhammer can potentially be used as the primary form of authentication.

However, using FP-Rowhammer for authentication comes with non-zero risk to benign users. While FP-Rowhammer only seeks to trigger bit flips within its own memory [60], it could potentially crash another application or the OS by flipping a sensitive bit reserved for a different application or reserved by the OS. While background applications did not crash in our experiments (§6.7), we cannot rule out their possibility. Although they were rare, we also report instances of devices crashing in our experiments, potentially due to bit flips in memory reserved for the OS.

Hammering those rows that are not at the edge of 2 MB chunks mitigates this concern. We observe most bit flips on rows that are adjacent to the aggressor rows with the number of bit flips successively decreasing on rows that are farther away [41]. Thus, this solution would reduce the chance of unintended bit flips by ensuring that most bit flips are confined to the desired 2 MB block. OS vendors can help mitigate these concerns by allocating specific DRAM rows for FP-Rowhammer that are far away from the rows allocated to the OS or other applications.

Another risk presented by FP-Rowhammer is that it could wear out memory modules if it is used to constantly trigger bit flips for fingerprinting. FP-Rowhammer's approach of triggering bit flips with fewer accesses to aggressors helps mitigate this concern. Such concerns can also be mitigated by only employing other fingerprinting techniques for the common cases and sparingly employing FP-Rowhammer to only handle the critical cases.

7.3 Comparison with other Rowhammer attacks

In this section, we compare FP-Rowhammer against other Rowhammer attacks in terms of reliably fingerprinting devices. While fingerprinting is not the intent behind traditional Rowhammer attacks, access to additional information from the attack could contribute toward a more fine-grained fingerprint. For example, consider a Rowhammer attack to obtain root privileges on a particular device. Since access to certain information (e.g., network interfaces [66]) is restricted to root, the entity triggering Rowhammer could combine this information with the information they already possess to create a more fine-grained fingerprint [17, 69].

FP-Rowhammer's fingerprints would have higher stability and higher uniqueness than such fingerprints. Concretely, device information including those only available to root could naturally change over time [76] or could be intentionally altered [36, 69], causing fingerprints to change. FP-Rowhammer's fingerprints are not affected by changes to such information. Our experiments also show that FP-Rowhammer's fingerprints remain stable across time (§6.4) even in presence of external factors that affect the behavior of bit flips (§6.7). Other Rowhammer attacks do not evaluate their approach in presence of such factors nor do they discuss the non-deterministic behavior of bit flips [13, 34, 71, 75]. Additionally, even when manually modifying multiple devices to contain the same device information along with identical hardware and software configurations, FP-Rowhammer would still be able to distinguish between them (§6.3) since it captures the side-effects of process variation for fingerprinting. Our experiments (§6.5) also show that the time taken by FP-Rowhammer to extract fingerprints is comparable to other Rowhammer attacks [13, 34, 75].

Lastly, we note that FP-Rowhammer is less invasive than traditional Rowhammer attacks. In contrast to such traditional Rowhammer attacks that seek to trigger bit flips in the memory reserved for the OS or other applications, FP-Rowhammer only seeks to trigger bit flips within its own memory [60]. As a result, FP-Rowhammer can only be used to uniquely identify devices while other Rowhammer attacks can also compromise the integrity and confidentiality of the OS and other applications for more nefarious attacks.

8 Related Work

Drammer [75] proposes a technique to overcome the operating system's abstractions to force a victim to allocate memory in a region that is susceptible to Rowhammer. However, to trigger the same bit flip again, their proposed technique requires the overall memory layout to remain unchanged (no allocation/deallocation of memory by other processes), which makes it impractical for fingerprinting. Researchers have also leveraged memory deduplication and MMU paravirtualization to overcome memory restrictions and trigger bit flips on memory allocated to a victim VM on cloud machines [64, 79]. However, these capabilities are either not enabled or unavailable on most end-user devices.

TRRespass [22] and Blacksmith [32] introduce fuzzers that discover hammering patterns to overcome TRR and trigger bit flips. However, fingerprinters get limited insights on employing these patterns to trigger bit flips on a large number of DIMMs since both papers do not provide guidance on which patterns to employ beyond those that produce bit flips.

9 Conclusion

We presented FP-Rowhammer to extract unique and stable fingerprints even for devices with identical hardware and software configurations. To this end, FP-Rowhammer leverages Rowhammer to capture the side-effects of process variation in the underlying manufacturing process of memory modules. FP-Rowhammer's design involves a novel sampling strategy to overcome memory allocation constraints, identification of effective hammering patterns to bypass Rowhammer mitigations and trigger bit flips at scale, and handling non-deterministic bit flips through multiple hammering iterations and divergence analysis of probability distributions. Our evaluation of FP-Rowhammer on 98 DIMMs across 6 sets of identical DRAM modules from two manufacturers showed that it can extract high entropy and stable fingerprints with an overall accuracy of 99.91% while being robust and efficient. FP-Rowhammer cannot be trivially mitigated without fundamentally fixing the underlying the Rowhammer vulnerability, which – despite existing countermeasures – is expected to escalate as the density of DRAM chips increases in the future.

References

[1] Morton Abramson and WOJ Moser. More birthday surprises. *The American Mathematical Monthly*, 77(8):856–858, 1970.

[2] Nampoina Andriamilanto, Tristan Allard, Gaëtan Le Guelvouit, and Alexandre Garel. A large-scale empirical analysis of browser fingerprints properties for web authentication. *ACM Trans. Web*, 16(1), sep 2021.

[3] Apple. User Privacy and Data Use. https://developer.apple.com/app-store/user-privacy-and-data-use/.

[4] Zelalem Birhanu Aweke, Salessawi Ferede Yitbarek, Rui Qiao, Reetuparna Das, Matthew Hicks, Yossi Oren, and Todd Austin. Anvil: Software-based protection against next-generation rowhammer attacks. In *Proceedings of the Twenty-First International Conference on Architectural Support for Programming Languages and Operating Systems*, ASPLOS '16, page 743–755, New York, NY, USA, 2016. Association for Computing Machinery.

[5] Kuljit S. Bains and John B. Halbert. Distributed row hammer tracking, 2012. US Patent US20140095780A1.

[6] Benjamin Seufert. Apple to Ad Tech: "Fingerprinting is Never Allowed". https://mobiledevmemo.com/apple-to-adtech-fingerprinting-is-never-allowed/.

[7] Erik Bosman, Kaveh Razavi, Herbert Bos, and Cristiano Giuffrida. Dedup est machina: Memory deduplication as an advanced exploitation vector. In *2016 IEEE Symposium on Security and Privacy (SP)*, pages 987–1004, 2016.

[8] Erik Bosman, Kaveh Razavi, Herbert Bos, and Cristiano Giuffrida. Dedup est machina: Memory deduplication as an advanced exploitation vector. In *2016 IEEE Symposium on Security and Privacy (SP)*, pages 987–1004, 2016.

[9] Brave. Fingerprinting Protection Mode. https://github.com/brave/browser-laptop/wiki/Fingerprinting-Protection-Mode.

[10] Brave Privacy Team. Fingerprinting defenses 2.0. https://brave.com/privacy-updates/4-fingerprinting-defenses-2.0/.

[11] Yushi Cheng, Xiaoyu Ji, Juchuan Zhang, Wenyuan Xu, and Yi-Chao Chen. Demicpu: Device fingerprinting with magnetic signals radiated by cpu. In *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security*, CCS '19, page 1149–1170, New York, NY, USA, 2019. Association for Computing Machinery.

[12] Lucian Cojocar, Kaveh Razavi, Cristiano Giuffrida, and Herbert Bos. Exploiting correcting codes: On the effectiveness of ecc memory against rowhammer attacks. In *2019 IEEE Symposium on Security and Privacy (SP)*, pages 55–71, 2019.

[13] Finn de Ridder, Pietro Frigo, Emanuele Vannacci, Herbert Bos, Cristiano Giuffrida, and Kaveh Razavi. SMASH: Synchronized many-sided rowhammer attacks from JavaScript. In *30th USENIX Security Symposium (USENIX Security 21)*, pages 1001–1018. USENIX Association, August 2021.

[14] Zainul Abi Din, Hari Venugopalan, Henry Lin, Adam Wushensky, Steven Liu, and Samuel T. King. Doing good by fighting fraud: Ethical anti-fraud systems for mobile payments, 2021.

[15] Dominik Brodowski, Nico Golde, Rafael J. Wysocki and Viresh Kumar. Linux CPUFreq CPUFreq Governor. https://www.kernel.org/doc/Documentation/cpu-freq/governors.txt.

[16] easy ANTI-CHEAT. Don't bear with the cheaters. https://www.easy.ac/en-US.

[17] Peter Eckersley. How unique is your web browser? In *Proceedings of the 10th International Conference on Privacy Enhancing Technologies*, PETS'10, page 1–18, Berlin, Heidelberg, 2010. Springer-Verlag.

[18] elinux. PandaBoard. https://elinux.org/PandaBoard.

[19] FingerprintJS. FingerprintJS. https://github.com/fingerprintjs/fingerprintjs.

[20] FingerprintJS. FingerprintJS Android. https://github.com/fingerprintjs/fingerprintjs-android.

[21] FingerprintJS. FingerprintJS iOS. https://github.com/fingerprintjs/fingerprintjs-ios.

[22] Pietro Frigo, Emanuele Vannacc, Hasan Hassan, Victor van der Veen, Onur Mutlu, Cristiano Giuffrida, Herbert Bos, and Kaveh Razavi. Trrespass: Exploiting the many sides of target row refresh. In *2020 IEEE Symposium on Security and Privacy (SP)*, pages 747–762, 2020.

[23] Lukas Gerlach, Fabian Thomas, Robert Pietsch, and Michael Schwarz. A rowhammer reproduction study using the blacksmith fuzzer. In Gene Tsudik, Mauro Conti, Kaitai Liang, and Georgios Smaragdakis, editors, *Computer Security – ESORICS 2023*, pages 62–79, Cham, 2024. Springer Nature Switzerland.

[24] GM Electronica. All-Purpose Temperature Controller. https://www.gmelectronica.com.ar/datasheets/ITC-1000F.pdf.

[25] Google. Advertising ID. https://support.google.com/googleplay/android-developer/answer/6048248.

[26] Daniel Goßen, Hugo Jonker, Stefan Karsch, Benjamin Krumnow, and David Roefs. Hlisa: towards a more reliable measurement tool. In *Proceedings of the 21st ACM Internet Measurement Conference*, IMC '21, page 380–389, New York, NY, USA, 2021. Association for Computing Machinery.

[27] Daniel Gruss, Clémentine Maurice, and Stefan Mangard. Rowhammer.js: A remote software-induced fault attack in javascript. In *Proceedings of the 13th International Conference on Detection of Intrusions and Malware, and Vulnerability Assessment - Volume 9721*, DIMVA 2016, page 300–321, Berlin, Heidelberg, 2016. Springer-Verlag.

[28] Hasan Hassan, Yahya Can Tugrul, Jeremie S. Kim, Victor van der Veen, Kaveh Razavi, and Onur Mutlu. Uncovering in-dram rowhammer protection mechanisms:a new methodology, custom rowhammer patterns, and implications. In *MICRO-54: 54th Annual IEEE/ACM International Symposium on Microarchitecture*, MICRO '21, page 1198–1213, New York, NY, USA, 2021. Association for Computing Machinery.

[29] Hynix Semiconductor. Datasheet for 1Gb (32Mx32) GDDR5 SGRAM H5GQ1H24AFR. Technical Report H5GQ1H24AFR, 2009.

[30] Umar Iqbal, Steven Englehardt, and Zubair Shafiq. Fingerprinting the fingerprinters: Learning to detect browser fingerprinting behaviors. In *2021 IEEE Symposium on Security and Privacy (SP)*, pages 1143–1161. IEEE, 2021.

[31] Bruce Jacob, Spencer Ng, and David Wang. *Memory Systems: Cache, DRAM, Disk*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2007.

[32] Patrick Jattke, Victor Van Der Veen, Pietro Frigo, Stijn Gunter, and Kaveh Razavi. Blacksmith: Scalable rowhammering in the frequency domain. In *2022 IEEE Symposium on Security and Privacy (SP)*, pages 716–734, 2022.

[33] Patrick Jattke, Victor van der Veen, Pietro Frigo, Stijn Gunter, and Kaveh Razavi. BLACKSMITH: Rowhammering in the Frequency Domain, 2022-05. https://github.com/comsec-group/blacksmith.

[34] Patrick Jattke, Max Wipfli, Flavien Solt, Michele Marazzi, Matej Bölcskei, and Kaveh Razavi. Zenhammer: Rowhammer attacks on amd zen-based platforms, 2024.

[35] JEDEC. DDR5 SDRAM. Technical Report JESD79-5B, August 2022.

[36] Joe Hindy. How to change the MAC address on almost any device. https://www.androidauthority.com/how-to-change-mac-address-3192669/.

[37] Ingab Kang, Walter Wang, Jason Kim, Stephan van Schaik, Youssef Tobah, Daniel Genkin, Andrew Kwong, and Yuval Yarom. Sledgehammer: Amplifying rowhammer via bank-level parallelism, 2024.

[38] kernel development community. Transparent Hugepage Support. https://www.kernel.org/doc/html/latest/admin-guide/mm/transhuge.html.

[39] kernel.org. pagemap, from the userspace perspective. https://www.kernel.org/doc/Documentation/vm/pagemap.txt.

[40] Yoongu Kim, Ross Daly, Jeremie Kim, Chris Fallin, Ji Hye Lee, Donghyuk Lee, Chris Wilkerson, Konrad Lai, and Onur Mutlu. Flipping bits in memory without accessing them: An experimental study of dram disturbance errors. *SIGARCH Comput. Archit. News*, 42(3):361–372, jun 2014.

[41] Andreas Kogler, Jonas Juffinger, Salman Qazi, Yoongu Kim, Moritz Lipp, Nicolas Boichat, Eric Shiu, Mattias Nissler, and Daniel Gruss. Half-Double: Hammering from the next row over. In *31st USENIX Security Symposium (USENIX Security 22)*, pages 3807–3824, Boston, MA, August 2022. USENIX Association.

[42] Tomer Laor, Naif Mehanna, Antonin Durey, Vitaly Dyadyuk, Pierre Laperdrix, Clé mentine Maurice, Yossi Oren, Romain Rouvoy, Walter Rudametkin, and Yuval Yarom. DRAWN APART : A device identification technique based on remote GPU fingerprinting. In *Proceedings 2022 Network and Distributed System Security Symposium*. Internet Society, 2022.

[43] Pierre Laperdrix, Nataliia Bielova, Benoit Baudry, and Gildas Avoine. Browser fingerprinting: A survey. *ACM Transactions on the Web (TWEB)*, 14(2):1–33, 2020.

[44] Dawei Li, Di Liu, Yangkun Ren, Ziyi Wang, Yu Sun, Zhenyu Guan, Qianhong Wu, and Jianwei Liu. Fphammer: A device identification framework based on dram fingerprinting, 2023.

[45] Shang Li, Zhiyuan Yang, Dhiraj Reddy, Ankur Srivastava, and Bruce Jacob. Dramsim3: A cycle-accurate, thermal-capable dram simulator. *IEEE Computer Architecture Letters*, 19(2):106–109, 2020.

[46] man7.org. lspci(8) — Linux manual page. https://man7.org/linux/man-pages/man8/lspci.8.html.

[47] Michael Kerrisk. madvise(2) — Linux manual page. https://man7.org/linux/man-pages/man2/madvise.2.html.

[48] Micron. DDR4 SDRAM. https://www.micron.com/-/media/client/global/documents/products/data-sheet/dram/ddr4/8gb_ddr4_sdram.pdf.

[49] Micron Technology. TN-46-12: Mobile DRAM Power-Saving Features and Power Calculations. Technical Report TN46_12, 2005.

[50] Micron Technology. DDR2 SDRAM . Technical Report MT47H512M4,MT47H256M8,MT47H128M16, 2006.

[51] Micron Technology. TN-41-01: Calculating Memory System Power for DDR3. Technical Report TN41_01DDR3, January 2007.

[52] Micron Technology. 1.35V DDR3L SDRAM SODIMM. Technical Report MT16KTF51264HZ, MT16KTF1G64HZ, 2011.

[53] Micron Technology,. DDR4 SDRAM. Technical Report MT40A2G4, MT40A1G8, MT40A512M16, 2015.

[54] Micron Technology. TN-ED-03: GDDR6: The Next-Generation Graphics DRAM . Technical Report TN-ED-03: GDDR6, 2017.

[55] Mike Perry, Erinn Clark, Steven Murdoch and Georg Koppen. The Design and Implementation of the Tor Browser. https://2019.www.torproject.org/projects/torbrowser/design/.

[56] Mozilla. Using HTTP cookies. https://developer.mozilla.org/en-US/docs/Web/HTTP/Cookies.

[57] MultiLogin. Hardware: Canvas. https://docs.multilogin.com/l/en/article/7gNVYHcFKG-canvas.

[58] Notes on AI. Jensen–shannon divergence. Accessed: 2024-09-12.

[59] Lukasz Olejnik, Gunes Acar, Claude Castelluccia, and Claudia Díaz. The leaking battery - a privacy analysis of the html5 battery status api. In *DPM/QASA@ESORICS*, 2015.

[60] Lois Orosa, Ulrich Rührmair, A. Giray Yaglikci, Haocong Luo, Ataberk Olgun, Patrick Jattke, Minesh Patel, Jeremie Kim, Kaveh Razavi, and Onur Mutlu. Spyhammer: Understanding and exploiting rowhammer under fine-grained temperature variations, 2024.

[61] David A. Patterson and John L. Hennessy. *Computer Architecture: A Quantitative Approach.* Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1990.

[62] Peter Pessl, Daniel Gruss, Clémentine Maurice, Michael Schwarz, and Stefan Mangard. DRAMA: Exploiting DRAM addressing for Cross-CPU attacks. In *25th USENIX Security Symposium (USENIX Security 16)*, pages 565–581, Austin, TX, August 2016. USENIX Association.

[63] Gaston Pugliese, Christian Riess, Freya Gassmann, and Zinaida Benenson. Long-term observation on browser fingerprinting: Users' trackability and perspective. *Proceedings on Privacy Enhancing Technologies*, 2020:558–577, 05 2020.

[64] Kaveh Razavi, Ben Gras, Erik Bosman, Bart Preneel, Cristiano Giuffrida, and Herbert Bos. Flip feng shui: Hammering a needle in the software stack. In *25th USENIX Security Symposium (USENIX Security 16)*, pages 1–18, Austin, TX, August 2016. USENIX Association.

[65] Joel Reardon, Álvaro Feal, Primal Wijesekera, Amit Elazari Bar On, Narseo Vallina-Rodriguez, and Serge Egelman. 50 ways to leak your data: An exploration of apps' circumvention of the android permissions system. In *28th USENIX Security Symposium (USENIX Security 19)*, pages 603–620, 2019.

[66] Redhat. Chapter 17. Network Configuration. https://docs.redhat.com/en/documentation/red_hat_enterprise_linux/5/html/deployment_guide/ch-network-config?utm_source=chatgpt.com#ch-network-config.

[67] Paul Rosenfeld, Elliott Cooper-Balis, and Bruce Jacob. Dramsim2: A cycle accurate memory system simulator. *IEEE Computer Architecture Letters*, 10(1):16–19, 2011.

[68] Samsung Electronics. DDR4 SDRAM. Technical report, 2014.

[69] Iskander Sanchez-Rola, Igor Santos, and Davide Balzarotti. Clock Around the Clock: Time-Based Device Fingerprinting. In *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*, CCS '18, page 1502–1514, New York, NY, USA, 2018. Association for Computing Machinery.

[70] Andre Schaller, Wenjie Xiong, Nikolaos Athanasios Anagnostopoulos, Muhammad Umair Saleem, Sebastian Gabmeyer, Stefan Katzenbeisser, and Jakub Szefer. Intrinsic rowhammer PUFs: Leveraging the rowhammer effect for improved security. In *2017 IEEE International Symposium on Hardware Oriented Security and Trust (HOST)*. IEEE, may 2017.

[71] Mark Seaborn and Thomas Dullien. Exploiting the DRAM Rowhammer bug to gain kernel privileges. In *Black Hat USA*, 2015.

[72] Sensirion. EEPROM data decoder for SDRAM DIMM modules. https://github.com/Sensirion/i2c-tools/blob/master/eeprom/decode-dimms.

[73] William Shockley. Problems related to p-n junctions in silicon. *Solid-State Electronics*, 2(1):35–67, 1961.

[74] The Khronos Group Inc. clFlush(3) Manual Page. https://registry.khronos.org/OpenCL/sdk/3.0/docs/man/html/clFlush.html.

[75] Victor van der Veen, Yanick Fratantonio, Martina Lindorfer, Daniel Gruss, Clementine Maurice, Giovanni Vigna, Herbert Bos, Kaveh Razavi, and Cristiano Giuffrida. Drammer: Deterministic rowhammer attacks on mobile platforms. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, CCS '16, page 1675–1689, New York, NY, USA, 2016. Association for Computing Machinery.

[76] Antoine Vastel, Pierre Laperdrix, Walter Rudametkin, and Romain Rouvoy. Fp-stalker: Tracking browser fingerprint evolutions. In *2018 IEEE Symposium on Security and Privacy (SP)*, pages 728–741, 2018.

[77] Hari Venugopalan, Zainul Abi Din, Trevor Carpenter, Jason Lowe-Power, Samuel T. King, and Zubair Shafiq. Aragorn: A privacy-enhancing system for mobile cameras. *Proc. ACM Interact. Mob. Wearable Ubiquitous Technol.*, 7(4), jan 2024.

[78] Catherine S. Weir, Gary Douglas, Martin Carruthers, and Mervyn Jack. User perceptions of security, convenience and usability for ebanking authentication tokens. *Computers & Security*, 28(1):47–62, 2009.

[79] Yuan Xiao, Xiaokuan Zhang, Yinqian Zhang, and Radu Teodorescu. One bit flips, one cloud flops: Cross-VM row hammer attacks and privilege escalation. In *25th USENIX Security Symposium (USENIX Security 16)*, pages 19–35, Austin, TX, August 2016. USENIX Association.

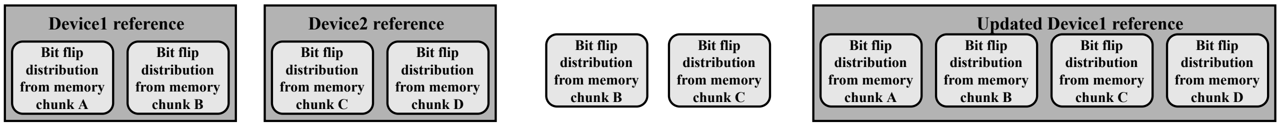
A Incrementally building up references

We discuss an alternate design of FP-Rowhammer that does not leave a distinct footprint by having to allocate multiple 2 MB chunks in the initial stages before reaching the limiting case (§5.3).

Fingerprinters can confine their hammering to fewer chunks (say, 2 chunks) per session to incrementally build up their references. In this case, fingerprinters would initially have multiple sets of references for devices without being able to link references that come from the same device. With this approach, fingerprinters would also not be able to fingerprint devices during their initial sessions. Eventually, after aggregating distributions from devices across multiple sessions, fingerprinters would be able to link sets of references to the same device and thereon fingerprint devices across all sessions. We demonstrate this approach with an example in Figure 13. Say, during a user's first session when running FP-Rowhammer on a new device, the fingerprinter obtains the distribution of bit flips in 2 distinct chunks of 2 MB of memory, chunk A and chunk B. Since each chunk has a unique distribution of bit flips, neither chunk would match any existing reference chunk known to the fingerprinter. The fingerprinter would create a fresh reference for these two chunks. In the next session on that device, say the fingerprinter obtains the distributions of 2 other chunks, chunk C and chunk D. Again, since the distribution of bit flips in each chunk is unique, the fingerprinter will not be able to identify that this session corresponds to the same device, and would store them as a separate reference. In the next session with the user, say the fingerprinter obtains the bit flip distribution to chunk B and chunk C. Now, the fingerprinter would be able to fingerprint the device, and also combine the references containing chunks A and B with the references containing chunks C and D as chunks obtained from the same device. In a subsequent session on the same device, say the fingerprinter obtains the bit flip distributions to chunk C and chunk E. For this session too, the fingerprinters would be able to fingerprint the device and also extend their references for the user to contain the distribution of chunk E. Thus, when incrementally building up references, fingerprinters would be able to fingerprint users without leaving behind a distinct memory footprint. However, fingerprinters would have to give up being able to fingerprint devices during their initial sessions.

B Deriving the number of 2 MB chunks to access in each session to guarantee access to at least one overlapping chunk across sessions

Suppose a user's device has N distinct 2 MB chunks of memory. As fingerprinters, we want to find d, the minimum number of distinct chunks to access in each session to guarantee that at least one chunk overlaps between sessions. Once we have an overlapping


Device1 reference
Bit flip distribution from memory chunk A
Bit flip distribution from memory chunk B
Device2 reference
Bit flip distribution from memory chunk C
Bit flip distribution from memory chunk D
Bit flip distribution from memory chunk B
Bit flip distribution from memory chunk C
Updated Device1 reference
Bit flip distribution from memory chunk A
Bit flip distribution from memory chunk B
Bit flip distribution from memory chunk C
Bit flip distribution from memory chunk D

Initial distribution of bit flips from two distinct chunks of memory from the same device across two sessions stored as references for two separate users

Distribution of bit flips from overlapping chunks obtained in a subsequent session

Updated set of reference distributions that collapse the previous set of reference distributions to map to the same device, thereby growing the number of references for the device

Figure 13: Incrementally building up references. In this figure, we assume that the fingerprinters first obtained bit flip distributions of two separate sets of contiguous chunks of memory from the same device across two separate sessions. Each set contains the distribution of bit flips observed on two such chunks. Since the distribution of bit flips on each chunk is unique, the fingerprinters treat these sets as belonging to two different devices. When they subsequently obtain bit flip distributions from chunks that overlap across both sets, they collapse them into a single reference that pertains to the same device.

chunk, we can use the unique distribution of bit flips on the chunk to identify the user's device.

To solve for d, we set up equations that describe the probability that at least one 2 MB chunk would overlap across 2 sessions. Let A be the event that at least one chunk overlaps between 2 sessions. For a given N, we want to find the minimum d such that $P(A \mid N, d) \approx 1$

$P(A \mid N, d) = 1 - P(\overline{A} \mid N, d)$

Here, $P(\overline{A} \mid N, d)$ is the probability that no chunk overlapped in the two sessions. Calculating $P(\overline{A} \mid N, d)$ is easy since it is the probability of choosing d chunks among N chunks in one session multiplied by the probability of choosing d chunks among $(N - d)$ chunks in the next session. Mathematically,

$$P(\overline{A} \mid N, d) = \binom{N}{d} / \binom{N}{d} \times \binom{(N-d)}{d} / \binom{N}{d}$$

$$\implies P(\overline{A} \mid N, d) = \binom{(N-d)}{d} / \binom{N}{d}$$

$$\implies P(\overline{A} \mid N, d) = \binom{(N-d)}{d} / \binom{N}{d}$$

$$\implies P(\overline{A} \mid N, d) = \frac{(N-d)!(N-d)!}{N!(N-2d)!}$$

$$\implies P(\overline{A} \mid N, d) = \frac{(N-d)(N-d-1)\cdots(N-2d+1)}{N(N-1)\cdots(N-d+1)}$$

$$\implies P(A \mid N, d) = 1 \text{ - } \frac{(N-d)(N-d-1)\cdots(N-2d+1)}{N(N-1)\cdots(N-d+1)}$$

Plugging in $N = 512$ which represents 1 GB of memory and $d = 64$ yields a probability of $0.9998 \approx 1$.

Using the same formulation, we can define the following function that determines the number of chunks to sample to get an overlapping chunk against the number of reference chunks:

$$P = 1 \text{ - } \left(\binom{N}{S} / \binom{(N-S)}{d} \times \binom{N}{S} / \binom{N}{d}\right)$$

where, N refers to the total number of chunks, S refers to the number of chunks previously sampled as part of the reference and d refers to the number of chunks to be sampled in the current session to ensure that we have a probability of P that at least one chunk overlaps with the reference.

Once we setting $P = 0.999$ and $N = 512$ for 1 GB of memory, we can calculate values of d for varying values of S. We visualize this plot in Figure 14.

C How does FP-Rowhammer compare against prior research?

In this section, we compare FP-Rowhammer's approach of hammering the same chunk multiple times and comparing probability distributions to match fingerprints against the approach of hammering once and comparing the Jaccard similarity of the set of bit flips as proposed by prior research [44, 70]. When accessing aggressors 1 million times and repeating the hammering sweep operation 4 times, FP-Rowhammer has an accuracy of 99.41%, corresponding to 96.33% precision and 89.93% recall. The same number of accesses to aggressors on the same set of DIMMs, leads to Jaccard similarity 98.01% accuracy, corresponding to 89.59% precision 64.64% recall.

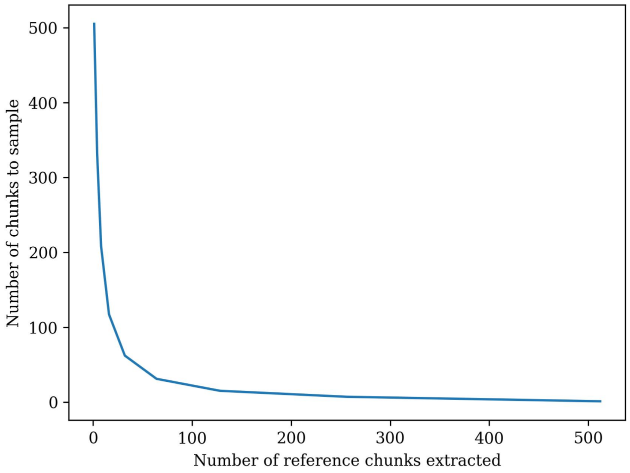


Figure 14: Plot shows the number of chunks to sample in one session to ensure at least one overlapping chunks against the number of reference chunks extracted. We see that the number of chunks to sample drops significantly as the number of reference chunks increases.

Furthermore, in presence of external factors, such as the CPU frequency, FP-Rowhammer reports 99.09% accuracy, 94.2% precision and 81.56% recall. On the same DIMMs, Jaccard index yields 96.3% accuracy, 100% precision and a significantly low 20.09% recal. These results show that FP-Rowhammer's approach of hammering chunks multiple times allows fingerprinters to account for the non-deterministic behavior of bit flips.

D Distirbution of JS divergence values across pairs of fingerprints on identical DIMMs from alternate manufacturer

Figures 15(a), 15(b) and 15(c) show the distribution of JS divergence values across pairs of fingerprints from identical 2Rx8, 1Rx8 and

1Rx16 DIMMs, respectively from a different DRAM manufacturer to the one mentioned in §6.3.

E Potential defenses against FP-Rowhammer

E.1 Eliminating Rowhammer

To the best of our knowledge, there exists no defense that can completely overcome Rowhammer. However, if such a defense were to be developed, it would also overcome FP-Rowhammer since FP-Rowhammer relies on observing bit flips produced by Rowhammer for fingerprinting.

E.2 Restricting access to contiguous rows within a bank

Any memory configuration that prevents access to contiguous rows within a single bank of a DIMM can be used to defend against FP-Rowhammer. For example, changing the DRAM address mapping such that the row bits are not present within the lower order bits would ensure that fingerprinters cannot pick appropriate aggressor rows to trigger Rowhammer without being able to allocate large amounts of contiguous memory. Such a mapping would practically make it impossible to execute double-sided Rowhammer, which is more reliable than single-sided Rowhammer in producing bit flips. More importantly, even if a fingerprinter is able to trigger bit flips using single-sided Rowhammer, they cannot observe them since bit flips are typically triggered in rows that are adjacent to those being triggered. However, it is difficult to change the mapping in the memory controller of legacy devices which currently have their own DRAM mapping. Changing the mapping in software would incur significant performance overhead. Even with the proposed mapping, fingerprinters may still be able to extract fingerprints if they get fortuitous with memory allocation that grants them access to contiguous rows in a bank.

F Inapplicability of common defenses

F.1 Standard fingerprinting defenses

Standard mitigations against fingerprinting such as normalization [76] or enforcing permissioned access [59] cannot be employed against FP-Rowhammer. Our results demonstrate unique and stable fingerprints even among homogeneous devices. Normalization as a defense against FP-Rowhammer would require eliminating process variation in the manufacture of DRAM chips, which is difficult to implement. Since all applications including benign applications access memory, blocking access to memory or requiring user permission to access memory would not be a viable mitigation against FP-Rowhammer. To the best of our knowledge, triggering additional bit flips to obfuscate or spoof the distributions extracted by FP-Rowhammer are not viable since they risk hurting benign memory usage.

F.2 Standard Rowhammer mitigations

With FP-Rowhammer, we extracted fingerprints on DIMMs that use in-DRAM TRR to mitigate Rowhammer. Different hammering patterns being able to evade TRR and trigger bit flips on different DIMMs shows that fingerprinters can overcome most TRR implementations. Since we assume that fingerprinters can run experiments on their own devices to discover ways to trigger bit flips, we anticipate that they can also overcome other defenses that attempt to mitigate Rowhammer. For example, researchers have explored the possibility of using ECC to defend against Rowhammer. However, existing research [12] has already shown that bit flips can also be triggered on ECC equipped systems. Thus, ECC does not guarantee a defense against FP-Rowhammer. Most recently, researchers have also managed to overcome in-DRAM ECC [35] to trigger bit flips on DDR5 DIMMs [34].

DIMM	**Num. Bit flips** $15°C$	**Num. Bit Flips** $40°C$
DIMM A	79417	88793
DIMM B	135148	146154
DIMM C	2861	2653
DIMM D	2713	2353
DIMM E	261083	211955
DIMM F	91840	102918
DIMM G	3956	2945
DIMM H	2480	2177
DIMM I	67129	77632
DIMM J	4092	3960

Table 3: Even when comparing fingerprints extracted over a span of a few minutes, we see that there is a significant drop in recall only when we re-seat DIMMs.

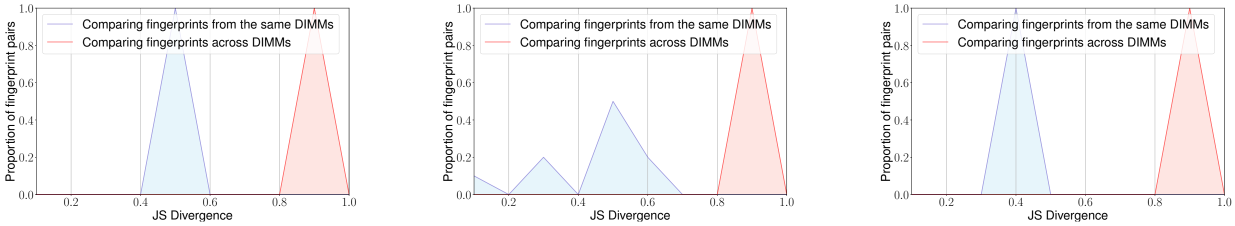
G Number of bit flips observed at different temperatures

Table 3 shows the number of bit flips observed at $15°C$ and $40°C$ in our experiments in §6.6.

H DRAM mapping and timing side-channel to determine DIMM geometry

We can figure out the geometry of a DIMM from a given address. For this example, let us assume that our machine is an Intel Core i7-7700 (Kaby Lake) equipped with a 16 GB DIMM, where the highest possible address is confined within 34 bits. Let's say that our given physical address is `0x2abcd1234`. Previous research has shown that Intel uses XOR functions to decode physical addresses to map it to the device [62]. We used DRAMA to reverse engineer the mapping functions for our machine [22, 62]. Figure 16 shows the mapping functions look like after decoding the address.

Now, the same information can be obtained from timing side-channels. This is partly how DRAMA works as well. This is particularly important for us as we need to figure out the number of ranks, banks and width of a given DIMM without administrative privileges. Code Listing 1 shows the mapping functions on an Intel Kaby Lake with 2 ranks (`2Rx1`) and 1 rank (`1Rx1`). Keeping our search space limited in this example, let us assume that the user's machine is equipped with either of these aforementioned CPUs. In order to start the process of finding the number of banks, we first need to read the rowbuffer hit time (t_{hit}). We allocate a 2 MB transparent huge page, which gives us access to the lower 21 bits in the physical address. These bits can be manipulated from the



(a) Distribution of JS divergence values across all pairs of fingerprints from 4 identical 2Rx8 DIMMs

(b) Distribution of JS divergence values across all pairs of fingerprints from 10 identical 1Rx8 DIMMs

(c) Distribution of JS divergence values across all pairs of fingerprints from 2 identical 1Rx16 DIMMs

Figure 15: Plots showing the distribution of JS divergence values when comparing bit flip distributions obtained from the same pair of DIMMs and across different DIMMs.

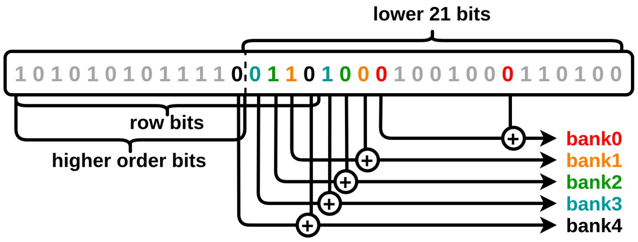


Figure 16: Highlighting the XOR bank functions used on an Intel Kaby Lake processor.

userspace [13]. Assume that the starting address is `0x0` and the ending address is `0xfffff` for the 2 MB chunk.

```
bank0 = [0x2040, 0x2040];     // 2R or 1R DIMM
bank1 = [0x44000, 0x24000];   // 2R or 1R DIMM
bank2 = [0x88000, 0x48000];   // 2R or 1R DIMM
bank3 = [0x110000, 0x90000]; // 2R or 1R DIMM
bank4 = [0x220000];           // only for 2R DIMM
```

Listing 1: Bank functions for a Kaby Lake machine with 1 rank and 2 ranks.

Suppose we access `0x0` and `0x1`, we should have a rowbuffer hit. Let this time be t_{hit}. Allocated data is interleaved in the DRAM [31, 61]. In order to cause a rowbuffer miss, we need another row from the same bank. This leaves us with the fact that we need to find pairs of bits that are XORed to determine the bank bits. Column bits take 10 or 11 bits to be represented [45, 53, 67, 68]. The lowest 3 bits are used to align addresses with byte-sized data. Therefore, it is down to the 13th or the 14th bit to represent the first bit for a bank function pair (bank0). This is historically seen as true, as it holds from 6th generation Intel Core processors to 11th generation Intel Core processors. It's corresponding bit is usually at 6th or 7th bit. The important bit pair here is b1, which starts at either 14th or the 15th bit, depending upon bank0. It's corresponding bit will be at the beginning of the row bits. If the DRAM DIMM has 1 rank, then the row bits will start at (bank1[0] + 3)th bit. So, we will guess that the row bit start at either bit index 17 or 18 and increment by 1. This also implies that we need to set its corresponding XOR bit to 1. This address will be `0x24000`. If accessing addresses `0x0` and `0x24000` results in a time say t_0, such that $t_0 > t_{hit}$ by a *large margin*, then we conclude that the DIMM is a single ranked DIMM. If not, then we repeat the process by assuming that the starting row bit is at bit index 18 (`0x0` and `0x44000`). This will imply that we have a DIMM with 2 ranks.

For the case of `x16`, the starting row index is either 16 or 17 depending upon the number of ranks. We repeat the aforementioned experiment for this case as well. The only point of contention is when we have a `2Rx16` and a `1Rx8` DIMMs. The starting row index is same, *i.e.* 17. In this case we simply run all the known Rowhammer patterns for both the cases in the hopes of finding the correct pattern which produces bit flips.

I Extension to configurations with multiple DIMMs across channels

If a user's device has multiple DIMMs across channels, contiguous addresses are interleaved in DIMMs across channels in addition to being interleaved across banks. Thus, while manipulating the available 21 bits in the address of a transparent huge page to set primary aggressors (and to scan for bit flips), we have to make sure that we do not alter the bits that correspond to the channel. For example, in case of 1Rx8 DIMMs on Kaby Lake machines, we can alter the row within a bank by modifying bits above the 18th bit (bit at index 17 with the least significant bit being bit 0) in the address of the huge page. From our experiments with DRAMA [62], we suspect that the channel bits can be derived from a combination of the bits at indices 7, 9, 14, and 17. We suspect that the bits at indices 27 and 28 also contribute to the channel, but they are not relevant since they fall outside the bits we can manipulate. Thus, we do not change the bit at index 17 to ensure that we do not change the channel.