Centauri: Practical Rowhammer 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*—Fingerprinters leverage the heterogeneity in hardware and software configurations to extract a device fingerprint. Fingerprinting countermeasures attempt to normalize these attributes such that they present a uniform fingerprint across different devices or present different fingerprints for the same device each time. We present Centauri, a Rowhammer fingerprinting approach that can build a unique and stable fingerprints even across devices with homogeneous or normalized/obfuscated hardware and software configurations. To this end, Centauri leverages the process variation in the underlying manufacturing process that gives rise to unique distributions of Rowhammer-induced bit flips across different DRAM modules. Centauri's design and implementation is able to overcome memory allocation constrains without requiring root privileges. Our evaluation on a test bed of about one hundred DRAM modules shows that Centauri achieves 99.91% fingerprinting accuracy. Centauri's fingerprints are also stable with daily experiments over a period of 10 days revealing no loss in fingerprinting accuracy. We show that Centauri is efficient, taking as little as 9.92 seconds to extract a fingerprint. Centauri is the first practical Rowhammer fingerprinting approach that is able to extract unique and stable fingerprints efficiently and at-scale.**

I. Introduction

Stateless tracking is becoming more prevalent [20], [5] in response to recent countermeasures against stateful tracking using browser cookies [39] and device identifiers (e.g., IDFA on iOS and AAID on Android) [3], [16]. Stateless tracking involves probing for distinguishing features of a device to construct a *fingerprint* without needing to store any client-side state [28]. For a fingerprint to be useful for tracking, it needs to possess two attributes: uniqueness and stability. First, a fingerprint should have sufficiently high entropy to uniquely identify a device within a given population of devices [12]. Second, it should remain sufficiently stable over an extended duration, so it can be linked to other fingerprints from the same device for re-identification [54].

Fingerprinting typically leverages the heterogeneity in hardware and software configurations to extract fingerprints. For example, FingerprintJS [14], a widely deployed fingerprinting library, aggregates a variety of software and hardware attributes such as browser version, screen resolution, and the number of processors to construct device fingerprints. 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 standardize or normalize the attributes that capture device configuration to present the same values across different devices [38], [6].

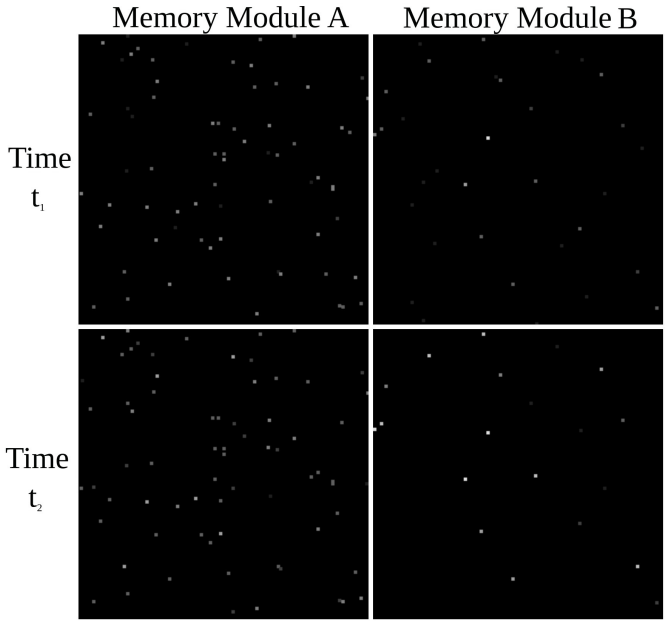


Fig. 1. Visualization of the Rowhammer bit flip distribution with a brighter spot representing a higher bit flip probability. The **top row** shows the distributions on two different but identical DRAM modules. The **bottom row** shows the distribution on the same DRAM modules at a later point in time.

Fingerprinters also have to contend with how the captured attributes change over time with the goal of producing a consistent fingerprint. These changes can either arise from natural evolution of device configuration (e.g., software updates) or from fingerprinting countermeasures that intentionally randomize values of attributes on the same device [7], [40]. To attain high stability, fingerprinters attempt to predict fingerprint changes [54] or employ attribute value stemming to improve stability while minimally impacting uniqueness [45].

In this work, we investigate a stronger threat model where a fingerprinter aims to extract unique and stable fingerprints

Network and Distributed System Security (NDSS) Symposium 2024
26 February - 1 March 2024, San Diego, CA, USA
ISBN 1-891562-93-2
https://dx.doi.org/10.14722/ndss.2024.23xxx
www.ndss-symposium.org

for devices with identical hardware and software configurations over extended periods of time. To this end, we aim to capture fundamental differences in the physical properties of the device's hardware as unique fingerprints. Our key insight is that a fingerprinter may be able to extract fingerprints from inherent differences that arise as a result of *process variation* in the hardware (CMOS) manufacturing process. As users seldom modify their device hardware, these fingerprints remain stable, as long as they account for differences resulting from process variation in the same hardware. While prior research has explored variations in internal clocks [50], GPUs [27] and CPUs [8], we are the first to successfully leverage memory (DRAM) for fingerprinting.

We leverage Rowhammer [26] 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 show that the pattern of bit flips due to Rowhammer can be leveraged to build a fingerprint. We also show that the pattern of Rowhammer bit flips is sufficiently unique and stable to build a reliable fingerprint for the population of computing devices (billions of devices). To build intuition, Figure 1 visualizes the distribution of bit flips produced by executing Rowhammer at the same locations on two identical DRAM modules[1] at two different points in time. The results look promising—the distribution of bit flips is *reasonably similar* on the same DRAM modules at different points in time while being *noticeably different* across the pair of DRAM modules.

In this paper, we present Centauri, a *practical* Rowhammer-based fingerprinting approach that exploits bit flip distributions to extract highly *unique* and *stable* fingerprints even among homogeneous devices with identical software and hardware configurations over an extended period of time. Centauri overcomes three main challenges that make it practical for fingerprinting. ***First***, as alluded to in Figure 1, the bit flips triggered by Rowhammer are non-deterministic (i.e., hammering the same location does not flip the same set of bits). Thus, a fingerprinter has to account for this non-determinism to extract stable fingerprints. We identify certain practical scenarios that exacerbate this non-determinism where comparing set similarity to match fingerprints falls short (§VI-F, VI-G). With Centauri, we hammer the same locations multiple times to extract a probability distribution of bit flips as fingerprints. We then compare the divergence of these distributions that leads to better re-identification of devices even where there is a drastic difference in the set of bits that flipped. ***Second***, fingerprinters are constrained by the abstractions provided by the operating system to allocate memory. These abstractions provide limited access to contiguous physical memory and hide information about their allocation on the DRAM. Without root privileges, these constraints prevent fingerprinters from trivially tracking the location of bit flips to fingerprint devices. We use the insight from our measurement study (§IV-B)that the distribution of bit flips in contiguous 2 MB chunks of memory is unique and persistent to overcome this challenge. Armed with the insight, we sample enough 2 MB chunks to guarantee access to the same chunk for fingerprinting. ***Third***, memory modules implement mitigations against Rowhammer, such as Target Row Refresh (TRR) [31]. While prior research has demonstrated ways to craft hammering patterns [15], [22] to bypass TRR, they provide limited insights towards operationalizing them to trigger bit flips at scale. Centauri systematically identifies effective patterns for at-scale fingerprinting using Rowhammer.

We evaluate Centauri on a set of 98 DIMMs across 6 sets of identical DRAM modules across 2 major DRAM manufacturers. Centauri demonstrates high entropy with a highest fingerprint accuracy of 99.91% corresponding to a precision of 100%, and recall of 97.06%. Centauri also demonstrates high stability with daily experiments to extract fingerprints from the same devices over a period of ten days without any degradation in fingerprint accuracy. Our experiments show that Centauri only suffers a minor loss in accuracy of 0.9% in presence of external factors that are not under the control of fingerprinters but affect the distribution of bit flips (such as the CPU frequency). We also investigate the trade-off between the accuracy of Centauri's fingerprints against the efficiency of Centauri's approach in terms of the time taken to extract fingerprints. Centauri is able to extract a fingerprint in as little as 9.92 seconds, reducing the overhead by more than 95.01% while degrading accuracy by just 0.64%.

Our key contributions include:

- Practically extracting highly unique and stable fingerprints using Rowhammer: We practically demonstrate Centauri on the largest scale of DRAM modules in current literature.
- Handling non-deterministic bit flips: We handle non-deterministic bit flips by hammering the same memory chunks multiple times and using the divergence between probability distributions of bit flips to re-identify devices.
- Overcoming memory allocation constraints: We overcome memory allocation constraints by devising a novel sampling strategy that guarantees access to the same chunk of memory for fingerprinting.
- Operationalizing bypass techniques for Rowhammer mitigations: We bypass Rowhammer mitigations by identifying effective hammering patterns that can trigger bit flips at-scale.

II. Background

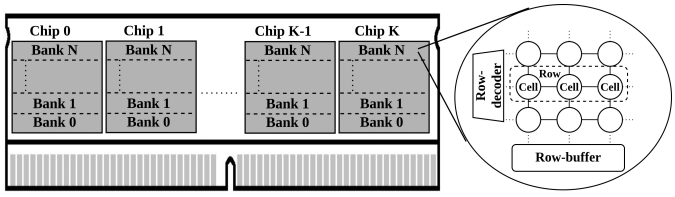


Fig. 2. This figure shows a single rank of a DRAM DIMM. Each rank contains multiple logical structures called banks that are interspersed across multiple physical structures called chips. Each bank is an array of cells in the form of rows and columns.

[1]also called dual in-line memory modules or DIMMs

	Rowhammer PUF [51]	Drammer, Cross-VM Rowhammer FFS [53], [46], [55]	Rowhammer.js [17]	SMASH [10]	Blacksmith, TRRespass [22], [15]	Centauri [This paper]
Studies bit flip uniqueness	◑	○	○	○	○	●
Studies bit flip stability	●	◑	◑	○	◑	●
Overcomes OS abstractions to reproduce bit flips	○	●	○	○	○	●
Overcomes Rowhammer mitigations (TRR)	○	○	○	●	●	●
Scale	1	≤60	10	5	∼40	**98**

TABLE I. COMPARING CENTAURI AGAINST OTHER ROWHAMMER RESEARCH. ●, ◑ AND ○ INDICATE FULL CONSIDERATION, PARTIAL CONSIDERATION AND NO CONSIDERATION OF A GIVEN TOPIC, RESPECTIVELY. CENTAURI IS THE FIRST APPROACH TO DEMONSTRATE THE EXTRACTION OF UNIQUE AND STABLE FINGERPRINTS ON THE LARGEST SCALE USING ROWHAMMER WHILE OVERCOMING PRACTICAL LIMITATIONS ENFORCED BY THE OPERATING SYSTEM AND BY ROWHAMMER MITIGATIONS (TRR).

A. DRAM basics

All DRAM technologies follow the same basic architecture [33], [34], [36], [23], [19], [37], [32], [35]. We concentrate on DIMM-based DRAM packages in this paper for ease of testing, but the findings apply to other packaging techniques as well.

Each physical DRAM Dual Inline Memory Module (DIMM) is installed on a DRAM channel on the motherboard. Channels enable issuing concurrent requests to multiple DIMMs. Figure 2 represents one side of a DIMM, also called a rank. Each individual DIMM contains multiple chips, which are uniformly divided into logical structures called banks on one or both of their ranks. 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. Each row in a DDR4 DIMM contains 65,536 capacitors. The number of ranks, banks, rows etc describe a DIMM's geometry.

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 also periodically issues `REFI` commands that refresh the charge held by the capacitors since the charge naturally drains over time. Every DRAM capacitor typically gets refreshed at least once every 64 milliseconds.

B. Rowhammer

Modern DIMMs are susceptible to memory corruption as a result of electrical interference among cells. Rowhammer [4], [26] 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 rate of charge dissipation of the the capacitors in the victim rows. Once these capacitors have lost a sufficient amount of charge, refreshing the DRAM cannot restore their value, resulting in memory corruption. Modern DDR4 DIMMs implement Target Row Refresh or TRR to mitigate Rowhammer [31]. While different implementations of TRR exist, all of them essentially track memory accesses to identify aggressor rows and issue additional refreshes to the associated victim rows [15], [18].

1) Rowhammer for fingerprinting: The rate at which a capacitor loses its charge depends on its physical properties [43], [21]. These properties are not uniform on all chips due to process variation induced during manufacturing [52]. As a result, the capacitors that lose their charge (i.e., the bits that flip) should also depend on the particular chip being subjected to 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. In this paper, we show how to exploit the distribution of bit flips across DIMMs for fingerprinting.

C. Related work

Rowhammer PUF [51] demonstrates that different memory regions within a PandaBoard [13] exhibit unique and consistent sets of bits that flip as a result of Rowhammer. However, their work does not compare the uniqueness of bit flips across devices. Furthermore, ensuring fingerprint stability is easier in a weaker threat model that allows fingerprinters to pick aggressor rows such that they always trigger bit flips on the same victim row. Thus, when comparing sets of bit flips from different points in time to re-identify a given device, fingerprinters are able to assume that the bits that flipped were on the same row. This assumption side-steps the need to account for differences in the bit flips across multiple rows on the same device. In contrast, a fingerprinter under a realistic threat model would be constrained by the abstractions provided by the OS for memory allocation. These abstractions hide information about the physical memory allocation, thereby making it difficult to

ensure that bit flips were only triggered from a particular row when re-identifying a device. Rowhammer PUF also presented results on DDR2 memory which did not incorporate any mitigations against Rowhammer.

While Drammer [53] does not aim to exploring the fingerprinting capabilities of Rowhammer, it 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. The proposed technique, Phys Feng Shui requires the overall memory layout to remain unchanged (no allocation/deallocation of memory by other processes). Fingerprinters could seek to trigger bit flips on a device at arbitrary points in time, across which they cannot expect the memory layout to remain constant. Thus, Phys Feng Shui cannot be adopted to overcome restrictions enforced by the operating system to execute Rowhammer for fingerprinting. Research from Razavi et al. [46] and Xiao et al. [55] leverage memory deduplication and MMU paravirtualization respectively to overcome memory restrictions and trigger bit flips on memory allocated to a victim VM on cloud machines. However, these capabilities are not enabled or available on most end-user devices.

TRRespass [15] and Blacksmith [22] introduce fuzzers that discover many-sided and non-uniform hammering patterns respectively to overcome TRR and trigger bit flips. While both papers present results on reproducing bit flips, they do not study differences in the distribution of bit flips across DIMMs since they do not explore the fingerprinting capabilities of Rowhammer. Fingerprinters also get limited insights on employing these patterns to trigger bit flips on a large number of DIMMs since both TRRespass and Blacksmith do not provide guidance on which discovered pattern to employ beyond those that produce bit flips. In their threat models, both TRRespass and Blacksmith also do not have to overcome the limited memory abstractions provided by the OS.

Rowhammer.js [17] and SMASH [10] show how to trigger bit flips when confined to running JavaScript on the browser. However, they do not explore differences in bit flip distributions or their reproducibility as they don't intend to use Rowhammer for fingerprinting.

With Centauri, we first make the crucial observation that the bit flips in each contiguous 2 MB chunk of memory (the largest contiguous chunk that can be reliably allocated without requiring administrative privileges) are highly unique and persistent (§IV-B). Armed with this observation, we present a novel sampling strategy as part of Centauri's design (§V) to overcome the restrictions imposed by the operating system's memory abstractions. To operationalize hammering patterns at scale, Centauri increases the CPU frequency to maximum to speed up the discovery of patterns and proposes to prioritize those patterns that can trigger a larger number of bit flips and can generalize to multiple DIMMs. To account for the inherent non-determinism in bit flips, we reset and hammer the same 2 MB chunk multiple times to extract a probability distribution of bit flips. We compare the similarity of these probability distributions to fingerprint DIMMs. We summarize how Centauri differs from prior research in Table I. In summary, Centauri 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 operating system and by Rowhammer mitigations such as TRR.

III. Centauri overview

In this paper, we present Centauri, a Rowhammer-based fingerprinting approach that extracts unique and stable fingerprints even among devices that have identical hardware and software configurations. In this section, we first define our threat model in terms of our assumptions and goals and then provide an overview of Centauri's design.

A. Threat model

In our threat model, the fingerprinter runs code on a user's device. Users either run an app or visit a website that includes the fingerprinting SDK/library. In this paper, we focus on the former and discuss extension to the latter in §VII-A.

We assume that the fingerprinter has a wide array of devices and DRAM modules (DIMMs) with different configurations in their possession. Within their own set of devices, fingerprinters have no restrictions in terms of what they can execute. We also assume that the fingerprinter has access to powerful servers where they can store and match fingerprints.

In this paper, we take the role of the fingerprinter and our goal is to extract unique and stable fingerprints from devices even among those that have identical configurations.

B. Centauri architecture

At a high level, Centauri triggers bit flips on multiple contiguous 2 MB chunks of memory on a user's device and uses the distribution of the triggered bit flips as a fingerprint. Centauri 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.

Centauri'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.

IV. Measuring the entropy and persistence of Rowhammer bit flips

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 2 MB chunks since this is the largest contiguous chunk of memory that we can reliably obtain from the OS without requiring root privileges (using Transparent Huge Pages or THP [24]). 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.

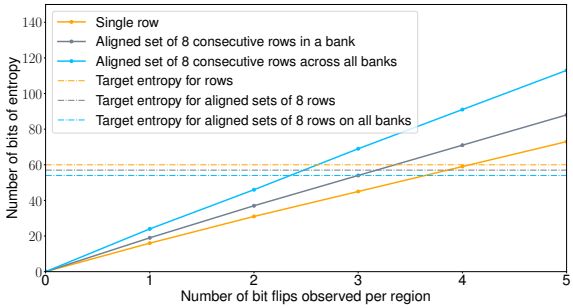


Fig. 3. Plots showing the variation in the number of bits of entropy that can be obtained to represent different regions of memory across a trillion DIMMs with varying number of bit flips.

A. 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.

As discussed in §II, 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 3 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. From the figure, 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 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 3 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.

B. Empirical entropy analysis

Existing Rowhammer research has primarily focused on developing techniques to trigger bit flips [26], [17], [53], [15], [10], [22] 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 §I, 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.

1) Test Bed: Our test bed for this measurement consists of 36 identical 2Rx8 DIMMs. We fuzz one of these DIMMs to discover a non-uniform hammering pattern that can trigger bit flips. We observe that the discovered pattern is able to produce bit flips on all 36 DIMMs.

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 [25]). 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.

3) Results: Across all 36 DIMMs, we hammered a total of 3,611 chunks. 99.77% of these chunks (3,603 chunks)

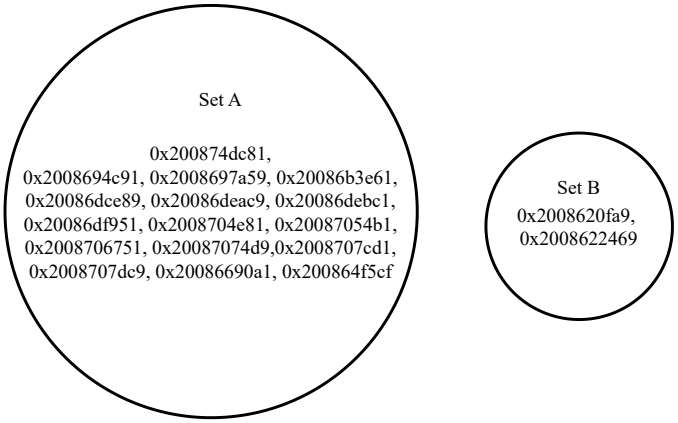


Fig. 4. Set A shows the addresses of bits that flipped when hammering a particular chunk of a DIMM. Set B shows the addresses of bits that flipped when restoring data to the chunk and hammering it again. The two sets are disjoint demonstrating the non-deterministic behavior of bit flips.

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

TABLE II. SUMMARY OF THE MEASURED DISTRIBUTION OF BIT FLIPS ACROSS 36 IDENTICAL DIMMS.

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. We summarize these findings in Table II. 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.

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 4 shows the sets of bits that flipped when hammering the same chunk on a particular

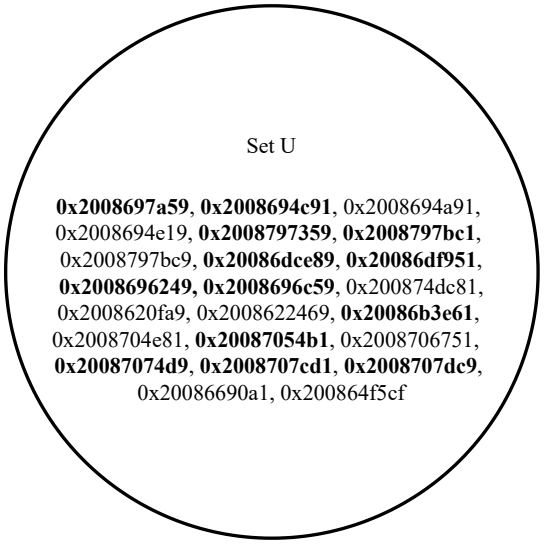


Fig. 5. Set U shows the set of all addresses that flipped across 8 attempts to restore the data and hammering the same chunk from Figure 4. The boldfaced addresses are addresses that flipped more than once across the 8 attempts.

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. Thus, we cannot re-identify a particular chunk by merely employing a set similarity metric like Jaccard index (as proposed by existing research [51], [29]).

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 5. This figure 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 multiple times and use the count of flips at different indices across all hammering attempts. Then, we use Jensen-Shannon (JS) divergence [41] to compute the similarity of distributions. In Figure 6, 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.

V. CENTAURI

We provide a detailed account of Centauri's three phases:

A. 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, we run Blacksmith's [22] fuzzer to discover non-uniform hammering patterns that can evade Target Row Refresh (TRR). To account for all TRR implementations that could exist in the wild, our goal is to discover a wide array of patterns that can 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 can trigger more bit flips as well as those that can trigger bit flips on more DIMMs. Patterns that trigger a large number of 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 (discussed in §IV-B) or as a result of external factors that fingerprinters cannot control on users' devices. For example, we observe fewer bit flips with the same pattern on the same DIMM when running at lower CPU frequency. Since we cannot control the CPU frequency of a user's device, we pick patterns that can account for changes to CPU frequency to ensure robustness of our fingerprints. 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.

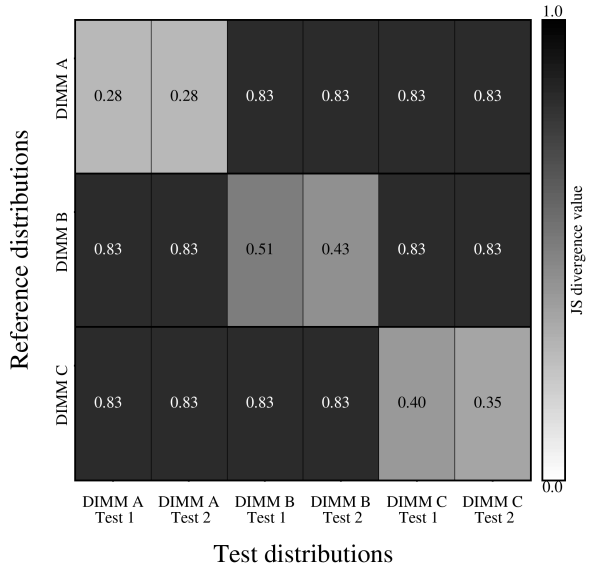


Fig. 6. Visualization of the relative persistence of bit flips within given 2 MB chunks of memory across multiple DIMMs.

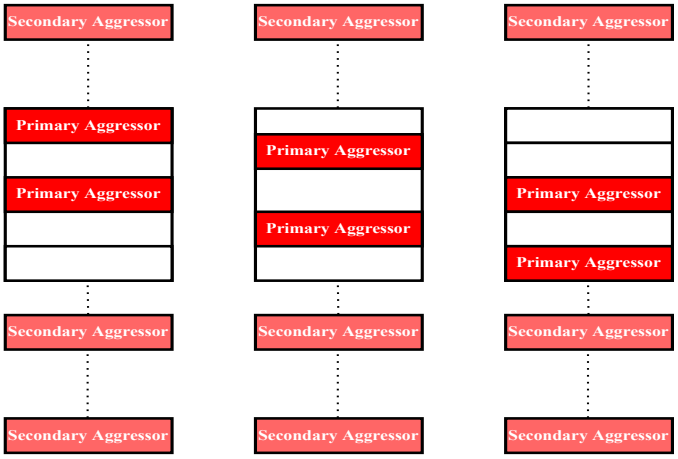


Fig. 7. Visualization of a hammering sweep performed within one bank of a contiguous 2 MB chunk. The central rectangular blocks in the visualization represent rows within the chunk. We map primary aggressors in the discovered patterns to rows within the chunk and secondary aggressors to random rows within 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.

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 evaluate Centauri's robustness while extracting fingerprints in context of varying CPU frequencies in §VI-F. 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.

B. 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 defined by a set of aggressors, a phase, an amplitude and a frequency [22]. The phase, amplitude and frequency are such that some aggressors engage with TRR (we refer to these as secondary aggressors) and the others trigger bit flips (we refer to these as 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. Concretely, for a pattern $a_1, a_2, \ldots a_i, a_{i+1}, \ldots a_n$, addresses places 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.

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 §VII-B. To execute the discovered patterns, we allocate transparent huge pages on a user's device to obtain contiguous chunks of 2 MB of memory without requiring 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 [44], [10]. 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.

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 7 visualizes the *hammering sweep*. To account for non-determinism in bit flips, we repeat the hammering sweep multiple times on the chunk.

C. Matching phase

With the observation from §IV-B 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 the contiguous 2 MB chunk (indexed from 0 to 1,048,576 in case of 1Rx8 DIMMs). We then 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.

However, we cannot guarantee access to the same 2 MB regions 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. One way to overcome this challenge would be to hammer multiple 2 MB chunks during each session. For example, suppose the user's device has 1 GB of memory which corresponds to 512 different 2 MB chunks of memory. Taking inspiration from the birthday paradox [1], if we were to 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 show how we derived this number from first principles in Appendix A.

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.

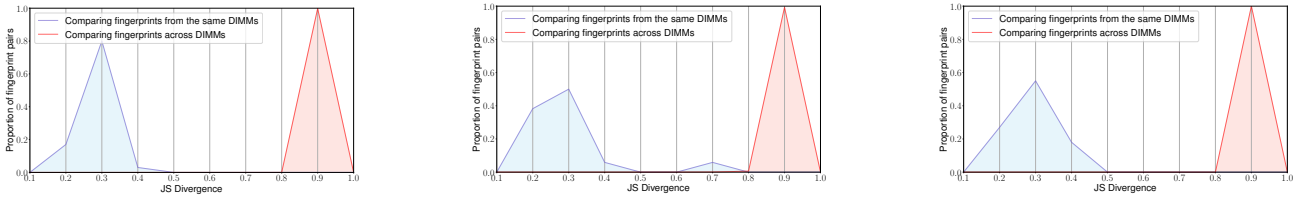
VI. Evaluation

This section seeks to answer the following questions about Centauri and its ability to extract fingerprints.

- How unique are the fingerprints extracted by Centauri?
- How stable are the fingerprints extracted by Centauri?
- How long does Centauri take to extract fingerprints?
- Can Centauri extract robust fingerprints in presence of external factors?
- How does Centauri compare against Rowhammer PUF?

A. Test bed

Since the likelihood of an unintended bit flip that results in a crash is non-trivial, we decide to evaluate Centauri on a controlled test bed. 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. To evaluate Centauri's fingerprints across manufacturers, our test bed also contains 10 1Rx8 DIMMs, 2 1Rx16 DIMMs and 4 2Rx8 DIMMs from another major DRAM manufacturer. Overall, our test bed contains 98 DIMMs that include 6 identical sets of DIMMs across two major DRAM manufacturers. We installed these DIMMs among 12 Intel Core i7 desktops for our experiments. Since we have more DIMMs than desktops in our possession, we hammer DIMMs in batches. While repeating experiments on a particular DIMM, we make sure that we seat it on the same slot on the same desktop.



(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

Fig. 8. Plots showing the distribution of JS divergence values when comparing bit flip distributions obtained from the same pair of DIMMs and across different DIMMs.

B. Experimental methodology

1) Templating phase: We use a set of 11 hammering patterns, which together are able to trigger bit flips on all 98 DIMMs in our test bed. We note that patterns generalize across DIMMs that were made by the same manufacturer. In other words, patterns that produce bit flips on DIMMs of a particular manufacturer, do not produce bit flips on DIMMs from other manufacturers. Thus, the pattern that triggers bit flips also helps identify DIMMs by revealing their manufacturer.

2) Hammering phase: For each DIMM in our test set, we perform a hammering sweep (visualized in Figure 7) 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 particular DIMM as its fingerprint. We vary the number of times we repeat the hammering chunk operation and the number of times we activate aggressors in different experiments to evaluate their impact on Centauri's fingerprints (§VI-E).

3) Matching phase: 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 (i.e., to correspond to the same DIMM) if the minimum JS divergence value across all pairs is below an empirically determined threshold.

C. How unique are the fingerprints extracted by Centauri?

For this evaluation, we extract 2 fingerprints from each DIMM in our test bed using the aforementioned methodology. We use the first fingerprint extracted from a particular DIMM as a reference for that DIMM and compare subsequently extracted fingerprints against the reference. In these experiments, we extracted both fingerprints within the space of few hours on the same day. We did not re-seat the DIMMs in the interim period. We repeated the hammering sweep operation 8 times and activated the aggressors 10,000,000 times.

Figure 8(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 8(b) and Figure 8(c) shows similar plots across 35 1Rx8 DIMMs and 11 1Rx16 DIMMs respectively. We report similar plots on DIMMs from the other manufacturer in Appendix B.

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[2], we attain an overall fingerprint accuracy of 99.91%, corresponding to a precision of 100% and recall of 97.06%. Overall, these results show that Centauri has very high discriminative power and can uniquely identify DIMMs across multiple sets of DIMMs with identical configurations.

D. How stable are the fingerprints extracted by Centauri?

We evaluate the stability of the fingerprints extracted by Centauri over time. Since we have more DIMMs than desktops, we first evaluate Centauri's 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 9 shows the variation in Centauri'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 in the values for any metric indicating that the fingerprints extracted by Centauri 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 threshold since we record similar accuracy, precision and recall even with slightly altered thresholds.

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

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

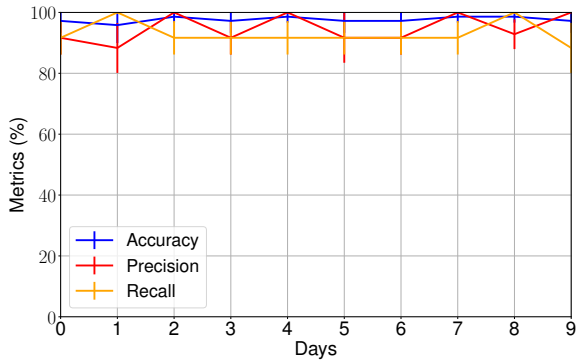


Fig. 9. Plot showing the variation in the accuracy, precision and recall of the fingerprints extracted by Centauri on a set of 10 DIMMs over a period of 10 days. We see that the metrics roughly remain constant with minor fluctuations which provides strong evidence that the fingerprints extracted by Centauri are stable.

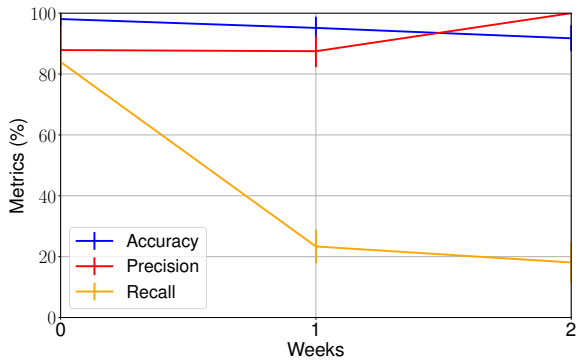


Fig. 10. Plot showing the variation in the accuracy, precision and recall of the fingerprints extracted by Centauri on our entire test bed of 98 DIMMs over a period of 2 weeks and again over a period of 4 weeks. Unlike what we observed in our previous experiment, there is a significant change in metrics with the recall declining significantly. We suspect that this decline is a result of having to re-seat DIMMs in our experimental setup.

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 10 shows how Centauri's accuracy, precision and recall change over two weeks and over four weeks on our entire set of DIMMs. At a common threshold of 0.6 for JS divergence, we only see minor fluctuations in precision and recall, 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 common threshold that gave us high precision and high recall.

We suspect that this instability in Centauri's fingerpints is a result of our experimental setup where we re-seat DIMMs. If this is indeed the case, Centauri's fingerprints would be stable in the wild since users rarely re-seat their DIMMs. The ideal way to confirm this would be to evaluate the stability of Centauri'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 an experiment that provides strong evidence that re-seating induces the instability in Centauri's

Experiment	Accuracy	Precision	Recall
No rebooting/re-seating	96.97%	83.33%	95.83%
Only Rebooting	94.44%	83.33%	83.33%
Re-seating (and rebooting)	91.67%	100%	50%

TABLE III. EVEN WHEN COMPARING FINGERPRINTS EXTRACTED OVER A SPAN OF A FEW MINUTES, WE SEE THAT THERE IS A SIGNIFICANT DROP IN RECALL ONLY WHEN RE-SEATING WE RE-SEAT DIMMS.

fingerprints. Concretely, we run 3 different experiments on 6 randomly chosen 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.

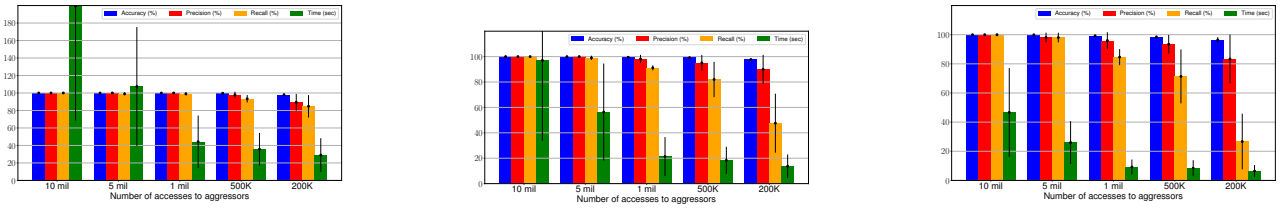
We present the highest accuracy attained, with the corresponding precision and recall for all 3 experiment in Table III. From the table we see a drastically low value of 50% recall for the experiment where we re-seated the DIMM. The other two experiments show high precision as well as recall. These observations support our suspicion that the instability in Centauri's fingerprints is a result of re-seating the DIMMs. Thus, we can expect the fingerprints extracted by Centauri to be stable in the wild since users rarely re-seat their DIMMs.

E. How long does Centauri take to extract fingerprints?

Efficiency, quantified by the time required to generate fingerprints, is another important metric to evaluate a fingerprinting technique [50]. An efficient fingerprinting technique should be able to extract fingerprints in a relatively short duration of time in order to remain practical.

When accessing aggressors 10,000,000 times to execute Rowhammer, performing the hammering sweep operation on a single 2 MB chunk takes an average of 20 seconds. Repeating the operation to account for non-determinism in bit flips further increases the time taken by Centauri to extract a fingerprint. Even in the limiting case where we have enough references to cover all 2 MB chunks in a given DIMM, Centauri takes almost 3 minutes to extract a fingerprint when accessing aggressors 10,000,000 times and repeating the hammering sweep 8 times. One way to improve Centauri's efficiency would be to reduce the number of times the aggressors are accessed to trigger bit flips. Another way would be to reduce the number of times we repeat the hammering sweep operation. Employing either approach is subject to making sure that they do not significantly degrade Centauri's fingerprinting accuracy.

In this section, we present a comprehensive analysis of the trade-off between fingerprint accuracy and efficiency when running Centauri. 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



(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

Fig. 11. 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.

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 11 shows the accuracy, precision, recall and time elapsed in all 15 configurations. These results show that while tuning either parameter to reduce the time taken by Centauri to extract fingerprints comes at the cost of its fingerprinting capability. However, accessing aggressors 1,000,000 times and repeating the hammering sweep operation twice only takes 9.92 seconds to extract fingerprints, while still maintaining a precision of 95.96% and a recall of 84.61% (accuracy of 99.27%). Based on their use case, fingerprinters can tune these parameters to choose between accuracy and efficiency.

F. Can Centauri extract robust fingerprints in presence of external factors?

In this section, we evaluate the robustness of Centauri's fingerprints to external factors (outside the control of the fingerprinter) that can influence the behavior of bit flips. Concretely, we evaluate Centauri's robustness in context of CPU frequency since fewer bits flip at lower frequencies as compared to higher frequencies. The operating frequency of a user's CPU is subject to change [11] and cannot be controlled by the fingerprinter.

In our experiments, we first extract fingerprints (bit flip distributions) on all DIMMs in our test bed when running the CPU at its highest frequency of 3600 MHz. Then, we extract fingerprints from the same DIMMs when running the CPU at a lower frequency of 2800 MHz. On average, we see that there is 2 orders of magnitude difference in the number of bit flips that trigger at the two frequencies. Comparing the fingerprints extracted at these two different frequencies, Centauri achieves accuracy of 99.09%, corresponding to a precision of 94.2% and recall of 81.56%. These results indicate that Centauri only suffers a modest drop in its ability to extract fingerprints even when matching bit flip distributions extracted at different frequencies. Since some CPU governors dynamically scale the CPU frequency based on the system load (e.g., ondemand [11]), Centauri can further improve its accuracy in such cases by running a CPU-intensive program to exert system load and increase the frequency.

Being robust to variations that result from external factors such as CPU frequency also demonstrates the impact of Centauri's design of picking hammering patterns that produce the most number of bit flips. We conduct experiments by intentionally picking a pattern discovered in the templating phase that does not trigger the most number of bit flips. We notice that this pattern generalizes to trigger bit flips on fewer DIMMs when compared to the pattern that triggers the most bit flips. We first run evaluate this pattern on a set of 10 DIMMs where it produces bit flips without altering the frequency. In this case, the pattern has a fingerprint accuracy of 95.92%, precision of 100% and recall of 71.43%. On the same set of DIMMs, this pattern has a recall of 0% when comparing fingerprints (bit flip distributions) at different CPU frequencies, since it was unable to trigger bit flips at the lower frequency. Results in the next section, §VI-G show that Jaccard similarity (proposed by Rowhammer PUF) has a lower accuracy than Centauri when matching fingerprints across frequencies.

G. How does Centauri compare against Rowhammer PUF?

We compare Centauri's fingerprinting accuracy against that of an adapted version of Rowhammer PUF that uses Centauri's techniques to overcome the challenges enforced by the OS for memory allocation as well as Rowhammer mitigations (TRR). Thus, in this section we evaluate Centauri's approach of hammering the same multiple times to extract probability distributions and comparing their divergence to match fingerprints against Rowhammer PUF's approach of hammering each chunk once to extract sets of bit flips and comparing their similarity. When accessing aggressors 1 million times (the optimal number of times to trade-off between accuracy and efficiency per our discussion in §VI-E), Centauri has an accuracy of 99.27%, corresponding to a precision of 95.96% and recall of 84.61%. With the same number of accesses on the same set of DIMMs, Rowhammer PUF has an accuracy of 98.01%, precision of 89.59% and recall of 64.64%.

Furthermore, in presence of external factors outside the fingerprinter's control, such as the CPU frequency, that affect the number of bit flips triggered, Centauri reports an accuracy of 99.09%, precision of 94.2% and recall of 81.56%. On the same set of DIMMs, using Rowhammer PUF to match fingerprints yields an accuracy of 96.3%, precision of 100% and a significantly low recall of 20.09%.

VII. Discussion

A. Extension to browser and mobile fingerprinting

So far, we discussed Centauri in context of device fingerprinting, where a fingerprinter can run native code on a user's desktop. In this section, we discuss how to adapt Centauri to operate in a situation where the user visits a website under the control of the fingerprinter or installs an app developed by the fingerprinter on their mobile phone. In both cases, fingerprinters cannot explicitly flush the cache which they could do with native code on desktops. Additionally, most Android devices do not support huge pages for the fingerprinter to allocate contiguous memory [53].

SMASH [10] describes self-evicting Rowhammer patterns that can evade TRR and trigger bit flips from JavaScript. While the SMASH paper presents results on CPUs that use Quad-age LRU [48] cache replacement policy, fingerprinters targeting the web can discover the replacement policies and correspondingly discover self-evicting Rowhammer patterns on other CPUs by running experiments on their own CPUs in Centauri's templating phase. Fingerprinters still have to know their user's CPU microarchitecture to pick the appropriate self-evicting pattern to trigger bit flips. With native code, fingerprinters can merely read this from `/proc/cpuinfo`, but cannot do so from the browser. Rowhammer.js [17] describes timing side channels to infer a user's CPU microarchitecture. Thus, fingerprinters running a website, can first infer the microarchitecture of a user by adopting Rowhammer.js and alter known non-uniform patterns into self-evicting patterns for that microarchitecture using SMASH to trigger bit flips. Once triggered, they can use Centauri's sampling and matching strategy to extract and compare fingerprints.

Drammer [53] uses Android's ION memory allocator for uncached access to contiguous memory to trigger bit flips. However, some devices have disabled access to contiguous memory via ION. More generally, as long as there are techniques to get uncached access to contiguous memory, they can be combined with Centauri to extract fingerprints on mobile.

B. 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 [44], we suspect that the channel bits can be derived from a combination of the bits at indices 7, 9, 14, and 17[3]. Thus, we do not change the bit at index 17 to ensure that we do not change the channel.

[3]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

C. Mitigating Centauri

1) Inapplicability of common defenses: **Standard fingerprinting defenses** Standard mitigations against fingerprinting such as normalization [54] or enforcing permissioned access [42] cannot be employed against Centauri. Our results demonstrate that we can extract unique and stable fingerprints even among homogeneous devices. Normalization as a defense against Centauri would require eliminating process variation in the manufacture of DRAM chips, which is difficult to implement. Since all applications including those that are benign require access to memory, blocking access to memory or requiring user permission to access memory would not be a viable mitigation against Centauri. To the best of our knowledge, triggering additional bit flips to obfuscate or spoof the distributions extracted by Centauri are not viable since they risk hurting benign usage.

Standard Rowhammer defenses With Centauri, 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 may be employed agaisnt Rowhammer. For example, DDR5 DIMMs are expected to have in-DRAM ECC [23] (Error Correction Codes) to mitigate Rowhammer. However, existing research [9] has already shown that bit flips can also be triggered on ECC equipped systems. Thus, ECC does not guarantee a defense against Centauri.

2) Potential defenses against Centauri: At its core, Centauri relies on observing bit flips produced by Rowhammer to extract fingerprints. Thus, any defense that can prevent bit flips from being triggered or observed would be able to overcome Centauri.

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 Centauri. For example, as a result of interleaving of contiguous memory across channels, a CPU that has 2 channels with each channel having 2 DIMMs would result in contiguous 2 MB chunks not spanning more than one row per bank. First, Such a configuration would make it impossible to execute a double-sided Rowhammer, which is more reliable in producing bit flips. Second, and more importantly, even if a fingerprinter is able to trigger bit flips with such a configuration, they cannot observe them since bit flips are typically triggered in the row adjacent to the row being aggressed.

Preventing bit flips during the manufacturing process In our experiments, we had difficulty triggering bit flips in DIMMs from a particular manufacturer. Most DIMMs from the manufacturer did not produce any bit flips. When repeatedly sweeping though contiguous 256 MB of memory, we occasionally observed a small number of bit flips (at most 200 bit flips) on 2 DIMMs. The DIMMs from this manufacturer are either robust to Rowhammer in that they do not trigger bit flips or they use a complex implementation of TRR that makes it difficult to consistently trigger bit flips. We were unable to fingerprint these DIMMs in our experiments.

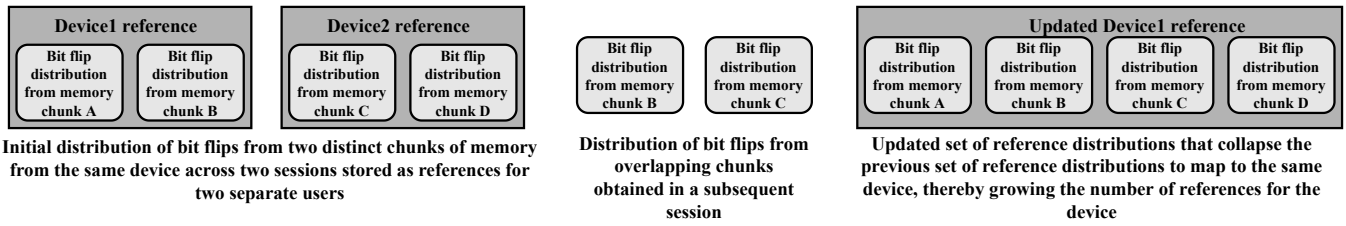


Fig. 12. Visualization of incrementally building up reference fingerprints. In this figure, we assume that the fingerprinters first obtained bit flip distributions of two separate sets of contiguous chunks of memory from the same device across two separate sessions. Each set contains the distribution 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.

Diversifying the number of TRR implementations The time taken for Centauri to extract a fingerprint increases as the number of possible TRR implementations increase, thereby decreasing Centauri's efficiency. Fingerprinters do not have a way to know if a given hammering pattern will trigger bit flips on a DIMM without executing the pattern. At best, fingerprinters can use timing side channels (see Appendix C) to determine a DIMM's geometry (number of ranks, width etc.) and limit themselves to only executing patterns discovered on DIMMs with the same geometry. Since multiple DIMMs with the same geometry can have different TRR implementations, fingerprinters may have to attempt multiple patterns to trigger bit flips. Thus, even if fingerprinters have found hammering patterns that can overcome all implementations of TRR, they have to execute all their patterns in the worse case, thereby increasing the time taken to extract fingerprints.

D. Incrementally building up references

In this section we discuss an alternate design of Centauri that does not leave a distinct footprint by having to allocate multiple transparent huge pages in the initial stages before reaching the limiting case (§V-C).

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 12. Say, during a user's first session when running Centauri 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, in the long run, fingerprinters would be able to fingerprint users without leaving behind a distinct memory footprint. However, in order to do so, fingerprinters would have to give up being able to fingerprint devices during their initial sessions.

E. Centauri for fraud detection

Centauri can significantly strengthen fraud detection by generating fingerprints that can unique identify fraudsters' devices even among a population of identical devices over a long period of time. Fraudsters will also find it difficult to alter or spoof the fingerprints extracted by Centauri, since Centauri captures fundamental properties of hardware as fingerprints. However, Centauri's promise in detecting fraudsters comes with a non-zero risk to benign users. While triggering bit flips to extract fingerprints, Centauri could accidentally crash a user's device by flipping a sensitive bit reserved for the OS. In our experience, however, we see that such occurrences are extremely rate. OS vendors can also help mitigate this concern by ensuring that memory allocated to the OS does not physically border around memory allocated to other applications. Another risk presented by Centauri is that it could wear out memory modules if it is used to constantly trigger bit flips for fingerprinting. Centauri'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 Centauri to only handle the critical cases.

VIII. Conclusion

We presented Centauri to extract unique and stable fingerprints even for devices with identical hardware and software configurations. To this end, Centauri leverages Rowhammer to capture the side-effects of process variation in the underlying manufacturing process of memory modules. Centauri'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 Centauri 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. Centauri 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] Kuljit S. Bains and John B. Halbert. Distributed row hammer tracking, 2012. US Patent US20140095780A1.

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

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

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

[8] 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.

[9] 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.

[10] 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.

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

[12] 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.

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

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

[15] 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.

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

[17] 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.

[18] 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.

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

[20] 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.

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

[22] 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.

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

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

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

[26] 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.

[27] 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.

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

[29] Dawei Li, Di Liu, Yangkun Ren, Ziyi Wang, Zhenyu Guan, and Jianwei Liu. Device identification in multimedia systems based on dram fingerprinting, 2022.

[30] 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.

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

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

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

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

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

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

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

[38] 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/.

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

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

[41] Notes on AI. Jensen-Shannon Divergence. https://notesonai.com/Jensen%E2%80%93Shannon+Divergence.

[42] Lukasz Olejnik, Gunes Acar, Claude Castelluccia, and Claudia Díaz. The leaking battery: A privacy analysis of the html5 battery status api. *IACR Cryptology ePrint Archive*, 2015:616, 2015.

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

[44] 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.

[45] 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.

[46] 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.

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

[48] Saarland Informatics Campus. Caches. https://uops.info/cache.html.

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

[50] 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.

[51] 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.

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

[53] 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.

[54] 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.

[55] 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.

APPENDIX A
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 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,

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

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

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

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

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

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

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

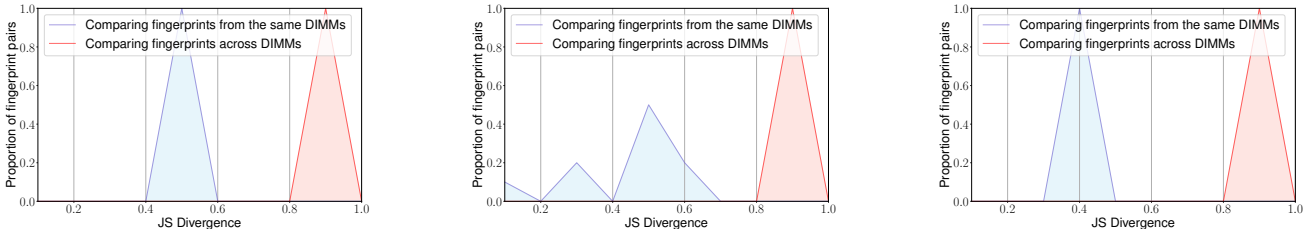
APPENDIX B
DISTIRBUTION OF JS DIVERGENCE VALUES ACROSS PAIRS OF FINGERPRINTS ON IDENTICAL DIMMS FROM ALTERNATE MANUFACTURER

Figures 13(a), 13(b) and 13(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 §VI-C.

APPENDIX C
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 [44]. We used DRAMA to reverse engineer the mapping functions for our machine [44], [15]. Figure 14 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 userspace [10]. Assume that the starting address is `0x0` and the ending address is `0xfffff` for the 2 MB chunk.



(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

Fig. 13. Plots showing the distribution of JS divergence values when comparing bit flip distributions obtained from the same pair of DIMMs and across different DIMMs.

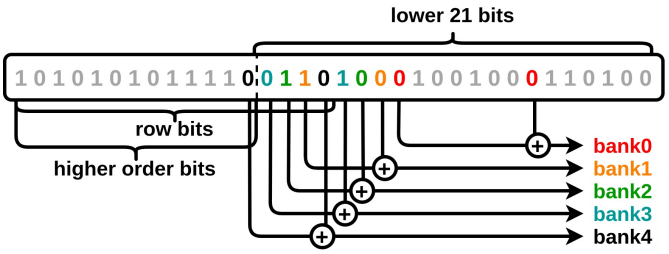


Fig. 14. Highlighting the XOR bank functions used on an Intel Kaby Lake processor.

```
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 [21], [43]. 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 [47], [30], [36], [49]. 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.