FracDRAM: Fractional Values in Off-the-Shelf DRAM

Fei Gao, Georgios Tziantzioulis, and David Wentzlaff
Department of Electrical and Computer Engineering
Princeton University
Princeton, NJ, USA
feig@princeton.edu, georgios.tziantzioulis@pm.me, wentzlaf@princeton.edu

***Abstract*—As one of the cornerstones of computing, dynamic random-access memory (DRAM) is prevalent across digital systems. Over the years, researchers have proposed modifications to DRAM macros or explored alternative uses of existing DRAM chips to extend the functionality of this ubiquitous media. This work expands on the latter, providing new insights and demonstrating new functionalities in unmodified, commodity DRAM. FracDRAM is the first work to show how fractional values can be stored in off-the-shelf DRAM. We propose two primitive operations built with specially timed DRAM command sequences, to either store fractional values to the entire DRAM row or to masked bits in a row. Utilizing fractional values, this work enables more modules to perform the in-memory majority operation, increases the stability of the existing in-memory majority operation, and builds a state-of-the-art DRAM-based PUF with unmodified DRAM. In total, 582 DDR3 chips from seven major vendors are evaluated and characterized under different environments in this work. FracDRAM breaks through the conventional binary abstraction of DRAM logic, and brings new functions to the existing DRAM macro.**

***Keywords*-DRAM; PIM; Processing-with-Memory; Memory Controller; PUF**

I. Introduction

The stalling of Moore's Law and end of Dennard scaling have motivated computer architects to investigate non-traditional media for computing. A prime example of this is research that investigates using and modifying DRAM, a ubiquitous tool for storing short term values, as a medium for computing. To the external user, a DRAM is a matrix of storage elements where information is stored in one of two states, zero or one. However, things in the world are not just black and white, and that holds for DRAM too. This paper demonstrates how to leverage that "grey part" of DRAM.

Near-memory [1], [2], [3], in-memory [4], [5], [6], [7], [8], and with-memory-computing [9], [10], [11], [12] are growing fields driven by the potential to reduce the energy consumption of data transfer between storage and processing units, as well as the search for an alternative and ubiquitous media that can be used for computation and related needs. This has fueled a reexamination of the operation and design of DRAM modules at all levels. One direction explored by prior work, such as Ambit [12] and RowClone [13], has investigated modifying DRAM macros to enable charge sharing between rows. A different approach has reexamined the operation of unmodified, commodity, DRAM by exploring how DRAM functions outside of the JEDEC specification. This reexamination has lead to discoveries such as Compute-DRAM [14] which has shown that with-memory computation can be realized in unmodified commodity DRAM modules by using out-of-specification timed command sequences.

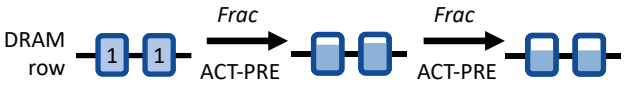


Figure 1: Utilizing a specially timed command sequence to store fractional values in off-the-shelf DRAM cells.

FracDRAM extends the later efforts and provides new insights and showcases new functionalities in unmodified, commodity, DRAM. Our work dissects the binary abstraction that data stored in DRAM cells can only be one or zero. In reality, this is not the case. Unlike SRAM, whose stable states are only two, the voltage level of a DRAM cell, which at its heart is a capacitor, can occupy any level between ground and V_{dd}. In fact, the DRAM cell is not always "full" due to leakage. Previous works have also shown DRAM working with a voltage lower than V_{dd} [15], [16], [17], [18]

The main requirement for exploiting this spectrum of voltages is the capability to set the voltage level with fine control. Achieving this seems to require a custom DRAM chip as there is no built-in functionality to set fractional values in commodity DRAM. However, closer examination of a DRAM module and its operation shows that in addition to the circuitry to drive voltage to V_{dd} or ground, there is also circuitry to drive voltage to $V_{dd}/2$. This circuitry is used during the `PRECHARGE` command (Section II). FracDRAM leverages this key insight and is **the first work to store fractional values in unmodified, commodity DRAM.** FracDRAM demonstrates the ability to generate three distinguishable states without any modification to DRAM and that the data-path to store these different voltage levels already exists in commodity DRAM. FracDRAM is capable of doing this by using novel timing of DRAM command sequences.

Building on this key insight, we create primitive operations that store fractional values to an entire row and to masked bits within a row. We also propose methods to verify the

978-1-6654-6272-3/22/$31.00 ©2022 IEEE
DOI 10.1109/MICRO56248.2022.00066

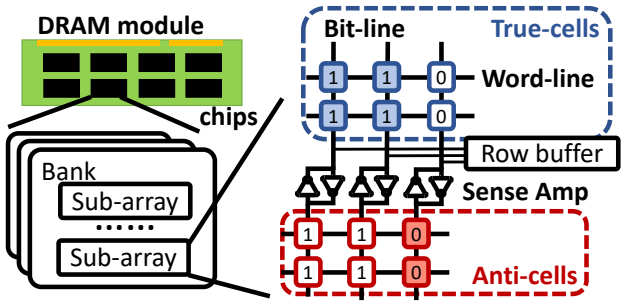


Figure 2: Basic DRAM structure within a sub-array. Shade amount in the cell represents the charge stored in it.

effectiveness of the primitives, which can be regarded as a destructive readout of the fractional value, and demonstrate those operations with unmodified, commercial, DRAM modules. We explore two use cases in detail: ① leveraging fractional values to expand the number of DRAM modules that can support ComputeDRAM-style majority operations with increased stability, and ②, using fractional values to implement a DRAM-based Physically Unclonable Function (PUF) in unmodified DRAM with state-of-the-art throughput. In addition to these examples, the primitives provided in FracDRAM can be leveraged to characterize DRAM data retention time, help people understand and reverse-engineer the "blackbox" design of the DRAM array, potentially increase the storage density of DRAM, and potentially support with-memory ternary computation.

The contributions of this work are the following:

- Demonstration of the first successful storage of fractional values in unmodified, commercial, DRAM.
- Utilization of fractional DRAM storage to expand ComputeDRAM-style, in-memory, majority operation to a larger set of DRAM module types.
- Utilization of fractional DRAM storage to enhance the reliability of existing ComputeDRAM-style majority operation, reducing the error rate from 9.1% to 2.2%.
- Demonstration of using fractional DRAM storage to implement a Physically Unclonable Function (PUF) in unmodified DRAM with state-of-the-art throughput.
- A careful evaluation and characterization of storing fractional values in DRAM and it's applications.

II. Background

FracDRAM is constructed using specially timed DRAM command sequences. In this section, we briefly introduce essential concepts for understanding FracDRAM : the DRAM structure and cell polarity, basic DRAM commands, and the multiple-row-activation.

A. DRAM Structure

Dynamic random-access memory (DRAM) consists of arrays of cells, each of which stores single-bit data using the voltage level in the cell capacitor. DRAM memory is hierarchically structured and organized in levels of: ranks, banks, and sub-arrays. For the purpose of this work we only focus on the DRAM structure within a sub-array; for a detailed description of DRAM organization and operation see: Jacob *et al.* [19] and Keeth *et al.* [20].

At the sub-array level, word-lines control the connection between the cells in a row and the bit-lines; bit-lines can be connected to all cells in the same column. At the end of each bit-line, there is a sense-amplifier which can drive the voltage to V_{dd} or 0 when enabled, and pass the sensed voltage to the row buffer for read/write. Figure 2 provides a graphical representation of the DRAM organization and the core components of a sub-array.

B. DRAM Commands

The operation of the DRAM module is controlled by the memory controller (MC). The MC translates high-level memory access instructions and their addresses to DRAM commands with detailed row and column addresses. FracDRAM uses two of the DRAM commands to construct fractional value storage in commercial DRAMs:

I. `PRECHARGE`: takes the bank address as an argument and has two steps. First, all opened rows are closed by zeroing all word-lines in the target bank, which disconnects the cells from the bit-lines. Second, all bit-lines are driven to $V_{dd}/2$ as an initial value.

II. `ACTIVATE`: takes the bank and row addresses as arguments, and has three steps. First, the target row's word-line is raised to high, connecting the cells of the row directly to the bit-lines. This enables charge sharing between a cell and its corresponding bit-line, which influences the voltage of the bit-line to slightly higher or lower than $V_{dd}/2$, depending on the value stored in the cell initially. At a second step, the sense amplifier is enabled, which amplifies the value held in the bit-line and drives the voltage to V_{dd} or 0. As the cell is still connected to the bit-line, its original voltage is recovered. Finally, the output of the activated local row buffer is transferred to the global row buffer.

Different DRAM commands perform different tasks and require different timing constraints (*i.e.*, memory cycles) to complete. It is the MC's responsibility to issue DRAM commands with enough idle cycles in between, following the timing constraints set by the JEDEC standard [21].

C. Cell Polarity: True-Cells vs Anti-Cells

DRAM design has been extensively optimized to improve its area efficiency. Modern DRAMs reuse another bit-line as the reference wire to the sense amplifier, as depicted in Figure 2. Since only one side of the sense amplifier is connected to the row buffer, the bit-line on the other side always holds an opposite state as the row buffer which

contains the read out value. The cells connected to the bit-lines on the "row buffer" side are called true-cells, while the cells connected to the "opposite" bit-lines are called anti-cells [22]. In true-cells, V_{dd} is read as a logic one, while in anti-cells V_{dd} represents a logic zero. We can identify anti-cells by pausing the refresh and checking if logic zero leaks to logic one (true-cells only leak from one to zero). In section V, we store opposite logic values to anti-cells by default, so that they physically hold the same voltage as true-cells. Since they are symmetric, for simplicity, we assume all cells are true-cells in the remainder of this work.

D. Multiple-Row-Activation

Under nominal operation, at most one row in a bank can be activated at a given time [21]. However, Gao *et al.* [14] demonstrated that multiple rows can be opened in unmodified DRAMs using a DRAM command sequence that violates the JEDEC timing constraints. The command sequence that was used is: ACTIVATE(R_1)-PRECHARGE-ACTIVATE(R_2), with 2.5*ns* per memory cycle and no idle cycles in between. R_1 and R_2 are two different row addresses within the same sub-array. Using the above command sequence, one or more other rows can be opened together with R_1 and R_2. The charge sharing among the cells in three opened rows is utilized to build a majority-of-three (MAJ3) operation. Olgun *et al.* [23] also found that the same command sequence can open four rows in DDR4 modules.

III. Primitive Operations

FracDRAM introduces *Frac* and *Half-m*, two novel primitive operations that utilize specially timed DRAM command sequences to store fractional values in off-the-shelf DRAM modules. Following, we elaborate on how each primitive operation is constructed and how it works at a low level. We explain how we validated that a fractional value is indeed generated and stored in the DRAM cell in Section IV-B.

A. The Frac Operation

The *Frac* operation allows us to store fractional values in an entire row. To construct *Frac*, we utilize two basic DRAM commands: ACTIVATE and PRECHARGE. The two commands need to be issued back-to-back without any extra cycles in between. At a high level, our goal is to use the PRECHARGE command to interrupt the process of row activation, and prevent the sense amplifier from being enabled.

Fig. 3 shows the voltage level of both the bit-line and the DRAM cell during *Frac*: First ①, we issue a PRECHARGE command to initialize the bit-line voltage to $V_{dd}/2$. The PRECHARGE command will close the activated row first. Thus, the bit-line voltage won't change until at least a memory cycle later. The initial voltage in the cell capacitor can be either V_{dd} or 0; we assume it is V_{dd} for this example. Following ②, we initiate the *Frac* operation by issuing an

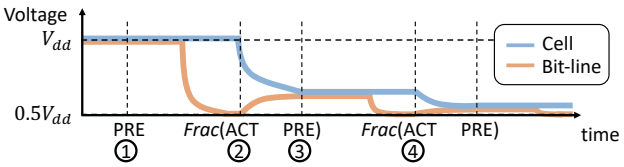


Figure 3: Voltage level of the cell capacitor and the bit-line during a *Frac* operation. In order to put all the commands together, the time scale in the horizontal axis is not even: step ② and ③ are in two consecutive cycles, while the time between step ① and ②, step ③ and ④, is at least 5 cycles. We assume each memory cycle is 2.5ns.

ACTIVATE to the target row, which raises up the word-line and connects the cell capacitor to the bit-line. After the charge sharing, both the cell and the bit-line reach the same voltage level which is slightly higher than $V_{dd}/2$. The equilibrium voltage is closer to the initial bit-line voltage because the bit-line capacitance is much larger than the cell's [13]. In the following cycle, step ③, we issue a PRECHARGE command to interrupt the process of row activation before the sense amplifier is enabled. After that, the cell capacitor is disconnected from the bit-line while holding a fractional value; neither V_{dd} nor 0, but a voltage slightly higher than $V_{dd}/2$. Since we need to wait for the PRECHARGE to finish, the total latency of a *Frac* operation is 7 memory cycles (two command cycles plus five idle cycles).

With a single *Frac* operation, we manage to store fractional values in DRAM cells in the same row. However, this fractional value depends on the initial state in the cell. For another column where the initial voltage is 0, the resultant voltage will be something between $V_{dd}/2$ and 0. If we want to make the stored voltage more accurate, closer to $V_{dd}/2$, and independent from the initial value, we can simply issue multiple *Frac* operations. At step ④, when the PRECHARGE command from the previous *Frac* is completed, we are ready to issue another *Frac* operation. We found that the more *Frac* operations we issue, the closer the resulted voltage is to $V_{dd}/2$, and the cell voltage will be more consistent across the row regardless of the initial value. We evaluate how the initial cell voltage and the number of *Frac* operations influence the final voltage that we generate in section V.

B. The Half-m Operation

Our second technique, *Half-m*, stores Half values to masked bits; *i.e.*, it stores a mixture of normal values and Half values in the same row. In *Half-m*, instead of interrupting a single row activation, as we do in *Frac*, we use a PRECHARGE command to interrupt the process of a four-row-activation.

As mentioned in Section II, the sequence ACTIVATE(R_1)-PRECHARGE-ACTIVATE(R_2) triggers a multiple-row-activation. We tested multiple groups of DDR3 modules

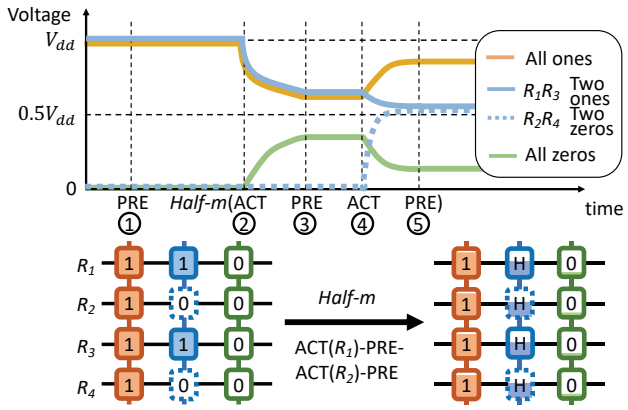


Figure 4: Voltage level of cell capacitors in three different columns during a *Half-m* operation. Lower half of the figure shows the data layout in the four opened rows. The curves in the voltage chart correspond to the cell whose outline has the same color or dotted line.

from SK Hynix, and found that besides opening three rows at the same time, we can open four rows as well using specific combinations of R_1 and R_2. For example, rows 0, 1, 8, 9 can be opened simultaneously by activating row 8 and row 1 back-to-back. QUAC-TRNG [23] also showed that four-row-activation can be achieved in commodity DDR4 modules. We construct *Half-m* by combining the command sequence of the four-row-activation, and a trailing PRECHARGE: ACTIVATE(R_1)-PRECHARGE-ACTIVATE(R_2)-PRECHARGE.

Fig. 4 shows the cell voltage in three different columns during a *Half-m* operation. As we store different initial values to the four opened rows (R_1 to R_4), different results (logical 0, 1, and Half value) are generated respectively. *Half-m* begins with ① a PRECHARGE command which closes all the rows and resets the bit-line voltage. Steps ② - ④ are essentially a four-row-activation. As explained in previous works [14], [23]: The PRECHARGE in step ③ doesn't have enough time to close the row, thus R_1 remains open. The glitch of the row decoder further implicitly opens extra rows, R_3 and R_4, after we issue ACTIVATE(R_2) at ④. We choose the address of R_1 and R_2 carefully to make sure that four rows are activated by the time of step ⑤. Consider the cells in the same column of these four rows, if they all have the same initial value (0 or 1), they will change the bit-line voltage to the same direction together, and the voltage at step ⑤ will be close to 0 or V_{dd}, as shown with the green and yellow line. If we store two ones and two zeros in these cells in the beginning, the final voltage will be close to $V_{dd}/2$, similar to the blue lines. Finally, ⑤ we issue the last PRECHARGE to prevent the sense amplifier from being enabled. Since the bit-lines are disconnected from the cells, the cell voltage stays unchanged.

Without the help from the sense amplifier, both logical one and zero are not fully recovered to V_{dd} and 0 as we show in Fig. 4, thus we call them "weak" ones and zeros. The Half value is not exactly $V_{dd}/2$ either, due to the asymmetry of those opened rows. But as long as we can distinguish these three generated values from each other and show that "weak" ones/zeros behave the same as normal ones/zeros, we could say that *Half-m* successfully stores a mixture of zero, one, and Half value in the same row.

Table I: Evaluated DRAM chips and their capability of performing *Frac*, three-row-activation, and four-row-activation.

Group	Vendor	Freq(MHz)	# Chips	*Frac*	Three-row-activation	Four-row-activation
A	SK Hynix	1066	16	✔		
B	SK Hynix	1333	80	✔	✔	✔
C	SK Hynix	1333	160	✔		✔
D	SK Hynix	1600	16	✔		✔
E	Samsung	1066	32	✔		
F	Samsung	1333	48	✔		
G	Samsung	1600	32	✔		
H	TimeTec	1333	32	✔		
I	Corsair	1333	32	✔		
J	Micron	1333	16			
K	Elpida	1333	32			
L	Nanya	1333	32			

C. Refresh

The fractional value stored in a DRAM cell can be destroyed by any row activation. Thus, whenever we have a fractional value stored in the DRAM array, we need to prevent the issuing of the REFRESH command to rows holding that fractional value. At the same time, if we also stored normal V_{dd} in other columns in the same row, we need to REFRESH that row to preserve the logical ones. Fortunately, the typical refresh cycle for a row – 64ms – is long enough for us to implement applications with fractional values. We only need to be cautious that REFRESH is not sent in the middle of the application.

IV. Methodology

In this section, we detail our experimental setup and our methodology for verifying the fractional values in off-the-shelf DRAMs. In our evaluation, we examine a total of 528 DDR3 chips from seven major DRAM vendors. Table I presents the full list of tested DRAM chips, as well as their nominal working frequency (speed level), and their capability of performing different in-memory operations. Among the in-memory operations, three-row-activation is the prerequisite of performing ComputeDRAM-style majority operation in the absence of *Frac*, and four-row-activation is necessary for *Half-m*. We divide the evaluated DRAM chips into 12 groups, from A to L, based on the vendor and DRAM configuration.

A. Platform

For our evaluation we used two setups with similar configurations as the one described in ComputeDRAM [14].

Figure 5: Evaluation platform including FPGA boards which the customized memory controller is programmed on.

We used a modified version of the SoftMC [24] software-controlled memory controller to generate the appropriate DRAM command sequences with special timing intervals. Each host computer sends DRAM command sequences to a Xilinx ML605 FPGA board through the PCIe bus, as shown in Figure 5. The hardware part of SoftMC, which is programmed in the FPGA, receives the command sequence and subsequently issues it to the real DRAM chip with specified timing. The working frequency of SoftMC is fixed to 400MHz, which means that the memory cycle is always 2.5ns no matter what speed level the DRAM chip has. Thus, whenever we mention "memory cycle", we assume 2.5ns per cycle.

B. Verification Methodology

Our goal is to demonstrate that a fractional value can indeed be generated and stored in commodity DRAM cells. However, the fractional value can not be simply read out of the DRAM module, as the sense amplifier will destroy the fractional value during row activation. Thus, we need a method to verify that the fractional value is there without actually probing into the cell. We propose two methods to achieve this: measuring the data retention time and checking the results of the majority-of-three (MAJ3) operation with fractional values.

1) Data Retention Time: The charge stored in a DRAM cell will gradually leak out of the cell capacitor if it's not activated for a while. In addition, there is a threshold voltage for the sense amplifier, below which the voltage level is regarded as a logical zero. When the voltage drops below that threshold such that the sense amplifier fails to recognize the logical one during activation, the data will be lost. The amount of time that a DRAM cell can hold the data is called the cell's *retention time*. Since the voltage is monotonically decreasing during the charge leakage, for the same cell, the higher the initial voltage is, the longer the "retention time" will be. Based on that, we can measure the retention time, and use it as an indicator for the starting voltage level in the cell.

2) MAJ3 operation with fractional value: Another method to verify the fractional value is to perform MAJ3 operations with it. Specifically, we perform MAJ3 twice, in both cases we store the same fractional value in the first two operands, and then we store different value of ones and zeros in the third operand. The detailed procedure is as follows:

1) Choose three rows that can be opened at the same time
2) Store fractional value into row R_1, R_2, and **one** in R_3
3) Perform MAJ3 on R_1, R_2, R_3. Record the result as X_1
4) Store fractional value into row R_1, R_2, and **zero** in R_3
5) Perform MAJ3 on R_1, R_2, R_3. Record the result as X_2

The two **different** MAJ3 results (X_1 and X_2) can be used to verify the existence of the fractional value.

The assumption we make is that, if we can store a fractional value, then we can produce the same fractional value on different rows in the same DRAM module using identical operations, and we can reproduce the same fractional value on the same row multiple times. This assumption can be proven using the "retention time" method, and we need it here to simplify the reasoning process. In the procedure above, we store a fractional value into row R_1 and R_2 twice, and we assume they all hold the same voltage V_{frac} after step 2 and 4. If we failed to produce the fractional value, in other words, if V_{frac} were still either 0 or V_{dd}, the MAJ3 result would be the same as the value we stored in R_1 and R_2, no matter what we set in R_3. Therefore, if we can ever get the result of $X_1 = V_{dd}$ and $X_2 = 0$ at the same time, it will prove that V_{frac} is neither V_{dd} nor 0, but rather a fractional value close to $V_{dd}/2$.

V. Evaluation

In this section, we present an evaluation and demonstration of proof-of-concept for the two primitive operations: *Frac* and *Half-m*. Specifically, we utilize the methods described in Section IV-B (retention time and MAJ3) to demonstrate the effectiveness of the proposed operations and the existence of fractional values in an entire DRAM row.

A. Evaluation of Frac: Retention Time Profile

The retention time collection is composed of two parts: first, we store ones in the entire target row, then we stop sending any memory commands in order to let the charge leak out of the cell. The second step is completed when after time t, we read out the data and record which bits have lost their data. If a bit holds one after time t_1 but loses its data after time t_2, the cell's retention time is narrowed down to the (t_1, t_2) range. Thus, by varying t and repeating the procedure multiple times, we can construct a profile of the retention time range of the cells in the target row. This profile shows how long the data can be preserved in each cell when full V_{dd} is stored, serving as a baseline. To evaluate *Frac*, the only difference in the procedure is that we issue one to five *Frac* operations right after all ones are stored in the target row, so that the retention time profile after *Frac* is collected.

To demonstrate our proof-of-concept, we performed a retention time experiment that included 16 chips from each of the 12 DRAM groups in Table I. For each DRAM chip, we randomly sample five rows from each bank to evaluate. The heatmap in Figure 6 presents the retention time profiles for the different DRAM groups. Columns present a Probability

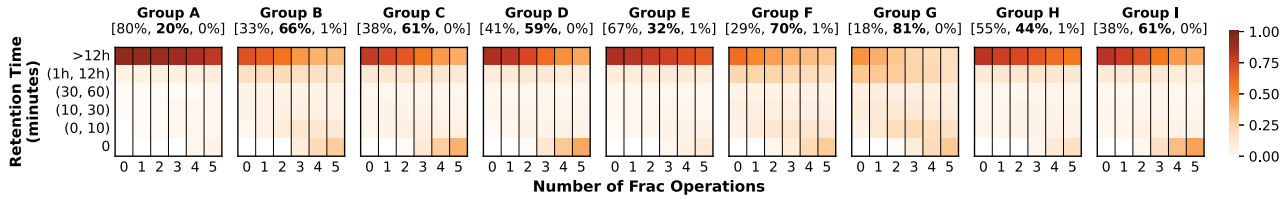


Figure 6: Change in the retention time as we issue 0-5 *Frac* operations to different DRAM groups. Each column of the heatmap represents a Probability Density Function (PDF) of the retention time. According to the change pattern, each individual cell is divided into three categories. The proportion of each category is shown by the numbers in brackets, as [long retention time, **monotonic decrease**, others] respectively. The second category demonstrates our proof-of-concept.

Density Function (PDF) of retention time, whereas rows represents retention time ranges. Darker shade in the box means that a larger portion of cells have a retention time in the corresponding range. DRAM retention time can be as short as several seconds, but those cells only make up a tiny portion; less than 10^{-4} [22]. Therefore, we choose a relatively large and coarse timescale, classifying the retention time into six ranges: 0, 0 to 10 minutes, 10 to 30 minutes, 30 to 60 minutes, 1 to 12 hours, and longer than 12 hours. A zero retention time means that the voltage in that cell is reduced below the threshold that the circuits regard as zero in the beginning; i.e., right after we issue the last *Frac*. This definition of zero retention time also complies with the monotonic relationship between the retention time and the cell voltage.

For groups J, K, and L, sending *Frac* operations has no effect in the retention time profile, thus we omit them from the plot. We speculate that those DRAM chips implement time checking circuits to prevent different DRAM commands being executed too close to each other.

For groups A to I, we analyse the change of the retention time on individual cells as more *Frac* operations are issued. According to the change pattern, we divide the cells into three categories. The cells in the first category have a long retention time. They are in the bucket of "$> 12h$" all the time, even after five *Frac* operations are sent. The cells in the second category exhibit a monotonic decrease of retention time ranges as we issue more *Frac* operations. This is the group of cells we are most interested in; the proportion of bits in this category is shown in bold numbers in Figure 6. The last category includes the remaining bits, they may exhibit an increased retention time as we issue *Frac*, or the retention time range can not be fixed in the experiment.

The cells in the second category - 55% on average - illustrate that *Frac* operations can reduce the cell's retention time incrementally. Given the monotonic relationship between the retention time and the voltage (discussed in Section IV-B1), we can further conclude that the *Frac* operation can reduce the cell voltage incrementally. According to our understanding of low-level DRAM circuits, with an initial voltage of V_{dd}, *Frac* can not generate a voltage lower than $V_{dd}/2$, which is the initial bit-line level, in a DRAM cell. Therefore, the reduced voltage value must be between V_{dd} and $V_{dd}/2$. The above leads us to draw two conclusions (for initial value of all ones): ① ***Frac* operation can store fractional voltages in DRAM cells**, and ② **consecutive *Frac* operations lower the fractional voltage towards $V_{dd}/2$.** Although the first category does not contribute to our point, it is not a counter example either. Those cells just have a retention time longer than we can profile. The third category exhibits irregular change patterns in the retention time, however, it only contains less than 1% of the cells, and we speculate that the affected cells have intrinsic variable retention time (VRT) [25], [26].

Despite the insights we extracted from, the retention time method in itself has several limitations. First, we only proved our claim on a portion of cells (55% in average). The main reason for that is the existence of cells that have long retention times that fall into the "$> 12h$" range. Extending the experiment beyond the 12 hour range significantly increases experimentation time, which makes it impractical to narrow down the retention time for all the cells. Second, the retention time of a cell is determined by multiple factors, such as the process variation, environmental changes, and a small portion of cells even exhibits variable retention times [25], [26]. If a fractional value is stored in one cell, it might still have a longer retention time than another cell which stores a full V_{dd}. Third, as charge only leaks from high voltage to ground, we can only test retention time with an initial value of one.

Based on the above and given the results of our retention time profiling, we can confidently say that *Frac* can produce a fractional value in DRAM chips for most DRAM vendors and for different positions in a DRAM chip. However, we still do not know if all of the bits in a row can reliable reproduce the fractional value, and if *Frac* works with an initial value of zero. We need the MAJ3 method to complete the characterization.

B. *Evaluation of* Frac*: MAJ3 Results*

In our second set of experiments, we follow the procedure described in Section IV-B2. We use the first three rows in each sub-array and execute the command sequence

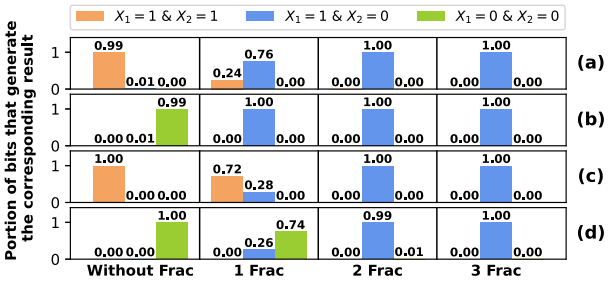


Figure 7: MAJ3 results as we change the number of *Frac* operations we send. (a) and (b) use R_1 and R_2 to store fractional values; (c) and (d) use R_1 and R_3 to store fractional values. (a) and (c) use all ones as the initial value thus the fractional values should be between $V_{dd}/2$ and V_{dd}; (b) and (d) use all zeros as the initial value thus the fractional values should be between 0 and $V_{dd}/2$.

ACTIVATE(R_1)-PRECHARGE-ACTIVATE(R_2), where $R_1 = 1$, and $R_2 = 2$; this leads to $R_3 = 0$ [14]. Before we issue the MAJ3 operation, we store fractional values to R_1 and R_2 using various numbers of *Frac*. As this test relies on the MAJ3 result, which requires three-row-activation, we can only conduct the experiment on group B. Thus, for every chip in group B, we scan every sub-array and record the result of X_1 and X_2. Figure 7 presents the results: each bar represents the proportion of columns that generate the corresponding result of X_1 and X_2.

For Fig. 7a, we store all ones as the initial value in R_1 and R_2 before we send any *Frac* operations. The configuration without any *Frac*, our baseline, is equivalent to a normal MAJ3 operation with operands in R_1 and R_2 being all ones. The observed baseline result meets our expectation: both X_1 and X_2 are one. Similarly, for Fig. 7b, the initial value in R_1 and R_2 is zero, and the MAJ3 result without any *Frac* is all zeros. However, as we issue *Frac* operations, the combination of $X_1 = 1$ and $X_2 = 0$ begins to dominate and it becomes the only result after two or more *Frac* operations are sent. This is because *Frac* operations make R_1 and R_2 hold fractional values and generate the different MAJ3 results of X_1 and X_2.

For Fig. 7c and Fig. 7d, we change the rows that hold fractional values to R_1 and R_3, and store all ones and all zeros to R_2. The result pattern with one *Frac* is a little different from the first two subplots, but the overall conclusion is the same: it proves that **fractional values can be stored in almost every bit in the DRAM chip**, and the initial value stored in the row doesn't affect the quality of the fractional value after multiple *Frac* operations are executed.

We want to highlight that **while *Frac* is based on unspecified behaviour of DRAM chips, it does not mean it is an unreliable operation**. Our above evaluation showed that *Frac* exhibited a stable, predictable, behaviour wherever we were able to study it (DRAM group B). For the DRAM groups that we were unable to thoroughly study *Frac* due to their lack of capability to perform a three-row-activation, we make the hypothesis that *Frac* has similar characteristics and can work on all bits. Our evaluation of the use case of *Frac* in Section VI-B2 provides supports for this hypothesis.

C. *Evaluation of* Half-m

Half-m, our second primitive, can store half values on masked bits in a row. It utilizes charge sharing among four simultaneously opened rows. If the initial values stored in these four rows are two ones and two zeros, it will hold a fractional value close to $V_{dd}/2$ after *Half-m*.

Similar to the evaluation method we used for *Frac*, we conduct both a retention time test and a MAJ3 test on the value generated from *Half-m*. We only evaluate the modules in group B during this experiment due to the limited support for the MAJ3 operation in groups C and D; However, groups C and D are also capable of performing four-row-activation and *Half-m*. For four-row-activation, we choose the rows 0, 1, 8, 9 in every sub-array and make $R_1 = 8, R_2 = 1$. After we issue the command sequence in *Half-m*, $R_3 = 0$ and $R_4 = 9$ will also be activated. We store one to R_1 and R_3, and zero to R_2 and R_4 as the initial value to generate a Half value with *Half-m* in that column. In contrast, the initial value of all ones/zeros in four rows would generate a result of "weak" ones/zeros after *Half-m*, as we show in Figure 4. After *Half-m* is performed, we plot the retention time PDF with the same six ranges as in Figure 6, for both the Half value and "weak" ones. Since the result of *Half-m* is stored in row 0 and 1 already, we can store ones/zeros to row 2 to perform the MAJ3 operation among the first three rows and get results of X_1 and X_2 respectively. The retention time and MAJ3 results are shown in Figure 8.

As a reference point, we also plot the retention time PDF of the fractional values generated by five *Frac* operations from the same row. Compared with that, the retention time of Half values present a similar distribution. That proves a fractional value close to $V_{dd}/2$ is indeed stored in some cells after *Half-m*. However, the fractional value is not consistent across the row. The MAJ3 result shows that only 16% of bits can generate the distinguishable Half value, i.e. having a MAJ3 result of $X_1 = 1$ and $X_2 = 0$. The "weak" ones and zeros do have decent quality, as the retention time of "weak" ones is same as normal ones, so does the MAJ3 result of "weak" ones/zeros.

Although not applicable to all the columns, this result is a proof-of-concept showing that *Half-m* can be used to store three distinguishable states in cells from the same row. Moreover, since it is built upon a four-row-activation, it has the potential to be extended to DDR4 modules [23] as well.

VI. Use Cases

In this section, we describe use cases that utilize the proposed operations: extending in-memory majority-of-three

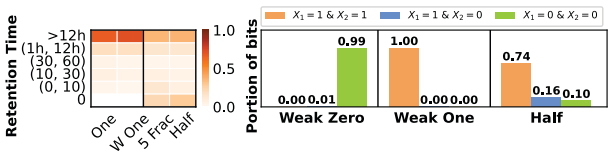


Figure 8: Both the retention time profile and the MAJ3 results for the "weak one", "weak zero" (no retention time profile), and Half value generated in *Half-m*.

operation to more modules and increasing the reliability of the existing in-memory majority operation, generating Physical Unclonable Function (PUF), and potentially expanding the storage capabilities of DRAM modules. We also characterize two applications in detail with real DRAM chips using the platform described in Section IV-A.

A. Extending in-memory majority operation in more modules

1) Proposal of F-MAJ: As mentioned in Section II, in-memory MAJ3 can be performed in off-the-shelf DRAM modules. However, according to Gao *et al.* [14], only a limited number of modules from SK Hynix are able to perform MAJ3. While more modules[1] are capable of opening multiple rows at the same time, the resultant value does not conform to the MAJ3 logic.

Our experiments revealed that on these special DRAM groups the root cause of the failure is that the command sequence that performs MAJ3 actually activates four rows instead of three. Specifically, we performed a thorough exploration using the sequence ACTIVATE(R_1)-PRECHARGE-ACTIVATE(R_2), with all possible combinations of row addresses R_1 and R_2. We found that within these modules (group C and D in Table I), only N rows can be opened simultaneously, where N is a power of two; *i.e.* 2, 4, 8, 16, *etc.* Moreover, all combinations of R_1 and R_2 that can open 2^k rows have k bits in difference. However, not all combinations of R_1 and R_2 that have k different bits can open 2^k rows. This result partially verifies the hypothetical row decoder circuit that Olgun *et al.* [23] used to explain the behavior of four-row-activation. As a detailed study of the row decoder structure is beyond the scope of this paper, we only want to briefly highlight that **for some modules, both DDR3 and DDR4, opening four rows is feasible, but not three**.

Having a method to concurrently open four rows in a sub-array raises the question: what more can this enable? An issue is that it is difficult to build logic that utilizes "majority-of-four". The challenge lies in defining the result when two operands are zeros and two others are ones. In contrast, charge sharing among three rows naturally generates the MAJ3 result. Following, we explain how we **successfully construct a MAJ3 operation using the charge sharing among four rows**; a major contribution of this work.

The key to transform a four-row-activation into MAJ3 is storing fractional values in one of the four operands. This is because the row that holds fractional values, which is close to $V_{dd}/2$, will have the least influence on the bit-line voltage during the charge sharing, and thus the final result will depend on the majority voltage among the other three rows. We call this operation, that utilizes the **F**our-row-activation to perform MAJ3, **F**-MAJ. The detailed procedure to perform F-MAJ is as follows:

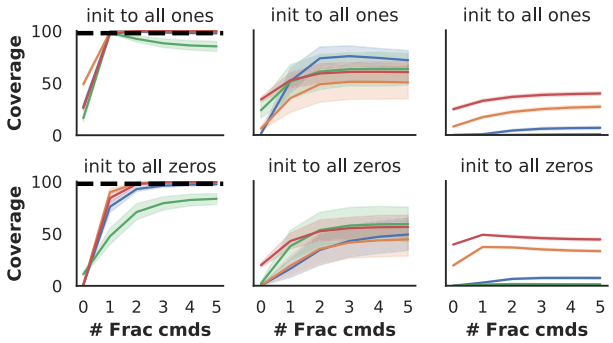
1) Choose four rows that can be opened at the same time
2) Store a fractional value into one row with one or multiple *Frac* operations. An initialization to all zeros/ones before *Frac* is preferred to make the fractional value more even across the row
3) Store three operands of MAJ3 to the other three rows
4) Issue ACTIVATE(R_1)-PRECHARGE-ACTIVATE(R_2), and the result of MAJ3 will be stored on all four rows

Compared to the original MAJ3 operation, the overhead of F-MAJ is the cost of storing a fractional value in an entire row. The initialization before *Frac* can be implemented with a single row copy proposed in ComputeDARM [14]. The subsequent *Frac* operation only consists of two memory commands (7 memory cycles), which is more light-weight than a row copy (18 memory cycles). Assuming the same strategy as ComputeDRAM, which exclusively uses reserved rows for computation, we need to copy the operands to the reserved locations and copy the result back as well. In that case, F-MAJ takes only 29% more memory cycles than the original MAJ3. However, with F-MAJ, we are capable of performing the majority-of-three operation and support ComputeDRAM-style computation in more DRAM modules, even DDR4 modules potentially [23], which cannot open three rows at the same time, but can open four rows. Moreover, we found that F-MAJ can even generate a more stable result compared to the original MAJ3 operation.

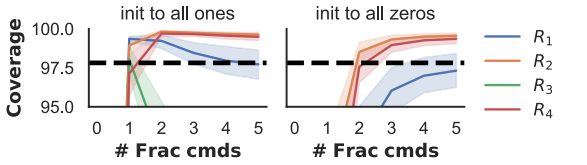
2) Evaluation of F-MAJ: In this section, we evaluate the coverage and stability of the F-MAJ operation which leverages the four-row-activation to perform logical majority-of-three. We use the word "coverage" when referring to how many bits in a row and how many chips in a DRAM group can perform F-MAJ, and the word "stability" when referring to the successful rate at which operation is performed on the same cell over multiple iterations.

Since F-MAJ requires storing a fractional value in one of the four rows, there are multiple variables to experiment with, such as the row to store the fractional value and the voltage level of the fractional value. We first explore the optimal configurations that generate the best coverage of F-MAJ. In this test, we examine all DRAM chips which are capable of performing four-row-activation. We fix $\{R_1, R_2, R_3, R_4\}$ to $\{1, 2, 0, 3\}$ for chips in group C and D, and to $\{8, 1, 0, 9\}$ for chips in group B. Group B supports

[1]SK Hynix DDR3 SDRAM with part number HMT351S6CFR8C-H9 and HMT451S6AFR8A-PB, group C and D in Table I



(a) Group B (b) Group C (c) Group D

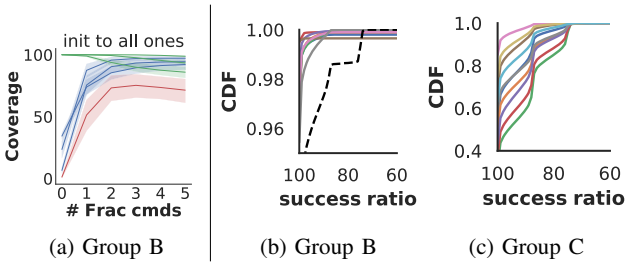


(d) Zoom in Group B

Figure 9: Coverage of the F-MAJ operation as a function of the number of *Frac* operations under different configurations for DRAM chips in groups B, C, and D; subfigures (a), (b), and (c), respectively. Subfigure (d) presents a more detailed version of the results for group B. The black dashed lines in subfigures (a) and (d) are the baseline MAJ3 operation.

the ComputeDRAM operations, thus if we activate row 1 and 2, three rows 0, 1, and 2 will be opened. We use the original MAJ3 operation built upon three-row-activation among the first three rows as the baseline design in group B to compare with. During this exploration, we vary the row that stores the fractional value from R_1 to R_4, and the voltage level of the "fractional value" all the way from 0 to V_{dd}. We can finely control the voltage level by adjusting the number of *Frac* operations we issue and the initial value in that critical row before we send *Frac*. All sub-arrays in every chip in group B, C, and D are tested. To verify the success of F-MAJ, we set the three operands six times as $\{1,0,0\},\{0,1,0\},\{0,0,1\},\{0,1,1\},\{1,0,1\},\{1,1,0\}$. A column is considered capable of performing F-MAJ only when it generates the correct majority result for all six sets of inputs.

Fig. 9 demonstrates the coverage of the F-MAJ operation in these three DRAM groups. Each line represents the percentage of bits that successfully performed F-MAJ when we use a different number of *Frac* operations to generate the fractional value. Different colors stand for using different rows to store fractional values. We plot the average value among the modules in that group and the shaded part indicates the 95% confidence interval. From Fig. 9, we can draw several conclusions:



(a) Group B (b) Group B (c) Group C

Figure 10: Subfigure (a) presents a detailed break down of F-MAJ coverage with different input combinations on group C. Green lines stand for input combinations with two ones and one zero; blue lines for two zeros and one one; red line shows the overall F-MAJ coverage. Focus on the condition of using R_1 to store fractional value with initial value of all ones. Subfigures (b) and (c) present the CDF of the reliability of F-MAJ operation across different modules in group B and C. The black dashed line in subfigure (b) presents the CDF for the MAJ3 baseline.

F-MAJ can be performed on all DRAM chips that are capable of opening four rows. A non-zero result for all the chips proves this point.

Different groups favor different configurations. For example, having a fractional value larger than $V_{dd}/2$ (meaning the initial value is all ones) in R_1 generates the best result for group C, while a fractional value smaller than $V_{dd}/2$ in R_4 is the favorite configuration for group D. Because of the "black box" nature of DRAM circuits, it is challenging for us to surmise the real cause behind the phenomenon. Going forward, we use the experimental results for choosing the best configuration for each group.

F-MAJ improves the coverage of the original MAJ3 operation. The best configuration of group B is storing a fractional value with two *Frac* operations starting with all ones in R_2, which has a 99.8% coverage of F-MAJ operation, while the original MAJ3 works on 98.0% of columns. This is because the original MAJ3 does not open the three rows at the exact same time, and they are not perfectly symmetric. Among the three opened rows, there is always a "primary" row which has more influence on the bit-line voltage compared with the other two. Thus, it has a larger error rate when the majority result is the opposite of the initial value of that primary row. However, for F-MAJ, we can put the fractional value in this primary row and adjust the voltage level of the fractional value. The insight is that **F-MAJ provides a chance for us to enhance the symmetry among the operands, therefore providing a wider coverage on the DRAM chip**.

To study how the number of *Frac* operations influences the coverage of F-MAJ in detail, we focus on one configuration and examine each individual input combination. Using group C with fractional values in R_1 and an initial value of all

ones as an example, we show the result in Figure 10a. Green lines represent the input combinations that generate a majority result of one, while blue lines represent the input combinations that generate a result of zero. The red line represents the combined F-MAJ success ratio, which is essentially the top line in Figure 9b. Without *Frac*, all ones are stored in R_1, thus more charge will be injected to the bit-line from R_1, and it is more likely to generate a result of one. We can verify this by the fact that the green lines in Figure 10a reach 100% without *Frac*, but the blue lines are pretty low under the same condition. As we issue more *Frac* operations, the voltage level in R_1 is reduced, and the majority result is more likely to be zero than one, which is confirmed by the fact that the blue lines are rising and green ones are declining. This is additional evidence showing the relationship between *Frac* and the voltage level in the cell.

At last, we test the stability of F-MAJ among group B and C. For each module, we randomly selected 500 different sub-arrays across all the banks, and perform F-MAJ 10000 times at each sub-array with random inputs. Within the sub-array, we use the same rows mentioned before and apply the best configuration that we found in the last experiment. The stability for a bit-line is the percent representing how many times it generated the correct majority result out of 10000 trials. The cumulative distributive function is plotted in Fig. 10c and Fig. 10b. Each line stands for a module in that DRAM group. Across the modules in group C, 33.0% to 85.2% of the columns can always generate the correct result. As for the modules in group B, at least 95.4% of the columns can reliably perform F-MAJ, beating the original MAJ3 operation which uses three-row-activation. The average error rate of in-memory majority thus can be reduced from 9.1% to 2.2%.

We conclude that F-MAJ not only expands in-memory majority operations to more modules – including potentially DDR4 modules, but also provides a better stability compared to the existing MAJ3 operation.

B. Physical Unclonable Function (PUF)

1) Proposal of Frac-based PUF: A Physical Unclonable Function (PUF) [27] is an intrinsic function attached to a physical object, which can be used for device authentication [28], [29], [30]. It takes a "challenge" as the input, and gives out a "response" based on the unique physical characteristics of the device. The uniqueness usually comes from manufacturing variations in its physical micro-structure, thus it is unpredictable and almost impossible to duplicate. An ideal PUF consistently generates the same response to the same challenge on the same device, while the responses to the same challenge generated by difference devices are distinct and random. This feature makes PUF a perfect candidate for generating fingerprints or signatures of digital devices. In addition, PUFs can be utilized to generate cryptographic keys [31], [32], and hardware obfuscation [33].

Due to the prevalence of DRAM memory across modern digital systems, a DRAM-based PUF is an attractive option. Moreover, the large address space in DRAMs naturally provides plenty of challenge-response pairs. Specifically, the address and size can be construed as the challenge, and the data read out will be the response. Previous DRAM-based PUFs have tried to leverage the variations in DRAM start-up values [34], DRAM cell retention failures [35], [36], [37], [38], and read-out data with reduced write duty-cycle [39] or other timing parameters like t_{RCD} [38] and t_{RP} [40]. However, past DRAM-based PUFs have several drawbacks such as long evaluation time, sensitivity to environmental changes (temperature or supply voltage), and requirement for a filter or error correction.

CODIC-based PUF [41] solved these issues using a modified DRAM substrate. Orosa *et al.* [41] proposed modifying control circuits inside the DRAM array to provide a new CODIC-sig command, which could drive the cell to $V_{dd}/2$. A subsequent read would enable the sense amplifier and thus "amplify" the cell voltage to either V_{dd} or 0, dependent on manufacturing variations. The read out data are both random and unique; thus they can be used as a PUF response. In addition, whether $V_{dd}/2$ is regarded as a zero or one is determined by the sense amplifier circuit, which is essentially a comparator. Thus, it is independent from the temperature and supply voltage. However, all of these benefits require hardware modifications to the existing DRAM circuits, which raises two hindrances to the realization of the technique with commercial DRAMs. First, DRAM manufacturers are reluctant to modify their designs due to the competitive and low-margin nature of the DRAM industry. Second, the introduction of a DRAM command requires modifications to the JEDEC specification. The CODIC authors did mention an alternative approach that works with off-the-shelf DRAMs. Specifically, they disabled DRAM refresh and waited for 48 hours to let the DRAM cell voltage leak towards $V_{dd}/2$ to emulate the effect of CODIC-sig. While this is an alternative that works with commercial DRAMs, it is too time-consuming to be considered for practical use.

With FracDRAM, we can use multiple *Frac* operations to achieve an effect similar to the CODIC-sig command, as the more *Frac* operations issued, the closer the cell voltage is to $V_{dd}/2$. According to our experiment results across nine memory groups from four major DRAM vendors, ten *Frac* operations are enough to generate a voltage close to $V_{dd}/2$ for PUF. A *Frac*-based PUF maintains all the merits of a CODIC-based PUF and can be implemented in off-the-shelf DRAMs without any hardware modification.

2) Evaluation of Frac-based PUF: In this section, we want to demonstrate that the proposed *Frac*-based PUF can generate a unique and reliable response to a challenge with real DRAM modules. We take the normalized *Hamming Distance*, which is commonly used in PUF research [31],

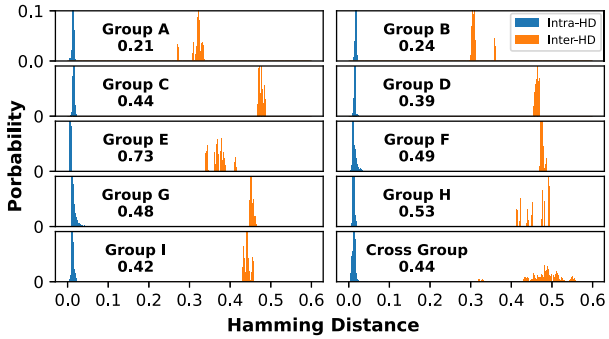


Figure 11: The distribution of Intra-HD and Inter-HD in group A to I, as well as cross-group. The number under group ID is the average *Hamming Weight* of the responses in that group.

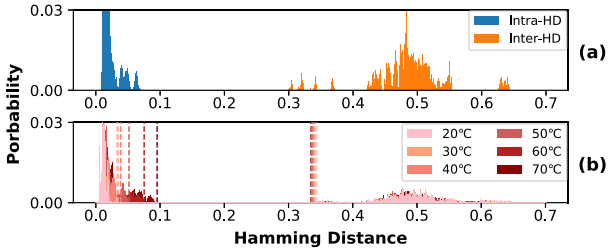


Figure 12: The distribution of Intra-HD and Inter-HD between responses with (a) different DRAM supply voltages and (b) different temperatures. The maximum intra-HD and minimum inter-HD are marked by the vertical dotted lines.

[34], [39], [40], [42], [43], [44], [45], [46], [47], [48], [49], as the metric to represent the difference between two responses. Specifically, that is the number of different bits in two responses divided by the total number of bits.

We use the notation of Intra-HD to indicate the *Hamming Distance* between two responses from the same DRAM module to the same challenge. Ideally, Intra-HD should be zero, meaning the PUF can reliably produce the same signature for the device. The notation of Inter-HD represents the *Hamming Distance* between two responses to the same challenge from different modules. Ideally, Inter-HD should be around 0.5 which is the distance between two independent random strings, meaning the PUF can produce unique responses.

In *Frac*-based PUF, the address and the size of the memory segment is used as a challenge, and the read out data from that segment is the response. We fixed the memory segment length to 8KB, which can fit in a single DRAM row. To get a challenge-response pair, we first choose an address (bank and row), and store all ones to that row as the initial value. Next we issue ten *Frac* operations to the target row to make the voltage close to $V_{dd}/2$. Finally, we read the entire row out. We tested at least two modules from each DRAM group, and all the groups capable of performing *Frac* are covered. For each module we collected 120 challenge-response pairs, and we send the same set of challenges to all the modules.

The distribution of Intra-HD and Inter-HD is shown in Figure 11. First, we found all the Intra-HDs exhibit an excellent property: they have a very narrow distribution and only concentrate around zero. The maximum Intra-HD ever seen is 0.051 from group G. Second, the Inter-HDs exhibit different patterns across different groups. For some groups like A, B, and E, the Inter-HD is clustered around 0.3 or 0.4, which shows some extent of similarity between the responses from different modules in the same group. The real cause of that is the responses from these groups have a *Hamming Weight* (i.e. the portion of the bits that hold one) away from 0.5. For example, in group A, only 21% of bits are one, which means most of the bits are zero, and that reduces the Inter-HD. *Hamming Weight* basically tells how many sense amplifiers regard $V_{dd}/2$ as one, which depends on the detailed circuit implementation of the bit-line and sense amplifier. Thus different DRAM groups produce different *Hamming Weight*. Although the randomness of the response is harmed, the uniqueness is still guaranteed: the minimum Inter-HD is 0.27, which is still way larger than the maximum Intra-HD. Notice that the *Hamming Weight* result is not aligned with the result in Figure 6. That's because we use different randomly selected rows in both tests.

Ten days from when we collected the first set of data, we generated the second set of responses to the same challenges again, but with a DRAM supply voltage of 1.4V (nominal supply voltage is 1.5V). We plot the Intra-HD and Inter-HD across these two data set in Figure 12(a). For the Inter-HD, we consider both within the same and between different DRAM groups. The highest Intra-HD is 0.07 and the lowest Inter-HD is 0.30. After an additional 3 months, we collected the third set of responses under different temperatures, and plot the intra-HD and inter-HD between those and the first data set collected three months ago under room temperature(20°C) in Figure 12(b). Although the average intra-HD increases as the temperature goes up, it does not change much, and most importantly, the maximum intra-HD remains far away from the minimum inter-HD. These results show that *Frac*-based PUF is robust againt environmental changes.

Other than the reliability and uniqueness, we also examined the randomness of the generated response with the random number test suite from NIST [50]. There are in total 15 different tests in the suite that check multiple characteristics including frequency, longest run, estimated entropy, template matching, etc. Because the raw response from the PUF is biased (the *Hamming Weight* is not 0.5), we use a modified Von Neumann randomness extractor to whitening the responses first and guarantee that the stream is balanced [41], [51]. Then, we concatenate the responses from different addresses

together. We use one million bits per module and feed that into the test suite. All 15 tests passed, which verifies the randomness of *Frac*-based PUF.

The total evaluation time for the *Frac*-based PUF will be: a preparation of 88 memory cycles (an initialization of all ones followed by 10 *Frac* operations) in addition to the readout time of the 8KB segment. In total, the evaluation time would be 1.5us (assuming a 2.5ns memory cycle), with the major time cost being the reading process. The evaluation time can be further reduced to 0.7us if an optimized memory controller is used so that the PUF response is read out at full speed.

C. Others

In addition to the cases we described above, the availability of a fractional value in DRAM can be utilized in other ways.

A possible use is to expand the storage capability of DRAM modules. Using the *Half-m* operation, we can store fractional value, one, or zero in arbitrary DRAM columns, which enables the cell to store three different states. Therefore, each cell represents a ternary bit. The overhead is that with *Half-m*, we need to write normal binary bits in four rows and issue a four-row-activation to generate one row of ternary bits. In addition, based on our evaluation the reading mechanism is not mature yet, since the way we have proposed to read out the fractional value requires four copies of the data (the MAJ3 method mentioned in Section IV-B), and the fractional value is destroyed after readout. We leave the readout and data recovery issue to future work; a possible solution would be a different sense amplifier design. Previous work [52] has proposed multi-level DRAM cells with "quantum dot gate" transistor. But only the layout of the multi-level DRAM cell is provided, and that's not fabricated yet.

Another use for fractional values is to assist the characterization of DRAM retention time. For example, focusing on a single cell, we can store different levels of fractional value and measure the retention time of each, thereby roughly tracing the voltage change during leakage. Different voltage levels can be generated in multiple ways: using a different number of *Frac* operations, changing the initial value in *Half-m*, and interrupting multiple-row-activation similar to *Half-m*.

Finally, it can be used in reverse-engineering DRAM designs and parameters, such as the sense amplifier threshold.

VII. Related Work

To the best of our knowledge, this is the first work that demonstrates fractional values being stored in off-the-shelf DRAMs. In this section, we present related work and identify its similarities and difference from FracDRAM.

Near-memory [1], [2], [3], in-memory [4], [5], [6], [7], [8], and with-memory-computing [9], [10], [11], [12] techniques are experiencing a "Renaissance" due to their potential to reduce the energy consumption from data transfer between storage and processing units. FracDRAM expands this area by identifying two novel techniques for storing fractional values in unmodified DRAM modules and evaluating two use cases for the new operations.

One of the key concepts in FracDRAM is the violation of timing specifications, which has been employed in previous works. The authors of [15], [16], [17], [53], [54] proposed pushing the limits of excessive process-margin in DRAM to enhance it's performance.

Moving a step further, recent work presented techniques that advance the concept of reducing the timing of DRAM commands outside of specification values. Specifically, timing is violated in such a way that new functionality is implemented by pushing the DRAM to undefined states. ComputeDRAM [14] was the first work that identified specially timed DRAM command sequencies that can perform with-memory computation in unmodified DRAM. ComputeDRAM utilized reduced t_{RAS} and t_{RP} to open three rows at the same time, and perform majority operation with the charge sharing among three rows. ComputeDRAM found that different number of rows can be opened with the proposed command sequence, however, only three-row-activation is leveraged to build majority operation on the only DRAM configuration that support it. FracDRAM expands the ideas of ComputeDRAM to enable fractional DRAM storage. The new capabilities extend ComputeDRAM-style, in-memory, majority operation to a larger set of DRAM module. We show how fractional DRAM storage can enhance the reliability of existing ComputeDRAM-style majority operation, reducing the error rate from 9.1% to 2.2%.

Operating outside of timing specifications means that correctness of reads and writes is not guaranteed. Previous works [38], [39], [40] leveraged the randomness and uniqueness of the erroneous readout data to build DRAM-based Physical Uncloneable Function (PUF). Orosa *et al.* [41] compared past DRAM-based PUFs and proved that CODIC-based PUF provides the state-of-the-art throughput and is more robust to environmental changes. Our proposed *Frac*-based PUF maintains all the merits of a CODIC-based PUF and is implementable with unmodified DRAM.

Kim *et al.* [55] proposed reducing the row activation latency to generate random numbers with high throughput and low latency. QUACTRNG [23] leveraged the command sequence in ComputeDRAM to open four rows simultaneously and explored different combinations of initial values in these four rows to generate random numbers using the charge sharing among them. Although we did not evaluate DDR4 modules due to the limitation of our experiment platform, the result of QUACTRNG proves four-row-activation can be performed in off-the-shelf DDR4 chips, which shows the potential of supporting F-MAJ and *Half-m* in DDR4 DRAMs.

Previous works [15], [16], [17], [18] have shown that voltage levels other than V_{dd} and 0 can be stored in DRAM cells. However, the concept of a "fractional value", which

is a new logical state, is first introduced in FracDRAM. For example, past work regards a 0.8Vdd voltage as a "weak" logical one, rather than a separate state of fractional value. In addition, the goal for which such voltage level was generated is different as well. Previous works aimed at reducing the DRAM access latency, whereas FracDRAM explicitly aims to generate a new logical state.

Finally, previous work has suggested modifications in hardware or operational values to store $V_{dd}/2$ in DRAM cells. DTRNG [56] proposed to use a lower supply voltage on word-line to store $V_{dd}/2$ in DRAM cell and issue a subsequent read to generate random numbers. CODIC [41] proposed exposing detailed DRAM control signals to users. One of the new operations they suggested is to set the cell voltage to $V_{dd}/2$, based on which they built a CODIC-based PUF. The rationale of the *Frac*-based PUF is also generating $V_{dd}/2$ in DRAM cells. However, their methodology of generating $V_{dd}/2$ is different from FracDRAM. Both of previous works are only verified in simulation, and can not work on unmodified DRAMs.

VIII. Conclusions

We have proposed and evaluated two novel primitive operations that store fractional values to the entire DRAM row, or to masked bits in a row. With these operations, we demonstrate for the first time the storing and destructive readout of fractional values in off-the-shelf, unmodified, DRAM chips. We show that with fractional value storage, in-memory majority operations can be expanded to more DRAM modules, and the error rate of the existing in-memory majority operations can be reduced from 9.1% to 2.2%. Furthermore, we presented a PUF based on fractional value storage, which has state-of-the-art throughput and can be implemented in commercial DRAM.

Acknowledgments

This material is based on research sponsored by the Air Force Research Laboratory (AFRL) and the Defense Advanced Research Projects Agency (DARPA) under agreement No. FA8650-18-2-7862. The U.S. Government is authorized to reproduce and distribute reprints for Governmental purposes notwithstanding any copyright notation thereon. The views and conclusions contained herein are those of the authors and should not be interpreted as necessarily representing the official policies or endorsements, either expressed or implied, of the Air Force Research Laboratory (AFRL), the Defense Advanced Research Projects Agency (DARPA), or the U.S. Government.

References

[1] D. Kim, J. Kung, S. Chai, S. Yalamanchili, and S. Mukhopadhyay, "Neurocube: a programmable digital neuromorphic architecture with high-density 3d memory," in *Computer Architecture (ISCA), 2016 ACM/IEEE 43rd Annual International Symposium on*. IEEE, 2016, pp. 380–392.

[2] J. Ahn, S. Yoo, O. Mutlu, and K. Choi, "Pim-enabled instructions: a low-overhead, locality-aware processing-in-memory architecture," in *Computer Architecture (ISCA), 2015 ACM/IEEE 42nd Annual International Symposium on*. IEEE, 2015, pp. 336–348.

[3] A. Gutierrez, M. Cieslak, B. Giridhar, R. G. Dreslinski, L. Ceze, and T. Mudge, "Integrated 3d-stacked server designs for increasing physical density of key-value stores," in *ACM SIGPLAN Notices*, vol. 49, no. 4. ACM, 2014, pp. 485–498.

[4] P. M. Kogge, "Execube - a new architecture for scaleable mpps," in *1994 International Conference on Parallel Processing Vol. 1*, vol. 1, Aug 1994, pp. 77–84.

[5] D. Patterson, T. Anderson, N. Cardwell, R. Fromm, K. Keeton, C. Kozyrakis, R. Thomas, and K. Yelick, "A case for intelligent ram," *IEEE Micro*, vol. 17, no. 2, pp. 34–44, March 1997.

[6] H. S. Stone, "A logic-in-memory computer," *IEEE Transactions on Computers*, vol. C-19, no. 1, pp. 73–78, Jan 1970.

[7] Q. Guo, X. Guo, R. Patel, E. Ipek, and E. G. Friedman, "AC-DIMM: associative computing with STT-MRAM," *SIGARCH Comput. Archit. News*, vol. 41, no. 3, pp. 189–200, Jun. 2013.

[8] S. Li, C. Xu, Q. Zou, J. Zhao, Y. Lu, and Y. Xie, "Pinatubo: a processing-in-memory architecture for bulk bitwise operations in emerging non-volatile memories," in *Proceedings of the 53rd Annual Design Automation Conference*. ACM, 2016, p. 173.

[9] S. Li, D. Niu, K. T. Malladi, H. Zheng, B. Brennan, and Y. Xie, "DRISA: A DRAM-based reconfigurable in-situ accelerator," in *Proceedings of the 50th Annual IEEE/ACM International Symposium on Microarchitecture*. ACM, 2017, pp. 288–301.

[10] A. Agrawal, A. Jaiswal, C. Lee, and K. Roy, "X-SRAM: enabling in-memory boolean computations in CMOS static random access memories," *IEEE Transactions on Circuits and Systems I: Regular Papers*, no. 99, pp. 1–14, 2018.

[11] S. Aga, S. Jeloka, A. Subramaniyan, S. Narayanasamy, D. Blaauw, and R. Das, "Compute caches," in *High Performance Computer Architecture (HPCA), 2017 IEEE International Symposium on*. IEEE, 2017, pp. 481–492.

[12] V. Seshadri, D. Lee, T. Mullins, H. Hassan, A. Boroumand, J. Kim, M. A. Kozuch, O. Mutlu, P. B. Gibbons, and T. C. Mowry, "Ambit: in-memory accelerator for bulk bitwise operations using commodity dram technology," in *Proceedings of the 50th Annual IEEE/ACM International Symposium on Microarchitecture*. ACM, 2017, pp. 273–287.

[13] V. Seshadri, Y. Kim, C. Fallin, D. Lee, R. Ausavarungnirun, G. Pekhimenko, Y. Luo, O. Mutlu, P. B. Gibbons, M. A. Kozuch, and T. C. Mowry, "Rowclone: fast and energy-efficient in-dram bulk data copy and initialization," in *2013 46th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, Dec 2013, pp. 185–197.

[14] F. Gao, G. Tziantzioulis, and D. Wentzlaff, "Computedram: In-memory compute using off-the-shelf drams," in *Proceedings of the 52nd Annual IEEE/ACM International Symposium on Microarchitecture*, ser. MICRO '52. New York, NY, USA: Association for Computing Machinery, 2019, p. 100–113. [Online]. Available: https://doi.org/10.1145/3352460.3358260

[15] D. Lee, Y. Kim, G. Pekhimenko, S. Khan, V. Seshadri, K. Chang, and O. Mutlu, "Adaptive-latency dram: Optimizing dram timing for the common-case," in *2015 IEEE 21st International Symposium on High Performance Computer Architecture (HPCA)*. IEEE, 2015, pp. 489–501.

[16] H. Hassan, G. Pekhimenko, N. Vijaykumar, V. Seshadri, D. Lee, O. Ergin, and O. Mutlu, "Chargecache: Reducing dram latency by exploiting row access locality," in *2016 IEEE International Symposium on High Performance Computer Architecture (HPCA)*. IEEE, 2016, pp. 581–593.

[17] Y. Wang, A. Tavakkol, L. Orosa, S. Ghose, N. M. Ghiasi, M. Patel, J. S. Kim, H. Hassan, M. Sadrosadati, and O. Mutlu, "Reducing dram latency via charge-level-aware look-ahead partial restoration," in *2018 51st Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*. IEEE, 2018, pp. 298–311.

[18] X. Zhang, Y. Zhang, B. R. Childers, and J. Yang, "Restore truncation for performance improvement in future dram systems," in *2016 IEEE International Symposium on High Performance Computer Architecture (HPCA)*. IEEE, 2016, pp. 543–554.

[19] B. Jacob, S. Ng, and D. Wang, *Memory systems: cache, DRAM, disk*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2007.

[20] B. Keeth, R. J. Baker, B. Johnson, and F. Lin, *DRAM Circuit Design: Fundamental and High-Speed Topics*, 2007.

[21] JEDEC, "Standard no 79-3f," *DDR3 SDRAM Specification*, 2012.

[22] J. Liu, B. Jaiyen, Y. Kim, C. Wilkerson, and O. Mutlu, "An experimental study of data retention behavior in modern dram devices: Implications for retention time profiling mechanisms," *ACM SIGARCH Computer Architecture News*, vol. 41, no. 3, pp. 60–71, 2013.

[23] A. Olgun, M. Patel, A. G. Yağlıkçı, H. Luo, J. S. Kim, F. N. Bostancı, N. Vijaykumar, O. Ergin, and O. Mutlu, "Quac-trng: High-throughput true random number generation using quadruple row activation in commodity dram chips," in *2021 ACM/IEEE 48th Annual International Symposium on Computer Architecture (ISCA)*. Los Alamitos, CA, USA: IEEE Computer Society, jun 2021, pp. 944–957. [Online]. Available: https://doi.ieeecomputersociety.org/10.1109/ISCA52012.2021.00078

[24] H. Hassan, N. Vijaykumar, S. Khan, S. Ghose, K. Chang, G. Pekhimenko, D. Lee, O. Ergin, and O. Mutlu, "Softmc: A flexible and practical open-source infrastructure for enabling experimental dram studies," in *2017 IEEE International Symposium on High Performance Computer Architecture (HPCA)*. IEEE, 2017, pp. 241–252.

[25] D. S. Yaney, C.-Y. Lu, R. A. Kohler, M. J. Kelly, and J. T. Nelson, "A meta-stable leakage phenomenon in dram charge storage-variable hold time," in *1987 International Electron Devices Meeting*. IEEE, 1987, pp. 336–339.

[26] M. K. Qureshi, D.-H. Kim, S. Khan, P. J. Nair, and O. Mutlu, "Avatar: A variable-retention-time (vrt) aware refresh for dram systems," in *2015 45th Annual IEEE/IFIP International Conference on Dependable Systems and Networks*. IEEE, 2015, pp. 427–437.

[27] B. Gassend, D. Clarke, M. Van Dijk, and S. Devadas, "Silicon physical random functions," in *Proceedings of the 9th ACM Conference on Computer and Communications Security*, 2002, pp. 148–160.

[28] W. Che, F. Saqib, and J. Plusquellic, "Puf-based authentication," in *2015 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*. IEEE, 2015, pp. 337–344.

[29] M. Majzoobi, M. Rostami, F. Koushanfar, D. S. Wallach, and S. Devadas, "Slender puf protocol: A lightweight, robust, and secure authentication by substring matching," in *2012 IEEE Symposium on Security and Privacy Workshops*. IEEE, 2012, pp. 33–44.

[30] M. Rostami, M. Majzoobi, F. Koushanfar, D. S. Wallach, and S. Devadas, "Robust and reverse-engineering resilient puf authentication and key-exchange by substring matching," *IEEE Transactions on Emerging Topics in Computing*, vol. 2, no. 1, pp. 37–49, 2014.

[31] R. Maes, A. Van Herrewege, and I. Verbauwhede, "Pufky: A fully functional puf-based cryptographic key generator," in *International Workshop on Cryptographic Hardware and Embedded Systems*. Springer, 2012, pp. 302–319.

[32] Z. Paral and S. Devadas, "Reliable and efficient puf-based key generation using pattern matching," in *2011 IEEE international symposium on hardware-oriented security and trust*. IEEE, 2011, pp. 128–133.

[33] J. B. Wendt and M. Potkonjak, "Hardware obfuscation using puf-based logic," in *2014 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*. IEEE, 2014, pp. 270–271.

[34] F. Tehranipoor, N. Karimian, W. Yan, and J. A. Chandy, "Dram-based intrinsic physically unclonable functions for system-level security and authentication," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 25, no. 3, pp. 1085–1097, 2016.

[35] C. Keller, F. Gürkaynak, H. Kaeslin, and N. Felber, "Dynamic memory-based physically unclonable function for the generation of unique identifiers and true random numbers," in *2014 IEEE international symposium on circuits and systems (ISCAS)*. IEEE, 2014, pp. 2740–2743.

[36] S. Sutar, A. Raha, D. Kulkarni, R. Shorey, J. Tew, and V. Raghunathan, "D-puf: An intrinsically reconfigurable dram puf for device authentication and random number generation," *ACM Transactions on Embedded Computing Systems (TECS)*, vol. 17, no. 1, pp. 1–31, 2017.

[37] W. Xiong, A. Schaller, N. A. Anagnostopoulos, M. U. Saleem, S. Gabmeyer, S. Katzenbeisser, and J. Szefer, "Run-time accessible dram pufs in commodity devices," in *International Conference on Cryptographic Hardware and Embedded Systems*. Springer, 2016, pp. 432–453.

[38] J. S. Kim, M. Patel, H. Hassan, and O. Mutlu, "The dram latency puf: Quickly evaluating physical unclonable functions by exploiting the latency-reliability tradeoff in modern commodity dram devices," in *2018 IEEE International Symposium on High Performance Computer Architecture (HPCA)*. IEEE, 2018, pp. 194–207.

[39] M. S. Hashemian, B. Singh, F. Wolff, D. Weyer, S. Clay, and C. Papachristou, "A robust authentication methodology using physically unclonable functions in dram arrays," in *2015 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE, 2015, pp. 647–652.

[40] B. B. Talukder, B. Ray, D. Forte, and M. T. Rahman, "Prelatpuf: Exploiting dram latency variations for generating robust device signatures," *IEEE Access*, vol. 7, pp. 81 106–81 120, 2019.

[41] L. Orosa, Y. Wang, M. Sadrosadati, J. S. Kim, M. Patel, I. Puddu, H. Luo, K. Razavi, J. Gómez-Luna, H. Hassan, N. Mansouri-Ghiasi, S. Ghose, and O. Mutlu, "Codic: A low-cost substrate for enabling custom in-dram functionalities and optimizations," in *2021 ACM/IEEE 48th Annual International Symposium on Computer Architecture (ISCA)*, 2021, pp. 484–497.

[42] S. S. Kumar, J. Guajardo, R. Maes, G.-J. Schrijen, and P. Tuyls, "The butterfly puf protecting ip on every fpga," in *2008 IEEE International Workshop on Hardware-Oriented Security and Trust*. IEEE, 2008, pp. 67–70.

[43] S. Sutar, A. Raha, and V. Raghunathan, "D-puf: An intrinsically reconfigurable dram puf for device authentication in embedded systems," in *2016 International Conference on Compliers, Architectures, and Sythesis of Embedded Systems (CASES)*. IEEE, 2016, pp. 1–10.

[44] I. Kumari, M.-K. Oh, Y. Kang, and D. Choi, "Rapid run-time dram puf based on bit-flip position for secure iot devices," in *2018 IEEE SENSORS*. IEEE, 2018, pp. 1–4.

[45] Q. Tang, C. Zhou, W. Choi, G. Kang, J. Park, K. K. Parhi, and C. H. Kim, "A dram based physical unclonable function capable of generating¿ 10 32 challenge response pairs per 1kbit array for secure chip authentication," in *2017 IEEE Custom Integrated Circuits Conference (CICC)*. IEEE, 2017, pp. 1–4.

[46] F. Tehranipoor, N. Karimian, W. Yan, and J. A. Chandy, "Investigation of dram pufs reliability under device accelerated

[49] S. Devadas, E. Suh, S. Paral, R. Sowell, T. Ziola, and V. Khandelwal, "Design and implementation of puf-based" unclonable" rfid ics for anti-counterfeiting and security applications," in *2008 IEEE international conference on RFID*. IEEE, 2008, pp. 58–64.

aging effects," in *2017 IEEE International Symposium on Circuits and Systems (ISCAS)*. IEEE, 2017, pp. 1–4.

[47] W. Liu, Z. Zhang, M. Li, and Z. Liu, "A trustworthy key generation prototype based on ddr3 puf for wireless sensor networks," *Sensors*, vol. 14, no. 7, pp. 11 542–11 556, 2014.

[48] A. Maiti, J. Casarona, L. McHale, and P. Schaumont, "A large scale characterization of ro-puf," in *2010 IEEE International Symposium on Hardware-Oriented Security and Trust (HOST)*. IEEE, 2010, pp. 94–99.

[50] A. Rukhin, J. Soto, J. Nechvatal, M. Smid, and E. Barker, "A statistical test suite for random and pseudorandom number generators for cryptographic applications," Booz-allen and hamilton inc mclean va, Tech. Rep., 2001.

[51] R. Shaltiel, "An introduction to randomness extractors," in *International colloquium on automata, languages, and programming*. Springer, 2011, pp. 21–41.

[52] S. Karmakar, "Design of multi-state dram using quantum dot gate non-volatile memory (qdnvm)," *Silicon*, vol. 11, no. 2, pp. 869–877, 2019.

[53] K. Chandrasekar, S. Goossens, C. Weis, M. Koedam, B. Akesson, N. Wehn, and K. Goossens, "Exploiting expendable process-margins in drams for run-time performance optimization," in *Proceedings of the Conference on Design, Automation & Test in Europe*, ser. DATE '14. European Design and Automation Association, 2014, pp. 173:1–173:6. [Online]. Available: http://dl.acm.org/citation.cfm?id=2616606.2616820

[54] K. K. Chang, A. Kashyap, H. Hassan, S. Ghose, K. Hsieh, D. Lee, T. Li, G. Pekhimenko, S. Khan, and O. Mutlu, "Understanding latency variation in modern dram chips: Experimental characterization, analysis, and optimization," in *Proceedings of the 2016 ACM SIGMETRICS International Conference on Measurement and Modeling of Computer Science*, 2016, pp. 323–336.

[55] J. S. Kim, M. Patel, H. Hassan, L. Orosa, and O. Mutlu, "D-range: Using commodity dram devices to generate true random numbers with low latency and high throughput," in *2019 IEEE International Symposium on High Performance Computer Architecture (HPCA)*. IEEE, 2019, pp. 582–595.

[56] K. Humood, B. Mohammad, and H. Abunahla, "Dtrng: Low cost and robust true random number generator using dram weak write scheme," in *2021 IEEE International Symposium on Circuits and Systems (ISCAS)*. IEEE, 2021, pp. 1–5.