How to Configure Row-Sampling-Based Rowhammer Defenses

Stefan Saroiu and Alec Wolman
Microsoft

***Abstract*—Row-Sampling is a simple, practical, and strong form of Rowhammer defense if properly configured. Unfortunately, previous papers either omit describing how to configure Row-Sampling, or when they do, their formulae suffer from limitations and unrealistic assumptions. This paper presents a mathematically rigorous description of how to configure Row-Sampling to protect an entire system for a given period of time.**

I. Introduction

Row-Sampling-based Rowhammer defenses are one of the oldest and simplest classes of defense techniques suitable for a memory controller. On each row activate, the memory controller flips a biased coin. With a low probability p ($p \ll 1$), the row address is *sampled* and the row is treated as if it is an aggressor row. The memory controller performs a mitigative action such as refreshing the corresponding victim rows. A sufficiently high sampling rate p thwarts a Rowhammer attack because it ensures that an aggressor row cannot escape being sampled with very high probability. Some of the earliest papers on Rowhammer introduced variants of sampling-based defenses under the names of "Probabilistic Adjacent Row Activation" (PARA)" [16] and "Probabilistic Row Activation" (PRA) [12].

Row-Sampling's main benefit lies in its simplicity: the memory controller need not maintain any state. This is in stark contrast with other forms of Rowhammer defenses suitable for memory controllers that need to store, lookup, and maintain large tables of rows that are tracked [6], [12], [30], [32], [23], [18], [19], [31], [21], [33], [35], [25], [11], [34], [14], remapped [15] or swapped [28].

These advantages make Row-Sampling very attractive to CPU vendors who are considering incorporating it into their memory controllers. In fact, older versions of Intel CPUs implemented a form of Row-Sampling to defend against Rowhammer targeting DDR3 DRAM, called pTRR [10]. Due to the initial impression of DDR4 being "Rowhammer-free" [20], [5], [17], Intel unfortunately dropped support for pTRR from their recent server SKUs. However, DDR4 has now been shown to still be vulnerable [13], [3], [27], [8] and recent work shows newer DRAM cells requiring fewer memory accesses until bits start to flip [24], [2], [29]. Given these trends, we expect a resurgence of interest in Row-Sampling-based techniques. An important question thus becomes: to what value should the sampling rate be set to provide an adequate level of defense? Answering this question must be done holistically for an entire system throughout its lifetime (rather than for a single bank or a single refresh window only), and must make realistic assumptions.

Despite the simplicity of Row-Sampling, previous work does not thoroughly answer this question. Some previous works characterize Row-Sampling's protection using simulations, and do not present enough detail to understand the formula used to derive the sampling rate [13]. Unfortunately, simulations are not a scalable way to provide guarantees for very rare events. For example, it is impractical to attempt to infer the value of p by observing Rowhammer failures that occur at a rate of $1e{-}10$ during simulations. Two previous papers do offer formulae for setting p [12], [25]. Unfortunately, the formulae are different from each other, make unrealistic assumptions about the Rowhammer remedy available to the memory controller, and assume a stronger threat model than what is realistic.

The absence of a rigorously-analyzed formula for setting p is dangerous. A mis-configured Row-Sampling implementation could leave a system vulnerable. Catching such mis-configurations by testing is difficult. With today's memory controllers, the only way to detect that an aggressor row has escaped sampling (i.e., a Row-Sampling mis-configuration) is by observing bit flips in DRAM.

This paper provides a mathematical analysis and a formula that let CPU vendors set p correctly in a Row-Sampling implementation. Our threat model is more realistic than previous models: the attacker knows the DRAM model, including items such as the t̊hreshold and the blast radius, as well as the value of the sampling probability p. Our model takes into account DRAM's auto-refresh behavior, but assumes that the attacker does not know which specific rows are refreshed during an auto-refresh command. This is important because, without knowing when a specific victim row is refreshed, the attacker's aggressor row must both escape sampling and avoid having its victim refreshed. Previous models did not take into account auto-refresh in modeling the behavior of Row-Sampling.

We implemented our formulae in Python using an arbitrary-precision library. With our code, a CPU vendor can determine the correct sampling rate value for a given level of protection. We present tables of values of p for DDR5 memory configurations that CPU vendors can use to guarantee *negligible* failure rates. Our code is open-sourced at https://github.com/microsoft/RHSampling.

II. DRAM Model

Most Rowhammer defenses assume a simple and uniform DRAM model. Upon an activation, each row creates disturbance in nearby rows. A victim row's degree of disturbance is affected only by its distance from the aggressor row. For example, an aggressor row R affects its two adjacent victim

rows $R \pm 1$ to the same degree. R affects $R \pm 2$ to a lesser degree than $R \pm 1$, $R \pm 3$ to even lesser than $R \pm 2$, and so on. The rate at which disturbance decreases with distance is referred to as *attenuation factor*. *Blast radius* is a term that indicates the distance between the aggressor row and its farthest victim. A DRAM module with a blast radius of 2 means that R disturbs four rows only: $R \pm 1$ and $R \pm 2$. No row outside of the blast radius is disturbed. Table I presents a summary of Rowhammer terminology.

Most Rowhammer defenses suitable for incorporation into a memory controller attempt to identify aggressor rows. Their goal is to never let an aggressor row receive more row activations than a fixed threshold, called a Rowhammer threshold (TH_{RH}), within a refresh interval (64ms in DDR4 and 32ms in DDR5). Once the number of row activations reaches TH_{RH}, a remedy is performed. The remedy is assumed to undo *all* disturbance created by the aggressor row.

Row-Sampling schemes configure their sampling rate p to provide a probabilistic guarantee, such as the likelihood of any row receiving TH_{RH} or more row activations is negligibly low. It is not very clear what a negligibly low failure probability should be, although some recent papers suggest $1e{-}15$ per hour of continuous hammering [13].

Earlier Row-Sampling schemes [16], [12] assumed that the remedy used by the memory controller is to activate victim rows. Unfortunately, internal DRAM row topology remains a closely-guarded secret by DRAM vendors [29]. This leaves the memory controller unable to identify those victim rows affected by a specific aggressor row. The two earliest Row-Sampling schemes incorporate this unrealistic form of remedy in their names: the last 'A' in both PARA [16] and PRA [12] stands for row Activation.

As an alternative, researchers have proposed the addition of a new DRAM command called Nearby Row Refresh (NRR) [19], [25]. When it detects an aggressor row, the memory controller issues NRR to report the aggressor's row address to the DRAM device which then allows the device to refresh the relevant victim rows. The rest of the paper assumes NRR as the remedy used by the memory controller. We expect NRR will be incorporated in upcoming DDR5 DRAM specifications and supported by DRAM vendors (see Errata).

Limitations: In practice, DRAM does not behave like this simplistic and uniform model suggests. Some rows require fewer row activations to induce bit flips than others. This corresponds to a need to have per-row TH_{RH} thresholds rather than a single value constant across all DRAM in the system. Also, DRAM disturbance is not uniform: some cells are more likely to be disturbed than others even if their distance to an aggressor row is the same. Finally, the blast radius varies depending on the aggressor row; certain rows have a wider blast radius than others.

Despite these limitations, the DRAM model remains useful because it lets us analyze a Rowhammer defense with mathematical rigor under a set of assumptions. To account for the differences in TH_{RH} values, DRAM disturbance effects, or blast radii, a practical Row-Sampling-based Rowhammer defense can set its global parameters to the most conservative values as appropriate. For example, the system-wide TH_{RH} should be set to the minimal value of all rows' and all cells' TH_{RH}. Similarly, the global blast radius parameter should be the maximum blast radius of any DRAM row in the system.

Single-sided attack	A Rowhammer attack where one aggressor row is activated repeatedly with the goal of inducing bit flips on adjacent (or nearby) rows in a bank.
Double-sided attack	A Rowhammer attack where two aggressor rows are located one row apart. The row between the two aggressors is a victim row.
Rowhammer threshold	The maximum number of activations a row can sustain in a single refresh window until a Rowhammer mitigation action must be performed.
Blast radius	The physical distance (i.e., the number of rows apart) between an aggressor and a victim row. A blast radius of 1 corresponds to the case when the aggressor and victim rows are adjacent. Distant rows correspond to a blast radius greater than 1.
Attenuation factor	A factor representing the reduction of disturbance errors as the blast radius increases. This factor is assumed to directly correlate with the increase in the number of activations an aggressor row requires to flip bits in victim rows located farther away. For example, an attenuation factor of 10 means that a victim row requires 10 times more activates to flip bits in a victim row located two rows away than an adjacent victim.
Neighbor row refresh (NRR)	New DRAM command that takes as input the address of an aggressor row. Upon detecting an aggressor row, the memory controller issues an NRR to the DRAM. In turn, the DRAM refreshes all victim rows in the blast radius.
Sampling rate p ($p \ll 1$)	Probability of sampling a row activation and treating the row as if it is an aggressor.

TABLE I
SUMMARY OF DRAM MODEL AND TERMINOLOGY.

III. THREAT MODEL

To analyze Row-Sampling schemes, we assume a worst-case, but realistic threat model. The attacker knows the DRAM model and the implementation of the Row-Sampling scheme including the value of p. The attacker is free to activate any row in any order but without violating the DRAM bus timings and correctness. The DRAM is configured to run at a normal refresh rate: the memory controller issues 8192 refresh commands to a rank every refresh window of 64ms for DDR4 and 32ms for DDR5. Such assumptions correspond to a scenario in which an attacker can run arbitrary code on the host system but cannot modify the hardware, the firmware, or the BIOS/UEFI settings. Prior work [1] has described instruction sequences carefully constructed to bypass all CPU caches and *hammer* memory at the highest rates allowed by DRAM (i.e., at a rate of one row activate every t_{RC}).

The worst-case attack strategy activates the same row continuously (i.e., a single-sided attack). An attack strategy that activates another row is sub-optimal for two reasons. First, activating the other row does not bring the attack on the first row any closer to success. Second, a successful attack that

hammers multiple rows remains successful (and even increases its chances of success) if the attacker hammered only one row.

Prior work assumed a stronger but less realistic threat model. Prior work [12], [25], [13] assumed a single necessary (but insufficient) condition for an attack to be successful: within a refresh window, one row must be activated more than TH_{RH} times without being sampled. This condition is insufficient because it assumes the absence of background auto-refresh. A series of back-to-back row activates lasting longer than TH_{RH} is *not* a successful attack if the victim row is auto-refreshed in the meantime.

To the best of our knowledge, an attacker does not have a way to infer or learn when a *specific* victim row is auto-refreshed. Thus, the attacker cannot avoid the case when the "lucky" series of back-to-back unsampled row activations still fails to flip bits because it overlaps with the auto-refresh of the victim row. Thus, our threat model is weaker than that assumed by prior work, yet more realistic.

The threat model affects the whole system throughout its entire lifetime. Prior work [12], [25], [13] describes the configuration of a Row-Sampling-based Rowhammer defense for a single bank and/or within a single refresh window. However, a more realistic scenario must assume a determined attacker who could launch a Rowhammer attack on all banks *in parallel* and can hammer continuously for long periods of time. This degree of parallelism and the attack's lifetime must be taken into account when configuring the Row-Sampling-based Rowhammer defense.

IV. Limitations of Prior Work

To the best of our knowledge, no prior work has done an in-depth correct analysis of how to configure an Row-Sampling-based Rowhammer defense. One prior source of inaccuracy is the less realistic threat model described in Section III: prior works fail to take into account the existence of background auto-refresh. No prior works incorporate in their analysis a degree of parallelism (i.e., an attack affecting all system's banks in parallel). Most, but not all [13], provide an analysis based on discretizing time in intervals corresponding to refresh intervals. This approach implicitly assumes that attacks can only start and end within the same interval and fails to consider attacks spanning multiple intervals. Finally, the analysis and formulae of prior works suffer from varying degrees of imprecision and errors as detailed below. Table II summarizes the limitations of prior work.

A recent prior work performed simulations to characterize the performance of a Probabilistic Row Adjacency Activation (PARA) to protect newer DRAM [13]. Unfortunately, the paper does not contain sufficient detail to understand how the sampling rate p was set other than "[we set p] such that the bit error rate (BER) does not exceed $1\text{e}{-15}$ per hour of continuous hammering". The simulations report the DRAM bandwidth overhead as a percentage for various Rowhammer defense schemes including PARA. Initially, we thought this overhead corresponds to the sampling rate and can be simply inferred from the graphs. Our intuition stemmed from the observation that PARA's overhead scales linearly with p *irrespective* of the workload. Thus a sampling rate of 1% would roughly correspond to a DRAM bandwidth overhead of 1%.

	Unrealistic threat model	Lack of parallelism	Long lasting attack	Correct formula & analysis
Revisiting [13]	✖	✖	✔[1]	✖
PRA [12]	✖	✖	✖	✖
Graphene [25]	✖	✖	✖	✔[2]

TABLE II
SUMMARY OF PRIOR WORK'S LIMITATIONS. REVISITING [13] CONSIDERED LONG-LASTING ATTACKS UP TO 1 HOUR (✔[1]). GRAPHENE [25]'S FORMULA SUFFERS FROM A OFF-BY-ONE ERROR (✔[2]).

Unfortunately, it appears this is not the case. Some of the outliers shown by the simulations report an overhead higher than 100% (for example, see the bottom whisker of the point with an x-axis value of 150 in Figure 10a [13]). This cannot correspond to a sampling rate value because p cannot be higher than 1.

One of the original papers [12] on Row-Sampling includes the derivation of a closed-form formula on the probability of Rowhammer failure for a given p, TH_{RH}, and a term called "rounds" (abbreviated by k). Rounds refers to the time needed for an attacker to issue TH_{RH} row activations back-to-back, and does not correspond to a refresh window. The closed form formula is:

$$P_{\text{failure}} = 1 - (1 - e^{-p \times \text{TH}_{\text{RH}}})^k$$

Unfortunately, this formula underestimates the sampling rate value. Its derivation rests on the assumption that each of the rounds are independent and no disturbance carries from one round to the next (assumption made by Equation 3.11 in [12]). For example, an attack that issues half its row activations at the very end of a round, and the other half at the very beginning has a high chance of escaping sampling. We confirmed with the authors that their assumption is not realistic.

The Graphene paper [25] also lists a brief analysis and a recurrence formula for computing the sampling rate of PARA:

$$P(e_N) = P(e_{N-1}) + p(1 - \frac{1}{2}p)^{\text{TH}_{\text{RH}}}(1 - P(e_{N-\text{TH}_{\text{RH}}-1}))$$
$$P(e_N) = 0 \text{ when } N < \text{TH}_{\text{RH}}$$

where e_N refers to a failure event (i.e., at least TH_{RH} consecutive row activations) in a series of N row activations.

The ending clause of the recurrence is trivial: if the number of row activations is lower than TH_{RH} the probability of a Rowhammer failure is zero. The recurrence formula itself is based on a simple, but insightful observation. There are only two possibilities to encounter a failure event in a series of N row activations: either the failure event occurred sometime during the first $N-1$ row activations or the failure occurred right on the N-th activate. This second possibility occurs only if the last $\text{TH}_{\text{RH}} + 1$-th row activations follow the pattern:

PATTERN=*sampled-unsampled-unsampled-...-unsampled*

Two conditions must be met to ensure the failure occurs right on the N-th activate and not earlier: 1) the first row activation in the pattern above must be sampled, and 2) no failures occur before the $\text{TH}_{\text{RH}} + 1$-th row activation. The

p	sampling rate
TH_{RH}	Rowhammer threshold
b	# of banks in the system
t_{RC}	row cycle time (45ns in DDR4/5)
t_{REFW}	refresh window (32ms in DDR5; 64ms in DDR4)
$\text{ACT}_{\text{TOTAL}}$	the maximum # of row activations to a bank throughout the attack's lifetime
t_{RFC}	REF command duration (used to calculate $\text{ACT}_{\text{TOTAL}}$)
$P(e_N)$	prob. of a sequence of TH_{RH} back-to-back unsampled row activations in a sequence of N row activations
$P(v_{\text{TH}_{\text{RH}}})$	prob. of a victim row **not** being refreshed during a sequence of TH_{RH} row activations
P_{failure}	prob. of a Rowhammer failure in a system

TABLE III
FORMULA PARAMETERS.

probability for both these two conditions is: $\frac{p}{2} \times (1-\frac{p}{2})^{\text{TH}_{\text{RH}}} \times (1 - P(e_{N-\text{TH}_{\text{RH}}-1}))$.

The presence of the $\frac{1}{2}$ factor comes from the formula being derived *specifically* for PARA. In PARA, the memory controller uses an unrealistic form of mitigation: activating one row adjacent to the victim. Since PARA chooses one of the adjacent victims at random, the probability of a mitigating one *specific* victim row is $\frac{p}{2}$ and, conversely, the probability of the victim row remaining un-mitigated is $1-\frac{1}{2}p$ explaining the presence of the $\frac{1}{2}$ factor. The presence of two victim rows (Graphene assumes blast radius to be 1 for this formula) doubles this probability giving us the second term in the recurrence: $p(1-\frac{1}{2}p)^{\text{TH}_{\text{RH}}}(1 - P(e_{N-\text{TH}_{\text{RH}}-1}))$.

Since the memory controller lacks knowledge about the internal DRAM topology [29], it cannot determine the address of a nearby row. Instead, we assume (and anticipate) the existence of an NRR command that refreshes all victim rows within an aggressor row's blast radius. Assuming NRR, the formula can be rectified to:

$$P(e_N) = P(e_{N-1}) + p(1-p)^{\text{TH}_{\text{RH}}}(1 - P(e_{N-\text{TH}_{\text{RH}}-1}))$$
$$P(e_N) = 0 \text{ when } N < \text{TH}_{\text{RH}}$$

Unfortunately, this still suffers from an off-by-one error in the recurrence. When the number of row activations N equals TH_{RH}, the probability of failure $P(e_N)$ (i.e., no activation is sampled) is $(1-p)^{\text{TH}_{\text{RH}}}$. Instead, the formula returns $p(1-p)^{\text{TH}_{\text{RH}}}$. We will address this last inaccuracy and provide a complete formula in the next section.

V. Correct Formula and Configuration

To arrive at the correct formula and configuration, we perform the following three steps to the earlier rectified formula from Graphene [25]:

1. Fixing the recurrence's termination condition. The recurrence needs an additional termination condition for the case when $N = \text{TH}_{\text{RH}}$. In this case, the probability of N unsampled row activations is simply $(1-p)^{\text{TH}_{\text{RH}}}$.

$$P(e_N) = P(e_{N-1}) + p(1-p)^{\text{TH}_{\text{RH}}}(1 - P(e_{N-\text{TH}_{\text{RH}}-1})) \quad (1)$$
$$P(e_N) = (1-p)^{\text{TH}_{\text{RH}}} \text{ when } N = \text{TH}_{\text{RH}} \quad (2)$$
$$P(e_N) = 0 \text{ when } N < \text{TH}_{\text{RH}} \quad (3)$$

2. Assuming a realistic threat model in which the attacker does not know when a victim row is being auto-refreshed. As described in Section III a Rowhammer failure requires two conditions to hold true: (1) TH_{RH} unsampled row activations and (2) the absence of the victim being auto-refreshed. We introduce $P(v_{\text{TH}_{\text{RH}}})$, the probability of a victim row **not** being refreshed during a sequence of TH_{RH} row activations. This probability is proportional to the ratio of two time intervals: (1) the portion of a refresh window that falls outside the TH_{RH} back-to-back row activations and (2) the refresh window.

$$P(v_{\text{TH}_{\text{RH}}}) = \frac{t_{REFW} - t_{RC} \times \text{TH}_{\text{RH}}}{t_{REFW}} \quad (4)$$

Since the two conditions are independent, the probability of failure is simply the product of $P(e_N)$ and $P(v_{\text{TH}_{\text{RH}}})$.

3. Considering all banks in the system and the attack duration. The correct formula for the probability of a Rowhammer failure must consider the whole system (i.e., not just a single bank) and the attack duration. We thus introduce two additional parameters: (1) b, the total number of banks in the entire system that can be under a Rowhammer attack simultaneously and (2) $\text{ACT}_{\text{TOTAL}}$, the total maximum number of row activations to a bank throughout the attack duration.

Once we derive the probability of failure of a single bank, we can scale it up to the whole system based on the following observation: the system will experience no failures if *none* of the banks fail. Thus, if the chance of a failure in a single bank is $P_{\text{one-bank-failure}}$, the chance of the whole system experiencing a failure is $1-(1-P_{\text{one-bank-failure}})^b$.

Table III summarizes the parameters used by our formula. The final formula is:

$$\boxed{P_{\text{failure}} = 1 - [1 - P(e_{\text{ACT}_{\text{TOTAL}}}) \times P(v_{\text{TH}_{\text{RH}}})]^b}$$

where: $P(e_{\text{ACT}_{\text{TOTAL}}})$ is derived from equations (1)–(3) and $P(v_{\text{TH}_{\text{RH}}})$ from equation (4).

Blast radius: Although our formula does not appear to incorporate a blast radius value explicitly, configuring TH_{RH} must take into account the blast radius (see Errata). For more on how to set TH_{RH}, see [25], [29]. In addition, our analysis assumes that the NRR command refreshes *all* rows within the blast radius.

VI. Configuration Examples

This section presents the Rowhammer failure rates for different sampling rates for two different hardware configurations. Configuration A corresponds to one server similar to those found in a datacenter environment: a dual-socket with 8 DDR5 channels per socket, 2 DIMMs per channel (DPC), and dual-rank DIMMs. Configuration B corresponds to a fleet of 100,000 such servers. Table IV summarizes these two configurations.

We implemented our formula in Python 3 using the *decimal* module [26] that provides support for fast, correctly-rounded decimal floating point arithmetic. We expose the level of precision as an input parameter and warn the user when the answer is being rounded by the library. When the answer is rounded, the script outputs a warning that instructs the user to re-run the script with increased precision.

We also derived and implemented the dual of our formula. Our formula recursively enumerates all possible cases of a

	Servers	Sockets	Channels per Socket	DPC	Ranks	Banks per Rank	Banks in Total
Cfg. A	1	2	8	2	2	32	2048
Cfg. B	100,000	2	8	2	2	32	200,480,000

TABLE IV
TWO CONFIGURATIONS: SINGLE SERVER VERSUS SERVER FLEET.

Rowhammer failure. The dual computes the probability of no Rowhammer failure by counting all possible cases when at least one of the TH_{RH} row activations is sampled. The dual formula is a more complex recursion because enumerating all possibilities of no Rowhammer failures requires more cases.

We omit describing the derivation of the dual of the formula. We performed a series of tests of our Python code to ensure that both formulae provide the same answer. The code computing the dual formula is significantly slower due to the complexity of the formula.

Tables V and VI list the probability of a Rowhammer failure for different sampling rates for configurations A and B. For the sampling rates, we used reciprocals of a power of two (i.e., 1 in 32, 1 in 64) because we anticipate such sampling rates to be easy to implement in a memory controller. However, our formula and code can work with any sampling rate values. We also used low TH_{RH} values corresponding to the recent trends that show newer DRAM cells requiring fewer memory accesses until bits start to flip [24], [2], [29].

These tables illustrate how sampling rates must be adjusted depending on the hardware configurations they aim to protect. For example, a sampling rate of 1 in 256 leads to a low rate of Rowhammer failures (7e−6) on a single server with a TH_{RH} value of 8,192. However, the same sampling rate is simply unacceptable to protect against a Rowhammer attack targeting an entire server fleet because the attack's success rate is 48.1%.

VII. DISCUSSION

Our analysis assumes the attacker cannot control or induce refresh postponement [9]. To date, we are unaware of software-based attacks that enable control over the refresh schedule of commodity memory controllers, but we cannot preclude this possibility. Although refresh postponement does not affect the recurrence formula shown in equations (1)–(3), it could reduce the likelihood of a victim row being refreshed as shown in equation (4). In the worst-case, an attacker with control over the refresh schedule could reduce the number of refresh commands overlapping with the attack.

Our threat model assumes the attack can be massively parallelized on all banks in a system (or in a fleet). This assumption is not realistic due to restrictions on parallelism imposed by DDR bus timings. For example, t_{RRD} restricts the rate of row activations to different banks within a bank group or within a rank. Similarly, t_{FAW} is a timing window that limits the number of row activations to a single rank to four. Unfortunately, incorporating these timings constraints into equation (4) is not trivial.

One concern is that the additional refreshes caused by NRR could induce additional disturbance not accounted by Row-Sampling. The memory controller has no opportunity to sample the additional row activations done by NRR. These attacks are known as *half-double* [4] or *transitive* [29]. Unfortunately, this NRR limitation is fundamental and can only be addressed by knowing the internal DRAM topology [29]. This concern cannot be mitigated by adjusting the sampling rate.

		Sampling Rate (p)				
		1 in 512	1 in 256	1 in 128	1 in 64	1 in 32
TH_{RH}	**8192**	99.9%	7e−6	1e−19	2e−47	5e−104
	4096	99.9%	99.9%	1e−5	2e−19	1e−47
	2048	99.9%	99.9%	99.9%	2e−5	3e−19
	1024	99.9%	99.9%	99.9%	99.9%	3e−5

TABLE V
$P_{FAILURE}$ IN CONFIGURATION A (SINGLE SERVER) FOR DIFFERENT SAMPLING RATES GIVEN A ONE HOUR-LONG ATTACK.

		Sampling Rate (p)				
		1 in 512	1 in 256	1 in 128	1 in 64	1 in 32
TH_{RH}	**8192**	99.9%	48.1%	1e−14	2e−42	5e−99
	4096	99.9%	99.9%	71.0%	2e−14	1e−42
	2048	99.9%	99.9%	99.9%	88.8%	3e−14
	1024	99.9%	99.9%	99.9%	99.9%	96.6%

TABLE VI
$P_{FAILURE}$ IN CONFIGURATION B (SERVER FLEET) FOR DIFFERENT SAMPLING RATES GIVEN A ONE HOUR-LONG ATTACK.

Another concern is the memory controller's ability to generate true random numbers on each row activation. In practice, the true random generator can periodically, but frequently, seed a pseudo-random number generator.

Many DRAM devices have in-DRAM Rowhammer defenses, such as TRR [22], [3]. Unfortunately, TRR defenses are both proprietary (i.e., relying on security by obscurity) and incomplete [13], [3], [7], [8]. These shortcomings are making CPU, cloud, and mobile vendors consider deploying their own Rowhammer defenses in memory controllers or in software. These defenses overlap partially with TRR leading to duplicate victim refreshes for some forms of Rowhammer attacks. As long as DRAM's defenses remain secret, it is difficult to incorporate them into our model.

VIII. CONCLUSIONS

This paper describes how to configure Row-Sampling, a memory controller-based Rowhammer defense. Row-Sampling is an attractive defense technique because it is simple to implement, effective, and can provide robust protection when properly configured.

Our goal is to describe a rigorous analysis of how to configure a Row-Sampling implementation. We present a DRAM model in an effort to reduce the ambiguity and increase the clarity of the assumptions made by our analysis. We then describe a more realistic threat model than those used by prior work. We identify and rectify the errors in earlier formulae. We expand these formulae in the earlier work to arrive at a final formula that incorporates our threat model. Finally, we present the correct parameters for a Row-Sampling-based Rowhammer defense deployed in a single server and a fleet of 100K servers.

We encourage CPU and DRAM vendors to use the formula and code to properly derive (or even sanity check) the configuration parameters used by their Row-Sampling-based Rowhammer defenses.

Acknowledgments. We would like to thank the anonymous reviewers for their thorough feedback.

REFERENCES

[1] L. Cojocar, J. Kim, M. Patel, L. Tsai, S. Saroiu, A. Wolman, and O. Mutlu, "Are We Susceptible to Rowhammer? An End-to-End Methodology for Cloud Providers," in *IEEE S&P*, 2020.

[2] L. Cojocar, K. Loughlin, S. Saroiu, B. Kasikci, and A. Wolman, "mFIT: A Bump-in-the-Wire Tool for Plug-and-Play Analysis of Rowhammer Susceptibility Factors," *Technical Report – Microsoft Research*, vol. MSR-TR-2021-25, 2021.

[3] P. Frigo, E. Vannacci, H. Hassan, V. van der Veen, O. Mutlu, C. Giuffrida, H. Bos, and K. Razavi, "TRRespass: Exploiting the Many Sides of Target Row Refresh," in *S&P*, 2020.

[4] Google, "Introducing Half-Double: New hammering technique for DRAM Rowhammer bug," https://security.googleblog.com/2021/05/introducing-half-double-new-hammering.html, 2021.

[5] M. Greenberg, "Row Hammering: What it is, and how hackers could use it to gain access to your system," https://blogs.synopsys.com/committedtomemory/2015/03/09/row-hammering-what-it-is-and-how-hackers-could-use-it-to-gain-access-to-your-system/, 2015.

[6] Z. Greenfield, J. B. Halbert, and K. S. Bains, "Method, apparatus and system for determining a count of accesses to a row of memory," Patent No. US 2014/0085995, 2014.

[7] H. Hassan, Y. C. Tugrul, J. Kim, V. van der Veen, K. Razavi, and O. Mutlu, "Uncovering In-DRAM RowHammer Protection Mechanisms: A New Methodology, Custom RowHammer Patterns, and Implications," in *MICRO*, 2021.

[8] P. Jattke, V. Veen, P. Frigo, S. Gunter, and K. Razavi, "Blacksmith: Scalable Rowhammering in the Frequency Domain," in *IEEE S&P*, 2022.

[9] JEDEC, *Double Data Rate 5 (DDR5) SDRAM Standard*, 2020.

[10] M. Kaczmarski, "Thoughts on Intel Xeon E5-2600 v2 Product Performance Optimisation," 2014.

[11] O. Kang, E. Lee, and J. H. Ahn, "CAT-TWO: Counter-Based Adaptive Tree, Time Window Optimized for DRAM Row-Hammer Prevention," *IEEE Access*, vol. 8, pp. 17 366–17 377, 2020.

[12] D.-H. Kim, P. J. Nair, and M. K. Qureshi, "Architectural Support for Mitigating Row Hammering in DRAM Memories," *CAL*, 2015.

[13] J. Kim, M. Patel, A. G. Yaglikci, H. Hassan, R. Azizi, L. Orosa, and O. Mutlu, "Revisiting RowHammer: An Experimental Analysis of Modern DRAM Devices and Mitigation Techniques," in *ISCA*, 2020.

[14] M. J. Kim, J. Park, Y. Park, W. Doh, N. Kim, T. J. Ham, J. W. Less, and J. H. Ahn, "Mithril: Cooperative Row Hammer Protection on Commodity DRAM Leveraging Managed Refresh," in *HPCA*, 2022.

[15] M. Kim, J. Choi, H. Kim, and H.-J. Lee, "An Effective DRAM Address Remapping for Mitigating Rowhammer Errors," in *TC*, 2019.

[16] Y. Kim, R. Daly, J. Kim, C. Fallin, J. H. Lee, D. Lee, C. Wilkerson, K. Lai, and O. Mutlu, "Flipping Bits in Memory Without Accessing Them: An Experimental Study of DRAM Disturbance Errors," in *ISCA*, 2014.

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

[18] E. Lee, S. Lee, G. E. Suh, and J. H. Ahn, "TWiCe: Time Window Counter Based Row Refresh to Prevent Row-Hammering," *CAL*, 2018.

[19] ——, "TWiCe: Preventing Row-hammering by Exploiting Time Window Counters," in *ISCA*, 2019.

[20] J.-B. Lee, "Green Memory Solution," http://aod.teletogether.com/sec/20140519/SAMSUNG_Investors_Forum_2014_session_1.pdf, 2014.

[21] C. Li and J.-L. Gaudiot, "Detecting Malicious Attacks Exploiting Hardware Vulnerabilities Using Performance Counters," in *COMPSAC*, 2019.

[22] J. Lin and M. Garrett, "Handling Maximum Activation Count Limit and Target Row Refresh in DDR4 SDRAM," Patent No. US 2015/0200002 A1, 2015.

[23] O. Mutlu, "The RowHammer Problem and Other Issues We May Face as Memory Becomes Denser," in *DATE*, 2017.

[24] L. Orosa, A. G. Yaglıkçı, J. Luo, A. Olgun, J. Park, H. Hassan, M. Patel, J. S. Kim, and O. Mutlu, "A Deeper Look into RowHammerâĂŹs Sensitivities: Experimental Analysis of Real DRAM Chips and Implications on Future Attacks and Defenses," in *MICRO*, 2021.

[25] Y. Park, W. Kwon, E. Lee, T. J. Han, J. H. Ahn, and J. W. Lee, "Graphene: Strong yet Lightweight Row Hammer Protection," in *MICRO*, 2020.

[26] Python, "decimal - Decimal fixed point and floating point arithmetic," https://docs.python.org/3/library/decimal.html, 2022.

[27] F. Ridder, P. Frigo, E. Vannacci, H. Bos, C. Giuffrida, and K. Razavi, "SMASH: Synchronized Many-sided Rowhammer Attacks from JavaScript," in *USENIX Security*, 2021.

[28] G. Saleshwar, B. Wang, M. Qureshi, and P. J. Nair, "Randomized Row-Swap: Mitigating Row Hammer by Breaking Spatial Correlation between Aggressor and Victim Rows," in *ASPLOS*, 2022.

[29] S. Saroiu, A. Wolman, and L. Cojocar, "The Price of Secrecy: How Hiding Internal DRAM Topologies Hurts Rowhammer Defenses," in *IPRS*, 2022.

[30] S. M. Seyedzadeh, A. K. Jones, and R. Melhem, "Counter-based Tree Structure for Row Hammering Mitigation in DRAM," *CAL*, 2017.

[31] ——, "Mitigating Wordline Crosstalk Using Adaptive Trees of Counters," in *ISCA*, 2018.

[32] M. Son, H. Park, J. Ahn, and S. Yoo, "Making DRAM Stronger Against Row Hammering," in *DAC*, 2017.

[33] Y. Wang, Y. Liu, P. Wu, and Z. Zhang, "Detect DRAM Disturbance Error by Using Disturbance Bin Counters," in *CAL*, 2019.

[34] A. G. Yaglıkçı, M. Patel, J. S. Kim, R. Azizi, A. Olgun, L. Orosa, H. Hassan, J. Park, K. Kanellopoulos, T. Shahroodi, S. Ghose, and O. Mutlu, "BlockHammer: Preventing RowHammer at Low Cost by Blacklisting Rapidly-Accessed DRAM Rows," in *HPCA*, 2021.

[35] J. M. You and J.-S. Yang, "MRLoc: Mitigating Row-hammering based on memory Locality," in *DAC*, 2019.

ERRATA

Erroneous Claim in Section V: In the last paragraph of Section V, we claimed that *configuring TH_{RH} must take into account the blast radius*. This claim is correct for Rowhammer defenses that rely on counters to track aggressor rows. For such defenses, an attack strategy that uses many aggressor rows is more likely to escape detection (i.e., is more effective) than one using few aggressor rows. Also, the larger the blast radius, the higher the number of aggressor rows that disturb one victim row. Thus, TH_{RH} must be set more conservatively (i.e., to a lower value) in a DRAM device with a large blast radius.

However, our claim is incorrect for Row-Sampling-based Rowhammer defenses. As we argue in Section III, a single-sided attack is the most effective attack strategy for Row-Sampling defenses. Also, an attack strategy that uses many aggressor rows is equally likely to escape detection as one using few aggressor rows. Thus, as long as the NRR command refreshes *all* rows within the blast radius, the TH_{RH} and the sampling rate values do not depend on the blast radius.

The authors would like to thank Michele Marazzi from ETH Zurich who pointed out this error.

NRR Command in DDR5: As of September 2022, the DDR5 specification has introduced a new DRAM command: Directed Refresh Management (DRFM). The functionality of DRFM is equivalent to the NRR command described herein.